

Dieses Skriptum entstand parallel zur Vorlesung und kurz danach mit dem Ziel, daß es mit möglichst geringer Verzögerung verfügbar sein soll. Es ist in seiner Qualität auf keinen Fall mit einem Lehrbuch zu vergleichen; insbesondere sind Fehler bei dieser Entstehensweise nicht nur möglich, sondern **sicher**. Dabei handelt es sich garantiert nicht immer nur um harmlose Tippfehler, sondern auch um Fehler bei den mathematischen Aussagen.

Das Skriptum sollte daher mit Sorgfalt und einem gewissen Mißtrauen gegen seinen Inhalt gelesen werden; falls Sie Fehler finden, teilen Sie mir dies bitte persönlich oder per e-mail (seiler@math.uni-mannheim.de) mit. Auch wenn Sie Teile des Skriptums unverständlich finden, bin ich, auch im Hinblick auf hoffentlich existierende künftige Kryptologie-Vorlesungen, für entsprechende Hinweise dankbar.

Biographische Angaben von Mathematikern beruhen größtenteils auf den entsprechenden Artikeln im *MacTutor History of Mathematics archive* (`www-history.mcs.st-andrews.ac.uk/history/`), von wo auch die meisten abgedruckten Bilder stammen. Bei noch lebenden Mathematikern bezog ich mich, soweit möglich, auf deren eigenen Internetauftritt.

Inhalt

KAPITEL I: AUFGABEN UND UMFELD DER KRYPTOLOGIE 1

§1: Einsatzgebiete der Kryptographie 1
a) Geheimhaltung 2
b) Verfälschungssicherheit 3
c) Sicherheit gegen erneutes Einspielen 3
d) Authentizität 4
e) Beweisbarkeit 4
f) Urheberrechtsschutz 5
g) Kopierschutz 6
h) Dokumentation des Wissensstands 6
i) Rechnen mit verschlüsselten Daten 6

§2: Alternativen zur Kryptographie 7

§3: Das Umfeld der Kryptologie 12

§4: Forderungen an ein Kryptosystem 15
a) Was weiß der Gegner über die Nachricht? 15
b) Was weiß der Gegner über das Kryptoverfahren? 17
c) Was kann der Gegner? 19

§5: Literaturhinweise 24

KAPITEL II: EINIGE KLASSISCHE KRYPTOVERFAHREN 27

§1: Monoalphabetische Substitutionen 27
a) Die Nullchiffre 27
b) Die CAESAR-Chiffre 28
c) Allgemeine monoalphabetische Substitutionen 31

§2: Polyalphabetische Substitutionen 54

§3: Der one time pad 64

§4: Transpositionschiffren 68

§5: Rotormaschinen 72

§6: Literaturhinweise 77

KAPITEL III: KLASSISCHE BLOCKCHIFFREN 79

§1: Anforderungen an eine Blockchiffre 81

§2: Der Aufbau einer Blockchiffre 85

§3: Der Data Encryption Standard DES 88

§4: Designkriterien und Kryptanalyse des DES 95
a) Geschichtliche Entwicklung 95
b) Designkriterien 96
c) Differentielle Kryptanalyse 98
d) Lineare Kryptanalyse 105
e) DES-Cracker 106

§4: Modifikationen 109
a) Mehrfacher DES 109
b) Doppelter DES 110
c) Dreifacher DES 111
d) DESX 112
e) Alternativen zu DES 113

§5: Operationsmodi 113
a) Electronic Code Book (ECB) 113
b) Cipher Block Chaining (CBC) 115
c) Cipher Feedback (CFB) 118
d) Output feedback (OFB) 120
e) Counter mode (CTR) 121

§6: Literatur 122

KAPITEL IV: DAS RSA-VERFAHREN 123
§1: New directions in cryptography 123
§2: Die Grundidee des RSA-Verfahrens 126
a) Allgemeine Vorüberlegungen 126
b) Modulararithmetik 129
c) Potenzfunktionen modulo einer Primzahl 131
d) Der erweiterte EUKLIDische Algorithmus 134
e) Die RSA-Verschlüsselungsfunktion 139
§3: Praktische Anwendung von RSA 142
a) Wie groß sollten die Primzahlen sein? 142
b) Wie werden Nachrichten zu Zahlen? 145
c) Probabilistische Verschlüsselung 149
d) Wie berechnet man die RSA-Funktion effizient? 150
e) Konkrete Implementierungen 152
1.) Maple 153
2.) Maxima 153
3.) Scheme/Racket 154
4.) Java 155
5.) C, C++, ... 156
§4: Was läßt sich mit RSA anfangen? 157
a) Identitätsnachweis 158
b) Elektronische Unterschriften 160
c) Bankkarten mit Chip 161
d) Elektronisches Bargeld 163
§5: Wie findet man Primzahlen für RSA? 165
a) Wie man es nicht machen sollte 166
b) Wie man es idealerweise machen sollte 167
c) Wie dicht liegen die Primzahlen? 169
d) Das Sieb des Eratosthenes 171
e) Der Fermat-Test 172
§6: Sicherheit und Sparsamkeit 176
a) Primzahlen sind Wegwerfartikel 176
b) Jeder braucht seinen eigenen RSA-Modul 177
c) Der chinesische Restesatz 179
d) Kleine öffentliche Exponenten und Kettenbriefe 181
e) Kleine private Exponenten 182
§7: RSA im wirklichen Leben 184
a) Allgemeine Struktur einer public key infrastracture 185
1.) Hierarchische Modelle 185
2.) Grassroot Modelle 185
b) SSL, TLS & Co 186
c) PKCS #1v1.5 189
d) Der Angriff von BLEICHENBACHER 190
e) Elektronische Unterschriften nach PKCS#1 193
f) BLEICHENBACHERs Angriff dagegen 196
§8: Faktorisierungsverfahren 198
a) Mögliche Ansätze zur Faktorisierung 198
b) Das quadratische Sieb 200
c) Varianten des quadratischen Siebs 208
1.) Die Multipolynomialversion 209
2.) Das Zahlkörpersieb 210
d) Faktorisierungsrekorde 211
e) Faktorisierung mit Spezialhardware 214
§9: Literatur 217

KAPITEL V: VERFAHREN MIT DISKRETEN LOGARITHMEN 219
§1: Schlüsselaustausch nach Diffie und Hellman 219
a) Das Verfahren 220
b) Die *man in the middle attack* 221
§2: Verschlüsselung und elektronische Unterschriften 223
a) Verschlüsselung nach ELGAMAL 223
b) Das Verfahren von MASSEY-OMURA 225
c) DSA 227

§3: Strategien zur Berechnung diskreter Logarithmen 230
a) Probieren 230
b) Gruppentheoretische Formulierung des Problems 230
c) Anwendung des chinesischen Restesatzes 234
d) Das Verfahren von POHLIG und HELLMAN 234
e) Folgerung für die Sicherheit von Kryptosystemen 236
f) Baby step und giant step 236
g) Zahme und wilde Kängurus 237
h) Indexkalkül 240
§4: Diskrete Logarithmen in anderen Gruppen 242
a) Die abstrakte Situation 242
b) Multiplikative Gruppen beliebiger endlicher Körper 243
c) Elliptische Kurven 244
§5: Literatur 247

KAPITEL VI: DER ADVANCED ENCRYPTION STANDARD RIJNDAEL . 249
§1: Geschichte und Auswahlkriterien 249
§2: Algebraische Vorbereitungen 252
a) EUKLIDische Ringe 252
b) Endliche Körper von Primzahlpotenzordnung 256
c) Der Körper mit 256 Elementen 259
§3: Spezifikation von Rijndael 261
a) Terminologie und Bezeichnungen 261
b) Die Grundoperationen 261
c) Der Aufbau der Runden 263
1.) Die Bytesubstitution 264
2.) Die Zeilenshifts 266
3.) Der Spaltenmix 266
4.) Schlüsselexpansion und Rundenschlüssel 267
d) Gesamtablauf von Rijndael 268
e) Geschwindigkeitsoptimierung 268
§4: Angriffe auf Rijndael 270
§5: Literatur 272

KAPITEL VII: KRYPTOGRAPHISCH SICHERE HASHVERFAHREN 273
§1: Nochmals elektronische Unterschriften 273
§2: Das Geburtstagsparadoxon 274
§3: Die Familie der SHA-Algorithmen 277
§4: Weitere Anwendungen sicherer Hashfunktionen 285
a) Schutz der Integrität von Daten 285
b) Wie zufällig müssen unsere Schlüssel sein? 286
c) Erzeugung großer Zufallszahlen aus kleinen 287
d) Primzahlen für DSA 288
e) Wie bekommt man echte Zufallszahlen? 290
§5: Literatur 292

KAPITEL VIII: KRYPTOGRAPHISCHE PROTOKOLLE 293
§1: Werfen einer Münze per Telephon 294
§2: Poker per Telephon 296
§3: Zero Knowledge Protokolle 299
§4: Schlußbemerkung 303
§5: Literatur 304

KAPITEL IX: KRYPTOLOGIE UND QUANTENPHYSIK 305
§1: Grundzüge der Quantenmechanik 306
§2: Quantenkryptographie 312
a) Informationsübertragung mit einzelnen Photonen 313
b) Protokolle zur Quantenkryptographie 316
c) Angriffsmöglichkeiten 319
d) Fehlerkorrektur 321
e) Elimination der gegnerischen Information 323
§3: Quantencomputer 324
a) Quantenregister und QBits 324
b) Quantencomputer 326
c) Der Algorithmus von SHOR 328

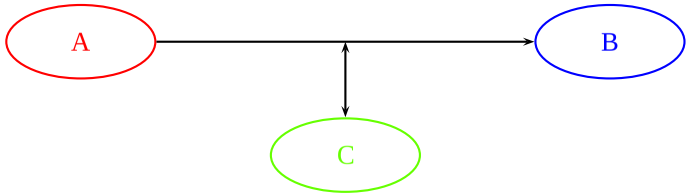
d) Was können Quantencomputer? 335
e) Experimentelle Realisierung 338
§4: Andere nichtkonventionelle Rechnerarchitekturen 339
a) Die Desoxyribonucleinsäure 341
b) Die Polymerase-Kettenreaktion 343
c) ADLEMANs Experiment 344
d) Wie geht es weiter? 347
e) Literaturhinweise 350

Kapitel 1
Aufgaben und Umfeld der Kryptologie

§ 1: Einsatzgebiete der Kryptographie

Kryptologie ist zusammengesetzt aus den beiden griechischen Wörtern κρυπτός = verborgen, versteckt und λόγος = Rede, Darlegung, Vernunft; sie ist also die Wissenschaft vom Geheimen. Sie besteht aus der Kryptographie (von γράφή = Das Schreiben), die Geheimschriften entwickelt, und der Kryptanalyse (von ἀναλύειν = auflösen, zerlegen), die versucht, letztere zu analysieren mit dem Ziel, sie zu knacken.

Die Grundsituation ist also die folgende:



A möchte eine Nachricht m an B übermitteln, jedoch besteht die Gefahr, daß alles, was er an B schickt, auf dem Weg dorthin von C gelesen und vielleicht auch verändert wird; außerdem könnte C eventuell versuchen, sich gegenüber B als A ausgeben oder umgekehrt.

Der Ausweg besteht darin, daß A anstelle von m eine Nachricht c schickt, aus der zwar B, nicht aber C die Nachricht m und gegebenenfalls weitere Informationen rekonstruieren kann.

Aufgabe der Kryptographie ist es, in dieser oder einer ähnlichen Situation eines oder mehrere der Ziele aus folgender Liste zu erreichen; Aufgabe der Kryptanalyse ist es, dies zu verhindern.

a) GEHEIMHALTUNG: Sie muß sicherstellen, daß zwar B, nicht aber C in der Lage ist, die Originalnachricht m aus c zu rekonstruieren.

b) VERFÄLSCHUNGSSICHERHEIT: Sie muß sicherstellen, daß C den übertragenen Text c nicht unbemerkt durch einen Text c' ersetzen kann, den B dann als eine Nachricht m' rekonstruiert.

c) SICHERHEIT GEGEN ERNEUTES EINSPIELEN: Sie muß sicherstellen, daß C den übertragenen Text c nicht unbemerkt ein zweites Mal in die Übertragung einspielen kann, so daß B glaubt, A habe ihm die Nachricht m zweimal geschickt.

d) AUTHENTIZITÄT: B muß sicher sein, daß die Nachricht m tatsächlich von A kam und nicht von C. Gelegentlich ist das sogar die einzige wesentliche Aufgabe der Kryptographie nämlich dann, wenn sie etwa bei Bankkarten oder Zugangskontrollsystemen speziell zur Identifikation berechtigter Personen eingesetzt wird.

e) BEWEISBARKEIT: B muß einem Dritten gegenüber beweisen können, daß die Nachricht m von A kam und nicht von C *oder ihm selbst* geschrieben wurde.

f) URHEBERRECHTSSCHUTZ: A muß einem Dritten gegenüber beweisen können, daß die Nachricht m ursprünglich von ihm kommt und nicht von C, der sie später kopiert hat.

g) KOPIERSCHUTZ: B soll zwar in der Lage sein, die Nachricht m aus c zu rekonstruieren, er darf aber nicht in der Lage sein, m an einen Dritten weiterzugeben.

h) DOKUMENTATION DES WISSENSSTANDS: Gelegentlich soll B gar nicht in der Lage sein, die Nachricht m zu rekonstruieren, aber A möchte zu einem späteren Zeitpunkt beweisen können, daß er zum Zeitpunkt des Absendens von c die Nachricht m kannte.

i) RECHNEN MIT VERSCHLÜSSELTEN DATEN: Hier soll B für A eine Rechnung ausführen, ohne die verwendeten Daten oder das Ergebnis kennenzulernen.

Betrachten wir diese Aufgaben etwas genauer.

a) Geheimhaltung

Dies ist die älteste unter den Aufgaben der Kryptographie und zugleich auch die, für die meisten Verfahren entwickelt wurden.

Heute unterscheiden wir vor allem zwei Arten von Verschlüsselungen:

- Bei der symmetrischen Kryptographie ist die Kenntnis des Verschlüsselungsalgorithmus äquivalent zu der des Entschlüsselungsalgorithmus
- Bei der asymmetrischen Kryptographie kann der Entschlüsselungsalgorithmus nicht mit vertretbarem Aufwand aus dem Verschlüsselungsalgorithmus abgeleitet werden, so daß letzterer öffentlich bekannt sein darf.

Was das im einzelnen bedeutet und welche Vor- und Nachteile die beiden Ansätze haben, wird uns im folgenden noch ausführlich beschäftigen.

b) Verfälschungssicherheit

Im elektronischen Zahlungsverkehr zwischen Banken legen natürlich alle Beteiligten größten Wert auf Geheimhaltung; noch wichtiger ist aber, daß die übertragenen Nachrichten nicht verfälscht werden, daß also aus einem Zahlungsauftrag über zehn Euro keiner über zehn Tausend Euro werden kann. Da alles weitgehend automatisch verläuft, müssen alle übertragenen Nachrichten in einem starr vorgegebenen normierten Format abgefaßt sein, und dieses Format läßt sich schon wegen der Größe des Bankennetzwerks nicht geheimhalten. Ohne zusätzliche Sicherungsmaßnahmen würde selbst eine zufällige Veränderung dieses Felds erheblichen Schaden anrichten. Eine gewisse Verfälschungssicherheit ist gegeben, wenn das verwendete Verschlüsselungsverfahren bei Manipulationen des Chiffretexts bei Entschlüsselung mit hoher Wahrscheinlichkeit keinen vernünftigen Klartext liefert, allerdings nur dann, wenn außer Sender und Empfänger niemand in der Lage ist, einen Text zu verschlüsseln. Bei Verwendung eines asymmetrischen Kryptoverfahrens braucht man auf jeden Fall zusätzliche Maßnahmen wie etwa kryptographisch sichere Prüfsummen oder ähnliches.

c) Sicherheit gegen erneutes Einspielen

Speziell der elektronische Zahlungsverkehr bietet auch ein Beispiel

dafür, daß eine Nachricht weder verstanden noch verfälscht werden muß, um damit Schaden anzurichten: Wenn etwa eine Zahlungsanweisung zugunsten des Lauschers von einer Clearingstelle an dessen Bank geschickt wird, kann dieser sie anhand von Zeitpunkt und Absender/Empfänger mit relativ hoher Wahrscheinlichkeit identifizieren. Falls er sie unbemerkt später noch einmal einspielt, muß verhindert werden, daß ihm die Bank das Geld ein zweites Mal gutschreibt. Dazu muß die Nachricht beispielsweise eine eindeutige und nicht verfälschbare Transaktionsnummer enthalten, anhand derer Dubletten erkannt werden.

d) Authentizität

Nicht nur bei Zahlungsanweisungen ist es oft von entscheidender Bedeutung, wer der Absender der Nachricht ist. Bei einem symmetrischen Kryptoverfahren, bei dem die genaue Ver- und Entschlüsselungsfunktion nur dem Absender und dem Empfänger bekannt sind und bei dem nur ein vernachlässigbarer Bruchteil aller theoretisch möglichen Chiffre-Nachrichten auf eine sinnvolle Entschlüsselung führt, kann sich der Empfänger einer sinnvollen Nachricht ziemlich sicher sein, daß eine nicht von ihm selbst produzierte Chiffre vom Absender stammt; bei asymmetrischen Kryptoverfahren müssen andere Wege gefunden werden.

e) Beweisbarkeit

Gelegentlich muß der Empfänger nicht nur selbst überzeugt sein, daß eine Nachricht wirklich vom angegebenen Absender stammt, sondern er muß dies auch gegenüber einem Dritten beweisen können, beispielsweise wenn der Absender eine eingegangene Verpflichtung nicht erfüllen will. Hier bietet der gerade skizzierte Einsatz eines symmetrischen Kryptoverfahrens keinen Schutz, denn der Absender kann ja jederzeit behaupten, der Empfänger habe die Nachricht selbst geschrieben. Wie wir sehen werden, kann man aber beispielsweise durch Vertauschung der Rollen von Ver- und Entschlüsselungsfunktion eines asymmetrischen Kryptosystems sogenannte *elektronische Unterschriften* erzeugen (die in Deutschland rechtsgültig sind).

f) Urheberrechtsschutz

Manchmal möchte der Absender später beweisen können, daß der Inhalt der Nachricht (etwa die Idee für ein neues Produktionsverfahren oder ein Musikstück) von ihm stammt; insbesondere möchte er den Empfänger daran hindern, es als eigene Leistung auszugeben oder unbefugt zu verbreiten. Dazu dienen meist sogenannte „Wasserzeichen“, d.h. Zusatzinformationen, die unsichtbar mit der Nachricht verknüpft sind. Die Techniken dazu stammen oft aus der sogenannten *Steganographie*, mit der wir uns im nächsten Paragraphen beschäftigen werden.

Gelegentlich werden sie auch dazu eingesetzt, Lecks zu finden: So soll etwa die damalige britische Premierministerin MARGARET THATCHER in den achtziger Jahren angeordnet haben, die Word Prozessoren ihrer Minister und engsten Mitarbeiten so umzuprogrammieren, daß jeder durch geringfügige Variationen im Zeilenvorschub einem Fachmann den Rückschluß auf den Autor gestattete. (Word Prozessoren waren elektrische Schreibmaschinen mit einem Mikroprozessor und einem Speichermedium, die einen Teil jener Grundfunktionen beherrschten, die heute in jedem Textverarbeitungsprogramm selbstverständlich sind.) Natürlich funktionierte die Identifikation des Autors nur, wenn das Original vorlag, aber die meisten Zeitungen druckten solche Dokumente im Faksimile ab um zu zeigen, daß wirklich alles echt war. Heute tippen Plattformen wie WikiLeaks alle geheimen Dokument neu, um solche Ansätze zu unterlaufen. Die Verfasser können freilich zumindest bei manchen Dokumenten auch leichte semantische Änderungen vornehmen; beispielsweise ist von der österreichischen Telephongesellschaft bekannt, daß sie in jedes Telephonbuch auch einen nicht existierenden Teilnehmer aufnahm, um so Plagiate zu enttarnen. Eine entsprechende Taktik bei internen Handbüchern mit in jedem Exemplar verschiedenem Zusatztext könnte wieder zur Enttarnung von Lecks führen.

Ein Hauptproblem guter Wasserzeichen besteht daher darin, daß sie robust sein müssen: Das Wasserzeichen in einem digitalen Musikstück sollte also beispielsweise im Idealfall auch in einer analogen Kopie noch nachweisbar sein; moderne Kryptoverfahren können auch das gewährleisten.

g) Kopierschutz

Früher gab es spezielle Farbstifte, deren Schrift für Schwarz/Weiß-Kopierer nicht lesbar war; heute haben Farbkopierer spezielle Software, die dafür sorgt, daß keine Geldscheine kopiert werden. Eine Computerdatei dagegen kann beliebig oft kopiert und an andere weitergegeben werden. Die einzige Möglichkeit für einen effizienten Kopierschutz besteht daher darin, die Informationen nur in verschlüsselter Form zur Verfügung zu stellen. Da auch jedes Entschlüsselungsprogramm problemlos kopiert werden könnte, muß die Entschlüsselung durch Spezialhardware erfolgen. Diese muß auch gegen Logikanalysatoren resistent sein, beispielsweise weil kritische Schlüssel in auslesesicheren Registern gespeichert sind. Da dies viel Aufwand erfordert, sind die meisten real existierenden Kopierschutzschemata nicht sonderlich effektiv.

h) Dokumentation des Wissensstands

Wer ein Patent anmeldet, muß sein Verfahren offenlegen, zahlt hohe Gebühren, und spätestens nach siebzehn Jahren kann es jeder frei nutzen. Wer allerdings kein Patent anmeldet, muß damit rechnen, daß ein anderer dieselbe Idee hat, diese patentieren läßt, woraufhin er selbst dann sein Verfahren nicht mehr oder nur noch nach Zahlen von Lizensgebühren anwenden darf. Ein solches Patent wird dem Konkurrenten allerdings nicht erteilt, wenn jemand nachweisen kann, daß dieser nicht der erste war, der die Idee hatte. Das sogenannte *time stamping* ist das digitale Analogon einer Stechuhr: Sie kann ein Dokument beweisbar einem Zeitpunkt zuordnen, ohne daß es einem Dritten bekanntgegeben werden muß.

i) Rechnen mit verschlüsselten Daten

Die meisten Computer haben die meiste Zeit fast nichts zu tun; nur selten fallen umfangreiche Rechnungen an, mit denen sie dann sogar eher überfordert sind. Daher liegt es nahe, alle Rechner eines Unternehmens zu einem sogenannten *grid* zusammenzufassen und anfallende Aufgaben jeweils auf solche Rechner zu verteilen, die gerade sonst nichts zu tun haben. Dabei muß man sich nicht unbedingt auf ein Unternehmen beschränken; beim *cloud computing* stellt ein externer Anbieter verschiedenen Unternehmen bedarfsorientiert Rechnerkapazität zur Verfügung.

Zumindest sensible Daten sollten dabei, wenn sie überhaupt in der *cloud* verarbeitet werden, vor dem Anbieter geschützt werden. Dazu könnte beispielsweise ein *homomorphes Verschlüsselungsverfahren* verwendet werden, das mit allen Rechenoperationen kompatibel ist. Solche Verfahren gibt es bereits; sie sind allerdings deutlich aufwendiger als die meisten Rechnungen, die man anschließend mit den Daten durchführen möchte, so daß dies heute noch nicht praktikabel ist.

§ 2: Alternativen zur Kryptographie

Am einfachsten läßt sich eine Nachricht geheim halten, wenn es gelingt, schon ihre bloße Existenz zu verschleiern. Entsprechende Techniken bezeichnet man als *Steganographie* von στεγανός = schützend, verdeckt.

Über die beiden wohl ältesten bekannten Anwendungen der Steganographie berichtet HERODOT (∼484 v.Chr.–∼424 v.Chr.) in seinen *Historien*. Die erste Episode ist aus der Zeit des ionischen Aufstands (500 v.Chr.) der kleinasiatischen und der zyprischen Griechen gegen die persische Oberherrschaft. Zur Vorbereitung schickte der Extyrann von Milet, HISTIAIOS (*vor* 520 v.Chr.–493 v.Chr.), der am persischen Hof in Susa lebte, eine Nachricht an seinem Nachfolger und Schwiegersohn ARISTAGORAS (*gefallen* 497). HERODOT schreibt dazu (Buch V, 35):

> Gerade damals kam nämlich auch jener Bote mit dem beschriebenen Kopf aus Susa an, den HISTIAIOS geschickt hatte, um ARISTAGORAS zum Abfall von dem König zu bewegen. Denn HISTIAIOS fand, weil alle Straßen bewacht wurden, kein anderes sicheres Mittel, ARISTAGORAS zum Abfall zu ermutigen, als seinem getreuesten Sklaven das Haar zu scheren, Zeichen auf seinen Kopf zu schreiben, das Haar wieder wachsen zu lassen und ihn dann nach Milet zu schicken. Der Sklave hatte bloß den Auftrag, ARISTAGORAS in Milet zu bitten, ihm das Haar zu scheren und seinen Kopf zu betrachten. Die Zeichen auf dem Kopf aber mahnten, wie ich schon sagte, zum Abfall.

Die zweite Episode ist im siebten Buch der Historien zu finden: Der 491 v.Chr. von seinem Mitkönig KLEOMENES abgesetzte und im persischen Exil lebende Ex-König DEMARATOS von Lakedaimon (Sparta)

erfuhr von einer geplanten Aufrüstung der Perser für einen Feldzug gegen Griechenland. HERODOT schreibt (Buch VII, 239):

> DEMARATOS, der Sohn des ARISTON, der nach Persien entflohen war, war den Lakedaimoniern, wie ich glaube und wie die Umstände nahelegen, nicht eben wohlgesinnt; man kann daher auch annehmen, daß er nicht aus Wohlwollen, sondern aus Schadenfreude gehandelt hat. Genug, DEMARATOS, der in Susa lebte und wußte, daß XERXES den Zug gegen Hellas im Sinne hatte, wollte den Lakedaimoniern Kunde davon geben. Weil sich dies auf andere Weise nicht bewerkstelligen ließ – er mußte die Entdeckung fürchten – verfiel er auf folgenden Gedanken: Er nahm eine doppelte Schreibtafel und schabte den Wachsüberzug ab. Dann schrieb er auf das Holz des Täfelchens den Plan des Königs und überzog die Schriftzüge wieder mit dem Wachs. Das leere Täfelchen sollte den Wächtern an der Straße keinen Argwohn erwecken. Als die Tafel nach Lakedaimon gelangte, verstanden die Lakedaimonier nicht, was die leere Tafel bedeuten sollte. KLEOMENES' Tochter GORGO, LEONIDAS' Gemahlin, war es endlich, wie man mir erzählt, die den Sinn erriet. Sie sagte, man solle das Wachs abschaben; dann werde man auf dem Holz die Buchstaben finden. Sie taten es, fanden die Botschaft und lasen sie, teilten sie dann auch den übrigen Hellenen mit. So soll sich jene Kunde verbreitet haben.

(zitiert nach A. HORNEFFERS *Übersetzung der* Historien, *erschienen als* Kröners Taschenausgabe 224, *Kröner Verlag Stuttgart, 1955)*

Beides Mal war die Steganographie kriegsentscheidend: Der ionische Aufstand war erfolgreich, und die Griechen begannen nach Erhalt der Nachricht von DEMARATOS, ihrerseits eine Flotte zu bauen. Als die Flotte des Perserkönigs XERXES schließlich fertig war und er seinen vermeintlichen Überraschungsangriff startete, waren die Griechen gut vorbereitet und konnten die Perser zwar nur mit Glück (der Wind wehte in die richtige Richtung), dafür aber umso vernichtender schlagen.

Zumindest im zweiten Fall freilich hing der Erfolg davon ab, daß zwar GORKO auf die Idee kam, das Wachs von der Tafel abzukratzen, nicht

aber einer der Wächter. Sobald jemand die Existenz einer Nachricht vermutet, wird er sie mit ziemlicher Sicherheit auch finden.

Aus diesem Grund wird Steganographie oft mit einer Verschlüsselung kombiniert. Ein Beispiel, das zwar eher der klassische Kryptographie als der Steganographie zuzurechnen ist, bietet das mesopotamische Arzneimittellexikon *Uruanna*, das im Auftrag des letzten assyrischen Herrschers ASSURBANIPAL (†627 *v.Chr.*) zusammengestellt wurde. Dort findet man Rezepte der Art *Menschenkot verarbeitest Du mit dem Urin eines Hundes zu einem Brei und verbindest [den Patienten] damit.* Bei Ausgrabungen wurde aber auch eine zweispaltige Tafel gefunden, die jeweils in der linken Spalte den Namen einer Pflanze enthielt und in der rechten ein Wort wie *Menschenkot, Fledermauskopf, Taubendreck, ...* Ganz offensichtlich waren diese Wörter also Chiffren für Heilpflanzen. Trotz des Ekels, den die wörtlich genommenen Rezepte verursachen, war der steganographische Effekt aber so groß, daß die Rezepte über die Jahrtausende tradiert und als „Weisheit der Alten“ sogar praktiziert wurden. *(s. Bild der Wissenschaft, Heft 6/2007, S. 40–41)*

Im 18. Jahrhundert sehr populär war beispielsweise KRISTIAN FRANTZ PAULINIS *Heylsame Dreck-Apothecke*, wo es unter anderem heißt:

> Im Koth und im Urin liegt GOTT und die Natur. Kuhfladen können dir weit mehr als Balsam nützen. Der blosse Gänsedreck geht Mosch und Ambra für. Was Schätze hast du offt im Kehricht und Mistpfützen. Der beste Theriak liegt draußen vor der Thür. *(zitiert nach* Deutsches Ärzteblatt **101**, *Ausgabe 47 vom 19.11.2004, Seite A-3184)*

Auch einige heutige Bücher mit Titeln wie *Lebenssaft Urin* oder *Gesund durch Eigenharn* könnten ihren Ursprung letztlich in dieser erfolgreichen Steganographie haben.

Heute sind Nachrichten auf der Kopfhaut oder unter der Wachsschicht einer antiken Tafel sicherlich keine attraktiven Alternativen zu einer E-Mail; dafür bieten Computer aber ganz neue Möglichkeiten zur Steganographie:

Speichert man etwa Bild- oder Audiodaten in nichtkomprimierter Form, ändert es im allgemeinen den visuellen oder auditorischen Eindruck



nicht, wenn man das letzte Bit des digitalisierten Werts verändert: Wie Experimente zeigen, kann unser Auge nicht mehr als etwa 64 verschiedene Grauwerte unterscheiden; da Grauwerte aber üblicherweise als Bytes und damit mit 256 möglichen Werten abgespeichert werden, läßt sich problemlos das letzte Bit oder gar Bitpaar zur Übertragung zusätzlicher Information verwenden – zumindest solange dies niemand vermutet: Die Korrelation zwischen den Endbits benachbarter Bytes ist bei echten Bild- oder Audiodaten im Falle exakter Abtastung erheblich kleiner als beim Aufmodulieren einer steganographischen Nachricht; bei einer verrauschten Abtastung dagegen wird sie wohl eher größer sein.

Auch in Texten lassen sich Nachrichten verstecken; unter dem URL `www.spammimic.com` etwa kann man eine Nachricht in eine *spam*-Email einbetten, die sich wahrscheinlich niemand genauer ansehen möchte. Aus *Uruanna* beispielsweise wurde

```
Dear Business person ; Your email address has been
```

submitted to us indicating your interest in our newsletter ! If you no longer wish to receive our publications simply reply with a Subject: of "REMOVE" and you will immediately be removed from our database . This mail is being sent in compliance with Senate bill 1622 , Title 5 , Section 305 . THIS IS NOT A GET RICH SCHEME ! Why work for somebody else when you can become rich in 88 MONTHS ! Have you ever noticed nobody is getting any younger plus nobody is getting any younger ! Well, now is your chance to capitalize on this ! WE will help YOU decrease perceived waiting time by 200% and turn your business into an E-BUSINESS ! The best thing about our system is that it is absolutely risk free for you ! But don't believe us . Mr Ames of Massachusetts tried us and says "My only problem now is where to park all my cars" ! We are licensed to operate in all states ! We beseech you - act now . Sign up a friend and your friend will be rich too ! Thank-you for your serious consideration of our offer !

Gibt man diese Nachricht auf www.spammimic.com/encode.shtml ein, erhält man wieder das Wort Uruanna.

Die Steganographie ist nicht die einzige Methode, um ohne spezielle Geheimschrift Nachrichten geheim zu halten: Auch eine den zu erwartenden Gegnern unbekannte natürliche Sprache oder Schrift kann diese Funktion erfüllen.

Im alten China beispielsweise, wo fast niemand lesen und schreiben konnte, war die gewöhnliche Schrift schon geheim genug; spezielle Geheimschriften wurden dort nie entwickelt. (Als Schutz vor Lesekundigen war allerdings eine Form der Steganographie gebräuchlich: Die Nachricht wurde auf ein Seidentuch geschrieben und dieses zusammengeknüllt und mit Wachs umhüllt, so daß es aussah wie eine einfache Wachskugel.)

Heute gibt es in jedem Staat einen nicht vernachlässigbaren Prozentsatz von Bürgern, die lesen und schreiben können; mit wenig bekannten

Sprachen konnte man aber auch im zwanzigsten Jahrhundert selbst im jahrelangen Großeinsatz noch Erfolge erzielen: Der einzige amerikanische Code im zweiten Weltkrieg, den die Japaner nie knacken konnten, war der der Navajos.

Die Navajos waren einer der wenigen Stämme, die noch nie Kontakte zu deutschen Forschern gehabt hatten (Japan und Deutschland kämpften im zweiten Weltkrieg auf derselben Seite), und ihre Sprache ist mit keiner europäischen oder asiatischen Sprache verwandt. Nach damaligen Schätzungen gab es weniger als dreißig Nicht-Navajos, die diese Sprache beherrschten. Die meisten von ihnen waren als Kinder von Missionaren gemeinsam mit Navajo-Kindern aufgewachsen; keiner war Japaner oder Deutscher. Außerdem gab es zu dieser Sprache keine Schrift, und sie ist so kompliziert, daß es für einen Erwachsenen praktisch unmöglich ist, sie zu lernen: Zum einen ist die Grammatik sehr komplex, zum anderen hat – wie im Chinesischen, nicht aber im Japanischen – derselbe Laut je nach Ton völlig verschiedene Bedeutungen.

Entsprechend erwies sich 1960, beim damaligen UN-Einsatz im vormals belgischen Kongo das Gaelisch der irischen Soldaten als die effektivste Kryptographie.

Für die Hauptlast der heutigen Kryptographie freilich, die Kommunikation und den Handel über das Internet, sind solche Verfahren nicht tauglich; hier geht nichts ohne „Geheimschriften", d.h. ohne eine algorithmische Transformation der Nachricht m in einen Chiffretext c. Dies und mögliche Angriffe dagegen wird daher den Hauptinhalt dieser Vorlesung ausmachen.

§ 3: Das Umfeld der Kryptologie

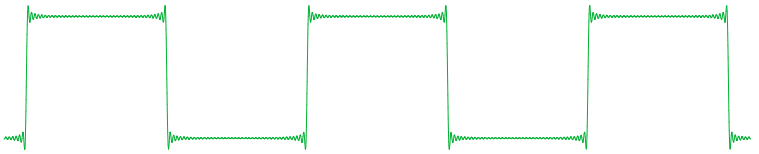
Auch das beste Kryptoverfahren ist nutzlos, wenn der Gegner die Entschlüsselungsfunktion kennt oder sich den Klartext auf andere Weise unabhängig vom Chiffretext verschaffen kann. Die Übertragung einer Nachricht geht über eine ganze Reihe von Schritten, und ein etwaiger Gegner kann sich aussuchen, wo er angreifen will. Natürlich wird es sich immer das aus seiner Sicht schwächste Glied der Kette aussuchen.

Zusätzlich zur Kryptanalyse hat er beispielsweise folgende Möglichkeiten:

1. Durch Bestechung oder sogenanntes *human engineering* oder *social engineering*, d.h. durch Ausnutzen der Dummheit und/oder Naivität von Mitarbeitern im Umfeld des Absenders (oder auch von diesem selbst!) kann er versuchen, den Inhalt wichtiger Nachrichten zu erfahren, bevor diese auch nur abgeschickt werden. Auch durch Abhören von Telephonen, Einbrüche *usw.* kann er Informationen gewinnen.
2. Mit denselben Methoden oder durch klassisches Hacken kann er sich Zugriff auf den Computer des Absenders verschaffen und dafür sorgen, daß entweder die unverschlüsselte Nachricht oder alle im Computer gespeicherten Schlüssel an ihn geschickt werden. Einen gewissen Schutz dagegen bieten nur Betriebssysteme einer hohen Sicherheitsklasse, und das sind nicht die, mit denen neue Rechner standardmäßig ausgeliefert werden.
3. Auch wenn er nicht bis zum Computer vordringen kann und auch keinen freiwilligen oder unfreiwilligen Komplizen in dessen Nähe hat, kann er versuchen, den Bildschirminhalt zu lesen: Die Pixel werden im Prinzip geschaltet durch Rechteckimpulse, da Leitungen für hochfrequente Ströme jedoch nicht nur einen OHMschen Widerstand, sondern auch eine Kapazität haben, fungieren sie als RC-Kreis und damit als ein sogenannte Tiefpaßfilter. Durch das Abschneiden der hohen Frequenzen entstehen an den Flanken der Rechtecke Überschwingungen (GIBBS-Phänomen), die auch noch in einer Entfernung von etwa fünfzig Metern mit einer Antenne aufgefangen werden können und die Rekonstruktion des Bildschirminhalts gestatten. Schutz dagegen ist nur durch aufwendige physikalische Abschirmungsmaßnahmen möglich: Der gesamte Computer muß in einem FARADAYscher Käfig sitzen und alle Kabel müssen abgeschirmt sein.

Beim Empfänger entstehen natürlich wieder im wesentlichen genau dieselben Probleme.

Betrachten wir zur Illustration die wesentlichen Schritte auf dem Weg einer Textnachricht von einem Absender A zu einem Empfänger B im



Das Gibbs-Phänomen für Rechteckimpulse

Hinblick auf Angriffsmöglichkeiten eines Gegners C:

1. Möglicherweise hat sich A bereits Notizen über den vorgesehenen Inhalt der Nachricht gemacht; fallen diese irgendwie vorher oder nachher (Suche im Abfall) in die Hände von C, kennt dieser zumindest den wesentlichen Inhalt der Nachricht.
2. Wenn A hinreichend bedeutend ist, diktiert er die Nachricht einer Sekretärin oder einem persönlichen Referenten. Falls C die Fensterscheiben mit einem Laserstrahl abtastet oder gar ein Mikrophon oder einen Spion im Raum plazieren konnte, oder aber die Sekretärin gekauft hat, kennt er die Nachricht.
3. Während A tippt oder tippen läßt, erscheint die Nachricht auf dem Bildschirm. Falls Bildschirm, Computer und Tastatur nicht aufwendig abgeschirmt sind, kann C mit einer nicht garzu weit entfernten Antenne die Signale auffangen und die Nachricht rekonstruieren. Falls ein Trojaner auf dem Computer aktiv ist, kann dieser den Klartext der Nachricht weiterleiten.
4. Nächster (fakultativer) Schritt ist die Quellenkodierung (oder Datenkomprimierung): Zumindest lange Nachrichten mit vielen Anhängen sollten zwecks besserer Ausnutzung der Kanalkapazität komprimiert werden; wie wir bald sehen werden, erhöht das auch zumindest prinzipiell die kryptographische Sicherheit. Falls freilich das Komprimierungsprogramm ein Freeware-Programm von *Mob Enterprises Central Europe Ltd* ist, wird es vielleicht auch noch zusätzlich die Nachricht an C weiterleiten. Selbst wenn das Programm während der *Woche der Sicherheit* im Rahmen der Aktion *Die Kriminalpolizei rät* erworben wurde, besteht ein gewisses Restrisiko, daß es sich dabei vielleicht um einen sogenannten SCHÄUBLE-Trojaner handelt.
5. Nun wird die Nachricht verschlüsselt. Die Kryptologie ist für die Sicherheit des Verschlüsselungsverfahrens verantwortlich; falls die-

se nicht ausreicht, kann C entschlüsseln. Beim Programm, das die Verschlüsselung durchführt, hat A dieselben Probleme wie bei dem zur Quellenkodierung.

6. Da kein Übertragungskanal perfekt ist, folgt als nächstes meist noch eine Kanalkodierung, d.h. die Anwendung eines fehlerkorrigierenden oder zumindest fehlererkennenden Codes. Falls dieses Programm ein Trojaner ist, der den Computer durchsucht, haben wir auch hier dieselben Risiken; die Umsetzung selbst ist problemlos, da sie auf bereits verschlüsselten Text angewandt wird.
7. Die Nachricht wird übertragen. C kann sie auffangen und die (nicht geheime) Kanalkodierung rückgängig machen; danach hat er ein kryptanalytisches Problem zu lösen.
8. Die Nachricht kommt beim Empfänger B an, und die Schritte 1–6 werden in umgekehrter Reihenfolge rückgängig gemacht. An den Sicherheitsproblemen ändert sich dabei nichts wesentliches.

Hier in der Vorlesung geht es ausschließlich um Sicherheit gegen einen möglichen Angriff in Schritt 7, nicht aber um solche gegen die anderen Schritte oder gar die sicherlich vielfachen weiteren Möglichkeiten. Alle Hörer müssen sich daher bewußt sein, daß sie auf Kryptographie allein kein Sicherheitskonzept aufbauen können und daß selbst dann, wenn sie selbst alle Probleme perfekt gelöst haben (so dies möglich sein sollte), ein einziger Fehler beim Kommunikationspartner alle Bemühungen zunichte machen kann.

§4: Forderungen an ein Kryptosystem

Unser Sicherheitsstandard sollte klar sein: Der Gegner darf nicht in der Lage sein, aus dem Chiffretext den Klartext zu rekonstruieren. Das Problem an diesem einfach klingenden Satz ist die Formulierung „darf nicht in der Lage sein“: Da der Gegner in der Wahl seiner Mittel frei ist, wissen wir weder, was er weiß, noch was er kann, noch was er tut.

a) Was weiß der Gegner über die Nachricht?

Spontan würde man wohl sagen, daß ihm nur der Chiffretext zur Verfügung steht; in der Kryptographie redet man dann von einem *Angriff nur mit Chiffretext*.

Eine Sicherheit nur gegen diese Art von Angriffen war allerdings noch nie in der Geschichte der Kryptographie akzeptabel: Wenn sich jemand die Mühe macht, eine Nachricht abzufangen und in eine (bei guter Kryptographie) aufwendige Kryptanalyse einsteigt, wird er sicherlich gewisse Vorkenntnisse über den Inhalt der Nachricht haben. Die klassischen Anwendungen der Kryptographie beschränkten sich bis vor etwa fünfzig Jahren hauptsächlich auf den militärischen und diplomatischen Bereich; beide sind eher nicht bekannt für große Individualität und Phantasie. Ein einigermaßen mit den Verhältnissen vertrauter Gegner kann mit ziemlich hoher Sicherheit erraten, womit die Nachricht beginnt (Oberstleutnant Knedderle im Generalstab der vierunddreißigsten Infanteriedivision an . . .) und endet. Bei den heute dominierenden Anwendungen im Bankenbereich und im Internet läuft die Kryptographie weitgehend unbemerkt vom Anwender im Hintergrund ab und muß daher, um von Computern allein verstanden zu werden, mit stark formalisierten Nachrichtenformaten arbeiten. Deren Spezifikation findet man in RFCs und ähnlichen Dokumenten, die sich jedermann mühelos verschaffen kann. Man muß daher realistischerweise davon ausgehen, daß ein Gegner Teile des Klartexts kennt, und man muß fordern, daß ihm dies nicht dabei hilft, auch die restlichen Teile der Nachricht zu entschlüsseln oder gar die gesamte Entschlüsselungsfunktion zu rekonstruieren. Wir brauchen also auch *Sicherheit gegen Angriffe mit bekanntem Klartext*.

Zumindest seit der Verbreitung von Chipkarten müssen wir dem Gegner sogar noch mehr Möglichkeiten zubilligen: Die Möglichkeit, frei gewählte Chiffretexte zu entschlüsseln. Da Kryptographie heutzutage nicht mehr mit Papier und Bleistift durchgeführt wird, heißt das konkret, daß der Gegner sich für eine gewisse Zeit in Besitz einer Entschlüsselungsmaschine oder Chipkarte setzen und frei über diese verfügen kann. Da jeder vernünftige Mensch seine Schlüssel ändert sobald er so etwas bemerkt, muß der Gegner die entwendete Hardware wieder unbemerkt zurückgeben; die Kenntnis, die er zwischenzeitlich gewonnen hat, kann ihm aber niemand nehmen. Damit auch künftige verschlüsselte Nachrichten sicher sind brauchen wir also fast immer auch noch *Sicherheit gegen Angriffe mit frei wählbarem Chiffretext* – wobei der aktuelle Chiffretext natürlich ausgeschlossen ist: Fällt er in die Hand des Gegners,

während dieser die Möglichkeit zur Entschlüsselung hat, ist er kompromittiert. Jeder spätere Text muß aber sicher sein.

b) Was weiß der Gegner über das Kryptoverfahren?

Idealerweise natürlich nichts. Aber das ist noch unrealistischer als die Annahme, daß er nichts über den Klartext weiß: Wie schon AUGUSTE KERCKHOFFS 1883 in seiner grundlegenden Arbeit *La cryptographie militaire* feststellte, muß man bei jedem in größerem Umfang eingesetzten Verfahren davon ausgehen, daß es sich nicht über einen längeren Zeitraum hinweg geheimhalten läßt. Anstelle einer einfachen Verschlüsselungsfunktion f, die jeder Nachricht m einen Chiffretext $c = f(m)$ zuordnet, soll man eine Funktion benutzen, die außer von m auch noch von einem zweiten Parameter s abhängt, dem *Schlüssel*. Somit ist also $c = f(m, s)$.

Im zweiten Kapitel seiner Schrift stellt er folgende Forderungen an einen Verschlüsselungsalgorithmus:

1. Das System muß praktisch, wenn schon nicht mathematisch, unentschlüsselbar sein.
2. Es darf den Schlüssel nicht preisgeben und kann ohne nachteilige Folgen in die Hand des Gegners fallen.
3. Es muß möglich sein, den Schlüssel ohne schriftliche Notizen zu übermitteln und aufzubewahren, und er muß sich ändern lassen, wann immer die Korrespondenten dies wünschen.
4. Das System muß sich für telegraphische Übermittlung eignen.
5. Das Verschlüsselungssystem muß tragbar sein und weder seine Handhabung noch seine Funktion darf die Zusammenarbeit mehrerer Personen erfordern.
6. Schließlich ist es auf Grund der Anforderungen seiner Anwendung notwendig, daß das System leicht anwendbar ist und weder geistige Anspannung noch die Kenntnis einer langen Reihe zu beachtender Regeln erfordert.



JEAN - GUILLAUME - HUBERT - VICTOR - FRANÇOIS - ALEXANDRE - AUGUSTE KERCKHOFFS VON NIEUWENHOF
(1835–1903) wurde in der heute niederländischen Ortschaft Nuth geboren. Er studierte an der Universität Liège, wo er mit dem Doktor der Literaturwissenschaften abschloß. Nachdem er mehrere Stellen als Lehrer in den Niederlanden und in Frankreich bekleidet hatte, wurde er schließlich Professor für Deutsch an der Ecole des Hautes Etudes Commerciales in Paris. Außer für seine Arbeit zur Militärkryptographie ist er vor allem auch noch für linguistische Studien bekannt, insbesondere auch zur heute weithin vergessenen Kunstsprache Volapük.

An diesen Forderungen hat sich im wesentlichen bis heute nichts geändert. Anstelle telegraphischer Übermittlungen haben wir zwar heute meist Rechnernetze, aber auch da ist es aus Effiziensgründen durchaus sinnvoll, mit Standard ASCII Code zu arbeiten statt mit frei erfundenen Hieroglyphen. Auch an der Forderung nach leichter Anwendbarkeit hat sich nichts geändert: Es wäre völlig unrealistisch, vom typischen Internetbenutzer mehr Intelligenz zu erwarten als von einem Militär mitten im Gefecht.

Regel drei muß man heute allerdings neu interpretieren: Schriftliche Notizen sind selbstverständlich weiterhin tabu, wir haben aber das Dilemma, daß einerseits ein sicherer Schlüssel einfach zu lang ist, als daß er mündlich übermittelt und auswendig gelernt werden könnte, daß aber andererseits Aufbewahrung im Computer oder gar Übermittlung per E-Mail zu nicht akzeptablen Sicherheitsrisiken führen. Wie wir bei der Diskussion von SSL/TLS sehen werden, gibt es zum Glück Möglichkeiten zur sicheren Schlüsselübermittlung per Computer.

Schwieriger ist das Problem, Schlüssel sicher zu speichern. Eine verhältnismäßig sichere, aber aufwendige Methode besteht darin, den Schlüssel wird auf einer Chipkarte zu speichern. Da diese in falsche Hände geraten kann oder möglicherweise auf einem präparierten Computer eingesetzt wird, ist klar, daß dabei noch zusätzliche Sicherungsmaßnahmen eingesetzt werden müssen. Eine bestünde etwa darin, den Schlüssel selbst zu verschlüsseln. Natürlich stellt sich dabei sofort die Frage, was man mit

dem *dazu* benötigten Schlüssel (dem *key encryption key* KEK) macht. Eine Strategie besteht etwa darin, den KEK auf Grund eines Passworts zu berechnen. Dazu kann etwa ein kryptographisch sicheres Hashverfahren verwendet werden – siehe dazu das entsprechende Kapitel dieser Vorlesung.

Ein mit einem passwortbasierten KEK verschlüsselter Schlüssel ist zwar (wenn wir getreu dem KERCKHOFFSschen Prinzip davon ausgehen, daß das Verfahren dazu nicht wirklich geheim gehalten werden kann) nicht sicherer als das Passwort, aber der Gegner braucht zu einem Angriff sowohl die Chipkarte als auch das Passwort. Selbst wenn die Chipkarte hinreichend lange in seinem Besitz ist, daß er alle Möglichkeiten für das Passwort durchprobieren kann, besteht immerhin noch die Chance, daß der Verlust bemerkt wird und der Schlüssel zumindest für künftige Kommunikationen nicht mehr verwendet wird. Denkbar, wenn auch für den alltäglichen Einsatz recht teuer, sind auch Chipkarten, die sich nach einer gewissen Anzahl falscher Eingaben selbst zerstören.

Alternativ zu einem Passwort könnte man auch mit biometrischen Daten arbeiten; erschwinglich und bereits relativ weit verbreitet sind beispielsweise Fingerabdrucksensoren. In der Realität bieten jedoch viele davon keine ausreichende Sicherheit gegen von einem berührten Gegenstand abgenommene und auf eine geeignete Folie geritzte Fingerabdrücke.

Die erste der KERCKHOFFSschen Regeln beschreibt ein Dilemma der Kryptographie, das auch noch heute bestimmend für das gesamte Gebiet ist und mit dem wir uns daher gleich noch viel ausführlicher befassen müssen. Die zentrale Frage ist:

c) Was kann der Gegner?

Sicher wissen wir nur, daß er es uns nicht verraten wird; meist verrät er uns schließlich nicht einmal, daß er unser Gegner ist. Wir sind daher auf Vermutungen angewiesen und sollten ihn, um mit relativ großer Wahrscheinlichkeit auf der sicheren Seite zu sein, im Zweifelsfall eher deutlich überschätzen.

Die Kryptologie hat als Idealgestalt des überschätzten Gegners den sogenannten BAYESschen Gegner eingeführt. Er verfügt über unbegrenzte

Rechenkraft, nicht aber über hellseherische Fähigkeiten. Er trifft seine Entscheidungen aufgrund der BAYESschen Statististik und stört sich nicht daran, daß die in der BAYESschen Formel auftretenden Terme in realistischen Anwendungen für alle praktischen Zwecke nicht berechenbar sind. Dies ist der Hintergrund der KERCKHOFFSschen Forderung, daß das Verfahren nur praktisch, nicht aber auch mathematisch sicher sein muß.

Die Sicherheit eines Verfahrens gegen den BAYESschen Gegner kann mit informationstheoretischen Methoden ziemlich gut abgeschätzt werden; zumindest was absolute Sicherheit betrifft, ist das Ergebnis allerdings deprimierend: Wie wir im Kapitel über Informationstheorie sehen werden, ist absolute Sicherheit unmöglich, sofern der Schlüssel kürzer ist als die Gesamtheit aller je damit verschlüsselten Nachrichten.

Im nächsten Kapitel werden wir sehen, daß sich mit hinreichend langen Schlüsseln tatsächlich absolut sichere Verfahren realisieren lasen (immer vorausgesetzt natürlich, daß auch im Umfeld alles *absolut* sicher ist . . .), und im Höchstsicherheitsbereich werden diese auch tatsächlich angewendet. Für die meisten Alltagsanwendungen der Kryptographie sind sie jedoch viel zu aufwendig; hier müssen wir unsere Anforderungen deutlich zurückschrauben.

Wenn wir KERCKHOFFS folgen, genügt es, daß ein Verfahren zumindest praktischen Sicherheit bietet. Wir müssen uns also überlegen, wie wir zumindest diese garantieren können.

An Ansätzen fehlt es nicht:

Der wohl erste „Sicherheitsbeweis“ geht zurück auf den GIROLAMO CARDANO: Seine Strategie bestand darin, ein Verfahren zu wählen, bei dem die Menge der möglichen Schlüssel so groß ist, daß sie unmöglich vollständig durchsucht werden kann.

Konkret schlug CARDANO vor, zur Verschlüsselung eines Texts eine zwischen Absender und Empfänger vereinbarte Permutation der 26 Buchstaben des Alphabets durchzuführen. Dafür gibt es

$$26! = 403\,291\,461\,126\,605\,635\,584\,000\,000$$

Möglichkeiten, also 403 Quadrillionen 291 Trilliarden 461 Trillionen 126 Billiarden 605 Billionen 635 Milliarden und 584 Millionen. Wie er völlig zu recht bemerkt, würden „viele Bücher nicht ausreichen," um alle Möglichkeiten zu fassen.



GIROLAMO CARDANO (1501–1576) war einer der bedeutendsten Ärzte und Naturforscher seiner Zeit. In der Mathematik ist er vor allem bekannt für seine Arbeiten über Gleichungen dritten und vierten Grades, auch wenn wesentliche Teile dieser Arbeiten nicht auf ihn zurückgehen. Er hat allerdings als erster durch Verwendung negativer Zahlen die vielen bis dahin betrachteten Fälle auf eine einzige Formel zurückgeführt. Nach seinem Medizinstudium schlug er sich zunächst mit Glückspiel durch, wobei ihm seine Kenntnisse der Wahrscheinlichkeitstheorie sehr nützlich waren. Erst 1539 konnte er eine Stelle als Arzt antreten und wurde schnell international berühmt.

Heute verwenden wir zum Entschlüsseln nur noch selten Bücher, aber auch Computer hätten Schwierigkeiten mit einer so großen Zahl von Möglichkeiten:

Nehmen wir an, wir hätten Tausend Chips, von denen jeder mit einer Taktfrequenz von zehn Gigahertz arbeitet und die so spezialisiert sind, daß jeder in jedem Takt eine ganze Probeentschlüsselung durchführen kann. Pro Sekunde kann also jeder Chip zehn Milliarden Möglichkeiten durchprobieren, und alle zusammen kommen auf zehn Billionen. Ein Jahr hat durchschnittlich ungefähr

$$365\frac{1}{4} \cdot 24 \cdot 60 \cdot 60 = 31\,557\,600$$

Sekunden, also schafft die Maschine pro Jahr etwas mehr als 315 Trillionen Entschlüsselungen; bis sie alle rund 403 Quadrillionen Möglichkeiten durchprobiert hat, braucht sie über eine Million Jahre; ein gewöhnlicher PC bräuchte gar weit länger als das Alter unseres Universums. Selbst wenn wir an Stelle von Tausend Chips eine Million verwenden, bräuchte die Maschine immer noch rund 1278 Jahre, und es ist sehr unwahrscheinlich, daß der Inhalt der Nachricht auch dann noch geheim bleiben muß. Das Verfahren sollte also für alle praktischen Zwecke absolut sicher sein.

Ähnlich klingen die Werbeaussagen vieler heutiger Anbieter von Kryptoverfahren, obwohl nicht viele davon mit über 403 Quadrillionen Varianten aufwarten können. Was dabei meist verschwiegen wird: Durchprobieren aller Möglichkeiten ist zwar *ein* Weg zur Entschlüsselung, aber oft ist es bei weitem nicht der einzige. Wir müssen uns immer der Tatsache bewußt sein, daß es der *Angreifer* ist, der entscheidet, wie er vorgeht, nicht wir. Wenn wir unser Verfahren gegen eine Art von Attacke immunisiert haben, müssen wir stets damit rechnen, daß er einfach eine andere wählt.

Im nächsten Kapitel werden wir sehen, daß der Aufwand zum Knacken von CARDANOS Verfahren schon bei Nachrichten moderater Länge (60–100 Buchstaben) eher im Bereich von Minuten als im Bereich von Stunden liegt, und nicht viel besser sieht es aus bei modernen Kryptoverfahren, die sich nur auf die „unüberschaubar große Anzahl von Möglichkeiten" verlassen. Bevor jemand ein solches Verfahren anwendet, sollte er zum Beispiel bei `pwcrack.com` nachschauen, für welch geringe Beträge (meist 40-100 US-$) die Kryptographie heutiger Office-Programme dort geknackt wird – und das mit Erfolgsgarantie. Verglichen mit den Werten, die im kommerziellen Bereich durch solche Kryptographie geschützt werden sollen, ist dieser Aufwand lächerlich gering. In der seriösen Kryptographie läßt sich daher schon lange niemand mehr durch eine bloße Vielzahl von Möglichkeiten blenden.

Das zwanzigste Jahrhundert kam mit neuen Methoden wie der sogenannten Komplexitätstheorie; dazu lesen wir im Buch *Privacy on the Line* von WHITFIELD DIFFIE und SUSAN LANDAU (MIT Press, ^{2}2007, Anmerkung 15 zu Kapitel 2):

> The vast number of keys was offered as an argument for the unbreakability of ciphers during the Renaissance (. . .) and probably earlier. The more general modern theories, including the theory of *non-deterministic polynomial time* or NP computing (. . .) are far more mathematical but little more satisfactory.

(Die beiden Auslassungen sind Verweise auf das Literaturverzeichnis)

(Wir werden WHITFIELD DIFFIE bald als einen der beiden Väter der asymmetrischen Kryptographie kennenlernen; er arbeitete von 1991 bis

2009 als *chief security officer* bei Sun Microsystems und ist seit 2010 *Vice President for Information Security and Cryptography* bei ICANN, der *Internet Corporation for Assigned Names and Numbers.*)

Worum geht es? Idealerweise wüßten wir gerne, wie groß der Mindestaufwand zur Lösung eines Problems ist. Darüber läßt sich allerdings nur selten eine Aussage machen, da man dazu entweder alle möglichen Ansätze zur Lösung des Problems kennen müßte oder aber beispielsweise eine Untergrenze für die Länge des Ergebnisses. Letztere ist im Falle der Kryptographie zwar kein Problem: Fast immer ist der Klartext genauso lang wie oder nur unwesentlich länger als der Chiffretext; außerhalb der Steganographie dürfte es wohl kein Verfahren geben, in dem er mehr als doppelt so lang ist; dafür ist er bei Einsatz einer guten Quellenkodierung gelegentlich auch deutlich kürzer als der Klartext. Somit liefert dies keine brauchbare Untergrenze.

Die Komplexitätstheorie betrachtet daher Obergrenzen oder (seltener) obere Grenzen für den mittleren Aufwand. Es ist klar, daß diese wertlos sind, wenn es um die Beurteilung der Sicherheit einer konkreten Verschlüsselung geht.

Schlimmer noch: Da auch konkrete Obergrenzen schwer zu finden sind, begnügt man sich meist mit *asymptotischen* Aussagen über das Verhalten der Obergrenze, wenn die Länge der Eingabe (zum Beispiel die eines öffentlichen Schlüssels) gegen unendlich geht; wie jeder, der seine *Analysis I* verstanden hat, wissen sollte, folgt daraus natürlich nichts für eine konkrete vorgegebene Länge.

Da selbst asymptotische Obergrenzen nicht einfach sind, beschränken sich viele sogar noch darauf, nur zu untersuchen, ob die Obergrenze polynomial wächst oder stärker – selbst da gibt es noch viele offene Probleme. Spätestens hier werden die Ergebnisse allerdings völlig bedeutungslos für praktische Anwendungen auf die Kryptographie: Die zahlentheoretischen Algorithmen, die vielen der in dieser Vorlesung behandelten Kryptoverfahren zugrunde liegen, haben typischerweise eine asymptotische Komplexität, die für Eingabewerte der Länge n in erster Näherung durch die Funktion

$$L_{\alpha,c}(n) = e^{cn^{\alpha}(\ln n)^{1-\alpha}}$$

beschrieben werden mit reellen Konstanten $0 \leq \alpha \leq 1$ und $c > 0$. Für $\alpha = 0$ ist dies ein ein Polynom in n, für $\alpha = 1$ eine Exponentialfunktion. Für $0 < \alpha < 1$ liegt das asymptotische Verhalten zwischen diesen beiden Grenzfällen, und der tatsächliche Aufwand hängt außer vom Parameter c auch noch stark von sonstigen Konstanten ab, die bei einer $O(\ldots)$-Abschätzung unter den Tisch fallen.

Deshalb kann die Komplexitätstheorie genauso wenig seriöse Aussagen über die Sicherheit eines konkreten Verfahrens machen wie die Mächtigkeit der Schlüsselmenge.

Wenn ein Kryptoverfahren nicht im informationstheoretischen Sinn beweisbar sicher ist, kann man nach heutigem Stand höchstens dann Vertrauen in seine Sicherheit haben, wenn sich bereits viele erfahrene Kryptologen hinreichend lange mit seiner Kryptanalyse beschäftigt haben und keine Attacke mit vertretbarem Aufwand fanden. Das gibt zwar keine Garantie, daß nicht doch einer in Kürze eine finden wird, aber damit müssen wir leben – es sei denn, wir sind bereit, den hohen Aufwand für ein beweisbar sicheres Verfahren zu tragen.

§ 5: Literaturhinweise

Die Geschichte der Kryptographie findet man wohl immer noch am besten in

DAVID KAHN: *The Codebreakers – the comprehensive history of secret communication from ancient time to the internet,* Scribner, New York, ^{2}1996

Abgesehen von einem Anhang über public key Kryptographie ist das Buch weitgehend identisch mit der ersten Auflage von 1967, die in der Mannheimer Universitätsbibliothek zu finden ist.

Beispiele, wie man durch *social engineering* Sicherheitsmaßnahmen aushebeln kann, findet man zum Beispiel in den beiden Büchern des früheren Hackers und heutigen Sicherheitsberaters KEVEN MITNICK:

KEVIN D. MITNICK, WILLIAM L. SIMON: *The Art of Deception: Controlling the Human Element of Security,* Wiley, 2002; deutsche Ausgabe *Die Kunst der Täuschung: Risikofaktor Mensch,* Verlag Moderne Industrie, 2003

KEVIN D. MITNICK, WILLIAM L. SIMON: *The Art of Intrusion – The Real Stories Behind the Exploits of Hackers, Intruders & Deceivers,* Wiley, 2005; deutsche Ausgabe *Die Kunst des Einbruchs,* Mitp-Verlag, 2006

Beide Bücher stehen auch (wahrscheinlich nicht ganz legal) als Volltext im Internet.

Zur Steganographie sind in den letzten Jahren eine Reihe neuer Bücher erschienen, die sich hauptsächlich mit deren elektronischer Version befassen, darunter

PETER WAYNER: *Disappearing cryptography – Information Hiding: Steganography and Watermarking,* Morgan Kaufmann, ^{3}2009

INGEMAR J. COX, MATTHEW L. MILLER, JEFFREY A. BLOOM, JESSICA FRIDRICH, TON KALKER: *Digital Watermarking and Steganography,* Morgan Kaufmann, ^{2}2008

Eher geschichtlich orientiert ist

KLAUS SCHMEH: *Versteckte Botschaften – Die faszinierende Geschichte der Steganografie,* Heise, 2009

Mit der Entdeckung von Steganographie beschäftigen sich unter anderem

GREGORY KIPPER: *Investigator's Guide to Steganography,* Auerbach Publications (CRC Press), 2003

RAINER BÖHME: *Advanced Statistical Steganalysis,* Springer, 2010

Kapitel 2
Einige klassische Kryptoverfahren

In diesem Kapitel geht es nicht um eine Darstellung der Geschichte der Kryptographie; die vorgestellten Verfahren werden daher nicht in chronologischer Reihenfolge dargestellt, sondern mehrere Gruppen von miteinander verwandten Verfahren werden jeweils in logischer Reihenfolge vom einfachsten zum schwierigsten der behandelten Verfahren präsentiert – auch wenn eine ganze Reihe von zum Teil heute noch gebräuchliche Verfahren schon zum Zeitpunkt ihrer Einführung hoffnungslos veraltet waren.

§ 1: Monoalphabetische Substitutionen

Bei einer monoalphabetischen Substitution besteht die Verschlüsselung in einer Permutation des Alphabets, in dem die Nachricht geschrieben ist. Klassisch waren dies die Buchstaben des Alphabets; heute sind es, zumindest wenn man mit Computern arbeitet, eher Bytes.

a) Die Nullchiffre

Im Internet kann jeder tun und lassen was er will – mit nur ganz wenigen Ausnahmen. Zu diesen Ausnahmen gehört definitiv nicht, daß irgend jemand dazu gezwungen würde, sich vernünftig zu verhalten und auf Sicherheit zu achten. Dort, wo Verschlüsselung vorgesehen ist, sind die Anwender frei in der Wahl der einzusetzenden Verfahren, und fast überall ist als eine Variante auch die sogenannte Nullchiffre vorgesehen, die einfach darin besteht, überhaupt nicht zu verschlüsseln. Da im Internet zwar praktisch nichts vorgeschrieben, aber alles normiert

ist, hat auch dieses Nichtstun eine rund sechs Seiten lange Beschreibung in RFC 2410; beispielsweise wird dort darauf hingewiesen, daß man auch bei diesem „Verfahren“ mit Schlüsseln arbeiten kann, daß eine Erhöhung der Schlüssellänge die Sicherheit aber nicht wesentlich steigert. Für Einzelheiten siehe `www.faqs.org/rfcs/rfc2410.html`.

b) Die Caesar-Chiffre

Bei CAESAR selbst lesen wir, daß er eine Nachricht an CICERO mit griechischen statt lateinischen Buchstaben schrieb um zu verhindern, daß feindliche Soldaten (die im Gegensatz zu CICERO keine griechischen Buchstaben lesen konnten) den Inhalt verstehen konnten. Der Bote kam zwar nicht bis zu CICERO durch, aber, wie ihm CAESAR für diesen Fall geraten hatte, steckte er die Nachricht auf einen Speer, den er in CICEROs Lager warf. Nach zwei Tagen wurde der Speer gefunden und die unverständliche Nachricht zu CICERO gebracht, der sie natürlich lesen konnte.

Von einem geringfügig komplexeren Verfahren berichtet SUETON in Kapitel 56 des ersten Buchs DIVUS IULIUS *(der göttliche Julius)* seines Werks DE VITA CAESARUM:

> extant et ad ciceronem, item ad familiares domesticis de rebus, in quibus, si qua occultius perferenda erant, per notas scripsit, id est sic structo litterarum ordine, ut nullum verbum effici posset: quae si qui investigare et persequi velit, quartam elementorum litteram, id est d pro a et perinde reliquas commutet.
>
> Erhalten sind auch seine Briefe an CICERO, ebenso an seine engeren Freunde über private Angelegenheiten, in denen er, was etwa geheim zu überbringen war, in verschlüsselter Form schrieb, nämlich in einer solchen Anordnung der Buchstaben, daß kein einziges Wort herauskam. Falls hier jemand nachforschen und der Sache nachgehen will, möge er den vierten Buchstaben des Alphabets, d.h. D für A und so fort setzen.

(zitiert nach SUETON: Kaiserbiographien, *Akademie Verlag Berlin,* 1993

Das war zwar weit hinter der Kryptographie, die in anderen Weltgegenden schon Jahrhunderte früher praktiziert wurde, aber wie die meisten

Römer war eben auch deren Kryptographie recht primitiv.

GAIUS JULIUS CAESAR (100-44) und MARCUS TULLIUS CICERO (106–43) dürften wohl den meisten bekannt sein. GAIUS SUETONAUIS TRANQUILLUS war ein römischer Geschichtsschreiber, der um das Jahr 70 geboren wurde; er starb wahrscheinlich zwischen 130 und 140. Bekannt ist vor allem seine Lebensgeschichte der Caesaren, die aus je einem Buch für jeden der zwölf Caesaren besteht.

CAESAR verschob also das Alphabet zyklisch um drei Positionen; aus GALLIA EST OMNIS DIVISA IN PARTES TRES wurde das auch für Römer unverständliche JDOOL DHVWR PQLVG LYLVD LQSDU WHVWU HV.

Später soll ein ähnliches Verfahren auch von AUGUSTUS verwendet worden sein, allerdings verschob dieser nur um einen Buchstaben. Dafür soll CAESAR gelegentlich auch um eine andere Anzahl als drei verschoben haben. Für jemand, der seine Allianzen so häufig wechselte wie CAESAR, bot dies natürlich schon damals keine große Sicherheit, und heute kann man ein solches Verfahren erst recht vergessen: Zu CAESARs Zeiten, als das Alphabet aus nur 21 Buchstaben bestand und mangels Internet die Nullchiffre noch nicht als Verschlüsselungsverfahren betrachtet wurde, gab es schließlich nur zwanzig Möglichkeiten. (Die Buchstaben K, Y, Z existieren damals noch nicht, U und V sowie I und J wurden nicht unterschieden, und überflüssiger Komfort wie Kleinbuchstaben oder Satzzeichen wurden frühestens im achten oder neunten Jahrhundert gebräuchlich.)

Heute, egal ob wir mit 26 Buchstaben, 256 ASCII-Zeichen oder irgend etwas dazwischen liegendem arbeiten, kann ein Computer in Sekundenbruchteilen alle Möglichkeiten durchprobieren: Um etwa den Chiffretext

XYXFS DKOCO NCMRY VKONS CMSWE CCOXO

MKOZS CDEVK OWYBK VOCKN VEMSV SEWMF S

zu entschlüsseln, betrachten wir einfach alle 26 Möglichkeiten:

```
 1 YZYGT ELPDP ODNSZ WLPOT DNTXF DDPYP NLPAT DEFWL PXZCL WPDLO WFNTW TFXNG T
 2 ZAZHU FMQEQ PEOTA XMQPU EOUYG EEQZQ OMQBU EFGXM QYADM XQEMP XGOUX UGYOH U
 3 ABAIV GNRFR QFPUB YNRQV FPVZH FFRAR PNRCV FGHYN RZBEN YRFNQ YHPVY VHZPI V
 4 BCBJW HOSGS RGQVC ZOSRW GQWAI GGSBS QOSDW GHIZO SACFO ZSGOR ZIQWZ WIAQJ W
 5 CDCKX IPTHT SHRWD APTSX HRXBJ HHTCT RPTEX HIJAP TBDGP ATHPS AJRXA XJBRK X
 6 DEDLY JQUIU TISXE BQUTY ISYCK IIUDU SQUFY IJKBQ UCEHQ BUIQT BKSYB YKCSL Y
 7 EFEMZ KRVJV UJTYF CRVUZ JTZDL JJVEV TRVGZ JKLCR VDFIR CVJRU CLTZC ZLDTM Z
 8 FGFNA LSWKW VKUZG DSWVA KUAEM KKWFW USWHA KLMDS WEGJS DWKSV DMUAD AMEUN A
```



```
 9 GHGOB MTXLX WLVAH ETXWB LVBFN LLXGX VTXIB LMNET XFHKT EXLTW ENVBE BNFVO B
10 HIHPC NUYMY XMWBI FUYXC MWCGO MMYHY WUYJC MNOFU YGILU FYMUX FOWCF COGWP C
11 IJIQD OVZNZ YNXCJ GVZYD NXDHP NNZIZ XVZKD NOPGV ZHJMV GZNVY GPXDG DPHXQ D
12 JKJRE PWAOA ZOYDK HWAZE OYEIQ OOAJA YWALE OPQHW AIKNW HAOWZ HQYEH EQIYR E
13 KLKSF QXBPB APZEL IXBAF PZFJR PPBKB ZXBMF PQRIX BJLOX IBPXA IRZFI FRJZS F
14 LMLTG RYCQC BQAFM JYCBG QAGKS QQCLC AYCNG QRSJY CKMPY JCQYB JSAGJ GSKAT G
15 MNMUH SZDRD CRBGN KZDCH RBHLT RRDMD BZDOH RSTKZ DLNQZ KDRZC KTBHK HTLBU H
16 NONVI TAESE DSCHO LAEDI SCIMU SSENE CAEPI STULA EMORA LESAD LUCIL IUMCV I
17 OPOWJ UBFTF ETDIP MBFEJ TDJNV TTFOF DBFQJ TUVMB FNPSB MFTBE MVDJM JVNDW J
18 PQPXK VCGUG FUEJQ NCGFK UEKOW UUGPG ECGRK UVWNC GOQTC NGUCF NWEKN KWOEX K
19 QRQYL WDHVH GVFKR ODHGL VFLPX VVHQH FDHSL VWXOD HPRUD OHVDG OXFLO LXPFY L
20 RSRZM XEIWI HWGLS PEIHM WGMQY WWIRI GEITM WXYPE IQSVE PIWEH PYGMP MYQGZ M
21 STSAN YFJXJ IXHMT QFJIN XHNRZ XXJSJ HFJUN XYZQF JRTWF QJXFI QZHNQ NZRHA N
22 TUTBO ZGKYK JYINU RGKJO YIOSA YYKTK IGKVO YZARG KSUXG RKYGJ RAIOR OASIB O
23 UVUCP AHLZL KZJOV SHLKP ZJPTB ZZLUL JHLWP ZABSH LTVYH SLZHK SBJPS PBTJC P
24 VWVDQ BIMAM LAKPW TIMLQ AKQUC AAMVM KIMXQ ABCTI MUWZI TMAIL TCKQT QCUKD Q
25 WXWER CJNBN MBLQX UJNMR BLRVD BBNWN LJNYR BCDUJ NVXAJ UNBJM UDLRU RDVLE R
26 XYXFS DKOCO NCMRY VKONS CMSWE CCOXO MKOZS CDEVK OWYBK VOCKN VEMSV SEWMF S
```

Auch ohne Lateinkenntnisse sieht man leicht, daß die korrekte Entschlüsselung nur Zeile sechzehn sein kann: NON VITAE SED SCHOLAE DISCIMUS – EPISTULAE MORALES AD LUCILIUM CVI: *Wir lernen nicht für das Leben, sondern für die Schule,* wie LUCIUS ANAEUS SENECA (∼1–65) im 106. seiner Briefe an LUCILIUS die Schulmeister seiner Zeit verspottete.

Da sie so einfach zu entschlüsseln sind, sollten CAESAR-Chiffren heute völlig vergessen sein. Die kryptographische Realität sieht aber leider anders aus: Selbst dümmste Verfahren sind anscheinend unausrottbar. Der letzte bekannt gewordene prominente Anwender einer (leicht variierten) CAESAR-Chiffre war Mafia-Boss BERNARDO PROVENZANO, der bereits im Alter von acht Jahren die Schule verließ: Er ersetzte den Buchstaben „A“ durch die Zahl „4“ und so weiter bis zur Zahl „29“ für „Z“. Da alle hier benutzten zweistelligen Zahlen mit einer Eins oder Zwei beginnen, die einstelligen aber mindestens gleich vier sind, konnte er diese Zahlen ohne Zwischenraum hintereinander schreiben: 611241822115261218 etwa kann nur interpretiert werden als

$$6, 11, 24, 18, 22, 11, 5, 26, 12, 18 \quad = \quad \textit{Ciao Silvio} \ .$$

Da die meisten Strafverfolger länger zur Schule gegangen waren, stellte sie auch diese Modifikation vor kein unüberwindbares Problem, und so konnte er nach vierzig Jahren am 11. April 2006 endlich gefaßt werden.

Auch im Internet benutzt man in einigen Foren die CAESAR-Chiffre mit Verschiebung um 13 (ROT-13), allerdings nicht zur Geheimhaltung: Hier geht darum, daß Texte, die nicht nach jedermanns Geschmack sind, nur von denen gelesen werden, die das wirklich wollen.

c) Allgemeine monoalphabetische Substitutionen

Schwieriger wird es, wenn die Anzahl möglicher Verschlüsselungen zu groß wird, als daß man alle ausprobieren könnte. Solche Verfahren lassen sich leicht konstruieren: Wie bereits im vorigen Kapitel erwähnt, schlug GIROLAMO CARDANO vor, das Alphabet nicht nur wie bei den CAESAR-Substitutionen zyklisch zu verschieben, sondern es auf irgendeine *beliebige* Weise durcheinander zu bringen, so daß es $26! \approx 4 \cdot 10^{26}$ Möglichkeiten gibt – auch für Supercomputer zu viele, um alle auszuprobieren.

Trotzdem konnten schon die Militärkryptographen zur Zeit des ersten Weltkriegs jede so verschlüsselte Nachricht ab etwa einer Länge von fünfzig Zeichen problemlos entschlüsseln, und das ohne jede Maschine mit einem Aufwand von deutlich unter einer Stunde. Bei den meisten Verfahren, die heute beispielsweise als Teil von Office-Software angeboten werden, geht die Entschlüsselung zwar nicht ganz so einfach, dafür aber wenigstens sehr billig: Bei Anbietern wie `www.pwcrack.com` oder `www.lostpassword.com` kann man nachlesen, für welch geringe Beträge (ab etwa 40 US-$) die Verschlüsselungssysteme der meisten heute üblichen Softwarepakete geknackt werden können – natürlich nur zur Rekonstruktion „vergessener“ Passwörter. Wenn man bedenkt, daß manche interne Dokumente oder Kundendatenbanken für ein Unternehmen Werte im Bereich von Tausenden wenn nicht gar Millionen Euro repräsentieren können, wird klar, wie fahrlässig hier mit Kryptographie umgegangen wird.

Der Ansatzpunkt für den Kryptanalytiker ist praktisch immer derselbe: Die Dokumente, die wir schützen wollen, enthalten keine Zufallsfolgen, sondern sprachliche Texte oder sonstige strukturierte Information. Diese Struktur muß der Angreifer ausnutzen.

Wir wollen uns anhand eines Beispiels anschauen, wie er etwa im Fall von CARDANOs System vorgehen kann. Angenommen, wir haben die

Nachricht

```
MBKFB MPLNL NIEAN KXRBP KKUMK KUNJC NEKXR SMKBN
JENEC PKKEI XRLMB BNIEU MKMAX AJIEO LUNEC NEKXR
NEIEU INRFN REIXR LMBBN IEICK XRJNI ANEBN KNEPN
ALKIX RNIEQ NJEPN EDLIO SNKNE EIXRL MBBNI EIEJN
XREPE OKKMX RNEKF BBUNJ CNEKX RKIXR CPNRN CMXRN
EKFEU NJEMP XRUNJ SNIKR NILBN RJNEC PKKCM ECILQ
NJOEP NONER FNJNE UMKKU INKCI LQNJK LMEUO NKXRM
RSMJR NJJBN RJNJB MNCGN BUMCM VPEUC FJILY UINKN
ANIUN ECFXR LNEIR EUMJP CEIXR LBNIU NEUNE ESNJA
FNKNK LJNIX RNCMX RLOIA LEIXR LMPDU NEBNR JNJMX
RL
```

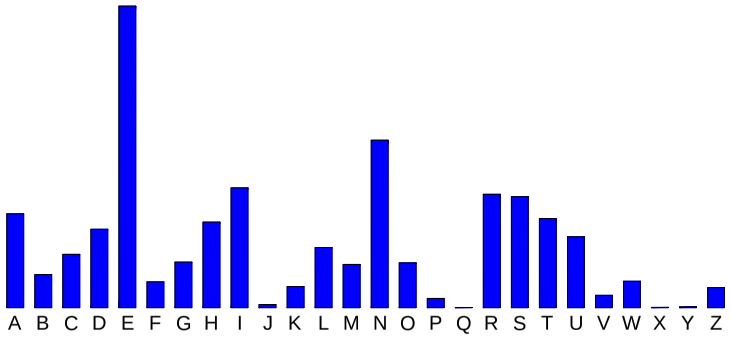
aufgefangen und wollen sie entschlüsseln. Ein erster Ansatz könnte darin bestehen, daß wir den häufigsten Buchstaben des Kryptogramms als E identifizieren, den zweithäufigsten als den nach E nächsthäufigsten Buchstaben in deutschem Klartext, *usw.*

Dazu müssen wir als erstes wissen, wie häufig welcher Buchstabe in einem typischen deutschen Text ist. Wie die Jahrhunderte alte Erfahrung der Kryptanalytiker (und auch die Forschungen der modernen Linguisten) zeigt, gibt es hier keine nennenswerten Unterschiede zwischen verschiedenen (hinreichend langen) Texten; wir können also einfach irgendeinen deutschen Klartext hernehmen und die Buchstaben zählen. Das folgende Diagramm beruht auf der Auszählung der 260 238 Buchstaben aus JEAN PAULs Novelle *Dr. Katzenbeisers Badereise*. (Wie in der der klassischen Kryptographie üblich, wurden Zwischenräume und Satzzeichen ignoriert und Umlaute sowie „ß“ umschrieben, so daß nur 26 Buchstaben verwendet werden. Dies macht das Lesen der entschlüsselten Nachrichten zwar etwas unbequemer, erhöht aber die Sicherheit.)

Ordnen wir die Buchstaben nach ihrer Häufigkeit in diesem Text, erhalten wir die Folge

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

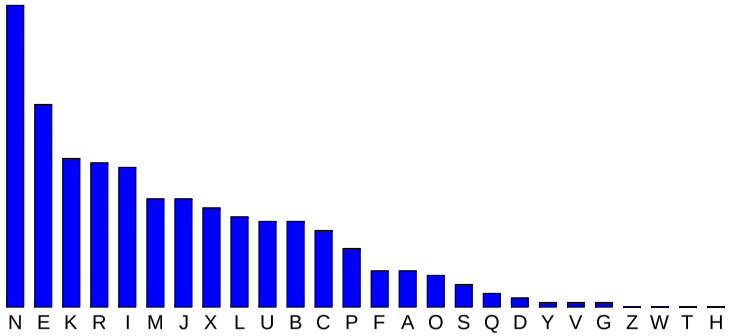
Das gleiche können wir auch mit dem Kryptogramm machen; hier erhalten eine ähnliche Abbildung (die uns, falls wir *a priori* nichts über



Buchstabenhäufigkeiten in deutschem Klartext

das verwendete Verfahren wissen, auch zeigt, daß wir es wahrscheinlich mit einer monoalphabetischen Substitution zu tun haben), jetz aber mit der Folge

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



Buchstabenhäufigkeiten im Kryptogramm

Ein naheliegender erster Versuch ist also, daß wir N als E entschlüsseln, E als N, K als I, R als R, I als S, und so weiter. Leider ist das Ergebnis nicht sehr vielversprechend:

```
ALIOL AGDED ESNME IHRLG IIUAI IUETC ENIHR WAILE
TNENC GIINS HRDAL LESNU AIAMH MTSNB DUENC ENIHR
```



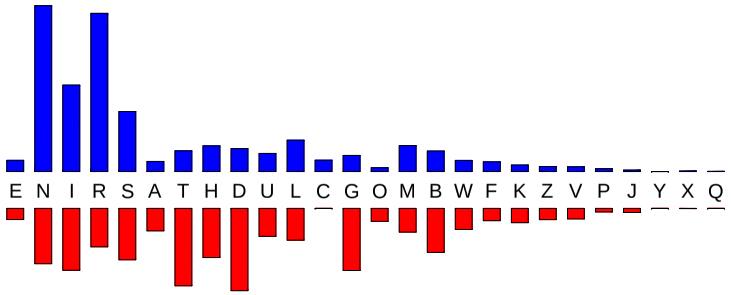
```
ENSNU SEROE RNSHR DALLE SNSCI HRTES MENLE IENGE
MDISH RESNF ETNGE NKDSB WEIEN NSHRD ALLES NSNTE
HRNGN BIIAH RENIO LLUET CENIH RISHR CGERE CAHRE
NIONU ETNAG HRUET WESIR ESDLE RTENC GIICA NCSDF
ETBNG EBENR OETEN UAIIU SEICS DFETI DANUB EIHRA
RWATR ETTLE RTETL AECPE LUACA VGNUC OTSDZ USEIE
MESUE NCOHR DENSR NUATG CNSHR DLESU ENUEN NWETM
OEIEI DTESH RECAH RDBSM DNSHR DAGKU ENLER TETAH
RD
```

Denkt man etwas nach, sollte dieses Ergebnis eigentlich niemanden verwundern: Die Häufigkeitsunterschiede zwischen ähnlich häufigen Buchstaben sind teilweise so gering, daß sie gerade bei einem relativ kurzen Texten von nur 402 Buchstaben noch im Bereich von Zufallsschwankungen liegen. Ein erfolgreicher Ansatz muß daher mehr von der Struktur der deutschen Sprache ausnützen.

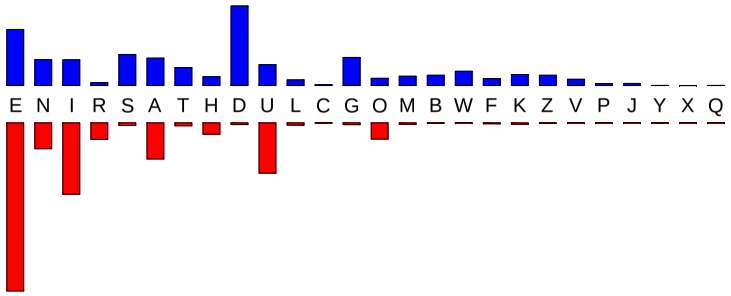
Innerhalb eines Texts ist die Wahrscheinlichkeit für das Auftreten eines Buchstabens stark vom Kontext abhängig: Auch wenn „E“ insgesamt gesehen der häufigste Buchstabe ist, wird man nach einem „C“ oder „Q“ nur selten eines finden und nach einem anderen „E“ auch nicht. Die Diagramme auf den folgenden Seiten zeigen, welche Buchstaben häufig vor (roter unterer Balken) *bzw.* nach (blauer oberer Balken) einem gegebenen Buchstaben auftreten.

Auch wenn in einem kurzen Text gelegentlich „N“ häufiger vorkommt als „E“: Mit diesen Diagrammen können die beiden Buchstaben leicht unterschieden werden: Beispielsweise kommt vor „N“ häufig „E“ oder „I“ vor, aber fast nie ein seltener Buchstabe, während die Verteilung vor „E“ sehr viel homogener ist. Dafür ist die Verteilung *nach* „N“ homogener als die nach „E“ mit ihren vier großen Zacken bei den häufigen Buchstaben. Zwar sind zu Beginn einer Entschlüsselung *alle* Buchstaben unbekannt, aber wenn man die Buchstaben in den Diagrammen nach Häufigkeit ordnet, kann man einigermaßen sicher sein, daß sich kein Buchstabe gar zu weit von seiner Position entfernt und damit zumindest die häufigen Buchstaben leicht identifizieren.

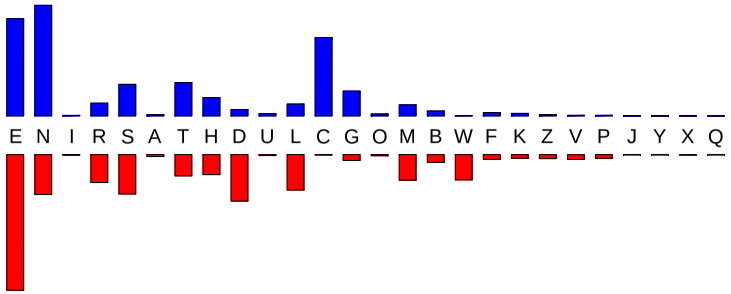
Kontaktdiagramm des Buchstabens E



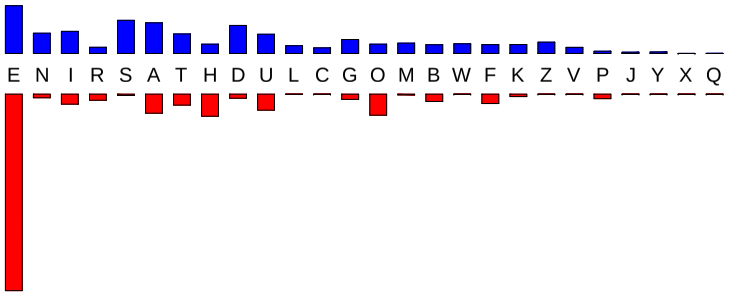
Kontaktdiagramm des Buchstabens N



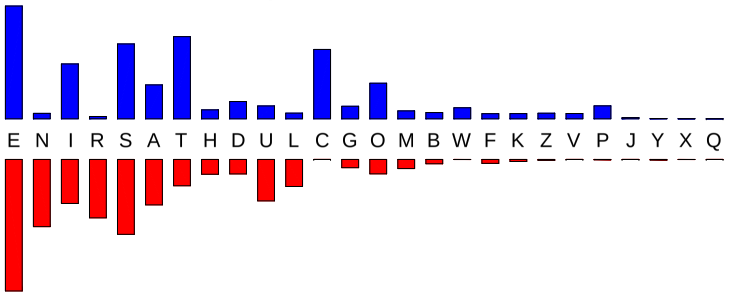
Kontaktdiagramm des Buchstabens I



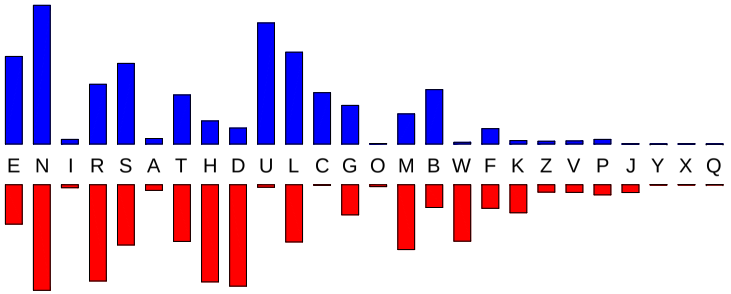
Kontaktdiagramm des Buchstabens R



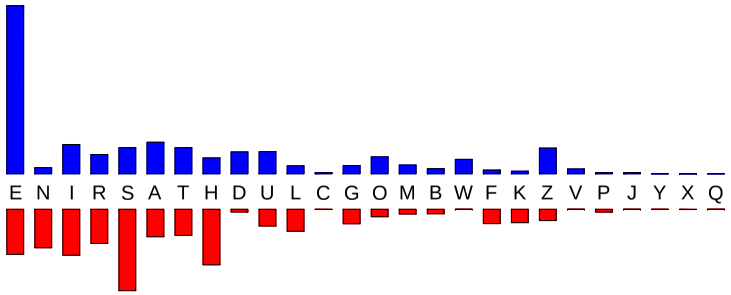
Kontaktdiagramm des Buchstabens S



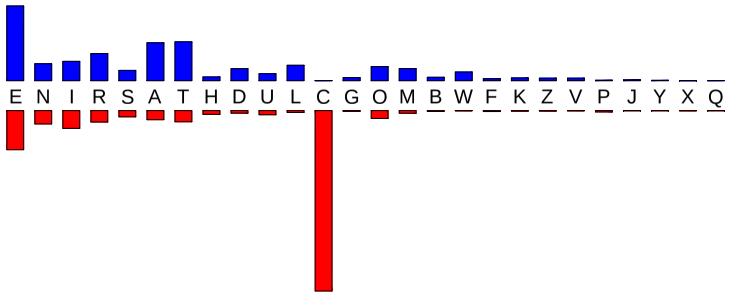
Kontaktdiagramm des Buchstabens A



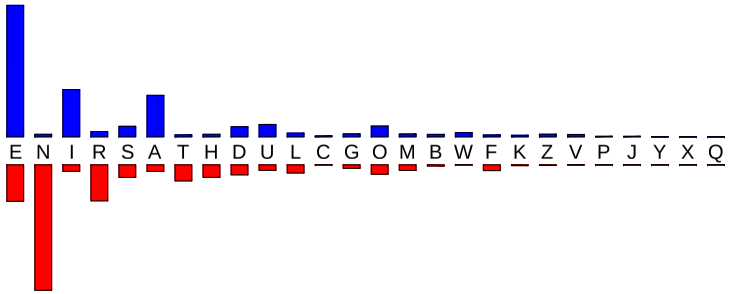
Kontaktdiagramm des Buchstabens T



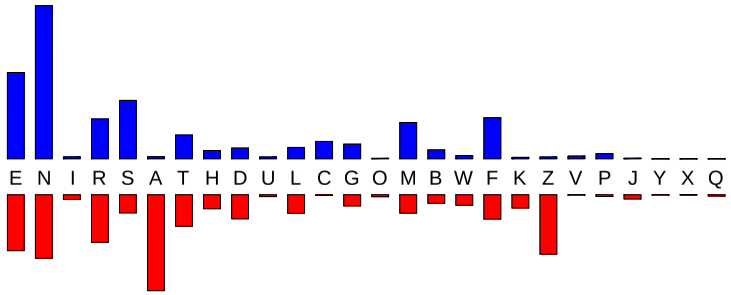
Kontaktdiagramm des Buchstabens H



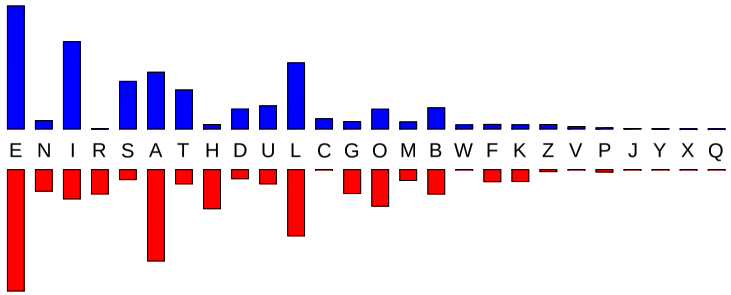
Kontaktdiagramm des Buchstabens D



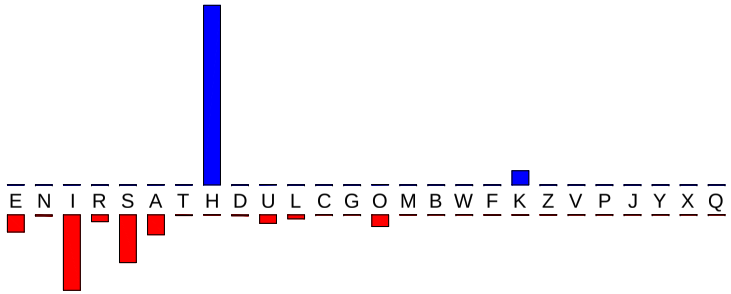
Kontaktdiagramm des Buchstabens U



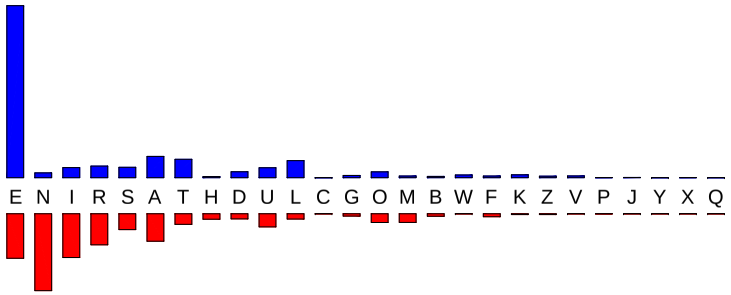
Kontaktdiagramm des Buchstabens L



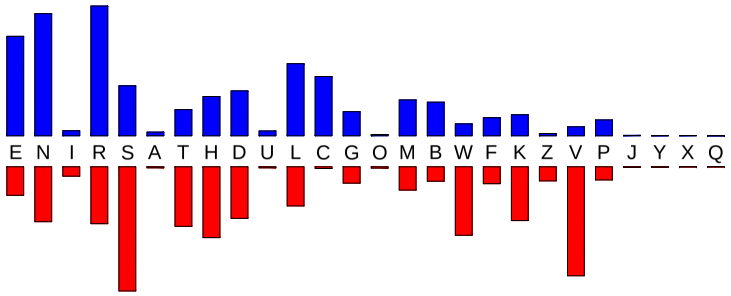
Kontaktdiagramm des Buchstabens C



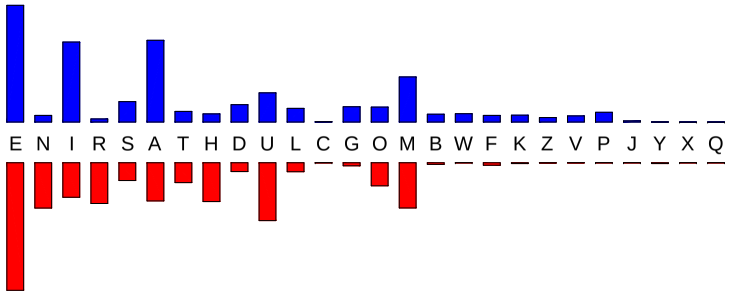
Kontaktdiagramm des Buchstabens G



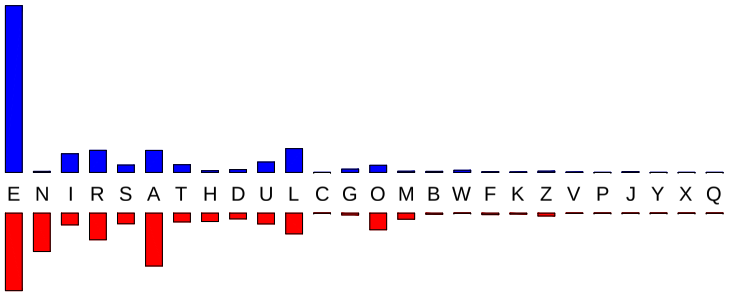
Kontaktdiagramm des Buchstabens O



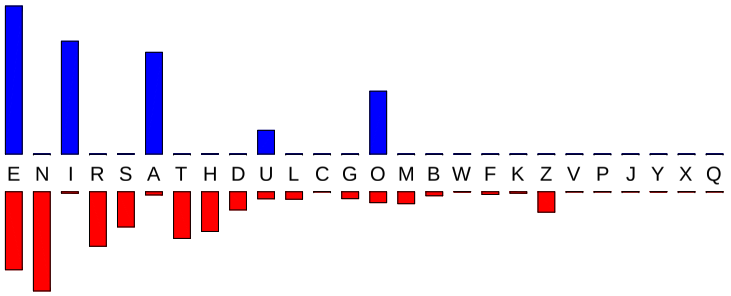
Kontaktdiagramm des Buchstabens M



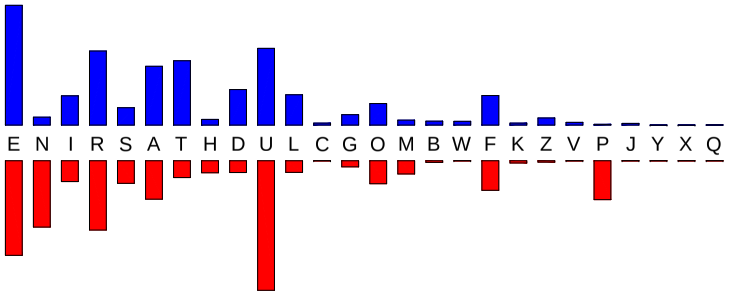
Kontaktdiagramm des Buchstabens B



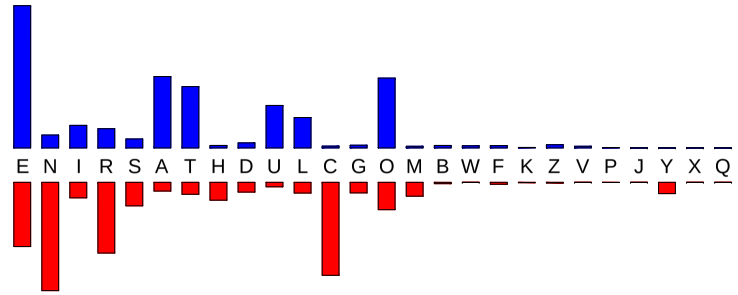
Kontaktdiagramm des Buchstabens W



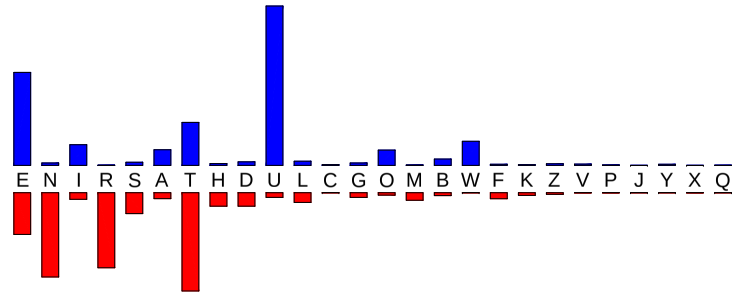
Kontaktdiagramm des Buchstabens F



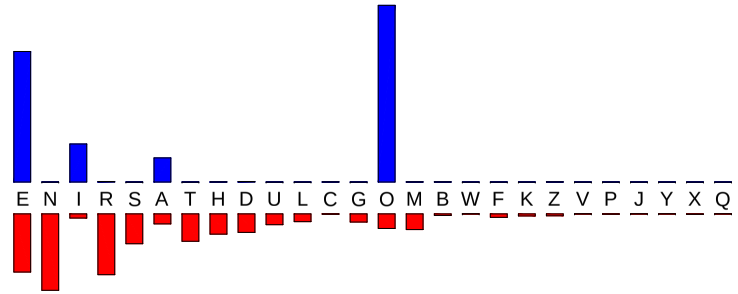
Kontaktdiagramm des Buchstabens K



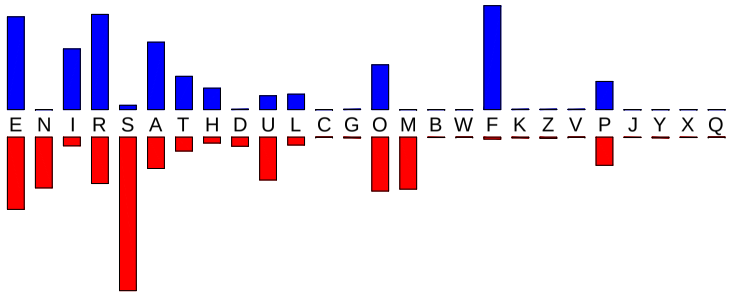
Kontaktdiagramm des Buchstabens Z



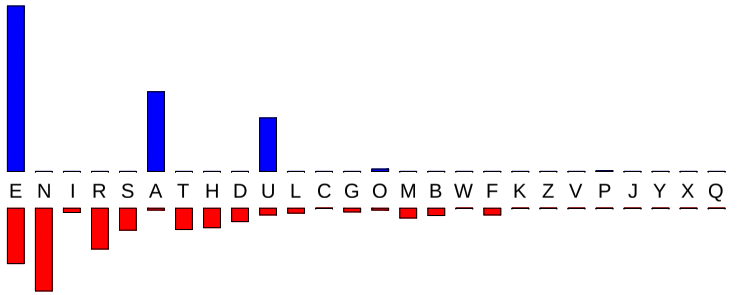
Kontaktdiagramm des Buchstabens V



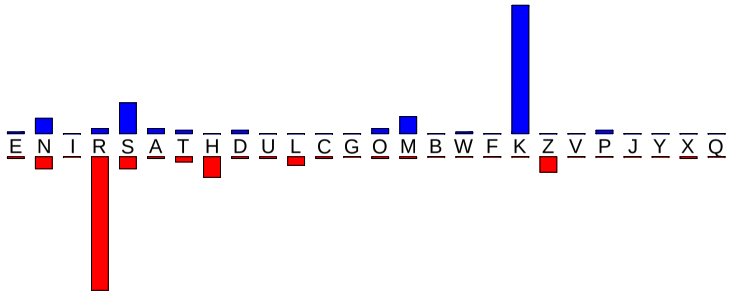
Kontaktdiagramm des Buchstabens P

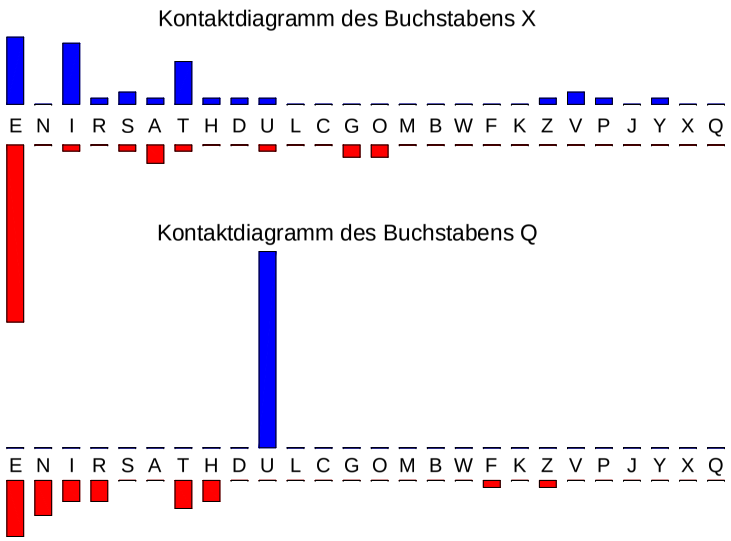


Kontaktdiagramm des Buchstabens J

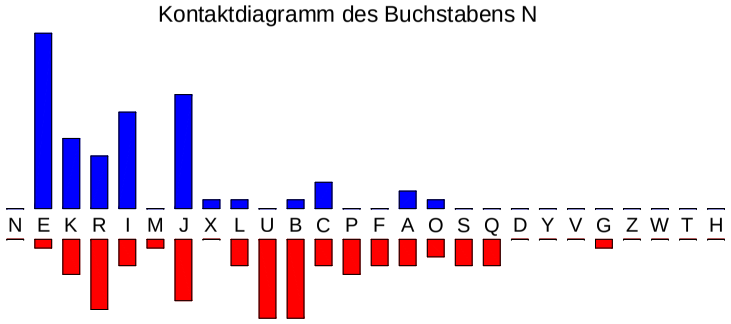


Kontaktdiagramm des Buchstabens Y

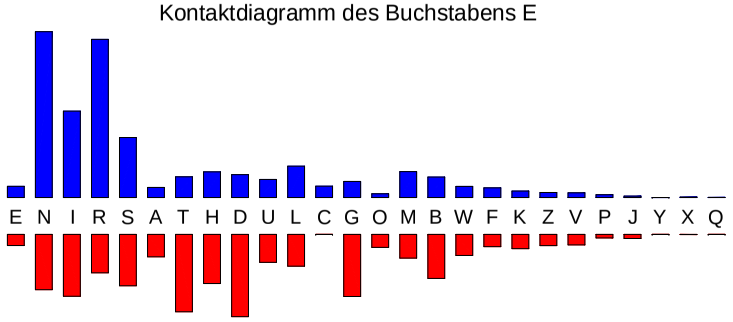




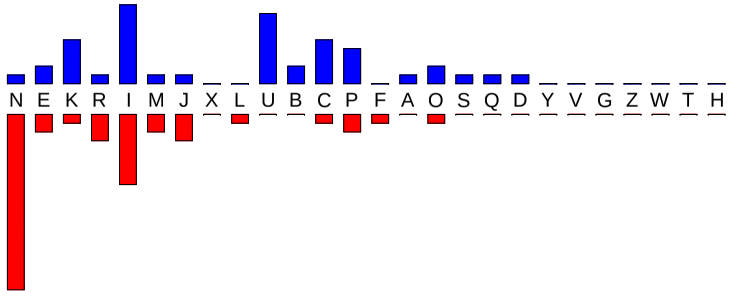
Wenden wir dies an auf unser Kryptogramm! Der häufigste Buchstabe dort ist N, was die Vermutung nahelegt, daß dies für ein Klartext-E stehen könnte. Hier ist das Kontaktdiagramm:



Ein Vergleich mit dem Kontaktdiagramm des Klartext-E zeigt deutliche Unterschiede:



Beispielsweise sehen wir nicht die auf-ab-auf-ab Struktur gleich am Anfang des Balkendiagramm über dem E. Andererseits müssen wir nach der Erfahrung mit unserer ersten Probeentschlüsselung damit rechnen, daß eine ganze Reihe von Buchstaben ihre Position verändert haben, und vor diesem Hintergrund sind die vier großen Balken im Anfangsbereich eigentlich alles, was wir realistischerweise erwarten können. Auch sonst ist die qualitative Übereinstimmung recht gut: Im unteren Teil des Diagramms ist die Verteilung deutlich homogener als im oberen, seltene Buchstaben kommen häufiger vor als in den Diagrammen zu den Klartextbuchstaben N, I, R, S und A, so daß wir ziemlich sicher sein können, daß der Kryptogrammbuchstabe N für ein Klartext-E steht.

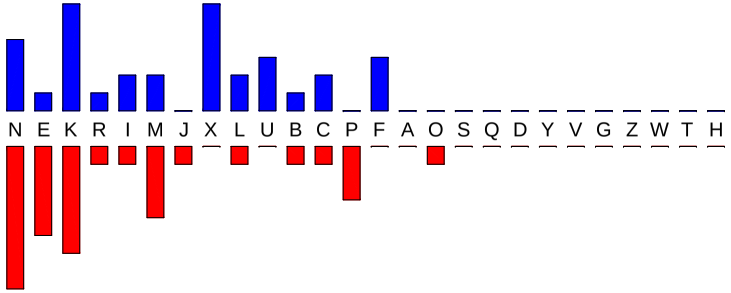


Der zweithäufigste Buchstabe im Kryptogramm ist das E. Auffällig an seinem Kontaktdiagramm ist der lange rote Balken unter dem bereits als Klartext-E erkannten häufigsten Kryptogrammbuchstaben N, der

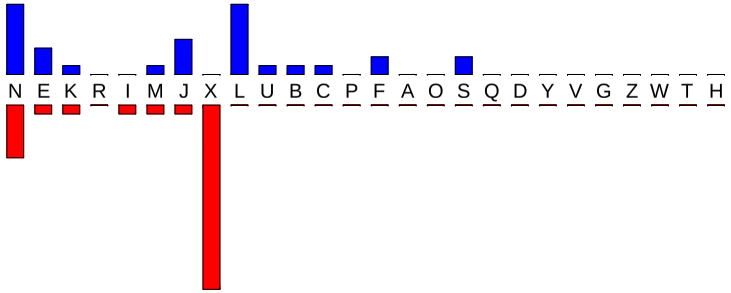
auch das Klartext-N auszeichnet; der zweite lange rote Balken unter dem I könnte dem unter Klartext-I oder eventuell auch U entsprechen. Auch die relativ homogene Verteilung der blauen Balken und die kaum vorhandenen roten Balken im Bereich der seltenen Buchstaben sprechen deutlich für ein N, so daß der Kryptogrammbuchstabe E wohl für ein Klartext-N steht.

Der dritthäufigste Buchstabe des Kryptogramms ist das K. Es zeichnet sich dadurch aus, daß die seltenen Buchstaben weder davor noch dahinter mit nennenswerter Häufigkeit vorkommen, daß häufig ein Klartext-E davorsteht, während N weder davor noch dahinter sonderlich oft vertreten ist. Dafür tritt K häufig als Doppelbuchstabe auf. Unter den in deutschen Klartexten häufigen Buchstaben verhält sich offenbar das S am ähnlichsten: Zwar passen die Balken ganz links besser zu einem R, aber die vielen weiteren langen Balken nach unten und die Häufigkeit von KK schließen dieses mit ziemlicher Sicherheit aus.

Kontaktdiagramm des Buchstabens K



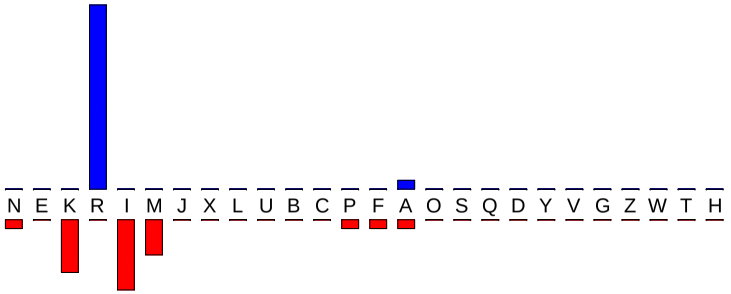
Kontaktdiagramm des Buchstabens R



Als nächstes müssen wir den Kryptogrammbuchstaben R identifizieren. Hier fällt als erstes der lange Balken unter dem X ins Auge; es gibt also einen mittelhäufigen Buchstaben, der das Geschehen vor dem gesuchten Buchstaben deutlich dominiert; ansonsten tritt dort nur noch Klartext-E mit nennenswerter Häufigkeit auf. Nach dem gesuchten Buchstaben ist die Verteilung deutlich homogener.

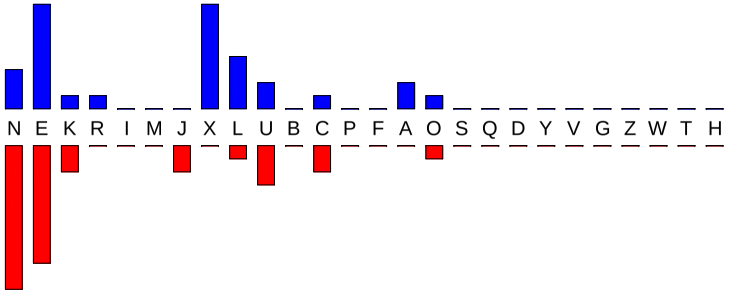
Damit ist eigentlich fast klar, daß R nur für ein H stehen kann, und daß der häufig davor stehende Buchstabe X im Klartext ein C sein muß. Ein direkter Vergleich des Kontaktdiagramms zu X mit dem von Klartext-C stützt diese Vermutung.

Kontaktdiagramm des Buchstabens X



Damit sind gleich zwei Buchstaben identifiziert; wenn wir nun wieder nach der Häufigkeit im Kryptogramm vorgehen, steht als nächstes das I auf dem Programm.

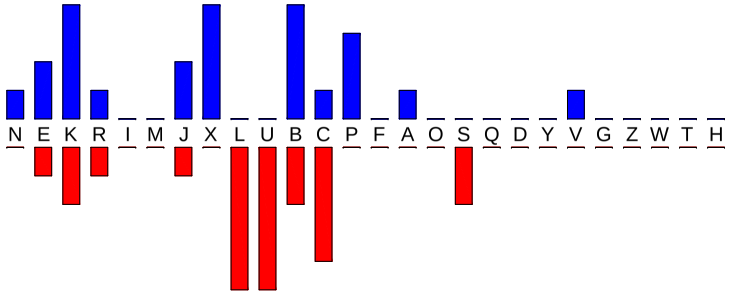
Kontaktdiagramm des Buchstabens I



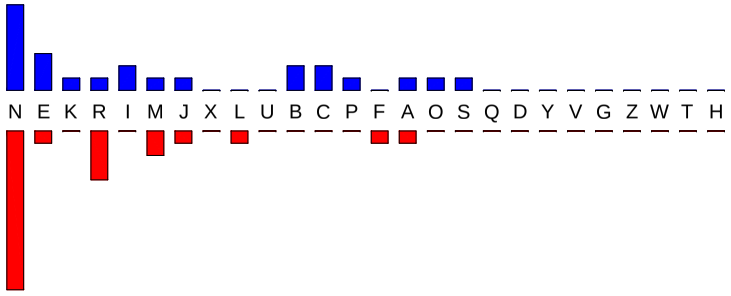
Es kommt häufig nach Klartext-E und N vor; insbesondere beim N auch vorher. Außerdem steht das gerade als C identifizierte X häufig dahinter. Unter den noch unbekannten Buchstaben ist offensichtlich I selbst der beste Kandidat; dieser Buchstabe wird also durch sich selbst verschlüsselt.

Als nächstes wartet das M auf seine Entschlüsselung. Es steht oft nach Buchstaben nur mittlerer Häufigkeit, praktisch nie nach E, aber gelegentlich davor. Zusammen mit I tritt es nie auf. Das deutet auf einen Vokal hin, und in der Tat paßt A sehr gut, wobei die Kombination AE natürlich von der Umschreibung der Umlaute herkommt.

Kontaktdiagramm des Buchstabens M



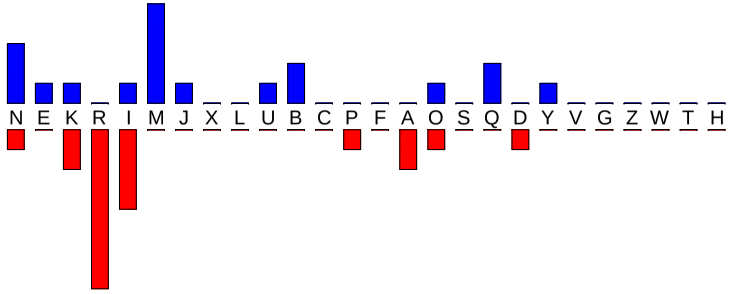
Kontaktdiagramm des Buchstabens J



Unser nächster Kandidat J mag keine seltenen Buchstaben um sich haben und versteht sich am besten mit dem E, insbesondere wenn es vor ihm steht. Die Entschlüsselung R bereitet damit keine Schwierigkeiten.

Das X haben wir bereits als C identifiziert, also kommt nun L an die Reihe.

Kontaktdiagramm des Buchstabens L



Es steht zwar gelegentlich hinter einem E, sehr viel häufiger aber hinter einem H oder I und praktisch nie hinter einem N. Dahinter sind A und E die häufigsten Buchstaben. Teile dieser Beschreibung passen zum T, andere zum D, aber alles zusammen paßt zu keinem der Buchstaben.

Unser Problem ist, daß der Buchstabe L im Kryptogramm nur zwanzig Mal vorkommt, und damit läßt sich keine sonderlich aussagekräftige Statistik über die Verteilung der 26 Buchstaben des Alphabets vor oder nach diesen zwanzig Stellen machen. Ab hier müssen wir also entweder mehrere Möglichkeiten in Betracht ziehen, oder aber zu einer anderen Strategie wechseln.

Mit acht von sechsundzwanzig Buchstaben haben wir zwar nur etwa 30% aller Buchstaben identifiziert, aber da es die acht häufigsten Buchstaben sind, kennen wir deutlich mehr als 30% des Klartexts: Nachzählen zeigt, daß wir von den 402 Buchstaben des Kryptogramms 278 identifiziert haben, also rund 70%. Damit sollte es nicht sehr schwierig sein, zumindest einige weitere Buchstaben aus dem Kontext zu erraten.

Hier ist noch einmal das Kryptogramm, wobei unter jedem bekannten Buchstaben dessen Entschlüsselung steht:

```
MBKFB MPLNL NIEAN KXRBP KKUMK KUNJC NEKXR SMKBN
A S   A  E  EIN E SCH   SS AS S ER  ENSCH  AS E
```

```
JENEC PKKEI XRLMB BNIEU MKMAX AJIEO LUNEC NEKXR
RNEN   SSNI CH A   EIN  ASA C  RIN    EN  ENSCH

NEIEU INRFN REIXR LMBBN IEICK XRJNI ANEBN KNEPN
ENIN  IEH E HNICH  A  E INI S CHREI  EN E SEN E

ALKIX RNIEQ NJEPN EDLIO SNKNE EIXRL MBBNI EIEJN
 SIC HEIN  ERN E N  I   ESEN NICH  A  EI NINRE

XREPE OKKMX RNEKF BBUNJ CNEKX RKIXR CPNRN CMXRN
CHN N  SSAC HENS     ER  ENSC HSICH   EHE  ACHE

EKFEU NJEMP XRUNJ SNIKR NILBN RJNEC PKKCM ECILQ
NS N  ERNA  CH ER  EISH EI  E HREN   SS A N I

NJOEP NONER FNJNE UMKKU INKCI LQNJK LMEUO NKXRM
ER N  E ENH  EREN  ASS  IES I   ERS  AN   ESCHA

RSMJR NJJBN RJNJB MNCGN BUMCM VPEUC FJILY UINKN
H ARH ERR E HRER  AE  E   A A   N    RI    IESE

ANIUN ECFXR LNEIR EUMJP CEIXR LBNIU NEUNE ESNJA
  EI E N  CH  ENIH N AR   NICH   EI  EN EN N ER

FNKNK LJNIX RNCMX RLOIA LEIXR LMPDU NEBNR JNJMX
ESES  REIC HE AC H  I   NICH  A    EN EH RERAC

RL
H
```

Gegen Ende der ersten Zeile haben wir im Klartext die Stelle

„UassUerCensch“ ,

Hier kann eigentlich nur „dass der Mensch“ gemeint sein, d.h. U steht für D und C für M.

In der zweiten Zeile sehen wir die Folge „ssnichLaB“; hier kann L eigentlich nur für T stehen, insbesondere da wir das L bereits provisorisch als D oder T identifiziert haben.

In der dritten Zeile ist der drittletzte Block entschlüsselt als „chrei“; davor steht ein „s“, dahinter ein unbekannter Buchstabe gefolgt von „en’. Dies spricht für eines der beiden Wörter „schreiben“ oder „schreiten“,

wobei letzteres nicht in Frage kommt, da wir bereits wissen, daß T durch L verschlüsselt wird, unser gesuchter Buchstabe aber durch A. Damit ist wohl A die Verschlüsselung des Buchstaben B.

Damit sind vier weitere Buchstaben identifiziert, und wenn wir sie an *allen* Stellen in die Entschlüsselung einsetzen, füllt sich der bereits entschlüsselte Teil des Texts und wir bekommen neue Kontexte zur Identifikation oder zumindest zum Erraten noch fehlender Buchstaben:

```
MBKFB MPLNL NIEAN KXRBP KKUMK KUNJC NEKXR SMKBN
A S   A TET EINBE SCH   SSDAS SDERM ENSCH  AS E

JENEC PKKEI XRLMB BNIEU MKMAX AJIEO LUNEC NEKXR
RNENM  SSNI CHTA   EIND ASABC BRIN  TDENM ENSCH

NEIEU INRFN REIXR LMBBN IEICK XRJNI ANEBN KNEPN
ENIND IEH E HNICH TA  E INIMS CHREI BEN E SEN E

ALKIX RNIEQ NJEPN EDLIO SNKNE EIXRL MBBNI EIEJN
BTSIC HEIN  ERN E N TI   ESEN NICHT A  EI NINRE

XREPE OKKMX RNEKF BBUNJ CNEKX RKIXR CPNRN CMXRN
CHN N  SSAC HENS    DER MENSC HSICH M EHE MACHE

EKFEU NJEMP XRUNJ SNIKR NILBN RJNEC PKKCM ECILQ
NS ND ERNA  CHDER  EISH EIT E HRENM  SSMA NMIT

NJOEP NONER FNJNE UMKKU INKCI LQNJK LMEUO NKXRM
ER N  E ENH  EREN DASSD IESMI T ERS TAND  ESCHA

RSMJR NJJBN RJNJB MNCGN BUMCM VPEUC FJILY UINKN
H ARH ERR E HRER  AEM E  DAMA   NDM  RIT  DIESE

ANIUN ECFXR LNEIR EUMJP CEIXR LBNIU NEUNE ESNJA
BEIDE NM CH TENIH NDAR  MNICH T EID ENDEN N ERB

FNKNK LJNIX RNCMX RLOIA LEIXR LMPDU NEBNR JNJMX
 ESES TREIC HEMAC HT IB TNICH TA  D EN EH RERAC

RL
HT
```

Ab dem dritten Block in der ersten Zeile steht nun

„einbeschBPssdassdermenschSasBernenmPss“ ,

Selbstverständlich muß P für einen Vokal stehen, und aus dem Zusammenhang ist klar, daß es ein U sein muß. Damit ist auch die Entschlüsselung von P als L klar, und S kann eigentlich nur für W stehen.

Da wir nun einen weiteren Vokal identifiziert haben und auch L ein in deutschen Klartext recht häufiger Buchstabe ist, lohnt es sich wahrscheinlich, vor der Suche nach neuen Identifikationen die drei gerade gefundenen im gesamten Text einzusetzen. Wir erhalten

```
MBKFB MPLNL NIEAN KXRBP KKUMK KUNJC NEKXR SMKBN
ALS L AUTET EINBE SCHLU SSDAS SDERM ENSCH WASLE

JENEC PKKEI XRLMB BNIEU MKMAX AJIEO LUNEC NEKXR
RNENM USSNI CHTAL LEIND ASABC BRIN  TDENM ENSCH

NEIEU INRFN REIXR LMBBN IEICK XRJNI ANEBN KNEPN
ENIND IEH E HNICH TALLE INIMS CHREI BENLE SENUE

ALKIX RNIEQ NJEPN EDLIO SNKNE EIXRL MBBNI EIEJN
BTSIC HEIN  ERNUE N TI  WESEN NICHT ALLEI NINRE

XREPE OKKMX RNEKF BBUNJ CNEKX RKIXR CPNRN CMXRN
CHNUN  SSAC HENS  LLDER MENSC HSICH MUEHE MACHE

EKFEU NJEMP XRUNJ SNIKR NILBN RJNEC PKKCM ECILQ
NS ND ERNAU CHDER WEISH EITLE HRENM USSMA NMIT

NJOEP NONER FNJNE UMKKU INKCI LQNJK LMEUO NKXRM
ER NU E ENH  EREN DASSD IESMI T ERS TAND  ESCHA

RSMJR NJJBN RJNJB MNCGN BUMCM VPEUC FJILY UINKN
HWARH ERRLE HRERL AEM E LDAMA  UNDM  RIT  DIESE

ANIUN ECFXR LNEIR EUMJP CEIXR LBNIU NEUNE ESNJA
BEIDE NM CH TENIH NDARU MNICH TLEID ENDEN NWERB

FNKNK LJNIX RNCMX RLOIA LEIXR LMPDU NEBNR JNJMX
 ESES TREIC HEMAC HT IB TNICH TAU D ENLEH RERAC

RL
HT
```


Niemand, der schon je ein Kreuzworträtsel gelöst hat, sollte Schwierigkeiten haben mit den wenigen Lücken, die jetzt noch übrig sind; der Klartext springt nun förmlich ins Auge: Natürlich kann die Lücke im ersten Block der ersten Zeile nur für ein O stehen, und die Lücke in der zweiten Zeile kann nur ein G sein. Selbst in der vierten Zeile, wo es in einem Wort noch drei Lücken gibt, fällt es nicht schwer, diese zu schließen. Mit Wortgrenzen und Satzzeichen geschrieben ist der Klartext

Also lautet ein Beschluss:
Dass der Mensch was lernen muss. –
Nicht allein das A-B-C
Bringt den Menschen in die Hoeh';
Nicht allein im Schreiben, Lesen
uebt sich ein vernuenftig Wesen;
Nicht allein in Rechnungssachen
Soll der Mensch sich Muehe machen;
Sondern auch der Weisheit Lehren
Muss man mit Vergnuegen hoeren.

Dass dies mit Verstand geschah,
War Herr Lehrer Laempel da. –
– Max und Moritz, diese beiden,
Mochten ihn darum nicht leiden;
Denn wer boese Streiche macht,
Gibt nicht auf den Lehrer acht.

(WILHELM BUSCH verwendet natürlich sowohl Umlaute als auch das „ß“; zum besseren Vergleich mit dem Kryptogramm habe ich sie aber so gelassen, wie ich sie vor der Verschlüsselung umschrieben habe.)

Damit haben wir also mit relativ geringem Aufwand ein Kryptogramm entschlüsselt, bei dem es auf den ersten Blick so aussah, als sei das nur möglich durch Ausprobieren von über 403 Quadrillion Möglichkeiten.

Wir haben zur Entschlüsselung zwar einen Computer als Hilfsmittel benutzt, aber auch wenn uns der das Leben etwas bequemer gemacht hat, war er nicht wirklich erforderlich: Die Buchstaben im Kryptogramm hätten wir mit nur unwesentlich größerem Aufwand auch von Hand durchzählen können, und die Kontaktdiagramme wurden in der Zeit,

als man noch alles von Hand machen mußte, gezeichnet, indem man einfach Querstriche über oder unter einen Buchstaben zeichnete, wenn dieser vor oder nach dem gerade untersuchten auftrat.

Nachdem die acht bis zehn häufigsten Buchstaben identifiziert sind, kann der Computer ohnehin nicht mehr weiterhelfen; jetzt ist eher Erfahrung mit der Sprache gefragt, am besten solche, die durch das Lösen vieler Kreuzworträtsel erworben wurde.

Als Kuriosität am Rande sei erwähnt, daß die Engländer im zweiten Weltkrieg das Personal für *Bletchley Park*, ihre Dechiffrierzentrale, tatsächlich zum Teil dadurch rekrutierten, daß sie Kreuzworträtselwettbewerbe in Zeitungen plazierten und die Sieger als potentielle neue Mitarbeiter ins Visier nahmen.

Die Häufigkeitsdiagramme für die seltenen Buchstaben nützen natürlich fast nie sonderlich viel, da hier die Schwankungen zwischen verschiedenen Texten besonders groß sind. Das beste Beispiel dafür ist das obige Diagramm zum Buchstaben Y: Es kommt im wesentlichen dadurch zustande, daß es in JEAN PAULS Roman einen häufig erwähnten Badearzt namens *Strykius* gibt.

Ein praktisches Problem bei der Schlüsselvereinbarung der hier betrachteten Substitutionschiffren bestand darin, daß eine Permutation der 26 Buchstaben im allgemeinen nur schwer zu übermitteln und auch zu merken ist, was unter anderem auch KERCKHOFFS dritte Forderung verletzt. Die klassische Lösung ging einfach aus von einem Wort oder einer Folge von Wörtern und schrieb diese von links nach rechts unter die Buchstaben A, B, C, ... Dabei mußte natürlich jeder Buchstabe, der schon dasteht, gestrichen werden. Falls am Ende noch nicht alle 26 Positionen gefüllt waren, nahm man dazu die noch verbleibenden Buchstaben in alphabetischer Reihenfolge.

Geht man aus von „Max und Moritz“, führt dies auf die Verschlüsselungstabelle

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

Auf diese Weise wurde das gerade behandelte Beispielkryptogramm verschlüsselt.

§ 2: Polyalphabetische Substitutionen

Da eine bloße Permutation des „Alphabets“ der Buchstaben oder Bytes offensichtlich keine nennenswerte Sicherheit bietet, kann man als nächstes versuchen, mehrere solcher Permutationen anzuwenden. Natürlich hat es keinen Sinn, sie alle jeweils auf denselben Buchstaben anzuwenden – das Produkt mehrerer Permutationen ist schließlich wieder nur eine einfache Permutation. Stattdessen unterteilt man die Nachricht in Blöcke einer festen (geheimgehaltenen) Länge n, wählt n Permutationen $\pi_i \in \mathbf{S}_n$ und verschlüsselt jeweils das i-te Zeichen eines Blocks mit der Permutation π_i.

Am bekanntesten ist die sogenannte VIGENÈRE-Chiffren, benannt nach BLAISSE DE VIGENÈRE, der sie 1586 in seinem Buch *Traictés des chiffres ou secrètes manières d'escrire* beschrieb; sie wurden allerdings bereits im 1518 erschienenen Buch *Polygraphia* des Abbé JEAN TRITHÈME erwähnt und gehen laut VIGENÈRE zurück auf die Hebräer. Hier sind alle π_i CAESAR-Chiffren; der Einsatz allgemeiner Permutationen π_i dürfte jedoch wenn überhaupt nur unwesentlich jünger sein.

L. SACCO, der ehemalige Chef des Chiffrierdienstes der italienischen Armee, schreibt in seinem *Manuel de cryptographie* (Paris, 1951):

> *Vigenère réussit à produire un chiffre plus faible et moins commode que les précédents, qui n'en connut pas moins une fortune imméritée, particulièrement dans les cent dernières années, où il fut adopté par de nombreuses armées, même après la publication d'un moyen de décryptement (1863), indice évident de décadence dans la domaine de la cryptographie.*

Die VIGÈNERE-Chiffre oder ähnliche Verfahren wurden bis Ende des neunzehnten Jahrhunderts unter anderem von der österreichischen, der französischen und der italienischen Armee benutzt; von letzterer sogar bis 1917. In verschiedenen Computerprogrammen ist ähnliches heute noch zu finden.

Um zu verstehen, warum SACCO trotzdem so abfällig darüber spricht, müssen wir ihre Sicherheit etwas genauer betrachten und insbesondere sehen, wann sie mit welchem Aufwand geknackt werden kann.

Hauptproblem des Kryptanalytikers ist offensichtlich die Bestimmung der Blocklänge n, denn sobald diese bekannt ist, weiß man, welche Buchstaben mit demselben Alphabet verschlüsselt wurden und kann (leicht modifiziert) die kryptanalytischen Verfahren aus dem vorigen Paragraphen anwenden.

Mit dieser Ermittlung der Periode hatten die Kryptanalytiker im letzten Jahrhundert lange ihre Probleme; den ersten Ansatz fand der preußische Offizier FRIEDRICH W. KASIKI (1805–1881) und veröffentlichte ihn 1863 in einem damals kaum beachteten nur 95 Seiten dicken Buch mit dem Titel *Die Geheimschriften und die Dechiffrierkunst.* Seine Idee war die folgende: Gewisse Wörter und Buchstabenkombinationen wie etwa die bestimmten Artikel sind in fast allen Texten sehr häufig. Wenn nun ein langer Text mit einem polyalphabetischen Verfahren kurzer Periode verschlüsselt wird, ist es sehr wahrscheinlich, daß solche Buchstabenfolgen mehrfach auf die gleiche Weise verschlüsselt werden. Man suche daher im Kryptogramm nach zweimal vorkommenden Buchstabenfolgen (Heute nennt man so etwas ein KASIKI-Paar) und berechne deren Abstand. Die Periode sollte ein Teiler von relativ vielen dieser Abstände sein, wobei Abstände, die zu langen Buchstabenfolgen gehören, natürlich höher zu gewichten sind als solche, die etwa nur zu Buchstabenpaaren gehören: Bei letzteren ist die Wahrscheinlichkeit, daß es sich um eine zufällige Koinzidenz handelt, erheblich größer.

Die Suche nach KASIKI-Paaren ist recht aufwendig, und sie führen im allgemeinen nur dann zu einer Lösung, wenn das Kryptogramm erheblich länger als ist als der verwendete Schlüssel. Eine Alternative fand um 1920 der wohl bedeutendste der klassischen Kryptologen, WILLIAM FRIEDMAN (1891–1969). Er wurde als WOLFE FRIEDMAN in Rußland geboren, aber als seine Eltern 1892 nach Amerika emigrierten, änderten sie seinen Vornamen. Er studierte zunächst Landwirtschaft, spezialisierte sich später auf Genetik und bekam 1915 eine Stelle als Genetiker bei dem Textilkaufmann GEORGE FABYAN, auf dessen Gut Riverbank in Geneva, Illinois. Dieser unterhielt dort Laboratorien für Akustik, Chemie, Genetik und Kryptologie – letztere mit dem Ziel zu beweisen, daß BACON der wahre Autor der SHAKESPEAREschen Schriften sei. Dadurch mußte sich FRIEDMAN zwangsläufig auch für Kryptologie interessieren

und war damit so erfolgreich, daß er bald Leiter der Laboratorien sowohl für Genetik als auch für Kryptologie war.

Mit dem Kriegseintritt der Vereinigten Staaten im April 1917 mußte sich auch die amerikanische Armee für Kryptographie interessieren, und da es außer Riverbank kein amerikanisches Zentrum für Kryptologie gab, wurden nicht nur aufgefangene Kryptogramme dorthin geschickt, sondern auch Armee-Offiziere, die bei FRIEDMAN in Kryptanalyse ausgebildet werden sollten. Diese Kurse wurden nach Kriegsende fortgesetzt, und 1921 verließ FRIEDMAN Riverbank, um anschließend in verschiedenen militärischen Funktionen sowohl Codes zu entwerfen als auch Codes zu knacken. Von ihm stammt das Wort *Kryptanalyse*, das das unbefugte Dechiffrieren vom legitimen unterscheidet.

Seine 1925 perfektionierte Idee zur Bestimmung der Periode ist folgende: Man betrachte für eine natürliche Zahl n die Wahrscheinlichkeit dafür, daß ein Buchstabe mit seinem n-ten Nachfolger übereinstimmt.

Falls n ein Vielfaches der Periode ist, wurden die beiden Buchstaben mit derselben Permutation verschlüsselt; da Summen nicht von der Reihenfolge der Summanden abhängen, ist die Wahrscheinlichkeit also

$$\sum_{i=1}^{26} p_i^2 \,,$$

wobei p_i die Häufigkeit des i-ten Buchstabens im Klartext ist.

Falls n dagegen *kein* Vielfaches der Periode ist, stammen ein Buchstabe und sein n-ter Nachfolger bei hinreichend langer Periode fast immer aus verschiedenen Permutationen, man kann sie also in allererster Näherung als voneinander unabhängig ansehen. Dann ist die Wahrscheinlichkeit einer Koinzidenz ungefähr gleich

$$\sum_{i=1}^{26} \left(\frac{1}{26}\right)^2 = \frac{1}{26} \,.$$

Betrachtet man die deutsche Sprache auf Grundlage von *Dr. Katzenbergers Badereise*, so ist

$$\sum_{i=1}^{26} p_i^2 \approx 0{,}0789 \quad \text{verglichen mit} \quad \frac{1}{26} \approx 0{,}0385 \,,$$

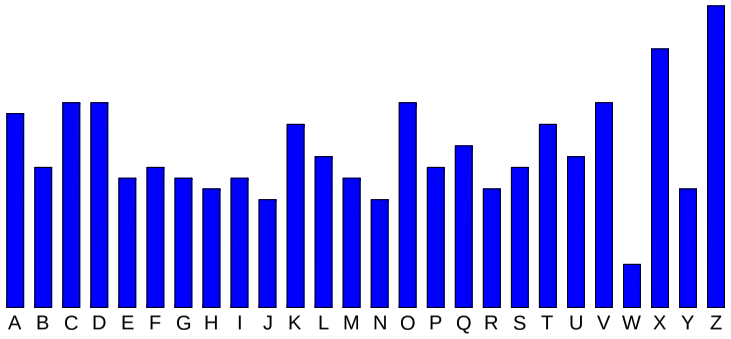
die beiden Werte unterscheiden sich also deutlich. In anderen Sprachen erhält man zwar andere Werte, aber der Unterschied ist auch dort unübersehbar.

In einem relativ kurzen Kryptogramm wird man selbstverständlich andere Werte berechnen, aber trotzdem sollte es im allgemeinen für gleiche Alphabete mehr Koinzidenzen geben als für verschiedene.

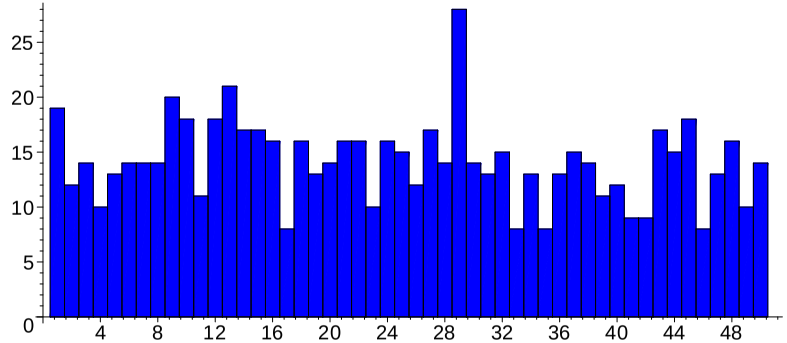
Betrachten wir als erstes Beispiel das Kryptogramm

```
PDOAA KKMQB LYORJ FTLXM OQGYU XTKCQ LXLVB ATCBU MJEDM SQZJJ
OZPHT AEZZD FFRSK XTYZV MVVZS QTZUO CTGDX ENOGX XGCOI GXHBN
OZXCC COJXJ PBQTV XTDOF ZRZND FHADX LZCQC NPBSL HTDVM ESKSP
YFCDB QHCEV ZDVIA DRKTR PGJDQ RFUUW AKQXP UZQVD AMXLF XGFGC
BTCFT KYRES GSALT VGZWT VSQOP UCALM ZFGMA VNDOZ ZNJOA HVDDQ
CMONK CPPBU ZZZYK PYRAD LZLYV GXNUB IURKX OEBBZ DNAWE UOVKH
TPISF DLIIQ LOXHO PAIZZ CAWEV XYSXU IZVUQ MAICE SKYXL AIZEH
KVNCR KUXYH IFKMK BJVHO TRSXX ZIEZJ
```

Auszählen der Buchstabenhäufigkeiten zeigt, daß die Buchstabenhäufigkeiten sehr viel gleichmäßiger verteilt sind als in deutschem Text; dies deutet darauf hin, daß keine *monoalphabetische*, sondern eine *polyalphabetische* Substitution angewandt wurde, d.h. die verschiedenen Buchstaben des Kryptogramms wurden nicht alle mittels ein und derselben Permutation aus den Klartextbuchstaben berechnet.



Buchstabenhäufigkeiten des Beispielkryptogramms

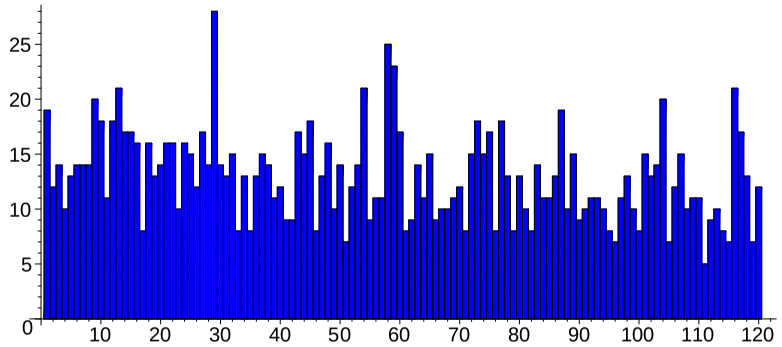


Koinzidenzen im Beispielkryptogramm für Abstände bis 50

Für das obige Kryptogramm zeigt Abbildung vier die Verteilung der Koinzidenzen bei n Buchstaben Abstand. Das erste Maximum bei $n = 1$ ist natürlich nicht ernst zu nehmen; sonst hätten wir eine monoalphabetische Substitution, deren Häufigkeitsverteilung deutlich anders aussieht als die hier beobachtete. Auch die Spitzen bei $n = 9$ und $n = 13$ sind wohl eher zufällig, denn bei $n = 18$ und $n = 26$ sind die Anzahlen eher klein. Unübersehbar ist das absolute Maximum bei $n = 29$; um zu sehen, ob dies das „richtige“ Maximum ist, müssen wir allerdings etwas mehr Werte berechnen. Die nächste Abbildung zeigt die entsprechenden Abstände bis einschließlich 120, und man sieht doch recht deutliche Ausschläge bei $n = 58$, 67 und 116. Wir wollen daher als erstes versuchen, das Kryptogramm unter der Annahme zu dechiffrieren, daß $n = 29$ ist.

Wir arbeiten also in diesem Ansatz mit der Hypothese, daß zwei Buchstaben genau dann mit derselben Permutation verschlüsselt sind, wenn ihr Abstand ein Vielfaches von 29 ist.

Deshalb teilen wir das Kryptogramm auf in 29 Buchstabenfolgen, die jeweils mit derselben Permutation verschlüsselt sein sollten. Ein Kryptanalytiker würde nun die 29 Häufigkeitsverteilungen dazu betrachten; da sie meisten (ganz grob) ungefähr so aussehen wie die von CAESAR-Substitutionen, würde er VIGENÈRE als wahrscheinlichste Möglichkeit ansehen.



Koinzidenzen im Beispielkryptogramm für Abstände bis 120

Die Entzifferung von gleich 29 CAESAR-Chiffren ist allerdings alles andere als kurzweilig; idealerweise sollten wir also auch das noch automatisieren.

Zum Glück läßt sich FRIEDMANs Idee auch darauf anwenden: p_i für $1 \leq i \leq 26$ sei die Wahrscheinlichkeiten dafür, daß ein Buchstabe in deutschem Klartext gleich dem i-ten Buchstaben des Alphabets ist, q_i entsprechend die, daß dies ein Buchstabe des Kryptogramms ist. Mit $\oplus$ bezeichnen wir eine Addition, bei der das Ergebnis durch Reduktion modulo 26 in das Intervall von 1 bis 26 gezwungen wird.

Falls r die Anzahl ist, um die wir zur Entschlüsselung verschieben müssen, ist $p_i \approx q_{i+r}$, also

$$\sum_{i=1}^{26} p_i q_{i\oplus r} \approx \sum_{i=1}^{r} p_i^2 \,.$$

Für jeden anderen Wert von r ist die Korrelation zwischen p_i und $q_{i\oplus r}$ schwächer, die Summe sollte also kleiner sein. Der Effekt ist sicherlich nicht so ausgeprägt, wie bei FRIEDMANs Test, und zumindest bei kurzen Kryptogrammen kann das Maximum auch gelegentlich bei einem falschen r liegen, aber das richtige r sollte im allgemeinen relativ weit oben liegen.

Wendet wir dies hier an, erhalten wir folgende Kandidaten für Schlüsselbuchstaben, wobei links jeweils der mit der größten $\sum p_i q_i$ steht, danach

die vier mit den nächstkleineren:

```
N 0,085 E 0,057 Z 0,055 O 0,050 A 0,048
A 0,083 E 0,062 N 0,047 Z 0,046 K 0,045
T 0,080 D 0,068 C 0,064 G 0,059 P 0,057
H 0,077 I 0,054 L 0,051 Y 0,049 G 0,049
M 0,077 Z 0,067 Q 0,055 D 0,051 I 0,051
B 0,055 I 0,054 F 0,053 J 0,053 P 0,048
P 0,074 T 0,071 C 0,059 M 0,058 D 0,056
T 0,069 G 0,055 E 0,052 R 0,050 I 0,048
A 0,081 N 0,070 J 0,061 L 0,050 K 0,047
G 0,092 C 0,075 F 0,064 P 0,056 Q 0,048
S 0,079 W 0,066 J 0,060 F 0,055 N 0,054
S 0,061 F 0,060 R 0,053 W 0,052 J 0,047
E 0,080 R 0,056 D 0,054 I 0,052 V 0,049
I 0,070 M 0,062 W 0,056 L 0,052 P 0,051
I 0,089 E 0,059 H 0,053 R 0,053 W 0,051
A 0,080 E 0,079 N 0,074 R 0,055 G 0,052
A 0,067 K 0,061 B 0,054 O 0,050 Z 0,050
R 0,076 N 0,072 W 0,056 B 0,054 I 0,049
K 0,072 B 0,060 E 0,059 V 0,056 U 0,051
R 0,075 E 0,055 N 0,054 Q 0,050 D 0,049
Y 0,069 S 0,059 P 0,058 J 0,054 W 0,054
P 0,099 C 0,061 L 0,053 M 0,051 Y 0,049
T 0,092 X 0,071 G 0,061 K 0,050 H 0,045
O 0,080 K 0,068 B 0,066 F 0,063 S 0,047
L 0,059 P 0,056 W 0,055 M 0,051 X 0,050
O 0,083 K 0,068 B 0,058 G 0,056 N 0,052
G 0,080 C 0,068 K 0,062 D 0,055 R 0,050
I 0,083 E 0,059 Z 0,053 V 0,053 R 0,051
E 0,107 N 0,059 A 0,056 R 0,054 I 0,048
```

Die Buchstabenfolge in der ersten Spalte legt nahe, daß hier kryptographisch unsorgfältig gearbeitet wurde: Der Schlüssel wurde wohl nicht zufällig gewählt, sondern als sinnvoller Teil der deutschen Sprache, wobei wir aber nicht alle Buchstaben auf Anhieb richtig erraten haben. Wir können nun entweder versuchen, den Schlüssel anhand der Buch-

staben aus den folgenden Spalten zu erraten (was hier wohl selbst ohne diese nicht sonderlich schwerfällt), oder aber wir entschlüsseln einfach mit dem wahrscheinlich falschen Schlüssel und korrigieren anhand des entschlüsselten Textes. Wenn wir diesen in Zeilen der Länge 29 aufschreiben, stehen jeweils die Buchstaben, die zum gleichen Alphabet gehören, untereinander, so daß es für jeden Schlüsselbuchstaben mehrere Überprüfungsmöglichkeiten gibt.

Der Entschlüsselungsungsversuch führt auf folgendes Ergebnis: (In der ersten Zeile steht der Schlüssel.)

```
N A T H M B P T A G S S E I I A A R K R Y P T O L O G I E

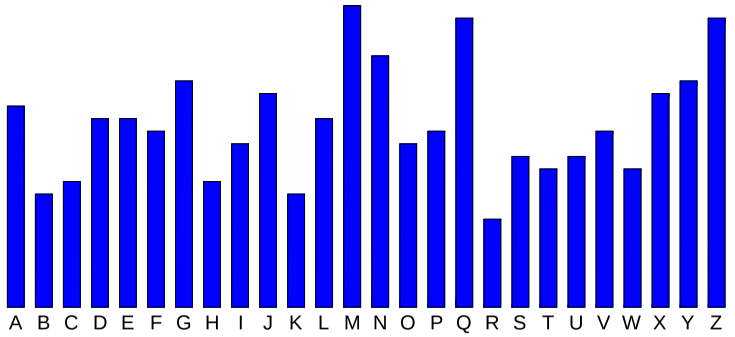
D E I I N M A G R I E R T A S G U D I E N G A N G M A T H
E M R T I D Q N D I N F O N M N T I K B I E T E T I H N E
N E Z N E U A R U F S O R E E A T I E R T E W I S S E N S
C H R F T E E C H E A U S X I Y D U N G I N D E N F A E C
H E I N M T P H E M A T I G U A D I N F O R M A T I K E R
S T E A C A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

Damit sollte wohl jeder Leser den Klartext rekonstruieren können. Bei einer Schlüssellänge von 29 und einer Textlänge von 480, bei etwa 16,5 Zeichen pro Alphabet also, ist das Vigenère-Verfahren somit schon völlig unsicher. Bei noch mehr Buchstaben pro Alphabet wird das Knacken noch einfacher: Betrachten wir als Beispiel denselben Klartext wie oben, chiffriert mit einem kürzeren Schlüssel:

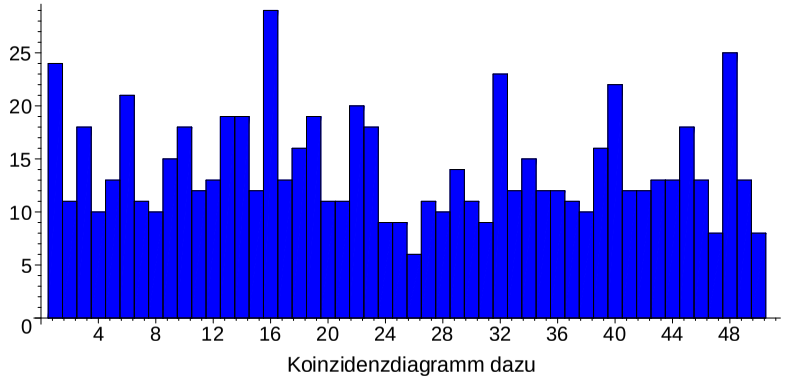
```
PZWDV AKLNZ ZDMDE MGYNZ VNGSC DVFAD YTFDP PVKOS BFMYT SUDND
JOMAO MJVIQ BMQUQ MZAAV XNAEO UXQFX IDXNM UYHDR AFEHO AQVZN
JPRIQ EBUGC QGRVJ XPLXS IROTX LMMUF ZILPT KKIHM MHWXD NYIIJ
NESVD VTGDX ZZSOA JNJEU ZZLHQ LUXMA CLXJE EZPXQ NXUYJ IIBYW
ETCFN MSXZH FOPLS FPZFG GCUGR JWHIA OPQEY PTLUM MPHCN BKWAZ
```



```
IQGCQ KNZNY MUGGO TCXND ELQYN KTVSR WKCQF ZFBWZ WJLLX IEGGA
FHZYA MRVBP QJNNV QAQQG PYJMM YYYAE WQBCQ GEOZY QLTOW YMQLH
ZWMGQ ZDLXF JJOME SGGSZ SBMTK NJJVY
```



Häufigkeitsverteilung der Buchstaben im Kryptogramm



Koinzidenzdiagramm dazu

Für dieses Kryptogramm sehen wir wieder anhand der relativ gleichmäßigen Verteilung der Buchstabenhäufigkeiten, daß es sich um eine polyalphabetische Verschlüsselung handeln muß.

Das Diagramm der Koinzidenzen bei n Buchstaben Abstand hat bei $n = 6$ eine erste Spitze; diese kann jedoch ignoriert werden, da es bei

$n = 12$ und $n = 24$ keine Maxima gibt. Die Maxima bei $n = 16, 32$ und 48 dagegen lassen eine Periodenlänge von 16 als wahrscheinlich erscheinen. Es gibt zwar auch eine ziemlich hohe Spitze bei $n = 40$, was vielleicht doch auf eine Periode acht hindeutet, da allerdings weder $n = 8$ noch $n = 24$ zu sonderlich großen Häufigkeiten führen, spricht das Diagramm doch eher für $n = 16$. Der pq-Test schlägt folgende Schlüsselbuchstaben vor:

```
N 0,066 M 0,052 Q 0,050 R 0,048 J 0,046
E 0,079 A 0,056 F 0,055 I 0,050 N 0,050
U 0,079 H 0,058 E 0,057 Q 0,056 D 0,051
E 0,064 D 0,055 P 0,049 F 0,048 T 0,048
R 0,083 V 0,059 E 0,050 I 0,046 Q 0,045
S 0,087 B 0,051 O 0,049 R 0,048 D 0,048
T 0,077 P 0,070 G 0,068 F 0,047 K 0,045
U 0,067 H 0,052 L 0,052 Y 0,052 V 0,049
D 0,075 Z 0,060 Q 0,056 U 0,049 H 0,046
I 0,072 Z 0,065 M 0,055 E 0,047 N 0,046
E 0,069 R 0,058 I 0,049 P 0,048 F 0,047
N 0,084 R 0,053 J 0,049 W 0,048 A 0,047
G 0,076 K 0,050 Z 0,049 X 0,047 Y 0,046
A 0,078 W 0,073 B 0,063 F 0,051 N 0,050
N 0,090 J 0,053 W 0,052 A 0,047 R 0,042
G 0,091 X 0,057 T 0,052 K 0,048 F 0,047
```

Hier steht gleich in der ersten Spalte der sehr wahrscheinlich aussehende Schlüssel `NEUERSTUDIENGANG`, der auch in der Tat zu verständlichem Klartext führt (entnommen aus einer Informationsbroschüre der Fakultät für Mathematik und Informatik für Schüler zu der Zeit, als der gerade auslaufende Diplomstudiengang noch neu war):

> *Der integrierte Studiengang Mathematik und Informatik bietet Ihnen eine berufsorientierte wissenschaftliche Ausbildung in den Fächern Mathematik und Informatik. Erst nach zwei Jahren entscheiden Sie sich für eine der beiden Ausrichtungen „Mathematik“ und „Informatik“ und damit auch für einen der beiden Abschlüsse „Diplom-Mathematiker“ oder „Diplom-Informatiker“. Während der ersten beiden Studienjahre sind alle Veranstaltungen gemeinsam.*

Informationstheoretische Betrachtungen zeigen, daß schon bei weniger als zwei Buchstaben pro Alphabet der Klartext und der Schlüssel mit hoher Wahrscheinlichkeit eindeutig durch den Chiffretext bestimmt sind. Zu einer auf der Informationstheorie beruhenden Kryptanalyse benötigt man allerdings praktisch *unbeschränkte* Ressourcen, denn die Größen, mit denen hier gerechnet wird, sind definiert als Summen über den Raum *aller* möglicher Schlüssel; im Falle des obigen Schlüssels der Länge 29 wäre das eine Summe über $26^{29} \approx 10^{42}$ Summanden. Kryptanalytiker suchen daher nach weniger aufwendigen Verfahren – und finden Sie auch in vielen Fällen.

§3: Der one time pad

Der *one time pad* („Einmalblock“) ist ein Spezialfall des VIGENÈRE-Verfahrens; daß er trotzdem einen eigenen Paragraphen bekommt, liegt an einem entscheidenden Punkt: Während die typische VIGENÈRE-Chiffre, wie wir im vorigen Paragraphen gesehen haben, keinerlei Sicherheit garantiert, ist der *one time pad* eines von nur zwei in dieser Vorlesung behandelten Verfahren, die beweisbare Sicherheit bieten. (Das zweite ist die Quantenkryptographie, die allerdings tatsächlich einfach ein Protokoll ist, um über räumliche Distanzen hinweg Schlüssel für den *one time pad* zu vereinbaren.)

Der *one time pad* hat seinen Namen von der Art und Weise, wie er früher benutzt wurde: Gedruckt wurden zwei identische Exemplare eines Blocks, der auf jeder Seite eine zufällige Folge von Buchstaben enthält; jeder der beiden Kommunikationspartner bekommt ein Exemplar.

Zur Verschlüsselung einer Nachricht nimmt der Absender das oberste Blatt, verschlüsselt den ersten Buchstaben der Nachricht à la CAESAR mit dessen erstem Buchstaben, den zweiten mit dem zweiten *usw.* Wenn alle Buchstaben auf dem Blatt aufgebraucht sind oder die Nachricht vollständig verschlüsselt ist, wird das Blatt vernichtet und es geht weiter mit dem nächsten Blatt. Der Empfänger kann die Nachricht mit Hilfe seines identischen Blocks problemlos entschlüsseln.

Falls KCHQR OFVFN FVSLA XRQBV E die ersten Buchstaben auf der aktuellen Seite sind, wird also die Nachricht „Angriff im Morgengrauen“

verschlüsselt gemäß

```
  ANGRI FFIMM ORGEN GRAUE N
+ KCHQR OFVFN FVSLA XRQBV E
= LQOIA ULESA UNZQO EJRWA S
```

Was kann ein Gegner mit diesem Chiffretext anfangen?

Wenn er das Verfahren kennt, weiß er, daß für die Verschlüsselung dieser Nachricht aus 21 Buchstaben auch 21 Schlüsselbuchstaben verwendet wurde; dazu gibt es $26^{21} \approx 5{,}181318713 \cdot 10^{29}$ Möglichkeiten, also über Tausend mal so viele wie bei der allgemeinen monoalphabetischen Substitution. Wie wir dort gesehen haben, können wir allerdings auf solche Zahlen nichts geben. Die Sicherheit des *one time pad* beruht auf einer ganz anderen Überlegung:

Angenommen, ein Gegner kommt irgendwie auf den richtigen Schlüssel KCHQR OFVFN FVSLA XRQBV E und entschlüsselt die Nachricht damit:

```
  LQOIA ULESA UNZQO EJRWA S
- KCHQR OFVFN FVSLA XRQBV E
= ANGRI FFIMM ORGEN GRAUE N
```

Damit kennt er die Nachricht. Aber weiß er das?

Da die Schlüssel zufällig gewählt wurden, ist KCHQR OFVFN FVSLA XRQBV E genauso wahrscheinlich wie etwa JYFUT PJSXN PZTVJ MWWCG J, und damit würde via

```
  LQOIA ULESA UNZQO EJRWA S
- JYFUT PJSXN PZTVJ MWWCG J
= BRING EBLUM ENFUE RMUTT I
```

mit einem völlig anderen Ergebnis entschlüsselt. Offensichtlich gibt es für jede Buchstabenfolge der Länge 21 genau einen Schlüssel, der genau diese Folge als „Entschlüsselung“ liefert, und da alle Schlüssel dieselbe Wahrscheinlichkeit haben, gilt dasselbe auch für alle Nachrichten der Länge 21.

Natürlich sind aus Sicht des Gegners, der im allgemeinen durchaus Informationen über das Umfeld der Kommunikation hat (warum sonst sollte

er schließlich abhören?) nicht alle diese Nachrichten gleich wahrscheinlich, aber durch das Auffangen des Chiffretexts bekommt er keinerlei neue Informationen, die seine Einschätzung der relativen Wahrscheinlichkeit der verschiedenen Möglichkeiten verändern könnten. Dieses Phänomen bezeichnet man als absolute informationstheoretische Sicherheit.

Er lernt allerdings zwei Dinge: Erstens, daß der Absender eine Nachricht an den Empfänger schickte und zweitens, wie lange diese Nachricht war. Auch das läßt sich verhindern, indem der Absender regelmäßig zu festgesetzten Zeiten eine Nachricht an den Empfänger schickt; falls es nichts zu sagen gibt, schickt er einfach irgendeinen Text.

Der *one time pad* wurde wohl erstmalig vom amerikanischen General VERNAM im ersten Weltkrieg benutzt; man redet daher gelegentlich auch von der VERNAM-Chiffre. Im zweiten Weltkrieg kommunizierte London auf diese Weise mit der französischen *Résistance*, und später sicherten FIDEL CASTRO und CHE GUEVARA ihre Kommunikation auf diese Weise.

Als *high tech*-Variante davon entstand im Kalten Krieg das *Rote Telephon* zwischen dem Weißen Haus und dem Kreml: Damals hielten viele (wohl zu Recht) die Gefahr eines Atomkriegs aus Versehen für erheblich größer als die eines absichtlichen. Um ersteren etwas weniger wahrscheinlich zu machen, einigten sich die beiden Großmächte im Juni 1963 in Genf darauf, das sogenannte *Rote Telephon* einzurichten; es funktioniert seit dem 30. August 1963.

Natürlich handelt es sich dabei nicht wirklich um ein Telephon, denn zu keinem Zeitpunkt des kalten Krieges reichten die Sprachkenntnisse eines amerikanischen Präsidenten oder eines Generalsekretärs der KPdSU auch nur für ein direktes Gespräch über das Wetter. Tatsächlich geht es um eine Fernschreibverbindung mit je vier (bei Siemens Mannheim gebauten) Fernschreibern an beiden Enden: jeweils zwei mit lateinischem und zwei mit kyrillischem Alphabet. Bislang verbrachten sie ihre meiste Zeit damit, stündliche Testnachrichten zu drucken wie amerikanische Baseball-Ergebnisse oder TURGENJEWs *Aufzeichnungen einer Jägers*.

Aus Sicherheitsgründen wurden zwei Leitungen eingerichtet, eine entlang der Route Washington-London-Kopenhagen-Stockholm-Helsinki-

Moskau, die andere via Tanger. Natürlich war es unmöglich, diese Leitungen auf ihrer ganzen Länge zu überwachen, so daß niemand ausschließen konnte, daß irgendwo zwischen Moskau und Washington eine vertrauliche Kommunikation abgehört oder – mit potentiell sehr viel katastrophaleren Folgen – eine gefälschte Nachricht eingespielt wurde. Aus diesem Grund mußten alle Nachrichten verschlüsselt werden

Dazu diente folgende einfache Variation des *one time pads:* Von Zeit zu Zeit tauschten die beiden Seiten per Kurier Magnetbänder mit zufallserzeugten Bitfolgen aus. Jedesmal, wenn eine Nachricht übermittelt werden sollte, übersetzte der Fernschreiber diese in eine Bitfolge, d.h. in einen Vektor $\vec{v}$ aus einem Vektorraum $\mathbb{F}_2^N$. Aus den ersten N bislang noch nicht benutzten Bits auf dem Magnetband wurde dazu ein weiterer Vektor $\vec{w} \in \mathbb{F}_2^N$ gebildet, und tatsächlich übertragen wurde die Summe $\vec{s} = \vec{v} + \vec{w}$.

Am anderen Ende der Leitung, wo eine Kopie des Magnetbands vorlag, war $\vec{w}$ bekannt, so daß die Nachricht

$$\vec{v} = \vec{v} + \vec{0} = \vec{v} + (\vec{w} + \vec{w}) = (\vec{v} + \vec{w}) + \vec{w} = \vec{s} + \vec{w}$$

leicht rekonstruiert werden konnte.

Ein Lauscher ohne Magnetband konnte nur die Länge N der Nachricht ermitteln, was bei seitenlangen in Diplomatensprache formulierten Texten so gut wie keine konkrete Information liefert – ganz abgesehen davon, daß man nie ausschließen kann, daß es sich vielleicht einfach um eine zusätzliche Testnachricht zu Wartungszwecken handelt.

Wichtig ist auch, daß jemand, der einfach irgendeinen Vektor $\vec{s}$ in die Leitung einspielt, so gut wie keine Chance hat, daß nach Addition von $\vec{w}$ daraus verständlicher Text wird; die Manipulation wird also mit an Sicherheit grenzender Wahrscheinlichkeit entdeckt.

In alltäglicheren Anwendungen ist der mit dem *one time pad* verbundene Aufwand meist zu hoch; man muß notgedrungen mit Schlüsseln arbeiten, die deutlich kürzer sind als die (Summe der) Nachrichten, die damit verschlüsselt werden.

Informationstheoretische Berechnungen legen nahe, daß VIGENÈRE bereits unsicher wird, wenn die Nachricht nur um 30% länger ist als der

Schlüssel. der Klartext nur etwa 30% länger ist als der Schlüssel. Wenn man bedenkt, daß es Programme gibt, die lange Dokumente oder gar eine gesamte Festplatte verschlüsseln in Abhängigkeit von einem nur wenige Zeichen langen Schlüssel mittels eines byte- statt buchstabenbasierten VIGENÈRE-Systems, wundert es nicht, daß so viele Programme auf dem Markt sind, die „vergessene“ Passwörter rekonstruieren.

Wie der bislang größte (bekannt gewordene) Unfall der sowjetischen Kryptographie zeigt, kann selbst der *one time pad* bei falscher Anwendung unsicher werden: Aus irgendeinem Grund wurden Anfang 1942 eine Zeit lang von jedem Einmalblock nicht zwei, sondern vier Exemplare produziert. Die „überflüssigen“ Exemplare wurden nicht zerstört, sondern wanderten ins Vorratslager und wurden später, als ihre Herkunft längst vergessen war, benutzt. Dies gab den amerikanischen und britischen Geheimdiensten die Möglichkeit, einen Teil der geheimsten sowjetische Kommunikation zu entschlüsseln, obwohl zusätzlich zum *one time pad* vorher noch eine Verschlüsselung nach einem Codebuch eingesetzt wurde. Die rekonstruierten Dokumente kann inzwischen auch die interessierte Öffentlichkeit nachlesen unter `www.nsa.gov/venona`.

§ 4: Transpositionschiffren

Permutationen lassen sich nicht nur anwenden, um ein Alphabet zu permutieren; sie können auch verwendet werden, um die Reihenfolge der Buchstaben des Klartexts so zu verändern, daß dieser nicht mehr erkennbar ist.

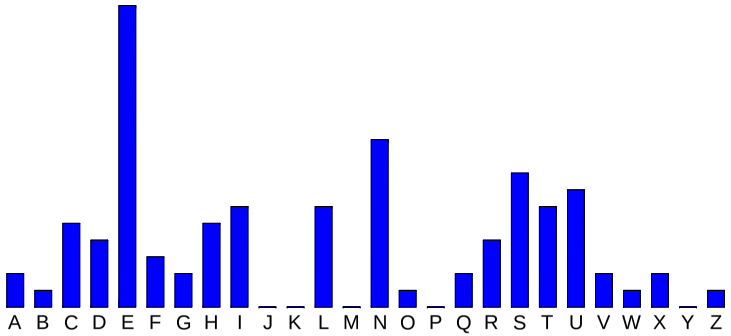
Konkret wählt man eine (geheime) Blocklänge n und eine Permutation $\pi \in \mathfrak{S}_n$. Der Klartext wird aufgeteilt in Blöcke der Länge n, wobei der letzte eventuell noch mit zufällig gewählten Buchstaben aufgefüllt werden muß; sodann wird der Block $a_1 a_2 \dots a_n$ ersetzt durch $a_{\pi(1)} a_{\pi(2)} \dots a_{\pi(n)}$.

Da nur die Reihenfolge der Buchstaben verändert wurde, ist die Verteilung der Buchstabenhäufigkeiten im Kryptogramm exakt dieselbe wie im Klartext; man erhält also ein Histogramm, daß genauso aussieht, wie man es von deutschem Klartext erwartet. Auf diese Weise erkennt ein Kryptanalytiker Transpositionschiffren.

Betrachten wir als Beispiel das Kryptogramm

```
CSHSI EUNUE EERRQ VNGDB EUINW HINES ZCUTE NELQL
UHNXI DTCRX EFTTF ESLSD TOALI TEESE DNLSU CEEHS
NRGLE VUEFE AINCN H
```

Wie das Diagramm zeigt, entsprechen die Buchstabenhäufigkeiten darin in der Tat der von deutschem Klartext:



Es hat eine Länge von 96 Zeichen; die Blocklänge n sollte also ein Teiler von 96 sein. (Ein geschickter Kryptograph würde freilich am Ende noch weitere Buchstaben anfügen, so daß das Kryptogramm keinen Hinweis auf n liefert.)

Versuchen wir etwa, das Kryptogramm zu entschlüsseln unter der Annahme, die Schlüssellänge sei gleich sechs. Dazu teilen wir es auf in Blöcke der Länge sechs:

1 CSHSIE	5 WHINES	9 RXEFTT	13 LSUCEE
2 UNUEEE	6 ZCUTEN	10 FESLSD	14 HSNRGL
3 RRQVNG	7 ELQLUH	11 TOALIT	15 EVUEFE
4 DBEUIN	8 NXIDTC	12 EESEDN	16 AINCNH

Da im ersten Block sowohl ein C als auch ein H vorkommen, im zweiter aber kein H steht, ist zu erwarten, daß C und H im Klartext ein CH sind; in Klartext sollte also die erste Spalte der obigen Blockdarstellung links vor der dritten stehen.

Im dritten Block steht an dritter Stelle ein Q, aber es gibt kein U. Das nächste U steht eine Zeile tiefer an vierter Stelle. Falls Q und U im Klartext ein QU bilden, muß also die dritte Spalte im Klartext die letzte sein und die vierte die erste.

Im sechsten Block haben wir ein C an zweiter Stelle, aber das nächste H steht eine Zeile tiefer an letzter Stelle; dies würde dafür sprechen, daß die zweite Spalte im Klartext an sechster Stelle steht und die sechste an erster.

Im siebten Block steht in der dritten Spalte ein Q und des einzige dazu passende U steht in Spalte fünf; denn im im achten Block gibt es keines. Demnach sollte die dritte Spalte im Klartext vor der fünften stehen.

In Block acht steht ein C, aber weder im achten noch im neunten Block gibt es ein H oder K.

In Spalte vier des dreizehnten Blocks steht ein C; dazu paßt eigentlich nur das H in der ersten Spalte des nächsten Blocks, d.h. die vierte Spalte sollte letzte sein und die erste auch im Klartext die erste.

Der letzte Block schließlich legt nahe, daß die vierte Spalte vor der sechsten stehen sollte.

Natürlich muß nicht jede dieser Folgerungen richtig sein; es kommt vor, daß nach einem C in deutschem Klartext weder ein H noch ein K steht, und es kommt auch vor, daß nach einem Q kein U steht. Hier haben aber so viele Widersprüche zwischen den getroffenen Folgerungen, daß so etwas der Regelfall sein müßte – dies spricht eher dagegen, daß die Verschlüsselung mit Blocklänge sechs erfolgte.

Orden wir den Chiffretext in Blöcke der Länge acht, bietet sich folgendes Bild:

1 CSHSIEUN	4 WHINESZC	7 RXEFTTFE	10 LSUCEEHS
2 UEEERRQV	5 UTENELQL	8 SLSDTOAL	11 NRGLEVUE
3 NGDBEUIN	6 UHNXIDTC	9 ITEESEDN	12 FEAINCNH

Hier legt Block eins nahe, daß die erste Spalte unmittelbar vor der dritten stehen sollte. Block zwei kann ein QU realisieren, wenn entweder Spalte sieben im Klartext vor der ersten steht, oder aber sie steht ganz hinten

und Spalte sechs ganz vorne. Block vier legt nahe, daß Spalte acht vor Spalte zwei stehen sollte, bei Block fünf haben wir dieselbe Situation wie bei Block zwei und bei Block sechs wie bei Block vier. Block zehn plaziert Spalte vier vor Spalte sieben und Block zwölf sechs vor acht.

Vermuten wir bei Block zwei die erste Möglichkeit, so stehen im Klartext einerseits die Spalten 4, 7, 1, 3 als Viererblock nebeneinander, andererseits Spalten 6, 8, 2 als Dreierblock. Entweder am Anfang oder am Ende oder dazwischen steht die fünfte Spalte.

Die möglichen Reihenfolgen sind also 47136825, 47135682, 54713682, 68247135, 68254713 und 56824713. Betrachtet man den Effekt dieser Permutationen auf die erste Zeile, ist klar, daß nur die erste Möglichkeit in Frage kommt; der Klartext ist also

SUCHEN SIE QUERVERBINDUNGEN ZWISCHEN
QUELLTEXT UND CHIFFRETEXT DAS SOLLTE
DIE ENTSCHLUESSELUNG VEREINFACHEN

Bei diesem Kryptogramm erforderte die Kryptanalyse wegen der relativ vielen Paare CH und QU wenig Arbeit; bei schwierigeren Texte hat er nicht unbedingt viele sichere Regeln, jedoch gibt es durchaus noch andere ziemlich sichere Kombinationen: So steht zum Beispiel, wie die Kontaktdiagramme zeigen, vor einem D sehr häufig ein N, und nach einem B oder G kommt zwar nicht immer, aber doch oft ein E, nach V steht meist O oder E. Bei kurzen Kryptogrammen mit wenigen typischen Buchstabenpaaren kann er für jedes Paar von Spalten die Wahrscheinlichkeiten dafür ausrechnen, daß die eine vor der anderen steht; auch das macht die Situation viel klarer. Oft springen dem Betrachter bei hinreichend langen Blöcken (die man aus Sicherheitsgründen braucht) auch Wörter ins Auge, die weiterhelfen können. Hier ist das Sprachgefühl des Kryptanalytikers oft wichtiger als jede mathematische Analyse.

Transpositionschiffren spielten bis zum ersten Weltkrieg eine große Rolle in der deutschen Militärkryptographie, allerdings wurden sie nie allein angewendet, sondern nur als Teil eines zweistufigen Verfahrens. Der Grund dürfte klar sein: Gegen einem Angriff mit bekanntem Klartext einer Länge, die über der Blocklänge liegt, bieten Transpositionschiffren keinerlei Schutz. Heute sind sie (zu Recht) weitgehend vergessen.

§5: Rotormaschinen

Alle bislang behandelten Kryptoverfahren sind hinreichend einfach, daß Nachrichten einfach mit Bleistift und Papier ver- und entschlüsselt werden können. Mit Ausnahme des logistisch sehr aufwendigen *one time pad* lassen sie sich allerdings auch alle relativ einfach knacken.

Viele klassische Kryptographen versuchten, die Sicherheit des *one time pad* mit der Einfachheit der VIGENÈRE-Chiffre zu kombinieren: Letztere könnte man auffassen als einen *one time pad*, bei dem die immer gleiche Buchstabenfolge endlos wiederholt wird, und das ist – wie wir gesehen haben – extrem unsicher.

Alternativ könnte man versuchen, aus einer relativ kurzen Schlüsselinformation eine komplizierte Buchstabenfolge zu erzeugen, die sich idealerweise verhält wie eine Zufallsfolge. Schließlich haben selbst viele Taschenrechner heute eine Taste, die Zufallszahlen erzeugt, und auch die meisten Programmiersprachen bieten „Zufallsgeneratoren“ an, manchmal auch unter dem korrekteren Namen „Pseudozufallsgenerator“.

Leider liest man immer wieder Untersuchungen, die vielen dieser Programme miserable Ergebnisse attestieren, aber es gibt durchaus Algorithmen, die Folgen liefern, die sich beispielsweise für die Zwecke einer Simulation nicht wesentlich von echt zufällig gewählten Zahlen unterscheiden – wie immer auch man letztere bekommen kann. (Wir werden uns am Ende der Vorlesung auch damit beschäftigen.)

Aber, und hier kommt das zweite *leider*, ein Algorithmus der gute Ergebnisse für Simulationszwecke liefert, liefert nicht notwendigerweise auch gute Ergebnisse für kryptographische Zwecke: Diese beiden Anwendungen sind praktisch disjunkt, denn die Algorithmen, die derzeit als kryptographisch sicher gelten, sind für Simulationszwecke viel zu aufwendig, und die bewährten Algorithmen, die für gute Simulationen eingesetzt werden, sind kryptographisch völlig unsicher.

Um 1920, als unabhängig voneinander in mehreren Länder erste Verschlüsselungsmaschinen auftauchten, wußte man von diesen Algorithmen des Computerzeitalters natürlich noch nichts; man versuchte ein-

fach, aus relativ kurzer und gut übermittelbarer Schlüsselinformation einen möglichst komplexen Schlüsselstrom zu erzeugen.

Wie üblich in der Kryptographie führten nahezu identische Ansätze zu dramatisch verschiedenen Ergebnissen:

Die Ansätze waren für Laien praktisch ununterscheidbar: Praktisch alle Maschinen bestanden aus sogenannten *Rotoren*. Dabei handelte es sich um Räder mit (im mitteleuropäischen Bereich) 26 äquidistanten elektrischen Kontakten auf der Oberfläche. Durch das Innere liefen Drahtverbindungen zwischen diesen Kontakten, die jeweils zwei miteinander verbanden; ein Strom der an einem dieser Kontakte angelegt wurde, verließ den Rotor also an einer festen anderen Stelle. Dadurch wurde eine Permutation aus $\mathfrak{S}_{26}$ realisiert, die sich als Produkt von 13 elementfremden Transpositionen schreiben ließ.

Die Tastatureingabe wurde über mehrere solcher Rotoren geleitet, so daß effektiv ein Produkt von drei bis fünf solcher Permutationen realisiert wurde. Für sich allein bietet das natürlich keinerlei Sicherheit, denn das Produkt von egal wievielen Permutationen ist schließlich wieder nur eine einfache Permutation. Deshalb drehte sich einer der Rotoren nach jedem Buchstaben um eins weiter um so eine andere Permutation zu realisieren.

Aus Sicht eines Kryptanalytikers ist auch eine solche reguläre Bewegung kein großes Hindernis; als zusätzliche Sicherheit gehört daher zu einer Rotormaschine, daß sich auch weiteren Rotoren gelegentlich aber nicht immer weiterdrehen. Bei den ersten Rotormaschinen wie etwa der von HEBERN geschah dies in einer völlig regelmäßigen Weise: Der fünfte Rotor bewegte sich nach jedem Buchstaben um eins weiter, der erste nach je 26 und der dritte nach je 676 Buchstaben. Der zweite und der vierte Rotor änderten ihre Position nicht. Bei anderen Maschinen wie der Enigma wurde die Bewegung der Rotoren durch unregelmäßig auf den einzelnen Rotoren angeordneten Haken realisiert und war damit auch von der Rotorkonfiguration abhängig.

Zur Entschlüsselung wird der Chiffretext einfach in eine identisch aufgebaute Maschine mit gleicher Anfangsstellung aber umgekehrter Ro-



torreihenfolge gegeben. (Das Bild hier zeigt eine Enigma-Maschine mit drei Rotoren.)

Bei der Enigma, im Gegensatz zu anderen Rotormaschinen, war der

letzte Rotor fest und leitet den Strom, nach einer weiteren Permutation, zurück durch die beweglichen Rotoren. Dadurch kann die Entschlüsselung mit derselben Rotorenkonfiguration erfolgen.

Im WWW sind verschiedene Simulatoren sowohl der vielen deutschen Enigma-Modelle als auch anderer Rotor-Maschinen zu finden; eine Enigma der deutschen Abwehr mit vier Rotoren findet man beispielsweise unter `http://home.caiway.nl/~antonh/enigma_ga.html` . Die mit solchen Maschinen erzielbaren Perioden liegen im Bereich über 100 000; wie jeder Leser inzwischen hoffentlich verstanden hat, ist das aber freilich keine Garantie für Sicherheit.

In der Tat wurde eine der ersten Rotormaschinen, der Kryptograph des deutschen Ingenieurs ALEXANDER VON KRYHA, die in Europa wie auch Amerika recht erfolgreich an Geschäftsleute verkauft wurde, im Januar 1933 auch der amerikanischen Armee zum Kauf angeboten. Diese ließ sich zu Testzwecken ein damit verschlüsseltes Kryptogramm aus 1135 Buchstaben (200 Wörtern) geben, und leitete dieses an ihren Chefkryptanalytiker ROBERT FRIEDMAN weiter. Diesem gelang es, zusammen mit drei Mitarbeitern, das Kryptogramm (ohne Maschinen) in zwei Stunden und 41 Minuten zu entschlüsseln. Die Öffentlichkeit erfuhr davon natürlich nichts; auch um 1950 verschlüsselte das Auswärtige Amt in Bonn noch seine diplomatische Korrespondenz mit solchen Maschinen.

Die Enigma-Maschine war deutlich sicherer, wurde aber trotzdem bereits während des zweiten Weltkriegs fast routinemäßig geknackt. Ein Grund dafür war, daß während der Gültigkeitsperiode eines Schlüssels (d.h. während eines Tages) alle Nachrichten mit derselben Anfangsstellung verschlüsselt werden müssen oder aber die Anfangsstellung auf andere Weise übermittelt werden muß.

Die deutsche Wehrmacht umging dieses Problem in einigen Netzen dadurch, daß mit dem jeweiligen Tagesschlüssel nur ein speziell für die folgende Nachricht gültiger zufällig gewählter Schlüssel übermittelt wurde; danach wurde die Maschine gemäß diesem Schlüssel in eine neue Anfangsstellung gebracht und die Nachricht verschlüsselt.

Grundsätzlich ist das eine recht sichere Option; falls allerdings durch einen Übermittlungsfehler bei den ersten drei Buchstaben diese nicht

korrekt beim Empfänger ankommen, kann dieser die Nachricht nicht entschlüsseln und muß um Wiederholung bitten.

Dies war den Militärs zu umständlich; deshalb wurden die drei Buchstaben der Anfangsstellung nicht einmal, sondern zweimal übertragen. Sobald die Kryptanalytiker in der britischen Dechiffrierzentrale in *Bletchley Park* dies merkten, hatten sie eine entscheidende Zusatzinformation: Die ersten sechs Buchstaben einer jeden Übertragung gehören zu einem Klartext, der aus zweimal derselben Dreiergruppe von Buchstaben besteht. Der Tagesschlüssel muß daher die Eigenschaft haben, daß alle diese aufgefangenen Sechsergruppen bei Entschlüsselung damit zu zwei aufeinanderfolgenden identischen Dreiergruppe werden, was bei zufällig gewählten Schlüsseln natürlich fast nie der Fall ist.

Um zu sehen, für welche(n) Schlüssel das funktioniert, muß man im Prinzip alle ausprobieren, was bei $26^3 = 17\,576$ möglichen Anfangsstellungen und anfänglich $3! = 6$, später $5 \cdot 4 \cdot 3 = 60$ möglichen Rotorkombinationen für die damalige Zeit ein beträchtlicher Aufwand war. Hierzu wurden in *Bletchley Park* spezielle Geräte konstruiert, die sogenannten *Bomben*, die mit hoher Geschwindigkeit viele Rotoren parallel arbeiten ließen.

Viele Angriffe waren auch Angriffe mit bekanntem Klartext, sogenannten *cribs*, die ebenfalls mit Hilfe der Bomben auf mögliche Schlüssel untersucht werden konnten. Meist gelang es, bis etwa elf Uhr morgens den Tagesschlüssel zu ermitteln und ab dann alle Kommunikation zu entschlüsseln.

Die Verdrahtung der einzelnen Rotoren war schon vor dem Krieg von polnischen Kryptanalytikern geknackt worden, nachdem der Deutsche HANS-THILO SCHMIDT (1888-1943) die Gebrauchsanweisung für die Enigma-Maschine und die Liste der Schlüssel für September und Oktober 1932 an einen französischen Kryptographen namens BERTRAND verkauft hatte. Dieser konnte die Maschine zwar nicht knacken, gab die Informationen aber nach England und nach Polen weiter; einer Gruppe polnischer Kryptographen unter Leitung von MARIAN ADAM REJEWSKI (1905–1980) gelang dann die Rekonstruktion der drei damals gebräuchlichen Rotoren und somit der Maschine.

Diese Rotoren blieben während des gesamten zweiten Weltkriegs im Einsatz; kein Rotor wurde je aus dem Verkehr gezogen. Zwar wurden in manchen Netzwerken wie etwa bei den U-Booten ein oder zwei neue Rotoren eingeführt, aber da diese zusammen mit den alten benutzt wurden, hatten die Kryptanalytiker dann nur noch das Problem, die Verdrahtung *eines* unbekannten Rotors zu ermitteln, was angesichts des großen Nachrichtenvolumens stets gelang.

Nach dem Krieg verkaufte die britische Regierung übrigens die erbeuteten Enigma-Maschinen an Regierungen ihrer Ex-Kolonien ohne denen über die erfolgreiche Entschlüsselung zu berichten; damit war sie erstens immer gut über die Vorkommnisse in diesen Ländern informiert und verdiente zweitens noch daran.

Seit etwa 1970 werden Rotormaschinen nicht mehr eingesetzt – außer in UNIX. Das dortige `crypt(1)`-Kommando simuliert eine Rotormaschine mit *einem* Rotor, der 256 Ein- und Ausgänge hat. Sehr sicher ist das nichts; selbst in der `man`-Seite dazu heißt es *Methods of attack on such machines are widely known; thus crypt provides minimal security.* Für Paßwörter verwendet UNIX daher auch nicht dieses Kommando, sondern das auf einem „gesalzenen“ DES beruhende deutlich bessere `crypt(3)`-Unterprogramm. Es gibt inzwischen ein universelleres Kommando `mcrypt`, das sich zwar mit Option `--enigma` genauso verhält wie `crypt(1)`, das aber auch alternative Algorithmen anbietet, die nach heutigen Standards sicher sind, z.B. AES und triple-DES. Es ist unter der GNU public licence erhältlich unter `mcrypt.sourceforge.net`.

§ 6: Literaturhinweise

Das (dickleibige) Standardwerk zur alten Kryptographie mit Schwerpunkt auf der geschichtlichen Darstellung ist das bereits im letzten Kapitel erwähnte Buch

DAVID KAHN: *The Codebreakers – the comprehensive history of secret communication from ancient time to the internet,* Scribner, New York, ^{2}1996

Für zahlreiche ältere Kryptologen war die erste Auflage von 1967 der Einstieg in ihr Arbeitsgebiet.

Deutlich kürzer, billiger und verfahrensorientierter ist

HELEN FOUCHÉ GAINES: *Cryptanalysis – a study of ciphers and their solution,* Dover, New York, 1956 *(Originalausgabe 1939)*

L. SACCO: *Manuel de cryptographie,* Payot, Paris, 1951

beschreibt die klassischen Verfahren und ihre Kryptanalyse aus seiner Sicht als Chef des Chiffrierdienstes der italienischen Armee. Eine Reihe entsprechender Bücher gibt es auch von WILLIAM FRIEDMAN, jedoch sind diese in Deutschland wenn überhaupt nur schwer zu finden.

Rotormaschinen und ihrer Kryptanalyse ist das Buch

CIPHER A. DEAVOURS, LOUIS KRUH: *Machine cryptography and modern cryptanalysis,* Artech House, Dedham MA, 1985

gewidmet. „Modern“ bezeichnet hier den Zeitraum von etwa 1920–1970. Weitere Informationen zur Kryptanalyse von Rotormaschinen im zweiten Weltkrieg findet man im Buch von KAHN sowie bei

BENGT BECKMAN: *Codebreakers – Arne Beurling and the Swedish crypto program during Word War II,* American Mathematical Society, Providence, R.I., 2002

Moderne Lehrbücher mit Kapiteln über klassische Kryptographie sowie über statistische und informationstheoretische Ansätze zur Kryptanalyse sind

JAN C.A. VAN DER LUBBE: *Basic Methods of cryptography,* Cambridge University Press, 1998

und

ALAN G. KONHEIM: *Computer Security and Cryptography,* Wiley, 2007

sowie dessen Vorgänger

ALAN G. KONHEIM: *Cryptography – A Primer,* Wiley, 1981

Kapitel 3
Klassische Blockchiffren

Im vorigen Kapitel haben wir gesehen, daß alle dort behandelten Kryptoverfahren mit Ausnahme des *one time pad* nicht einmal einfachsten Sicherheitsanforderungen genügen; der *one time pad* wiederum erfordert einen derart großen Aufwand beim Schlüsselaustausch, daß er für die meisten aktuellen Anwendungen wie etwa dem Handel über das Internet nicht in Frage kommt.

Nun hat sich aber bereits KERCKHOFFS mit *praktischer* Sicherheit begnügt, und in den meisten Fällen werden auch wir uns damit begnügen müssen. Zwei Ansätze bieten sich an:

Als erstes können wir den *one time pad* imitieren, indem wir zwar keinen wirklich zufälligen Einmalschlüssel von der Länge der Nachricht verwenden, dafür aber einen solchen Schlüssel erzeugen aus einem Anfangsschlüssel handhabbarer Länge und einem Algorithmus, der daraus eine zufällig aussehende Folge produziert. Solche Algorithmen bezeichnet man als *Pseudozufallsgeneratoren,* denn tatsächlich ist die erzeugte Folge ja nicht zufällig, sondern wird streng deterministisch aus dem Anfangsschlüssel berechnet.

Solche Folgen von Pseudozufallszahlen spielen nicht nur in der Kryptographie eine wichtige Rolle, sondern vor allem auch bei Simulationen aller Art; die meisten Pseudozufallsgeneratoren wurden für solche Anwendungen entwickelt. Einer der bekanntesten verwendet lineare Kongruenzen, indem er ausgehend von einem Anfangswerts x_0 die Folgewerte gemäß $x_n = ax_{n-1} \bmod p$ berechnet, wobei p eine hinreichend große Primzahl ist. Bei geeigneter Wahl von a und p sind die so erzeugten Zahlen für Simulationen gut geeignet; für kryptographische

Anwendungen jedoch sind sie wertlos: Wer im Rahmen einer Attacke mit bekanntem Klartext auch nur wenige Klartext/Chiffretext-Paare bestimmen kann, kennt eine Reihe von Folgengliedern x_n und kann daraus im allgemeinen ohne große Schwierigkeiten auf a und p schließen. Damit kann er den gesamten Schlüsselstrom berechnen. Ein kryptographisch sicherer Pseudozufallsgenerator muß natürlich so beschaffen sein, daß auch eine lange Folge der erzeugten Zahlen nicht ausreicht, um die Parameter des Algorithmus praktisch zu bestimmen. Wir werden solche Generatoren im Zusammenhang mit kryptographisch sicheren Hashfunktionen kennen lernen.

Chiffren, die auf diese Weise eine abgespeckte Version des *one time pad* realisieren, bezeichnet man als *Stromchiffren;* ihr Hauptanwendungsgebiet sind lange Datenströme, wie sie etwa bei der Übertragung zwischen einem Mobiltelephon und dem Sendemasten oder zwischen einem Beobachtungssatelliten und seiner Bodenstation anfallen; auch Pay-TV wird so verschlüsselt.

Bei der Kommunikation zwischen Computern oder zwischen Computern und ihrer Peripherie geht man meist einen anderen Weg: Die Angriffe, die wir im vorigen Kapitel betrachtet haben, beruhten auf der unterschiedlichen Häufigkeit der verwendeten Buchstaben, Buchstabenpaare und so weiter. Bei einem Alphabet aus nur 26 Buchstaben bereitet es auch bei kurzen Kryptogrammen keine Schwierigkeiten, sich eine entsprechende Statistik zu verschaffen; nimmt man aber ein Alphabet, dessen Buchstabenzahl die Anzahl der Buchstaben in einer typischen Nachricht deutlich übersteigt, wird es eher unwahrscheinlich, daß im Chiffretext überhaupt irgendwelche Doubletten zu finden sind, auf jeden Fall aber nicht genug, um eine sinnvolle Statistik aufzustellen. Bei diesem Ansatz spricht man von *Blockchiffren,* da vor der Verschlüsselung jeweils mehrere Buchstaben zu einem Block zusammengefaßt und dieser dann als ganzes verschlüsselt wird.

Da Blockchiffren hauptsächlich in Rechnernetzen eingesetzt werden, bestehen die Blöcke bei den heute üblichen Verfahren nicht aus Buchstaben, sondern aus Bits oder Bytes; bei den ersten kommerziell eingesetzten Blockchiffren arbeitete man mit Blöcken von 64 Bit; heute üblich sind mindestens 128 Bit.

§ 1: Anforderungen an eine Blockchiffre

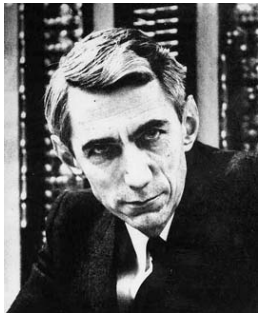
Idealerweise möchten wir erreichen, daß ein Gegner durch ein aufgefangenes Kryptogramm keinerlei Informationen über den Klartext erhält – außer natürlich der offensichtlichen, daß eine Nachricht gesendet wurde, und gewissen Hinweisen auf deren Länge. In der Kryptologie spricht man von *perfekter Sicherheit,* wenn die gegnerische Information über den Klartext ohne Kenntnis des Kryptogramms genauso groß ist wie mit.

Mathematisch läßt sich dieser Begriff wie folgt formalisieren: Der Gegner erwartet, daß irgendeine von r möglichen Nachrichten $m_1, \ldots, m_r$ übertragen wird, und auf Grund seiner Einschätzung der Situation ordnet er diesen Nachrichten Wahrscheinlichkeiten $p(m_i)$ zu: Ein Militärkryptanalytiker beispielsweise wird die Nachricht Angriff im Morgengrauen für wahrscheinlicher halten als Bringe Blumen fuer Mutti, und wer im Internet Kontodaten abgreifen will, wird Banken mit Sitz in der Nähe seines Opfers für wahrscheinlicher halten als solche aus anderen Erdteilen.

Sobald er einen Chiffretext c aufgefangen hat, kann er diesen untersuchen und dann die *bedingten* Wahrscheinlichkeiten $p(m_i|c)$ dafür berechnen, daß sich hinter dem Chiffretext c der Klartext m_i verbirgt. Wir reden von perfekter Sicherheit, wenn $p(m_i) = p(m_i|c)$ ist für alle i, wenn ihm also der aufgefangene Chiffretext nicht erlaubt, seine Anfangseinschätzung zu modifizieren. (Da man aus der Länge eines Kryptogramms fast immer Rückschlüsse auf die Länge der Nachricht ziehen kann, beschränkt man sich bei dieser Forderung meist auf Nachrichten einer festen Länge – sonst würde auch der *one time pad* keine perfekte Sicherheit bieten.)

Leider konnte SHANNON beweisen, daß eine *notwendige* Voraussetzung für perfekte Sicherheit darin besteht, daß die Schlüssellänge mindestens gleich der Summe der Längen aller je mit diesem Schlüssel übermittelten Nachrichten ist. Einen Beweis dieses Satzes in Lehrbuchdarstellung findet man zum Beispiel bei

DOMINIC WELSH: *Codes und Kryptographie,* Wiley-VCH, 1991, Theorem 7.3, oder in einigen der älteren Versionen dieses Skriptums.



CLAUDE ELWOOD SHANNON (1916–2001) wurde in Petoskey im US-Bundesstaat Michigan geboren; 1936 verließ er die University of Michigan mit sowohl einem Bachelor der Mathematik als auch einem Bachelor der Elektrotechnik, um am M.I.T. weiterzustudieren. In seiner 1938 geschriebenen Masterarbeit *A symbolic analysis of relay and switching circuits* entwickelte er die Schaltlogik, die seitdem eine wichtige Grundlage der digitalen Informationsverarbeitung bildet. Seine 1940 fertiggestellte Dissertation befaßte sich mit Anwendungen der Algebra auf die MENDELschen Gesetze. Nach seinem Studium arbeitete er bis 1956 bei den Bell Labs, wo er während des zweiten Weltkriegs insbesondere über die Sicherheit kryptographischer Systeme forschte. Seine *Mathematical theory of cryptography* wurde aus Geheimhaltungsgründen erst 1949 zur Veröffentlichung freigegeben. Seine wohl bekannteste Arbeit ist die 1948 erschienene *Mathematical theory of communication,* in der er die fehlerfreie Übertragung von Nachrichten über einen gestörten Kanal untersuchte. Von 1956 bis zu seiner Emeritierung 1978 lehrte er am M.I.T., das er dadurch zur führenden Universität auf dem Gebiet der Informationstheorie und Kommunikationstechnik machte. Zu seinen zahlreichen Arbeiten zählt auch eine über die mathematische Theorie der Jongliermuster, anhand derer Jongleure eine Reihe neuer Muster gefunden haben; außerdem konstruierte er mehrere Jonglierroboter.

Der zitierte Satz von SHANNON zeigt noch einmal von der theoretischen Seite, warum uns schon KERCKHOFFS riet, uns mit *praktischer* Sicherheit zu begnügen. Dafür müssen wir nicht unbedingt wissen, daß kein Gegner irgendwelche Informationen aus dem Kryptogramm gewinnen kann; wir können an mehreren Stellen abschwächen:

- Wir müssen uns nicht gegen *jeden* Gegner schützen, sondern nur gegen zu erwartende Gegner. Private Aufzeichnungen, die nicht in die Hände neugieriger jüngerer Geschwister fallen sollen, müssen nicht unbedingt auch gegenüber einer Attacke von Regierungsorganisationen sicher sein.
- Viele Geheimnisse haben ein Verfallsdatum, nach dem sie nicht mehr geschützt werden müssen. Entwürfe für eine öffentliche Ankündigung eines Unternehmens etwa müssen nur bis zu dieser Ankündigung geheim gehalten werden und müssen nicht wie etwa eine Kundenkartei gegen eine Attacke geschützt werden, die eine mehrmonatige Kryptanalyse erfordert.
- Wir müssen nicht unbedingt fordern, daß der Gegner *keinerlei* Infor-

mation aus dem Kryptogramm extrahieren kann; falls er nicht mehr als einige kleinere statistische Anomalien des Klartexts findet, kann man in vielen Fällen damit leben.

Diese Abschwächungen sind allesamt wenig konkret, denn natürlich wissen wir nicht, was welcher Gegner in welcher Zeit leisten kann. Um auf der sicheren Seite zu sein, müssen wir seine Fähigkeiten nach oben abschätzen, indem wir im Zweifelsfall mit einem sehr viel gefährlicheren Gegner rechnen, als dem tatsächlich erwarteten.

Der gefährlichste Gegner überhaupt ist der sogenannte BAYESsche Gegner, der über unbegrenzte Ressourcen für Rechnungen und Probeentschlüsselungen verfügt. Er ist benannt nach dem englischen Theologen und Mathematiker THOMAS BAYES, dessen Formeln wohl den meisten Lesern aus der Wahrscheinlichkeitstheorie bekannt sind. Mit diesen Formeln arbeitet auch der BAYESsche Gegner, indem er für jeden möglichen Klartext m und/oder Schlüssel s die bedingte Wahrscheinlichkeit dafür berechnet, daß der abgefangene Chiffretext eine Verschlüsselung von m *bzw.* mit Schlüssel s ist. Er entscheidet sich dann für den Klartext m und/oder den Schlüssel s mit der größten bedingten Wahrscheinlichkeit.



THOMAS BAYES (1702–1761) wurde in London geboren als ältestes von sieben Kindern eines der ersten nonkonformistischen Pastoren Englands. Da die englischen Universitäten Oxford und Cambridge keine Nonkonformisten akzeptieren, mußte er zum Studium 1719 nach Schottland an die Universität Edinburgh, wo er sich für Logik und Theologie immatrikulierte. Nach seinen späteren Äußerungen muß er sich auch bereits damals oder kurz danach mit Mathematik beschäftigt haben. Wie sein Vater wurde er Geistlicher; seine mathematischen Arbeiten, z.B. über die Grundlagen der Analysis, erschienen zu seinen Lebzeiten nur anonym. Trotzdem wurde er 1742 fellow der Royal Society, die 1764 auch posthum seinen *Essay towards solving a problem in the doctrine of chances* veröffentlichte.

Sicherheit gegenüber dem BAYESschen Gegner ist nur in einem Punkt schwächer als perfekte Sicherheit: Wir können zulassen, daß er etwas Information über den Klartext (oder den Schlüssel) erhält, allerdings so wenig, daß es ihm nichts nützt.

Mit SHANNONs Ansatz läßt sich genau bestimmen, wieviel Information der BAYESsche Gegner aus einem Chiffretext extrahieren kann; sobald die Schlüssellänge wesentlich kleiner wird als der Klartext, ist das leider deutlich mehr als alles, was wir tolerieren können.

Tatsächlich ist *keine* Blockchiffre sicher gegenüber einem BAYESschen Gegner, denn selbst wenn nur wenige Blöcke Text im ASCII Code verschlüsselt werden, gibt es mit an Sicherheit grenzender Wahrscheinlichkeit nur einen Schlüssel, mit dem die Entschlüsselung zu lesbarem Klartext führt. Der BAYESsche Gegner mit seinen unbegrenzten Ressourcen kann alle Schlüssel durchprobieren und so den einen richtigen identifizieren.

Man könnte ihm das Leben schwerer machen, indem man den Text vor der Verschlüsselung komprimiert; da kein heutiger Komprimierungsalgorithmus so gut ist, daß *jede* mögliche Bitfolge zu lesbarem Klartext dekomprimiert werden kann, heißt das aber nur, daß er eventuell einige Blöcke mehr braucht, um den korrekten Schlüssel zu finden.

Zum Glück sind die Gegner, vor denen wir uns schützen müssen, nur selten BAYESsche Gegner, und SHANNON machte sich auch Gedanken über die Konstruktion praktikabler Chiffren, die zumindest gegenüber den zu erwartenden Gegnern praktisch sicher sind. Nach seinen Vorstellungen beruht die Sicherheit eines Kryptosystems auf zwei Prinzipien, der *Konfusion* und der *Diffusion.*

Die *Konfusion* soll dafür sorgen, daß der Zusammenhang zwischen Schlüssel und Chiffretext möglichst undurchsichtig ist; mit seinen verfügbaren Mitteln soll der Kryptanalytiker aus dem Chiffretext nur wenig Information über den Klartext und den Schlüssel gewinnen können. Der *one time pad* ist ein Beispiel dafür, wie Konfusion zu perfekter Sicherheit führen kann; die anderen Beispiele aus dem ersten Kapitel zeigen, daß reine Substitution bei kürzeren Schlüssellängen nicht sonderlich hilfreich ist. Wie das Beispiel der HILL-Chiffre vom zweiten Übungsblatt zeigt, muß Konfusion insbesondere auch für Nichtlinearität sorgen; andernfalls reicht möglicherweise bereits die Lineare Algebra zur Kryptanalyse.

Die *Diffusion* soll die statistischen Besonderheiten des Klartexts möglichst weiträumig über den Chiffretext verteilen und somit ausmitteln. Jedes Zeichen im Klartext sollte möglichst viele Zeichen im Chiffretext beeinflussen, so daß insbesondere kein signifikanter statistischer Zusammenhang zwischen dem Zeichen an einer speziellen Position des Klartexts mit dem Zeichen an einer (eventuell anderen) speziellen Position des Chiffretexts bestehen sollte.

§ 2: Der Aufbau einer Blockchiffre

Moderne Blockchiffren arbeiten natürlich nicht über einem Alphabet aus 26 Buchstaben; heute brauchen wir Kryptosysteme, die nicht nur Texte, sondern Dateien aller Art sicher verschlüsseln können, und bei vielen Anwendungen wollen wir, daß die gesamte Kryptographie ohne Zutun des Anwenders im Hintergrund abläuft. Insbesondere muß die entschlüsselte Datei genau so aussehen, wie vor der Verschlüsselung. Daher müssen wir davon ausgehen, daß unsere Blöcke aus beliebigen Bitfolgen (einer festen Länge) bestehen, und daß sie auch als solche wieder entschlüsselt werden müssen.

Wir betrachten die Chiffrierung daher als eine Abbildung von $\mathbb{F}_2^n$ nach $\mathbb{F}_2^n$, wobei $\mathbb{F}_2^n = \{0,1\}$ den Körper mit den beiden Elementen 0 und 1 bezeichnet und n die Blocklänge.

Die Rechenoperationen in $\mathbb{F}_2$ sind die Addition und die Multiplikation modulo zwei:

$\oplus$	0	1
0	0	1
1	1	0

$\odot$	0	1
0	0	0
1	0	1

$\mathbb{F}_2^n$ mit den üblichen Operationen ist ein Vektorraum über $\mathbb{F}_2$; die Vektoraddition bezeichnen wir mit $\oplus$. Sie läßt sich auf Computern problemlos realisieren durch die logische Antivalenz, das exklusive Oder XOR.

Da gängigee Computer byteorientiert arbeiten, ist die Blocklänge n praktisch immer ein Vielfaches von acht; bei einigen alten Blockchiffren war $n = 64$, heute sollte $n \geq 128$ gewählt werden.

Getreu dem Prinzip der Konfusion darf die Verschlüsselung keine lineare Abbildung sein; eine solche ließe sich schließlich mit nur wenigen Paaren aus bekanntem Klartext und seiner Verschlüsselung rekonstruieren. Idealerweise sollte die Verschlüsselung irgendeine beliebige Permutation der *Menge* $\mathbb{F}_2^n$ sein.

Diese Menge hat 2^n Elemente; somit gibt es $2^n!$ verschiedene Permutationen. Schon für $n = 64$ ist das eine Zahl, die kein Taschenrechner mehr ausrechnen kann und vor der auch die meisten Computeralgebrasysteme nicht einmal eine Gleitkomma-Näherung berechnen können: Nach der STIRLINGschen Formel ist in erster Näherung $\log n! \approx n \log n$, wobei der Logarithmus links und rechts zur gleichen, aber beliebigen Basis genommen werden kann. Somit ist

$$\log_2(2^{64}!) \approx 2^{64} \log_2(2^{64}) = 2^{64} \cdot 64 = 2^{70} \quad \text{und}$$
$$\log_2(2^{128}!) \approx 2^{128} \log_2(2^{128}) = 2^{128} \cdot 128 = 2^{137};$$

die Anzahl aller möglicher Permutationen hat also etwa $2^{70} \approx 1{,}2 \cdot 10^{21}$ *bzw.* $2^{135} \approx 4{,}3 \cdot 10^{40}$ Binärziffern; die Anzahl der Dezimalziffern ist ungefähr 0,3 mal so viel.

Um nur *eine* Permutation $\mathbb{F}_2^{64} \to \mathbb{F}_2^{64}$ zu spezifizieren, brauchen wir demnach ungefähr 2^{70} Bit oder 2^{67} Byte oder 2^{57} kB oder 2^{47} MB oder 2^{37} GB oder 2^{27} TB oder ..., jedenfalls viel zu viel.

Damit ist klar, daß wir bei einer modernen Blockchiffre nicht mehr wie bei den monoalphabetischen Substitutionen einfach eine beliebige Permutation als Schlüssel zulassen können; die Übermittlung eines solchen Schlüssels wäre völlig unpraktikabel. Wir müssen uns daher beschränken auf eine deutlich kleinere Menge von Permutation, deren Elemente sich durch Schlüssel handhabbarer Länge beschreiben lassen.

Von einer guten Blockchiffre erwarten wir aber, daß die von ihr realisierten Permutationen wie zufallsverteilt in der Gruppe aller Permutationen liegen. Sie sollen beispielsweise weder eine Untergruppe bilden noch eine verhältnismäßig kleine Untergruppe erzeugen, denn die Kenntnis von deren Erzeugenden und Relationen könnte einem Gegner Ansatzpunkte zu einer Entschlüsselung geben, die deutlich schneller geht als das Durchprobieren aller Schlüssel.

Eine Blockchiffre gilt nach heutigen Standards als gut, wenn auch nach längerer Untersuchung durch Fachleute keine Attacken auftauchen, die schneller sind als das Durchprobieren aller Schlüssel; als praktisch sicher gilt sie derzeit, wenn die Anzahl möglicher Schlüssel mindestens gleich 2^{100} oder besser 2^{128} ist. Bis zum Ende dieses Kapitels, spätestens aber bis Mitte des Semesters, sollte jedem Hörer klar sein, daß letzterer Wert in 25 Jahren mit ziemlicher Sicherheit nicht mehr als ausreichend betrachtet werden kann.

Fast alle heutigen Blockchiffren arbeiten in mehreren Runden, wobei in jeder Runde eine relativ einfache Permutation in Abhängigkeit von einem sogenannten Rundenschlüssel realisiert wird; erst die Hintereinanderausführung hinreichend vieler Runden führt zu ausreichender Konfusion und Diffusion. Speziell für die Diffusion setzt man hier auf den sogenannten *Lawineneffekt:* Während die Transformationen in jeder einzelnen Runde recht einfach sind und ein verändertes Eingabebit nur wenige Ausgangsbits beeinflußt, sorgt die Hintereinanderausführung vieler Runden dafür, daß sich dieser Einfluß immer mehr ausweitet und schließlich das gesamte Endergebnis beeinflußt – zumindest bei einer gut aufgebauten Blockchiffre.

Bis gegen Ende des vorigen Jahrhunderts dominierten bei der Architektur der Blockchiffren die sogenannten FEISTEL-Netzwerke, bei denen in jeder Runde nur ein halber Block modifiziert wurde.

HORST FEISTEL (1915–1990) wurde in Berlin geboren. Er emigrierte 1934 in die USA, wo er am MIT (BSc) und in Harvard (MSc) Physik studierte. Als Deutscher stand er während des zweiten Weltkriegs zunächst unter Hausarrest, wurde aber Anfang 1944 eingebürgert und arbeitete dann gleich in einem Forschungszentrum der Air Force. Nach dem Krieg arbeitete er kurze Zeit am MIT und bei MITRE, dann wechselte er zu IBM, wo er ab etwa 1960 die ersten Blockchiffren entwickelte, insbesondere auch das Lucifer-System, auf dessen Grundlage der im nächsten Paragraphen vorgestellte DES entwickelt wurde.

Da FEISTEL-Netzwerke jeweils mit halben Blöcken arbeiten, muß die Blocklänge gerade sein; wir bezeichnen sie als $2N$. Kern des Verschlüsselungsalgorithmus ist eine Funktion

$$f\colon \mathbb{F}_2^k \times \mathbb{F}_2^N \to \mathbb{F}_2^N\,,$$

die sogenannte FEISTEL-Funktion, die aus k Schlüsselbits und N Nachrichtenbits wieder N Bits produziert. Sie wird folgendermaßen angewandt: Vor Beginn der Verschlüsselung wird die Nachricht aufgeteilt in ein Paar (L_0, R_0) aus der linken Hälfte L_0 und der rechten Hälfte R_0; in den verschiedenen Runden wird dieses Paar transformiert zu neuen Paaren (L_i, R_i), deren letztes das Ergebnis der Verschlüsselung ist. Konkret wird in der i-ten Runde das Paar (L_i, R_i) zunächst ersetzt durch

$$\big(f(s_i, R_{i-1}) \oplus L_{i-1},\ R_{i-1}\big)\,,$$

wobei s_i den Schlüssel der i-ten Runde bezeichnet; danach werden (außer in der letzten Runde) die beiden Hälften miteinander vertauscht. Die i-te Runde realisiert also die Substitution

$$L_i = R_{i-1} \quad \text{und} \quad R_i = f(s_i, R_{i-1}) \oplus L_{i-1}\,.$$

Die FEISTEL-Funktion ist somit die einzige Quelle von *Konfusion;* die Auswahl der Schlüsselbits für die jeweilige Runde sowie auch die Vertauschung von linker und rechter Seite in jeder Runde dienen (neben der FEISTEL-Funktion selbst) der Diffusion.

Da $\mathbb{F}_2$ nur die beiden Elemente 0, 1 enthält mit $0 \oplus 0 = 1 \oplus 1 = 0$, ergibt jeder Vektor $v \in \mathbb{F}_2^N$ zu sich selbst addiert den Nullvektor; somit läßt sich das Paar (L_{i-1}, R_{i-1}) aus (L_i, R_i) rekonstruieren durch

$$R_{i-1} = L_i \quad \text{und} \quad L_{i-1} = f(s_i, L_i) \oplus R_{i-1}\,.$$

Zur Entschlüsselung einer als FEISTEL-Netzwerk aufgebauten Blockchiffre kann die Verschlüsselung also einfach rundenweise rückgängig gemacht werden, wobei im wesentlichen derselbe Algorithmus wie zur Verschlüsselung benutzt wird.

§ 3: Der Data Encryption Standard DES

Logisch gehört dieser Paragraph eigentlich zum vorigen Kapitel: Der Data Encryption Standard DES ist kein Verfahren, das man heute noch anwenden sollte. Der Standard wurde 1977 eingeführt und war ursprünglich für eine Dauer von zehn Jahren vorgesehen. Spätestens seit 1998, als die *Electronic Frontier Foundation* öffentlich vorführte, wie einfach er geknackt werden kann, sollte jedem klar sein, daß er seine nützliche Lebensdauer inzwischen deutlich überschritten hat.

Tatsächlich aber dürfte er immer noch zumindest eines der in der Praxis am häufigsten eingesetzten Kryptoverfahren sein, wenn auch (hoffentlich) meist in der derzeit wahrscheinlich noch sicheren Variante Triple-DES. Ein Grund dafür dürfte in einer heute als falsch eingeschätzten Rahmenbedingung seines Entwurfs liegen: Er sollte sich nur schwer rein softwaremäßig implementieren lassen und im Regelfall mit Spezialhardware eingesetzt werden. Dies hielt man für einen Sicherheitsvorteil, da dadurch ein Angriff von Amateuren mit begrenzten Mitteln deutlich erschwert wurde. Aus diesem Grund mußten viele Anwender in Spezialhardware investieren, die sich zumindest in manchen Unternehmen nur schwer aussondern läßt, bevor sie nicht mehrfach abgeschrieben ist.

Professionelle Angreifer freilich (zu denen auf jeden Fall auch der Hauptsponsor des DES, die *National Security Agency* NSA der Vereinigten Staaten zählt), sind typischerweise bereit, zum Knacken eines Codes ein Vielfaches des Aufwands einzusetzen, den sparsame Buchhalter für die Verschlüsselung zulassen, so daß sie durch die Notwendigkeit von Spezialhardware nicht abgeschreckt werden können.

Heute, da auch Amateure mit reinen Softwareattacken keine großen Schwierigkeiten mehr haben, DES zu knacken, muß man trotzdem sagen, daß DES nach allem, was in den letzten dreißig Jahren bekannt wurde, abgesehen von der viel zu kleinen Schlüssellänge ein sehr gutes Verfahren war: Trotz vieler Versuche auch der erfahrensten unter den veröffentlichenden Kryptanalytikern ist es keinem von ihnen gelungen, eine Angriffsmöglichkeit zu finden, die schneller wäre als das Durchprobieren aller Schlüssel: Alle erfolgreichen Angriffe basieren darauf. Im Sinne der Sicherheitsdiskussion im ersten Kapitel ist das die ideale Situation für ein Verfahren, dessen Sicherheit wir nicht beweisen können: Wir können mit ziemlich großer Sicherheit sagen, wie groß der Aufwand des Gegners sein muß: Er muß die Schlüssel durchprobieren, was im Durchschnitt nach 2^{55} Versuchen zum Erfolg führt. Unser einziges Problem besteht darin, daß dieser Aufwand inzwischen mit recht geringen Kosten erbracht werden kann.

Der DES ist im wesentlichen als FEISTEL-Netzwerk aufgebaut, allerdings werden die Nachrichtenblöcke $(x_1, x_2, \ldots, x_{64})$ zunächst einer Anfangspermutation unterzogen, d.h. der Block wird ersetzt durch

$(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(64)})$, wobei die Folge der Zahlen $\pi(1), \ldots, \pi(64)$ der folgenden Tabelle entnommen wird:

58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7

Das Wort geht also auf $(x_{58}, x_{50}, x_{42}, \ldots, x_{23}, x_{15}, x_7)$. Erst dann beginnen die im Falle des DES sechzehn Runden.

Die Anfangspermutation hat keine kryptographische Funktion: Da sie nicht vom Schlüssel abhängt und allgemein bekannt ist, kann sie jeder Kryptanalytiker leicht rückgängig machen. Ihr Sinn bestand anscheinend in erster Linie darin, Software-Attacken zu erschweren, denn Permutation sind aufwendig zu programmieren. (Bei Hardware-Implementierungen sind Permutationen natürlich sehr einfach und schnell durch Leitungskreuzungen zu realisieren.)

Zur Definition der FEISTEL-Funktion f dienen acht sogenannte S-Boxen (S = Substitution), die als Wertetabellen einem Eingabewort aus sechs Bit einen vier Bit langen Funktionswert zuordnen; sie beschreiben also Abbildungen von $\mathbb{F}_2^6$ nach $\mathbb{F}_2^4$.

Diese Wertetabellen sind folgendermaßen angeordnet: Das Eingabewort mit seinen sechs Bit wird geschrieben als (a, m, e), wobei a das Anfangsbit, e das Endbit und m die aus vier Bit bestehende Mitte ist. Diese wird als Zahl zwischen Null und fünfzehn aufgefaßt, genau wie auch die Ausgabe der S-Box. Die S-Box wird angegeben durch vier Zeilen, die mit den verschiedenen Möglichkeiten für das Paar (a, e) indiziert sind, und die für die sechzehn Werte von m die Ausgabewerte enthalten:

Box 1

$a\ e$	m = 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0 0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0 1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
1 0	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
1 1	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Box 2

$a\,e$	m = 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0 0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
0 1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
1 0	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
1 1	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

Box 3

$a\,e$	m = 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0 0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
0 1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
1 0	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1 1	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

Box 4

$a\,e$	m = 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0 0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
0 1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
1 0	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
1 1	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

Box 5

$a\,e$	m = 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0 0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
0 1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
1 0	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
1 1	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

Box 6

$a\,e$	m = 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0 0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
0 1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
1 0	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
1 1	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

Box 7

$a\,e$	m = 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0 0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
0 1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1 0	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
1 1	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

Box 8

$a\,e$	m = 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0 0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
0 1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
1 0	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
1 1	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Diese S-Boxen werden in der offensichtlichen Weise zusammengesetzt zu einer Funktion $F\colon \mathbb{F}_2^{48} \to \mathbb{F}_2^{32}$: Ein Vektor der Länge 48 wird aufgeteilt in acht Vektoren der Länge sechs; auf den ersten davon wird die erste S-Box angewandt, auf den zweiten die zweite *usw;* dabei entstehen acht Vektoren der Länge vier, die zum Ergebnisvektor der Länge 32 zusammengesetzt werden.

Nach diesem Konfusionsschritt folgt noch ein Diffusionsschritt: Die Komponenten des Vektors werden untereinander permutiert mittels einer Permutation aus S_{32}, die durch folgende Wertetabelle geben ist:

16	7	20	21	29	12	28	17		1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9		19	13	30	6	22	11	4	25

Die Funktion F wird folgendermaßen eingesetzt: Zunächst wird die rechte Hälfte R der Eingabe der jeweiligen Runde auf 48 Bit vergrößert, indem man einen Vektor $(x_1, \ldots, x_{32})$ ersetzt durch $(x_{\tau(1)}, \ldots, x_{\tau(48)})$, wobei die Werte von τ der folgenden Tabelle entnommen werden:

32	1	2	3	4	5		4	5	6	7	8	9		8	9	10	11	12	13
12	13	14	15	16	17		16	17	18	19	20	21		20	21	22	23	24	25
24	25	26	27	28	29		28	29	30	31	32	1							

Das Ergebnis dieser Aufblähung wird zum Schlüssel der i-ten Runde addiert; der kryptologische Sinn besteht natürlich wieder in einer Diffusion, die dafür sorgt, daß ein Eingabebit möglichst viele Ausgabebits beeinflußt.

Damit kommen wir zur Verwendung des Schlüssels im Algorithmus. Der Schlüssel hat, wie bereits erwähnt, 56 Bit, wird aber mit 64 Bit gespeichert, wobei jedes achte Bit ein Paritätsbit ist, d.h. die Summe (in $\mathbb{F}_2$) der sieben davorstehenden Bits. Wir numerieren die Schlüsselbits daher von eins bis 64, verwenden aber nur die nicht durch acht teilbaren Indizes.

Aus dem Schlüssel werden zunächst zwei Schlüssel der Länge 28 extrahiert, bestehend aus den folgenden Komponenten:

```
57 49 41 33 25 17  9     1 58 50 42 34 26 18
10  2 59 51 43 35 27    19 11  3 60 52 44 36
```

und

```
63 55 47 39 31 23 15     7 62 54 46 38 30 22
14  6 61 53 45 37 29    21 13  5 28 20 12  4
```

Die so erhaltenen Teilschlüssel werden vor jeder Runde noch zirkulär nach links verschoben, und zwar vor der i-ten Runde um nochmals a_i Bit gegenüber der vorherigen Runde, wobei a_1 bis a_{16} die Zahlenfolge

$$1\ 1\ 2\ 2\ 2\ 2\ 2\ 2\ 1\ 2\ 2\ 2\ 2\ 2\ 2\ 1$$

ist. Man beachte, daß die Summe dieser Zahlen gleich 28 ist, jeder der Halbschlüssel wird also in den sechzehn Runden einmal komplett zyklisch verschoben.

Nachdem die beiden Teilschlüssel so präpariert sind und aneinandergehängt einen 56 Bit-Schlüssel $(s_1, \ldots, s_{56})$ bilden, wird daraus ein 48 Bit-Schlüssel für die i-te Runde gewählt, bestehend aus den folgenden Komponenten:

```
14 17 11 24  1  5  3 28    15  6 21 10 23 19 12  4
26  8 16  7 27 20 13  2    41 52 31 37 47 55 30 40
51 45 33 48 44 49 39 56    34 53 46 42 50 36 29 32
```


Die Rundenschlüssel werden also in einem reinen Diffusionsverfahren aus dem Gesamtschlüssel berechnet.

Die FEISTEL-Funktion f berechnet die Summe aus den 48 aufgeblähten Nachrichtenbits und den 48 Schlüsselbits der i-ten Runde und setzt diesen Vektor aus F_2^{48} in F ein; der Funktionswert ist der Wert der FEISTEL-Funktion.

Dieses Spiel wird insgesamt sechzehnmal gespielt, wobei nach der letzten Runde keine Vertauschung von links und rechts mehr stattfindet. Danach wird nur noch die Anfangspermutation rückgängig gemacht; die inverse Permutation hat die Wertetabelle

```
40  8 48 16 56 24 64 32    39  7 47 15 55 23 63 31
38  6 46 14 54 22 62 30    37  5 45 13 53 21 61 29
36  4 44 12 52 20 60 28    35  3 43 11 51 19 59 27
34  2 42 10 50 18 58 26    33  1 41  9 49 17 57 25
```

Man beachte, daß DES (wie jedes (balancierte) FEISTEL-Netzwerk) in jeder Runde nur einen halben Block verändert; die andere Hälfte bleibt erhalten. Schreiben wir den Nachrichtenblock nach Anwendung der Anfangspermutation also in der Form (m_0, m_1), so wird in der i-ten Runde (m_{i-1}, m_i) zu (m_i, m_{i+1}) mit

$$m_{i+1} = m_{i-1} \oplus f(s_i, m_i)\,,$$

wobei wieder s_i den Schlüssel der i-ten Runde bezeichnet. Das Ergebnis der sechzehn Runden, abgesehen von der Endpermutation, ist dann allerdings nicht (m_{16}, m_{17}), sondern (m_{17}, m_{16}), da nach der letzten Runde die Hälften nicht mehr miteinander vertauscht werden.

Der Grund dafür liegt in der Entschlüsselung: Wegen

$$m_{i+1} = m_{i-1} \oplus f(s_i, m_i) \Leftrightarrow m_{i-1} = m_{i+1} \oplus f(s_i, m_i)$$

läßt sich die Verschlüsselung bei Kenntnis des Schlüssels leicht rückgängig machen, sogar mit derselben Hardware. Da aber vor dem ersten Verschlüsselungsschritt die Hälften nicht vertauscht werden, sollten sie es dann auch nach dem letzten nicht mehr werden, denn die Entschlüsselung läuft ja rückwärts durch die Runden.

§ 4: Designkriterien und Kryptanalyse des DES

Die Beschreibung des DES im vorigen Paragraphen läßt die meisten Leser zunächst wohl mit ziemlicher Verwirrung zurück, und es erscheint schwierig, irgendeine Aussage über die kryptographische Sicherheit des Verfahrens zu machen. In der Tat wurde diese von Anfang an sehr kontrovers diskutiert.

a) Geschichtliche Entwicklung

Als das damalige *National Bureau of Standards* der USA (heute National Institute of Standards and Technology, NIST) im Januar 1977 den DES als Standard veröffentlichte (mit einer auf zehn Jahre veranschlagten Laufzeit) enthielt das Dokument im wesentlichen nur die hier reproduzierten Angaben; sowohl IBM als auch die National Security Agency (NSA) lehnten es ab, die Kriterien zu benennen, nach denen die S-Boxen und die Permutationen konzipiert worden waren.

Dies führte schnell auf den Verdacht, daß DES möglicherweise eine nur IBM und NSA bekannte „Falltür“ enthält, mit deren Hilfe eine Entschlüsselung ohne Schlüssel mit vertretbarem Aufwand durchgeführt werden kann. Außerdem gab es bereits damals Kritik an der mit 56 Bit sehr kurzen Schlüssellänge: Das Vorgängersystem LUCIFER hatte eine Schlüssellänge von 128 Bit, im militärischen Bereich waren Systeme mit mehr als zehn mal so langen Schlüsseln nichts Ungewöhnliches.

M.E. HELLMAN: A cryptanalytic time-memory tradeoff, *IEEE Trans. Inf. Theory* **26** (1980), 401–406

schlug ein Verfahren vor, mit dem durch eine Kombination von Vorberechnungen und Probieren die Komplexität der Schlüsselsuche von 2^{56} mit großer Erfolgswahrscheinlichkeit auf ungefähr die Kubikwurzel dieser Zahl reduziert werden konnte; als Baupreis seiner Maschine schätzte er zehn Millionen Dollar, als Zeitrahmen für die Vorberechnungen ungefähr ein Jahr. Da die NSA erheblich größere Geldmittel als nur zehn Millionen Dollar einsetzen kann, bestärkte dies den Verdacht, daß sie DES selbst dann knacken kann, wenn der Algorithmus keine Falltür enthalten sollte.

Im Laufe der Jahre wurden einige der Designkriterien durch *reverse engineering* gefunden; einige dann auch freiwillig veröffentlicht. Erst 1994 veröffentlichte einer der ursprünglichen Entwickler bei IBM die, wie er sagt, vollständige Liste der kryptographisch relevanten Kriterien in

D. COPPERSMITH: The Data Encryption Standard (DES) and its strength against attacks, *IBM J. Res. Develop.* **38** (1994), S. 243–250

– nachdem die kryptanalytische Technik, gegen die diese Kriterien schützen sollten, auch in der offenen Literatur erschienen war. Dabei zeigte sich, daß DES mit seinen nur 56 Bit zumindest gegen diese Technik eine eher größere Sicherheit bietet als Lucifer mit seinen 128 Bit und daß die Sicherheit von DES nicht unbedingt erhöht würde, indem man für jede der sechzehn Runden einen neuen 48 Bit-Schlüssel verwendet, so daß man insgesamt eine Schlüssellänge von $16 \times 48 = 762$ Bit hätte. NSA hielt diese Technik damals für so wichtig für den Angriff auf gegnerische Systeme, daß die speziell dagegen eingesetzten Designkriterien „aus Gründen der nationalen Sicherheit“ geheimgehalten wurden.

Die Technik, um die es hier geht, war bei IBM um 1974 unter dem Namen *T attack* bekannt; in der offenen Literatur erschienen erste Ansätze dazu ab etwa 1988, vollständige Beschreibungen erschienen ab 1990 unter dem Namen *differentielle Kryptanalyse*. Bevor wir sie genauer betrachten, wollen wir uns zunächst die inzwischen bekannten Designkriterien des DES ansehen.

b) Designkriterien

D. COPPERSMITH nennt in der oben zitierten Arbeit folgende Designkriterien für die S-Boxen (und sagt, daß dies *alle* kryptographisch relevanten gewesen seien; der Rest habe nur mit Implementierungsfragen zusammengehängt):

(S1) Jede S-Box hat sechs Eingabe- und vier Ausgabebits.

(S2) Kein Ausgabebit einer S-Box sollte zu nahe bei einer linearen Funktion der Eingabebits liegen.

(S3) Bei festgehaltenem linken und rechtem Bit der Eingabe sollte jeder der sechzehn möglichen Ausgabewerte genau einmal vorkommen.

(S4) Wenn sich zwei Eingaben einer S-Box um genau ein Bit unterscheiden, müssen sich die Ausgaben um mindestens zwei Bit unterscheiden.

(S5) Wenn sich zwei Eingaben einer S-Box genau in den beiden mittleren Bits unterscheiden, müssen sich die Ausgaben um mindestens zwei Bit unterscheiden.

(S6) Wenn sich zwei Eingaben einer S-Box in ihren beiden Anfangsbits, nicht aber in ihren beiden Endbits unterscheiden, müssen die Ausgaben verschieden sein.

(S7) Für jede von Null verschiedene Differenz Δ zwischen zwei Eingaben dürfen höchstens acht der 32 Paare mit Differenz Δ auf dieselbe Differenz zwischen den Ausgaben führen.

(S8) Ähnlich zu *(S7)*, aber mit stärkeren Eigenschaften für den Fall gleicher Ausgaben, wenn in einer Runde drei S-Boxen „aktiv“ sind. *(s.u.)*

Für die Permutation aus S_{32}, die in jeder FEISTEL-Funktion als Abschluß ausgeführt wird, sollten folgende Bedingungen erfüllt sein:

(P1) Die vier Ausgabebits einer S-Box werden so verteilt, daß in der nächsten Runde zwei von ihnen mittlere Bits der Eingabe einer S-Box sind und die beiden anderen nicht (d.h. die kommen an Position 1, 2, 5 oder 6).

(P2) Die vier Ausgabebits einer S-Box sind in der nächsten Runde Eingaben zu sechs verschiedenen S-Boxen; keine zwei von ihnen sind Eingabe derselben S-Box.

(P3) Für zwei (nicht notwendigerweise verschiedene) S-Boxen j, k gilt: Wenn ein Ausgabebit von j als eines der beiden mittleren Bits an k weitergegeben wird, kann kein Ausgabebit von k als mittleres Bit an j weitergegeben werden. Insbesondere darf also kein Ausgabebit von j an j selbst als mittleres Bit weitergegeben werden.

Der Sinn einiger dieser Kriterien ist unmittelbar einsichtig: *(S1)* etwa kommt daher, daß mit der Technologie von 1974 größere S-Boxen dazu geführt hätten, daß man den Algorithmus nicht auf einem Chip untergebracht hätte.

(S2) ist selbstverständlich: Da die S-Boxen der einzige nichtlineare Bestandteil des Algorithmus sind, müssen sie nichtlinear sein; ansonsten hätten wir eine (leicht zu knackende) HILL-Chiffre. Wenn einzelne Bits lineare Funktionen der Eingabebits wären, hätten wir möglicherweise für einzelne Ausgabebits des Algorithmus lineare Zusammenhänge mit den Eingabebits, was dazu führen würde, daß man zumindest einen Teil der Chiffre als HILL-Chiffre betrachten kann und damit die Komplexität des Algorithmus reduziert. Ähnlich verhält es sich, wenn Funktionen nicht exakt, aber doch ungefähr linear sind – mehr dazu gleich bei der *linearen Kryptanalyse*.

Die restlichen Kriterien dienen in erster Linie zur Förderung der Diffusion: Für zwei verschiedene Eingaben W, W' in Runde i sagen wir, eine S-Box sei *aktiv*, wenn sie für W und W' verschiedene Ausgaben liefert. Es muß nicht in jeder Runde aktive S-Boxen geben, aber die obigen Kriterien sollen dafür sorgen, daß im Durchschnitt über alle Runden möglichst viele S-Boxen pro Runde aktiv sind; wie man zeigen kann, sind es im Durchschnitt mindestens 1,6.

(Bei *(S7)* sind die Zahlen, so wie sie genannt wurden, offensichtlich um den Faktor zwei zu klein: Es gibt 64 Paare mit vorgegebener Differenz Δ, und für $\Delta \neq 0$ dürfen dann wohl höchstens 16 davon auf denselben Ausgabewert führen.)

Bevor wir solche Fragen vertiefen können, müssen wir uns zunächst mit der kryptanalytischen Attacke beschäftigen, vor der dies schützen soll:

c) Differentielle Kryptanalyse

Ihre Grundidee besteht darin, daß man nicht von einzelnen Klartextblöcken ausgeht, sondern von Paaren (W, W') aus zwei Klartextblöcken. Diese werden aufgefaßt als Elemente von $\mathbb{F}_2^{64}$; da über dem Körper mit

zwei Elementen Addition gleich Subtraktion ist, bezeichnen wir die Differenz zwischen den beiden Nachrichten als $W \oplus W'$. Praktisch handelt es sich hier einfach um das bitweise XOR zwischen den beiden Blöcken.

DES unterzieht die beiden Worte zunächst der Anfangspermutation; da XOR eine bitweise Operation ist, wird dabei auch die Differenz $W \oplus W'$ dieser Permutation unterzogen. Danach werden die rechten Hälften der entstandenen Nachrichten betrachtet; ihre Differenz ist natürlich einfach die rechte Hälfte der permutierten Differenz. Die entstandenen 32 Bit-Worte werden durch Bitauswahl auf 48 Bit Worte V, V' aufgebläht; auch diese Aufblähung ist kompatibel mit der Differenzbildung.

Als nächstes kommt der Schlüssel ins Spiel; sowohl V als auch V' werden zum 48-Bit Schlüssel s_1 der ersten Runde addiert; dann gehen die Ergebnisse $V \oplus s_1$ und $V' \oplus s_1$ in acht 6 Bit Stücke aufgespalten in die acht S-Boxen. Die Differenz zwischen den beiden Eingaben ist

$$(V \oplus s_1) \oplus (V' \oplus s_1) = (V \oplus V') \oplus (s_1 \oplus s_1) = V \oplus V' ,$$

d.h. der Schlüssel ist herausgefallen.

Nun kommen die S-Boxen ins Spiel. Falls diese linear wären, wäre die Differenz ihre Ausgabe für zwei gegebene Eingabewerte nur von der Differenz der Eingabewerte abhängig, aber da es gerade der Zweck der S-Boxen ist, die Verschlüsselung nichtlinear zu machen, können wir natürlich nicht erwarten, daß wir hier auch nur bei einer einzigen S-Box eine lineare Funktion finden: Schon die ersten experimentellen Untersuchungen von DES befaßten sich mit etwaigen linearen Zusammenhängen zwischen einzelnen Ausgabebits sowohl einer S-Box wie auch des gesamten DES und der jeweiligen Eingabe, und keine konnte eine lineare Funktion finden.

Nach der Anwendung der S-Boxen können wir also nicht mehr sagen, was die Differenz der Ausgabewerte ist, obwohl wir die Differenz der Eingabewerte auch unabhängig vom Schlüssel kennen.

Trotzdem zeigt sich, daß wir zumindest gewisse Informationen über die Differenz haben: Bei sechs Eingabebits und vier Ausgabebits pro S-Box müssen von den $2^6 = 64$ Eingabepaaren (E, E') mit einer gegebenen

Differenz ΔE im Durchschnitt jeweils vier auf jede der sechzehn möglichen Differenzen ΔA der Ausgabewerte A, A' führen. Im Einzelnen gibt es allerdings beträchtliche Schwankungen:

Für $\Delta E = 0$ ist natürlich auch $\Delta A = 0$, denn dasselbe Wort kann nicht auf zwei verschiedene Weisen verschlüsselt werden. Aber auch für andere Werte von ΔE gibt es keine Gleichverteilung der Ausgabewerte: Für $\Delta E = 100100$ etwa ergibt sich für die (dezimal geschriebenen) Differenzen ΔA bei der ersten S-Box folgende Verteilung:

ΔA	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Fälle	12	0	0	2	2	2	2	0	14	14	2	0	2	6	2	4

Wir haben also hier, wie auch bei den anderen Differenzen und anderen S-Boxen eine ziemlich inhomogene Verteilung. (Die Fallanzahlen für jede S-Box und jede Ausgabedifferenz sind aufgelistet im Anhang von

E. BIHAM, A. SHAMIR: Differential Cryptanalysis of the Data Encryption Standard, *Springer,* 1993;

in diesem Buch ist die differentielle Kryptanalyse des DES und anderer Blockchiffren vollständig beschrieben.)

Auf die Ausgabe der acht S-Boxen wird eine Permutation angewandt; die ist wieder mit Differenzenbildung kompatibel. Falls wir also die Differenzen der Ausgaben der S-Boxen kennen, bereitet diese Permutation keine Schwierigkeiten und wir kennen die Eingabedifferenzen für die zweite Runde, mit denen wir genauso weiter verfahren können.

Tatsächlich kennen wir die Ausgaben der acht S-Boxen der ersten Runde natürlich nicht; wir können nur für jede einzelne S-Box eine Wahrscheinlichkeitsverteilung der Ausgabedifferenzen angeben. Für einzelne Bits oder Bitgruppen kommen wir dabei durchaus auf recht ansehnliche Wahrscheinlichkeiten: Im obigen Beispiel für die Eingabedifferenz 100100 etwa ist mit jeweils einer Wahrscheinlichkeit von $14 : 64$, also in mehr als 20% aller Fälle, die Ausgabedifferenz gleich acht oder neun, binär geschrieben also 0100 oder 0101. Die Wahrscheinlichkeit dafür, daß die ersten drei Bits der Ausgabedifferenz gleich 010 sind, ist also $28 : 64 = 7 : 16$, und wenn wir uns auf das erste und dritte Bit beschränken, kommen wir auf eine Wahrscheinlichkeit von $40 : 64 = 5 : 8$

dafür, daß die Ausgabedifferenz die Form $0x0y$ hat. Das dritte Bit schließlich ist in 52 der 64 möglichen Fällen gleich Null, so daß wir zumindest dieses eine Bit mit der recht hohen Wahrscheinlichkeit von 13 : 16 kennen.

Eine Blockchiffre geht insbesondere deshalb durch mehrere Runden, daß sie solche Inhomogenitäten durch die Konfusion und Diffusion in den Folgerunden weitestgehend zu zerstören. Man wird erwarten, daß dies nicht für alle Klartextdifferenzen gleich gut gelingt; die Idee hinter der differentiellen Kryptanalyse ist, sich auf die zu konzentrieren, bei denen es möglichst schlecht gelingt. Wir müssen uns also genauer anschauen, wie eine Klartextdiffernz durch die Runden geht.

Definition: Eine *r-Rundencharakteristik* ist eine Folge

$$\Delta = (\delta_0, \delta_1, \ldots, \delta_r)$$

von Elementen aus $\mathbb{F}_2^{64}$.
Ein Klartextpaar $(x_0, y_0) \in \mathbb{F}_2^{64} \times \mathbb{F}_2^{64}$ *gehört zur Charakteristik* Δ, wenn für die Paare (x_i, y_i) der Ausgaben der i-ten Runde gilt: $x_i \oplus y_i = \delta_i$ für $i = 0, \ldots, r$.
Die *Wahrscheinlichkeit* einer Charakteristik ist die Wahrscheinlichkeit dafür, daß ein Klartextpaar (x_0, y_0) mit Differenz δ_0 zur Charakteristik Δ gehört.

Natürlich sind die Wahrscheinlichkeiten der meisten Charakteristiken sehr gering: In einem idealen Kryptosystem wären alle Ausgabewerte einer Runde gleich wahrscheinlich, die Wahrscheinlichkeit einer r-Rundencharakteristik sollte also im Mittel bei $1/2^{-64r}$ liegen, und schon die Wahrscheinlichkeit dafür, daß überhaupt ein Klartextpaar mit gegebener Differenz nach r-Runden eine ebenfalls vorgegebene andere Differenz hat, sollte im allgemeinen bei nur etwa 2^{-64} liegen.

Wenn wir eine gute r-Rundencharakteristik gefunden haben, deren Wahrscheinlichkeit deutlich besser ist als 2^{-64r}, können wir daher ziemlich sicher sein, daß ein zufällig gewähltes Klartextpaar (x_0, y_0) mit Anfangsdifferenz δ_0 und Enddifferenz δ_r in den r Runden so verschlüsselt wurde, wie es der Charakteristik entspricht. Ist etwa $p = 2^{-44}$, sollte die Wahrscheinlichkeit für alles andere bei nur etwa 2^{-20} oder etwa eins

zu einer Million liegen. Konkret heißt das: Wir kennen die Differenzen $x_i \oplus y_i$ mit hoher Wahrscheinlichkeit und können die Verschlüsselung durch die r Runden verfolgen.

Zunächst brauchen wir aber gute Charakteristiken. Für nur eine Runde ist das sehr einfach: Für jeden Halbblock $d_0 \in \mathbb{F}_2^{32}$ führt der mit 32 Nullen auf 64 Bit aufgefüllte Block δ_0 auf die Charakteristik (δ_0, δ_0) mit Wahrscheinlichkeit eins: Sind nämlich (m_0, m_1) und (m_0', m_1) zwei Klartexte mit (nach der hier stets ignorierten Anfangspermutation) gleicher rechter Hälfte, so wird die linke Hälfte ersetzt durch

$$m_2 = m_0 \oplus f(s_1, m_1) \quad \textit{bzw.} \quad m_2' = m_0' \oplus f(s_1, m_1)\,,$$

und die Differenz ist

$$m_2 \oplus m_2' = \big(m_0 \oplus f(s_1, m_1)\big) \oplus \big(m_0' \oplus f(s_1, m_1)\big) = m_0 \oplus m_0' = d_0\,,$$

wie wir das schon oben gesehen haben. Nach Durchgang durch die erste Runde haben wir also die beiden Paare (m_2, m_1) und (m_2', m_1), deren Differenz wieder δ_0 ist.

Leider können wir diese Charakteristik nicht iterieren, denn nach der ersten Runde werden ja die beiden Hälften vertauscht, so daß wir dann (m_1, m_2') und (m_1, m_2') haben, worüber wir nicht so viel sagen können, da nun die FEISTEL-Funktionen verschiedene Werte liefern.

Um trotzdem zu einer Zweirundencharakteristik zu kommen, benutzen wir die Nichtinjektivität der FEISTEL-Funktion f: Wir suchen zwei Halbblöcke m_0 und m_0' derart, daß $f(s, m_0) = f(s, m_0')$ für möglichst viele Schlüssel s; die Differenz $m_0 \oplus m_0'$ bezeichnen wir mit d_0.

Zwei beliebige Klartexte der Form (m_0, m_1) und (m_0', m_1) mit Differenz $m_0 \oplus m_0' = d_0$ gehen in der ersten Runde nach (m_2, m_1) und (m_2', m_1) mit

$$m_2 = m_0 \oplus f(s_1, m_1) \quad \text{und} \quad m_2' = m_0' \oplus f(s_1, m_1)\,,$$

danach werden die linke und die rechte Hälfte vertauscht, so daß die Eingaben zur zweiten Runde gleich (m_1, m_2) und (m_1, m_2') sind, wobei m_2 und m_2' die Differenz d_0 haben. In der zweiten Runde gehen die beiden Klartexte dann nach

$$\big(m_1 \oplus f(s_2, m_2), m_2\big) \quad \text{und} \quad \big(m_1 \oplus f(s_2, m_2'), m_2'\big)\,.$$

Da m_2 und m_2' Differenz d_0 haben, ist dabei mit einer gewissen Wahrscheinlichkeit

$$f(s_2, m_2) = f(s_2, m_2') .$$

Falls dem so sein sollte, haben wir $(0, d_0)$ als Differenz zwischen den Ausgabewerten. Es gibt daher eine Zweirundencharakteristik der Form

$$\big((d_0, 0), (d_0, 0), (0, d_0)\big) ,$$

die mit einer gewissen Wahrscheinlichkeit p auftritt. Diese Charakteristik kann offensichtlich beliebig oft iteriert werden, denn bevor ihre Ausgabewerte in die nächste Runde gehen, werden die linke und die rechte Hälfte vertauscht, so daß wir wieder die Ausgangsdifferenz haben. Damit haben wir für jedes n eine n-Rundencharakteristik gefunden; ihre Wahrscheinlichkeit ist $p^{[n/2]}$, wobei die GAUSS-Klammer $[x]$ die größte ganze Zahl $\leq x$ bezeichnet.

Diese Charakteristik ist umso nützlicher, je größer p ist. Wie sich zeigt, kann p nur dann größer als Null sein, wenn mindestens drei benachbarte S-Boxen aktiv sind; die größten Wahrscheinlichkeiten sind also zu erwarten bei *genau* drei aktiven S-Boxen. Systematisches Probieren zeigt, daß der beste erreichbare Wert dann

$$p = \frac{14}{64} \cdot \frac{8}{64} \cdot \frac{10}{64} = \frac{35}{8192} \approx 0{,}004272461 \approx \frac{1}{234} \approx 2^{-7{,}870716983}$$

ist. Er wird erreicht für

$$d_0 = (19\,60\,00\,00)_{\text{hex}} \quad \text{und} \quad d_0 = (1B\,60\,00\,00)_{\text{hex}} .$$

Damit haben wir also für beliebiges n eine n-Rundencharakteristik gefunden; leider ist sie aber nicht für jedes n brauchbar: Für 16 Runden ist ihre Wahrscheinlichkeit nur etwa

$$2^{-7{,}870716983 \times 8} \approx 2^{-62{,}96573586} ,$$

wir bräuchten also mindestens 2^{63} Klartextpaare bekannter Differenz, um eines zu dieser Charakteristik zu finden, während wir mit nur 2^{56} Versuchen alle Schlüssel durchprobieren könnten. Tatsächlich reichten sogar bereits 2^{55} Versuche, denn nimmt man das Einserkomplement von Schlüssel und Klartext, so entsteht das Einserkomplement des Chiffretexts. Deshalb sind auch die 14- und die 15-Rundencharakteristik, deren

Wahrscheinlichkeiten bei

$$2^{-7{,}870716983 \times 7} \approx 2^{-55{,}09501888}$$

liegen, nicht sonderlich interessant; erst die 13-Rundencharakteristik liefert mit

$$p \approx 2^{-7{,}870716983 \times 6} \approx 2^{-47{,}22430190}$$

eine halbwegs interessante Wahrscheinlichkeit.

Wählen wir ein Halbwort m_0 derart, daß $m_0 \oplus f(s, m_1)$ für möglichst viele Schlüssel s gleich ist! Für einen solchen Schlüssel sind dann

$$m_2 = m_0 \oplus f(s, m_1) \quad \text{und} \quad m_2' = m_0' \oplus f(s, m_1)$$

Außerdem interessieren wir uns nicht für die Wahrscheinlichkeitsverteilung der Chiffretexte – Chiffretext ist schließlich das, was wir immer haben – sondern für Klartext oder besser noch den Schlüssel bei gegebenem Chiffretext.

Dazu nutzen wir aus, daß der Eingabewert der S-Boxen der ersten Runde nicht der Klartext ist, sondern der mit gewissen Schlüsselbits geXORte Klartext. Wenn wir nun für ein Klartextpaar mit gegebener Differenz die Ausgabedifferenz kennen, haben wir die möglichen Eingabepaare der S-Boxen, deren Differenzen ja genau dieselben sind wie für das Klartextpaar, von 64 auf eine erheblich kleinere Zahl reduziert. Für jedes dieses möglichen Paare können wir die entsprechenden Schlüsselbits durch XOR mit dem tatsächlichen Klartext berechnen und haben somit eine relativ kleine Anzahl potentieller Schlüsselteile. Wenn wir das ganze für hinreichend viele Klartextpaare wiederholen, sollte der für die jeweilige S-Box zuständige Schlüsselanteil relativ bald eindeutig feststehen.

DES mit nur einer Runde ist auf diese Weise also relativ einfach zu entschlüsseln, falls wir genügend viele Paare von Klartext mit fester Differenz haben. Diese können wir uns nur verschaffen durch eine Attacke mit wählbarem Klartext, also der schwierigsten Form der Attacke.

Auch diese Attacke liefert aber nicht die Ausgabedifferenzen der ersten Runde, sondern nur die der *letzten*. Der Ansatz der differentiellen Kryptanalyse des DES ist daher folgender:

1. Man wähle eine geeignete Differenz zwischen Klartexten.
2. Dazu erzeuge man hinreichend viele Paare von Klartextblöcken mit dieser Differenz, verschlüssele sie und behalte nur die so berechneten Chiffretextpaare.
3. Durch Analyse der Klartextdifferenzen und des Verhaltens der S-Boxen in den verschiedenen Runden bestimme man die zu erwartende Wahrscheinlichkeitsverteilung der Eingabedifferenzen der letzten Runde.

Differentielle Kryptanalyse war der erste Ansatz, DES mit geringerem Aufwand als der vollständigen Durchsuchung des Schlüsselraums zu zu brechen. Da aber, wie bereits erwähnt, die Designer des DES die differentielle Kryptanalyse schon kannten lange bevor sie in der offenen Literatur auftauchte und den Algorithmus so gut wie möglich dagegen immun machten, ist diese Attacke nicht sehr praktikabel: Selbst bei Angriffen mit frei wählbarem Klartext braucht man über 2^{40} Paare aus Klartext und Chiffretext, um den Schlüssel zu finden.

d) Lineare Kryptanalyse

Eine leichte Verbesserung bietet die kurz später entdeckte *lineare Kryptanalyse:* Zwar sind die Ausgabebits der S-Boxen nach Design-Kriterium ($S2$) auch nicht näherungsweise lineare Funktionen der Eingabebits, aber es kann dennoch vorkommen, daß man eine Linearkombination von Ausgabebits mit einer Wahrscheinlichkeit, die deutlich über 50% liegt durch eine Linearkombination der Eingabebits vorhersagen kann. Mit hinreichend vielen Paaren aus Klartext und Chiffretext Läßt sich dadurch ein niedrigdimensionaler affiner Unterraum des Schlüsselraums $\mathbb{F}_2^{56}$ finden, in dem der Schlüssel mit hoher Wahrscheinlichkeit liegen muß. Die vollständige Durchsuchung dieses Unterraums ist unproblematisch, so daß der Schlüssel mit hoher Wahrscheinlichkeit gefunden werden kann.

Auch für diese Attacke sind allerdings unrealistisch viele Paare aus Klartext und Chiffretext erforderlich (in einer Variante genügen noch mehr reine Chiffretexte), so daß der Gesamtaufwand nicht wirklich geringer sein dürfte als die vollständige Durchsuchung des Schlüsselraums.

Nach allem, was in der offenen Literatur bekannt ist, gibt es also zum Knacken des DES keine wesentlich bessere Alternative zur vollständigen Durchsuchung des Schlüsselraums.

e) DES-Cracker

Der erste in der offenen Literatur dokumentierte realistische Angriff auf DES war denn auch die vollständige Durchsuchung des Schlüsselraums. Eine amerikanische Bürgerrechtsorganisation, die *Electronic Frontier Foundation (EFF)*, konstruierte eine Maschine mit Spezialhardware zum Knacken von DES mit Chiffretext allein.

Die *Electronic Frontier Foundation* wurde 1990 nach dem großen *Hacker Crackdown* in den USA gegründet; Initiatoren waren unter anderem JOHN PERRY BARLOW, bekannt vor allem durch die Lieder, die er für *The Grateful Dead* schrieb, JOHN GILMORE, einer der Pioniere sowohl von *Sun Microsystems* als auch der *Free Software Foundation*, MITCHELL KAPOR, der Gründer von *Lotus*, sowie STEVE WOZNIAK, einer der beiden Gründer von *Apple*.

Ihr Ansatz ist im wesentlichen der unseres guten alten Feinds, des BAYESschen Gegners: Kodiert man einen englisch- oder deutschsprachigen Klartext im ASCII-Code sollten dem BAYESschen Gegner ein bis zwei Blöcke Chiffretext ausreichen, um den Schlüssel zu finden.

Natürlich verfügen selbst die vier obengenannten Gründer der *Electronic Frontier Foundation* nicht über die unbegrenzten Mittel, die der BAYESsche Gegner einsetzen kann; verglichen mit vielen anderen Gegnern verfügt aber doch jeder von ihnen über beträchtliche Mittel. Trotzdem war das 1997 begonnene und 1998 beendete DES-Cracker-Projekt kein Angriff ohne Rücksicht auf die Kosten: Die *Electronic Frontier Foundation* wollte gerade zeigen, daß DES auch mit begrenzten Mitteln geknackt werden kann. Aus diesem Grund wurde der Ansatz des BAYESschen Gegners an mehreren Stellen optimiert:

Zunächst ist es nicht notwendig, wirklich für *jeden* Schlüssel die bedingte Wahrscheinlichkeit auf Grund des Chiffretexts zu bestimmen: In vielen Fällen werden bei der Entschlüsselung nicht druckbare Zeichen

entstehen, so daß schon nach wenigen Byte klar ist, daß die Wahrscheinlichkeit des Schlüssels Null ist.

DES Cracker beginnt daher mit der Aussonderung unmöglicher Schlüssel durch massiv parallele Hardware: Die Maschine arbeitet mit zwei Blöcken Chiffretext; sie hat als Kern von EFF entwickelte ASICs *application specific integrated circuits),* die einen 64-Bit-Block mit einem vorgegebenen Schlüssel dechiffrieren können und die Bytes des Ergebnisses auf vom Benutzer einstellbare Bitmuster überprüfen – beispielsweise darauf, ob es sich um ASCII-Codes druckbarer Zeichen handelt. Nur wenn alle Bytes den gewählten Kriterien genügen, wird auch der zweite Block entsprechend untersucht, und wenn auch hier kein im Klartext unmögliches Byte auftaucht, wird der Schlüssel zur weiteren Untersuchung an einen die Maschine steuernden PC weitergegeben, der eine genauere Untersuchung gemäß dem Ansatz des BAYESschen Gegners durchführt.

Von den 256 möglichen ASCII-Werten sind etwa ein Viertel druckbare Zeichen; da DES eine Ausgabe liefert, die sich nur wenig von einer Zufallsfolge unterscheidet, wird ein Block nur mit einer Wahrscheinlichkeit von etwa $1 : 4^8 = 1 : 65536$ den Test bestehen; für zwei Blöcke liegt die Wahrscheinlichkeit entsprechend bei $1 : 4^{16} = 1 : 2^{32}$. Von den 2^{56} zu untersuchenden Blöcken werden also nur etwa

$$2^{56-32} = 2^{24} = 16\,777\,216$$

an den PC weitergegeben, und die Untersuchung von etwa 17 Millionen Klartextkandidaten ist kein Problem für einen Standard-PC.

Der hauptsächliche Rechenaufwand liegt in der Voruntersuchung der Blöcke durch die ASICs; je nachdem, wie viele von diesen parallel arbeiten, kann dies mehr oder weniger schnell gehen.

Die tatsächlich gebaute Maschine enthält $1536 = 3 \times 2^9$ ASICs, von denen jedes aus 24 parallel arbeitenden Sucheinheiten besteht; insgesamt können also jeweils 36 864 Schlüssel parallel untersucht werden. Jede Sucheinheit kann zweieinhalb Millionen Schlüssel pro Sekunde untersuchen, die gesamte Maschine also etwas über 92 Milliarden.

Insgesamt müssen 2^{56} Schlüssel untersucht werden; im Mittel wird man nach 2^{55} Versuchen den richtigen gefunden haben. Dafür braucht man

$$\frac{2^{55}}{92\,160\,000\,000} \approx 390\,937 \text{ Sekunden} \approx 108{,}5 \text{ Stunden} \approx 4{,}5 \text{ Tage}\,.$$

Die Maschine ist skalierbar: Jeweils 64 Chips sitzen auf einem Board und 12 Boards in einer Chassis (einer ehemaligen SUN); die gebaute Maschine besteht aus dem steuernden PC zusammen mit zwei Chassis; der PC könnte aber auch mit deutlich mehr als zwei Chassis arbeiten und auch mehrere PCs wären denkbar.

In der gebauten Version kostete DES Cracker 210 000 $, wovon 80 000 $ Entwicklungskosten waren; der Bau eines zweiten Exemplars wäre also für 130 000 $ möglich, wobei der Preis bei Serienproduktion wohl deutlich niedriger gewesen wäre. Mit einer Investition in der Größenordnung von einer Million Dollar hätte am also bereits damals einen DES-Schlüssel innerhalb weniger Stunden finden können. Heute kann man auch rein softwaremäßig mit einem handelsüblichen PC.

Da eine Million Dollar auch für Geheimdienste kleiner Länder und (etwas kreative Buchhaltung vorausgesetzt) Großunternehmen kein Problem sind, war damit endgültig gezeigt, daß die Zeit für DES abgelaufen war.

Die EFF veröffentlichte sowohl die komplette Hardware-Spezifikation als auch die Software von DES-Cracker; in gedruckter Form findet man sie im Buch

ELECTRONIC FRONTIER FOUNDATION: *Cracking DES. Secrects of Encryption Research, Wiretap Politics & Chip Design,* O'Reilly, 1998

Online ist das Buch unter anderem verfügbar unter

`http://cryptome.org/cracking-des.htm` ;

die DES Cracker Seite der EFF ist

`www.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/` .

§4: Modifikationen

Viele Unternehmen, insbesondere im Bankenbereich, haben viel Geld in DES-Hardware investiert und haben daher wenig Interesse, auf ein neues Verfahren umzusteigen – ganz abgesehen davon, daß 1998 noch kein allgemein anerkannter Nachfolgealgorithmus zur Verfügung stand. (Mit dem inzwischen normierten „offiziellen“ Nachfolger AES werden wir uns in einen späteren Kapitel beschäftigen.)

Deshalb bot sich ein Verfahren an, das auf der Grundlage von DES eine Verschlüsselung mit deutlich mehr als nur 56 Schlüsselbit realisiert.

a) Mehrfacher DES

Die Kritik an der extrem kurzen Schlüssellänge des DES wurde bereits kurz nach dessen Einführung laut, und schon damals wurde vorgeschlagen, ihn zur Erhöhung der Sicherheit mehrfach und mit verschiedenen Schlüsseln anzuwenden.

Eine zweifache Verschlüsselung hängt von 112 statt von nur 56 Schlüsselbit ab; der Aufwand für eine Durchsuchung des gesamten Schlüsselraums steigt also von 2^{56} auf 2^{112}, was eine Maschine nach Art des DES Crackers auch nach heutigem Stand der Technik noch nicht in akzeptabler Zeit durchführen kann.

Eine mehrfache Verschlüsselung bietet allerdings nur dann Vorteile, wenn die Hintereinanderausführung zweier DES-Verschlüsselungen nicht äquivalent zur einfachen DES-Verschlüsselung mit einem anderen Schlüssel ist.

Um dies zu untersuchen, müssen wir noch einmal zurück zur grundsätzlichen Struktur von Blockchiffren: Blockchiffren sind Permutationen auf der Menge aller Blöcke; im Falle von DES also auf der Menge aller bijektiver Abbildungen von der Menge aller 64-Bit-Blöcke auf sich selbst.

Die 2^{56} durch DES definierten Permutationen bilden natürlich nur eine winzige Teilmenge der Gruppe aller $2^{64}!$ solcher Permutationen; falls diese Teilmenge eine Gruppe sein sollte (oder in einer relativ kleinen

Untergruppe liegt), kann die mehrfache Anwendung von DES keine (oder nur wenig) zusätzliche Sicherheit bieten.

Wir brauchen daher Informationen darüber, wie groß die kleinste Gruppe ist, die alle 2^{56} DES-Permutationen enthält. Über deren genaue Struktur und Elementanzahl ist leider nichts bekannt, man kann aber immerhin untere Schranken angeben: Für jeden einzelnen DES-Schlüssel kann man die zugehörige Substitution so lange wiederholen, bis die Identität entsteht; da $2^{64}!$ eine zwar große, aber endliche Zahl ist, muß dies nach endlich vielen Schritten der Fall sein.

Angenommen, bei solchen Berechnungen mit verschiedenen Schlüsseln ergeben sich die Ordnungen $n_1, n_2, \ldots, n_r$. Dann enthält die kleinste Gruppe, in der alle DES-Substitutionen liegen, zyklische Untergruppen der Ordnungen $n_1, \ldots, n_r$. Nach einem einfachen Satz der Gruppentheorie, dem Satz von LAGRANGE, müssen die Zahlen $n_1, \ldots, n_r$ dann die Ordnung der gesamten Gruppe teilen; diese ist also mindestens gleich dem kleinsten gemeinsamen Vielfachen der n_i.

Experimente zweier Wissenschaftler von Bell Northern Research in Ottawa zeigten, daß die Ordnung der erzeugten Untergruppe größer als $0{,}9 \times 10^{2499}$ sein muß; das liegt sehr deutlich über 2^{56}. Für Einzelheiten sei auf ihre Arbeit

KEITH W. CAMPBELL, MICHAEL J. WIENER: DES is not a Group, *Crypto '92, Springer Lecture Notes in Computer Science* **740** (1993), 512–520

verwiesen. Sie zeigt insbesondere, daß mehrfache Anwendung von DES die Sicherheit erhöhen kann.

b) Doppelter DES

Beim doppelten DES hat man einen Schlüsselraum mit 2^{112} Elementen. Leider muß ein Gegner mit hinreichend viel Speicherplatz diesen aber nicht vollständig durchsuchen, um die Schlüssel zu finden: Falls er nur ein Paar (x, y) aus Blöcken von einander entsprechendem Klartext und Chiffretext hat, reicht es, kann er in einer sogenannten *meet in the middle attack* den Klartext x mit allen 2^{56} möglichen Schlüsseln *ver*schlüsseln und den Chiffretext y mit allen 2^{56} Schlüsseln zu *ent*schlüsseln. Ist $y = \mathrm{DES}\big(s_2, \mathrm{DES}(s_1, x)\big)$, so ist $\mathrm{DES}^{-1}(s_2, y) = \mathrm{DES}(s_1, x)$, dieser

Block kommt also in beiden Listen vor. Da die von DES realisierten Permutationen weit von einer Gruppe entfernt sind, ist es sehr unwahrscheinlich, daß es noch einen anderen Block gibt, der in beiden Listen auftaucht; sobald man einen solchen Block gefunden hat, kann man also praktisch sicher sein, daß man s_1 und s_2 kennt.

Der Speicherbedarf dieser Attacke ist sicherlich nichts für Amateure am heimischen PC: Schließlich müssen für 2^{56} Schlüssel jeweils zwei Blöcke à 64 Bit oder 8 Byte gespeichert werden; insgesamt also 2^{60} Byte = 2^{50} Kilobyte = 2^{40} Megabyte = 2^{30} Gigabyte = 2^{20} Terabyte; das ist ungefähr eine Million mal soviel, wie eine große handelsübliche Festplatte heute faßt. Trotzdem handelt es sich hier um eine Kapazität, die nicht nur bei Regierungsorganisationen, sondern auch bei großen Unternehmen durchaus im Bereich des Möglichen liegt. In der Praxis spielt der doppelte DES daher keine Rolle.

c) Dreifacher DES

Der nächste Schritt zu Erhöhung der Komplexität besteht in einer dreifachen Anwendung von DES. Auch hier kann man wieder eine *meet in the middle attack* anwenden, aber auf dem Weg zur Mitte muß von mindestens einer der beiden Seiten aus DES zweimal mit verschiedenen Schlüsseln angewendet werden, der Aufwand liegt also in der Größenordnung von 2^{112}. Dreifacher DES oder, wie man meist sagt, Triple DES (kurz TDES), gilt daher im Augenblick noch als sicher: Nachdem die DES-Cracker-Attacke publik geworden war, zog die amerikanische Regierung die Zulassung von DES für weniger geheime Nachrichten im Regierungsbereich zurück und verlangte stattdessen Triple DES; entsprechend wurden in Deutschland die Euroscheckkarten ersetzt durch neue Karten, die auf Triple DES anstelle des einfachen DES beruhen. Daran hat sich bis heute wenig geändert: Die großen Kreditkartenunternehmen einigten sich auf einen gemeinsamen Standard, der nach den Initiatoren Europay, Mastercard und Visa als EMV bezeichnet wird, damit die Geldautomaten und Verkaufsstellenterminals einer Firma auch die Karten aller anderer beteiligter Unternehmen lesen können, und der verwendet weiterhin Triple DES als symmetrisches Kryptoverfahren. Die genau Spezifikation des Standards ist via `www.emvco.com` zu finden.

Triple DES wird praktisch immer so angewendet, daß man mit dem ersten Schlüssel *ver*schlüsselt, mit dem zweiten *ent*schlüsselt und mit dem dritten wieder *ver*schlüsselt. Oft ist dabei der dritte Schlüssel gleich dem ersten; zumindest für einen *meet in the middle* Angriff erleichtert dies die Arbeit des Angreifers nicht wesentlich. Trotzdem gehen inzwischen die Empfehlungen eher dahin, drei verschiedene Schlüssel zu verwenden.

d) DESX

Wie wir gesehen haben, wurde DES bewußt so spezifiziert, daß reine Softwareimplementierungen eher langsam sind. Bei Triple DES macht sich diese Langsamkeit gleich dreifach bemerkbar. Da es sich heute kaum mehr lohnt, neu in DES Hardware zu investieren, suchte man daher früh nach Alternativen, die einerseits das Problem der kurzen DES Schlüssellänge abmildern, andererseits aber keine mehrfache Anwendung von DES erfordern. RON RIVEST, den wir im nächsten Kapitel kennenlernen werden, schlug 1995 eine erstaunlich einfache solche Methode vor: Sein DESX kombiniert den gewöhnlichen DES einfach mit zwei VIGENÈRE-Verschlüsselungen: Zusätzlich zum DES-Schlüssel $s_1 \in \mathbb{F}_2^{56}$ gibt es noch zwei VIGENÈRE-Schlüssel $s_2, s_3 \in \mathbb{F}_2^{64}$, und ein Nachrichtenblock $x \in \mathbb{F}_2^{64}$ wird verschlüsselt als $s_3 \oplus DES(s_1, x \oplus s_2)$.

Nach unseren Erfahrungen mit der VIGENÈRE-Chiffre wäre es naiv zu glauben, daß ein Gegner zum Knacken dieser Chiffre den gesamten Schlüsselraum $\mathbb{F}_2^{56} \times \mathbb{F}_2^{64} \times \mathbb{F}_2^{64} \cong \mathbb{F}_2^{184}$ durchsuchen muß; er könnte beispielsweise durch eine differentielle Kryptanalyse zunächst den Schlüssel s_3 eliminieren, und wenn er – wie auch immer – s_1 und s_2 gefunden hat, genügt ein einziges Klartext/Chiffretext-Paar, um auch noch s_3 zu finden. Trotzdem ist DESX erstaunlich gut: Die beste bekannte Attacke muß auch bei 2^m bekannten Klartext/Chiffretext-Paaren immer noch 2^{119-m} mögliche Schlüssel durchprobieren; sie ist zu finden bei

> JOE KILIAN, PHILLIP ROGAWAY: How to protect DES against exhaustive key search (an analysis of DESX), *Journal of Cryptology* **14** (2001), 17–35;
> `http://www.springerlink.com/content/b1ljx8fky92nlhmk/`

e) Alternativen zu DES

Da DES nach heutigen Standard nicht mehr zeitgemäß ist, sind die gerade betrachteten Modifikationen höchstens als Übergangslösungen interessant; sinnvoller sind völlig neu konzipierte Alternativen. Insbesondere eine davon, denn offiziellen Nachfolger AES (Advanced Encryption Standard) werden wir in dieser Vorlesung noch ausführlich diskutieren; da hierzu allerdings mehr Algebra erforderlich ist, als wir bislang kennen, muß das entsprechende Kapitel nach weiter hinten verschoben werden, bis wir im Zusammenhang mit den sogenannten asymmetrischen Kryptoverfahren mehr Hilfsmittel aus der Algebra kennengelernt haben. Zum Abschluß dieses Kapitels wollen wir uns noch kurz mit der Frage befassen, wie man eine Blockchiffre nach Art von DES in der Praxis anwenden sollte.

§5: Operationsmodi

Bislang haben wir DES nur betrachtet für die Verschlüsselung eines einzelnen Blocks; tatsächlich besteht eine Nachricht aber meist aus einer ganzen Folge $x_1, x_2, \ldots, x_n$ von Blöcken. In diesem Paragraphen wollen wir uns überlegen, wie diese Nachricht am besten verschlüsselt wird. Dabei werden zum ersten Mal auf ein Phänomen stoßen, daß uns im Laufe dieser Vorlesung noch mehrfach begegnen wird: Auch eine relativ sichere Chiffre zeigt praktisch immer deutliche Schwächen, wenn sie einfach in der offensichtlichen Weise angewendet wird.

Die Betrachtungen hier beziehen sich nicht speziell auf DES, sondern gelten genauso auch für jede andere Blockchiffre. Daher gehen wir hier aus von *irgendeiner* Blockchiffre

$$B\colon S \times X \to X; \quad (s,x) \mapsto B(s,x)\,,$$

die einem Block x in Abhängigkeit von einem Schlüssel s den Chiffretext $B(s,x)$ zuordnet.

a) Electronic Code Book (ECB)

Die naheliegendste Weise, eine Nachricht $x_1, x_2, \ldots, x_n$ zu verschlüsseln, besteht darin, ihr den Chiffretext $y_1, y_2, \ldots, y_N$ zuzuordnen mit

$y_i = B(s, x_i)$, d.h jeder Block wird vor der Übertragung mit s verschlüsselt.

So einfach diese Methode auch ist, in der Praxis sollte man sie besser nicht anwenden. Ihre große Schwäche von ECB liegt in der Tatsache begründet, daß gleiche Klartextblöcke *immer* zu gleichen Chiffretextblöcken führen. Das bedeutet zwar nicht unbedingt, daß gleiche Klartextteile stets gleich verschlüsselt werden, denn wegen der Blockstruktur der Chiffre hängt der Chiffretext ja auch noch davon ab, wie weit der Textbeginn vom Blockanfang entfernt ist. Bei DES mit ASCII gibt es dafür aber nur acht Möglichkeiten, bei DES mit Unicode sogar nur vier, so daß bei längeren Texten in denen gewisse Namen und/oder Begriffe häufig vorkommen, durchaus die Gefahr einer größeren Anzahl identischer Chiffretextblöcke besteht.

Auch *magic bytes*, die bei vieler Dateiformaten als Dateianfang vorgeschrieben sind, führen stets zur selben Verschlüsselung; bei anderen Dateiformaten wie etwa ausführbaren Programmen oder gewissen Büroprogrammen gibt es innerhalb der Datei viele Blöcke von Nullen, *usw.*, so daß jemand, der alle Nachrichten eines Absenders abhört die Empfänger leicht in Klassen einteilen kann, die (ungefähr) dieselbe Information erhalten. Dadurch kennt er zwar noch nicht den Inhalt der Nachrichten, kann aber vielleicht doch sehr nützliche Informationen gewinnen.

Manchmal kann ein Gegner auch einfach dadurch Schaden anrichten, daß er unbemerkt die Reihenfolge von Nachrichtenblöcken vertauscht oder aber einen Nachrichtenblock *mehrfach* übermittelt. Er könnte auch eine neue Nachricht generieren, die aus Teilen bereits übermittelter Nachrichten zusammengesetzt ist; falls er die Struktur der Nachrichten auf Grund gemeinsamer Blöcke erkennt, hat er sogar eine gute Chance, daß die entstehende Nachricht sinnvoll ist. Bei Nachrichten mit festem Format, wie sie beispielsweise im elektronischen Zahlungsverkehr unter Banken üblich sind, hätte er eventuell sogar die Möglichkeit, zwei von ihm selbst initiierte Transaktionen zu identifizieren und zu seinem Vorteil zu manipulieren. Aber auch das bloße Einschleusen einer nicht als falsch zu erkennenden Nachricht etwa zu Sabotagezwecken kann bereits genügend Schaden anrichten.

Natürlich hat ein Angreifer bei einer guten Blockchiffre auch im ECB-Modus keine Chance, die Nachricht zu dechiffrieren oder gar den Schlüssel herauszufinden, aber wie wir gesehen haben, kann er sich bei gewissen Typen von Nachrichten doch einiges an Information verschaffen.

Man kann in der Kryptographie üblicherweise nicht davon ausgehen, daß ein Anwender über die Stärken und Schwächen des verwendeten Kryptosystems Bescheid weiß: Er verläßt sich darauf, daß das gekaufte oder von einem Experten eingerichtete System seine Geheimnisse zuverlässig schützt, egal worum es sich handelt. Daher sollte man den ECB-Modus im Normalfall nicht benutzen.

b) Cipher Block Chaining (CBC)

Hier wird die Nachricht $x_1, x_2, \ldots, x_n$ übermittelt als $y_1, y_2, \ldots, y_n$ mit

$$y_i = B(s, x_i \oplus y_{i-1}) .$$

Da es für $i = 1$ noch keinen Chiffretextblock y_0 gibt, muß dabei zusaätzlich zum Schlüssel noch ein Anfangsblock y_0 explizit festgelegt werden. Er muß nicht unbedingt geheimgehalten werden, sollte aber zwecks zusätzlicher Sicherheit möglichst für jede Verschlüsselung neu gewählt werden.

Unabhängig von der Wahl des Anfangsblock hängt bei CBC jeder übertragene Block y_i auch noch vom Vorgänger y_{i-1} ab; es ist daher nicht möglich, eine Nachricht durch Auslassen von Blöcken oder durch Zusammensetzen zweier existierender Nachrichten zu manipulieren ohne daß Blöcke verfälscht und damit unentschlüsselbar werden – was den Empfänger (hoffentlich) zum Nachfragen veranlaßt.

Ein weiterer Vorteil besteht darin, daß jeder übertragene Block y_i von *jedem* der Blöcke $x_1, x_2, \ldots, x_i$ abhängt; insbesondere hängt als der letzte Block y_N von jedem einzelnen Klartextblock ab. Falls also die übermittelte Nachricht noch elektronisch unterschrieben werden soll, reicht es, den Block y_N zu unterschreiben.

(Mit elektronischen Unterschriften werden wir uns im Zusammenhang mit der asymmetrischen Kryptographie beschäftigen. Wir werden dann

auch Verfahren kennenlernen, wie man auch bei anderen Übertragungsmodi oder gar der Übertragung von Klartext einen Block finden kann, der von der gesamten Nachricht abhängt. Einen solchen Block bezeichnet man als *message authentication code*, kurz MAC. Im Allgemeinen berechnet man ihn durch sogenannte sichere Hash-Verfahren; eine gute Blockchiffre im CBC-Modus liefert ein, wenn auch vergleichsweise aufwendiges, solches Verfahren.)

Die Abhängigkeit eines jeden Chiffreblocks von allen vorausgehenden Klartextblöcken hat nicht nur Vorteile: Sie führt auch dazu, daß Übertragungsfehler nicht nur einen Block betreffen. Tatsächlich führen sie aber bei CBC nur zur falschen Entschlüsselung zweier Blöcke: Die Entschlüsselungsfunktion ist bei CBC offenbar

$$x_i = B^{-1}(s, y_i) \oplus y_{i-1} ,$$

bereits $x_{i+2} = B^{-1}(s, y_{i+2}) \oplus y_{i+1}$ ist also von einen falsch übermittelten Block y_i nicht mehr betroffen.

Ein großes Problem beim ECB-Modus war, daß gleiche Nachrichten und auch gleiche Blöcke gleich übermittelt werden. Beim CBC-Modus ist dieses Problem zumindest insofern abgemildert, als gleiche Block durch das XOR mit dem vorangegangenen Chiffreblock verschieden chiffriert werden. Falls man allerdings den Anfangsblock y_0 konstant wählt – aus Sicht des Anwenders sicherlich die einfachste Lösung – werden identische Nachrichten weiterhin identisch chiffriert.

Die Sicherheit wird also auf jeden Fall erhöht, wenn für jede Übertragung ein neuer Anfangsblock benutzt wird. Dieser könnte beispielsweise ein Zufallsblock sein, der – damit ihn auch der Empfänger kennt – entweder unverschlüsselt oder ECB-verschlüsselt als erstes übertragen wird. Damit wird die zu übermittelnde Nachricht um einen Block verlängert, was im allgemeinen kein großes Problem ist – außer vielleicht in dem Fall, daß man sehr viele sehr kurze Nachrichten über eine teure oder stark kapazitätsbeschränkte Leitung übertragen muß.

Ein zufälliger Anfangsblock hilft noch nicht gegen das Problem, daß ein Angreifer einfach eine aufgefangene Nachricht ein zweites Mal in die Leitung einspielt. Da so etwas beispielsweise bei elektronischen

Finanztransfers unbedingt erkannt werden muß, enthalten entsprechende Nachrichten selbstverständlich eine eindeutige Buchungsnummer.

Auch in anderen Systemen ist es oft üblich, daß jede Nachricht ihre eindeutige Kennzeichnung hat, und das legt es nahe, zumindest in solchen Systemen entweder direkt diese Nachrichtennummer oder aber eine daraus abgeleitete Zahl (eine sogenannte *Nonce;* die Bezeichnung ist eine Kontraktion von *N*umber used *once*) zu verwenden. Da auch Nachrichtennummern Informationen enthalten, sollte diese Nummer zur Sicherheit mit der Blockchiffre verschlüsselt werden.

Ganz perfekt ist die Chiffre auch so noch nicht: Angenommen, der Chiffretextblock y_i ist gleich dem Block y_j. Dann können wir wie folgt argumentieren:

$$y_i = B(s, x_i \oplus y_{i-1}) \wedge y_j = B(s, x_j \oplus y_{j-1})$$
$$\Longrightarrow x_i \oplus y_{i-1} = x_j \oplus y_{j-1} \Longrightarrow x_i \oplus x_j = y_i \oplus y_j \,.$$

Somit läßt sich die Differenz $x_i \oplus x_j$ aus der Differenz der vorangegangenen Chiffretextblöcke $y_{i-1} \oplus y_{j-1}$ berechnen. Falls die Nachrichtenquelle eine ähnlich hohe Redundanz hat wie die deutsche Sprache, sollte diese Information ausreichen, um die beiden Blöcke (bis auf Reihenfolge) zu rekonstruieren.

Dies ist sicherlich ein Schwachpunkt, den man in der besten aller Welten gerne vermeiden würde; andererseits ist die Wahrscheinlichkeit, daß zwei gleiche Chiffretextblöcke auftreten, nicht sonderlich groß: Wenn wir davon ausgehen, daß sich Chiffretext im CBC-Modus wie eine Zufallsfolge verhält (was wahrscheinlich etwas zu optimistisch ist), liegt sie im Falle der Blocklänge N etwa bei $2^{-N/2}$. Bei einer 64-Bit-Blockchiffre wie DES heißt das, daß wir etwa $2^{32} = 4294967296$ oder rund 4,3 Milliarden Blöcke brauchen, bevor wir mit einer Wahrscheinlichkeit von mindestens 50% zwei gleiche Chiffretextblöcke finden. Bei einer Blockchiffre mit 128 Bit (was heute eigentlich Mindeststandard sein sollte, kommt man sogar auf $2^{64} \approx 1{,}8 \cdot 10^{19}$ oder rund 18 Trilliarden Blöcke. (Für Einzelheiten sei auf das Kapitel über sichere Hashverfahren verwiesen, wo wir das sogenannte *Geburtstagsparadoxon* genauer betrachten werden.)

Da wir in Wirklichkeit natürlich keine Zufallsfolge haben, dürften die tatsächlichen Wahrscheinlichkeiten wohl etwas größer sein, aber bei normalen Textdateien sollten sie weiterhin praktisch vernachlässigbar sein, und bei wirklich großen Dateien wie etwa Videofilmen sollte die Kenntnis einiger weniger einzelner Blöcke für einen Angreifer wohl nutzlos sein. Was bleibt, ist das Restrisiko, daß beispielsweise genau der eine Block, in dem ein besonders streng geheimzuhaltender Name oder Begriff steht zufälligerweise trotzdem genauso verschlüsselt wird wie ein anderer Block und damit einem ohne große Erfolgsaussichten auf genau dieses Restrisiko hoffenden Gegner bekannt wird – dies gehört zum unvermeidbaren Risiko eines jeden nicht absolut sicheren Kryptosystems.

Als Randbemerkung sollte erwähnt werden, daß obige Rechnung natürlich auch zeigt, daß verschiedene Klartextblöcke zu verschiedenen Chiffretextblöcken führen, falls die Chiffretextblöcke in den Vorgängerpositionen verschieden sind. Dies mag zwar auf den ersten Blick als nicht sehr informativ erscheinen, aber die Enigma wurde im zweiten Weltkrieg geknackt eben wegen der Beobachtung, daß sie nie einen Buchstaben durch sich selbst verschlüsselt. Bei einer Blockchiffre von 64 oder 128 Bit kann man mit so einer Information zwar sehr viel weniger anfangen, aber es handelt sich doch Information für den Gegner, von der wir nicht sicher sein können, was er damit anfangen kann.

c) Cipher Feedback (CFB)

Die nun folgenden Modi sind nützlich, wenn Daten in Echtzeit übertragen werden sollen, die kürzer sind als die Blocklänge; hier verwenden wir die Blockchiffre, um einen Schlüsselstrom zu erzeugen, der nach Art des *one time pad* verwendet wird. Der große Unterschied ist natürlich, daß die Entropie dieses Schlüsselstroms nur die des Schlüssels und des (ähnlich zu CBC verwendeten) Anfangsblocks ist: Unser guter alter Feind, der BAYESsche Gegner, hätte also keinerlei Schwierigkeiten, die Chiffre zu entschlüsseln. Unsere Hoffnung und der publik gewordene Teil der Erfahrung im Umgang mit Blockchiffren wie DES und AES beruht darauf, daß der Schlüsselstrom zu komplex ist für einen realen Gegner.

Bei CFB gehen wir davon aus, daß die Daten nicht als Blöcke anfallen, sondern in eventuell kleineren Einheiten zu k Bit. Typisch für Anwendungen ist der Wert $k = 8$, d.h. wir verschlüsseln einen Strom von Bytes, aber selbst der Fall $k = 1$, bei dem einzelne Bits verschlüsselt werden, kommt gelegentlich vor. Falls k kleiner ist als die Blocklänge N des Codes, ist dieser Modus also um den Faktor N/k langsamer als die bislang betrachteten Modi.

Auch hier gehen wir aus von einem Anfangsblock; er ist allerdings durch k fast vollständig festgelegt: In einem Register R, dessen Länge gleich der Blocklänge des verwendeten Codes ist, stehen rechts k Bit, zum Beispiel lauter Einsen, die restlichen Bits des Registers werden auf Null gesetzt. Sodann werden die *ersten* k Bit von $B(s, R)$ zu den ersten k Bit der Nachricht addiert und dies wird übertragen. Man beachte, daß das Verschlüsselungsergebnis nur für die Übertragung benutzt wird; der Inhalt des Registers behält seinen Wert.

In jedem der folgenden Schritt wird der Inhalt des Registers um k Bit (nichtzyklisch) nach links verschoben, und die k zuletzt übertragenen Bits werden am rechten Ende eingesetzt. Sodann werden die ersten k Bit des mit dem neuen Registerinhalt berechneten Blocks $B(s, R)$ zum nächsten Nachrichtenblock addiert und übertragen, *usw.*

Sofern die ersten k Bit, die ins Register geschrieben werden, konstant sind, werden gleiche Texte stets gleich verschlüsselt, und – was schlimmer ist – zum ersten Block der Nachricht wird stets derselbe Schlüssel addiert, so daß die statistischen Angriffe aus dem ersten Kapitel anwendbar sind. Falls k gleich der Blocklänge ist, könnte ein damit erfolgreicher Angreifer sogar den Wert $B(s, R)$ für den Anfangszustand des Registers rekonstruieren und, zumindest im Falle DES mit Hilfe von DES-Cracker oder einem ähnlichen Werkzeug den Schlüssel s ermitteln. Hier liefert also die Wahl eines deutlich unterhalb der Blocklänge liegenden Werts von k einen zusätzlichen Sicherheitsfaktor. Bei einer guten und zeitgemäßen Blockchiffre ist es natürlich unmöglich, aus einem Paar von Klar- und Chiffretextblöcken den Schlüssel zu rekonstruieren – es sei denn, man verfügt über die Rechenkraft des BAYESschen Gegners.

Auch in der praktischen Anwendung gibt es ein Problem, denn wie

bei CBC hängt wieder jedes übertragene Wort aus k Bit von allen Vorgängern ab. Aus Sicht des Empfängers allerdings hängt der Inhalt des Registers nur ab von den letzten r empfangenen Chiffretextblöcken, wobei r die kleinste ganze Zahl ist mit $rk \geq n$, denn nach r Übertragungen fällt jeder Chiffreblock wegen der zyklischen Verschiebung aus dem Register heraus. Ein Übertragungsfehler beeinflußt hier also insgesamt $r + 1$ Nachrichtenblöcke.

d) Output feedback (OFB)

Typische Anwendungen von Stromchiffren sind Satellitenübertragungen. Hier sind Bitfehler auf Grund atmosphärischer Störungen relativ häufig; obwohl sie natürlich durch fehlerkorrigierende Codes so weit wie möglich kompensiert werden, muß man doch immer wieder mit auch längerfristigen erhöhten Fehlerraten rechnen. Dabei ist die Eigenschaft des CFB-Modus, jeden Fehler gleich auf $r + 1$ Blöcke durchschlagen zu lassen, höchst unwillkommen.

Ein für solche Anwendungen nützlicher Modus ist *output feedback* (OFB). Auch dieser Modus erzeugt einen Schlüsselstrom mit Hilfe eines Registers R, allerdings hängt dessen Inhalt weder vom Klartext noch vom Chiffretext ab. Da der Schlüsselstrom zum Nachrichtenstrom addiert wird, betrifft daher ein Bitfehler bei der Übertragung hier nur ein einziges Bit.

Das Register R wird zu Beginn auf einen Anfangswert gesetzt. Im Gegensatz zu CFB wird das Register selbst in jedem Schritt verschlüsselt, sein Inhalt also durch $B(s, R)$ ersetzt. Danach werden die ersten k Bit zum Nachrichtenblock addiert, und vor dem nächsten Schritt wird das Register *zyklisch* um k Positionen nach links verschoben.

Bei dieser Vorgehensweise *muß* man natürlich für jede Übertragung einen neuen Anfangsblock und/oder Schlüssel wählen, denn ansonsten wird mehrfach derselbe Schlüsselstrom verwendet, ein Gegner kann also schon mit einer relativ kleinen Anzahl von Chiffretexten durch Häufigkeitsanalysen Informationen über den Klartext bekommen, die bis zur völligen Entschlüsselung führen können. Da die Blockchiffre hier nur zur Erzeugung eines Schlüsselstroms verwendet wird, ist die zu betrachtende Einheit aus Sicht des Kryptanalytikers kein Block, sondern

die kleinste Dateneinheit der Nachricht, typischerweise also ein Byte, so daß die Verfahren aus dem ersten Kapitel problemlos angewandt werden können. Außerdem muß darauf geachtet werden, daß der Schlüsselstrom natürlich periodisch ist. Die Periode ist zwar, abgesehen von einigen wenigen sogenannten *schwachen* Schlüsseln der Blockchiffre, sehr groß, aber je nach zu übertragendem Datenvolumen kann es trotzdem Probleme geben.

e) Counter mode (CTR)

Dieser Modus wird im Standard für DES nicht erwähnt, wurde aber 2001 vom NIST (dem *National Institute of Standards* der Vereinigten Staaten) als eine Methode zur Anwendung von Blockchiffren standardisiert.

Ausgangspunkt ist eine *Nonce,* d.h. eine Zahl a, die für genau eine Nachricht und danach nie wieder während der Gültigkeitsdauer des Schlüssels verwendet wird. Sie wird beispielsweise aus der Nummer oder dem Übertragungsdatum der Nachricht nach einem vorher definierten Verfahren erzeugt.

An diese Zahl wird wie bei OFB ein von der Nachricht unabhängiger Schlüsselstrom erzeugt, hier nach der Vorschrift

$$s_i = B(s, a||i)\,,$$

wobei $a||i$ für eine Vorschrift steht, wie die Blocknummer i hinter die Zahl a geschrieben wird. Konkret geht es also darum, daß a eine gewisse maximale Bitlänge hat, und die restlichen Bits werden für i reserviert.

Wie bei OFB *muß* auch hier natürlich sichergestellt sein, daß keine zwei Blöcke mit demselben s_i verschlüsselt werden, d.h. die Anzahl möglicher Werte für i muß größer sein als die maximale Länge einer zu übertragenen Nachricht. Bei 64 Bit-Chiffren kann dies den Wertebereich von i deutlich einschränken; bei einer Blocklänge von 128 Bit sollte es jedoch mit realistischen Nachrichten keine Probleme geben.

Die Verschlüsselung geschieht auch hier wie beim *one time pad,* d.h.

$$y_i = x_i \oplus s_i = x_i \oplus B(s, a||i)\,.$$

Auch hier muß wieder unbedingt sichergestellt werden, daß derselbe Schlüsselstrom nur einmal benutzt wird, d.h. die Zahl a darf auf keinen

Fall mehrfach benutzt werden, da sonst die Attacken aus dem ersten Kapitel greifen würden.

Sofern dies wirklich sichergestellt ist, dürfte CTR wohl der sicherste unter den hier diskutierten Modi sein.

§ 6: Literatur

Da DES rund 25 Jahre lang *das* Standardverfahren für ernsthafte symmetrische Verschlüsselung im zivilen Bereich war, ist er natürlich in praktisch jedem Lehrbuch der Kryptologie aus der damaligen Zeit ausführlich beschrieben, z.B. in

JAN C.A. VAN DER LUBBE: *Basic Methods of cryptography,* Cambridge University Press, 1998

oder

JOHANNES BUCHMANN: *Einführung in die Kryptographie,* Springer, ^{3}2003

Eine ausführliche Diskussion der Operationsmodi findet man unter anderem in

A.J. MENEZES, P.C. VAN OORSCHOT, S.A. VANSTONE: *Handbook of applied cryptography,* CRC Press 1997

sowie in

NIELS FERGUSON, BRUCE SCHNEIER: *Practical Cryptography,* Wiley, 2003

Speziellere Fragen sind außer in den bereits im Text zitierten Arbeiten und Büchern auch in fast jeder Konferenz über Kryptologie behandelt; insbesondere gilt dies für Tagungen wie *Crypto, Eurocrypt* und *Asiacrypt,* deren Proceeding jeweils in den Springer Lecture Notes in Computer Science erscheinen.

Kapitel 4
Das RSA-Verfahren

§ 1: New directions in cryptography

Bei allen bisher betrachteten Codes verläuft die Entschlüsselung entweder genauso oder zumindest sehr ähnlich wie die Verschlüsselung; insbesondere kann jeder, der eine Nachricht verschlüsseln kann, jede andere entsprechend verschlüsselte Nachricht auch entschlüsseln. Man bezeichnet diese Verfahren daher als *symmetrisch*.

Der Nachteil eines symmetrischen Verfahrens besteht darin, daß in einem Netzwerk jeder Teilnehmer mit jedem anderen einen Schlüssel vereinbaren muß. In militärischen Netzen war dies traditionellerweise so geregelt, daß das gesamte Netz denselben Schlüssel benutzte, der in einem Codebuch für jeden Tag im voraus festgelegt war; in kommerziellen Netzen wie beispielsweise einem Mobilfunknetz ist dies natürlich unmöglich.

1976 publizierten MARTIN HELLMAN, damals Assistenzprofessor an der Stanford University, und sein Forschungsassistent WHITFIELD DIFFIE eine Arbeit mit dem Titel *New directions in cryptography* (IEEE Trans. Inform. Theory **22**, 644–654; inzwischen auch im Netz zu finden), in der sie vorschlugen, den Vorgang der *Ver*schlüsselung und den der *Ent*schlüsselung völlig voneinander zu trennen: Es sei schließlich nicht notwendig, daß der Sender einer verschlüsselten Nachricht auch in der Lage sei, diese zu *ent*schlüsseln.

Der Vorteil eines solchen Verfahrens wäre, daß jeder potentielle Empfänger nur einen einzigen Schlüssel bräuchte und dennoch sicher sein

könnte, daß nur er selbst seine Post entschlüsseln kann. Der Schlüssel müßte nicht einmal geheimgehalten werden, da es ja nicht schadet, wenn jedermann Nachrichten *ver*schlüsseln kann. In einem Netzwerk mit n Teilnehmern bräuchte man also nur n Schlüssel, um jedem Teilnehmer zu erlauben, mit jeden anderen zu kommunizieren, und diese Schlüssel könnten sogar in einem öffentlichen Verzeichnis stehen. Bei einem symmetrischen Kryptosystem wäre der gleiche Zweck nur erreichbar mit $\frac{1}{2}n(n-1)$ Schlüsseln, die zudem noch durch ein sicheres Verfahren wie etwa ein persönliches Treffen oder durch vertrauenswürdige Boten ausgetauscht werden müßten.



BAILEY WHITFIELD DIFFIE wurde 1944 geboren. Erst im Alter von zehn Jahren lernte er lesen; im gleichen Jahr hielt eine Lehrerin an seiner New Yorker Grundschule einen Vortrag über Chiffren. Er ließ sich von seinem Vater alle verfügbare Literatur darüber besorgen, entschied sich dann 1961 aber doch für ein Mathematikstudium am MIT. Um einer Einberufung zu entgehen, arbeitete er nach seinem Bachelor bei Mitre; später, nachdem sein Interesse an der Kryptographie wieder erwacht war, kam er zu Martin Hellman nach Stanford, der ihn als Forschungsassistent einstellte. 1991–2009 arbeitete er als *chief security officer* bei Sun Microsystems, seit 2010 ist er bei ICANN für Sicherheit zuständig. `labs.oracle.com/people/mybio.php?uid=18607` .



MARTIN HELLMAN wurde 1945 in New York geboren. Er studierte Elektrotechnik zunächst bis zum Bachelor an der dortigen Universität; für das Studium zum Master und zur Promotion ging er nach Stanford. Nach kurzen Zwischenaufenthalten am Watson Research Center der IBM und am MIT wurde er 1971 Professor an der Stanford University. Seit 1996 ist er emeritiert, gibt aber immer noch Kurse, mit denen er Schüler für mathematische Probleme interessieren will. Seine home page ist unter `http://www-ee.stanford.edu/~hellman/` zu finden.

DIFFIE und HELLMAN machten nur sehr vage Andeutungen, wie so ein System mit öffentlichen Schritten aussehen könnte. Es ist zunächst einmal klar, daß ein solches System keinerlei Sicherheit gegen einen BAYESschen Gegner bieten kann, denn die Verschlüsselungsfunktion ist

eine bijektive Abbildung zwischen endlichen Mengen, und jeder, der die Funktion kennt, kann zumindest im Prinzip auch ihre Umkehrfunktion berechnen.

Wer im Gegensatz zum BAYESschen Gegner nur über begrenzte Ressourcen verfügt, kann diese Berechnung allerdings möglicherweise nicht mit realistischem Aufwand durchführen, und nur darauf beruht die Sicherheit eines Kryptosystems mit öffentlichen Schlüsseln. DIFFIE und HELLMAN bezeichnen eine Funktion, deren Umkehrfunktion nicht mit vertretbarem Aufwand berechnet werden kann, als *Einwegfunktion* und schlagen als Verschlüsselungsfunktion eine solche Einwegfunktion vor.

Damit hat man aber noch kein praktikables Kryptosystem, denn bei einer echten Einwegfunktion ist es auch für den legitimen Empfänger nicht möglich, seinen Posteingang zu entschlüsseln. DIFFIE und HELLMAN schlagen deshalb eine Einwegfunktion mit *Falltür* vor, wobei der legitime Empfänger zusätzlich zu seinem öffentlichen Schlüssel noch über einen geheimen Schlüssel verfügt, mit dem er (und nur er) diese Falltür öffnen kann.

Natürlich hängt alles davon ab, ob es solche Einwegfunktionen mit Falltür wirklich gibt. DIFFIE und HELLMAN gaben keine an, und unter den Experten gab es durchaus einige Skepsis bezüglich der Möglichkeit, solche Funktionen zu finden.

Tatsächlich existierten aber damals bereits Systeme, die auf solchen Funktionen beruhten, auch wenn sie nicht in der offenen Literatur dokumentiert waren: Die britische *Communications-Electronics Security Group* (CESG) hatte bereits Ende der sechziger Jahre damit begonnen, nach entsprechenden Verfahren zu suchen, um die Probleme des Militärs mit dem Schlüsselmanagement zu lösen, aufbauend auf (impraktikablen) Ansätzen von AT&T zur Sprachverschlüsselung während des zweiten Weltkriegs. Die Briten sprachen nicht von Kryptographie mit öffentlichen Schlüsseln, sondern von *nichtgeheimer Verschlüsselung*, aber das Prinzip war das gleiche.

Erste Ideen dazu sind in einer auf Januar 1970 datierten Arbeit von JAMES H. ELLIS zu finden, ein praktikables System in einer auf den 20. November 1973 datierten Arbeit von CLIFF C. COCKS. Wie im

Milieu üblich, gelangte nichts über diese Arbeiten an die Öffentlichkeit; erst 1997 veröffentlichten die *Goverment Communications Headquarters* (GCHQ), zu denen CESG gehört, einige Arbeiten aus der damaligen Zeit; eine Zeitlang waren sie auch auf dem Server `http://www.cesg.gov.uk/` zu finden, wo sie allerdings inzwischen anscheinend wieder verschwunden sind.

In der offenen Literatur erschien ein Jahr nach der Arbeit von DIFFIE und HELLMAN das erste Kryptosystem mit öffentlichen Schlüsseln: RON RIVEST, ADI SHAMIR und LEN ADLEMAN, damals alle drei am Massachussetts Institute of Technology, fanden nach rund vierzig erfolglosen Ansätzen 1977 schließlich jenes System, das heute nach ihren Anfangsbuchstaben mit RSA bezeichnet wird:

Das System wurde 1983 von der eigens dafür gegründeten Firma RSA Computer Security Inc. patentiert und mit großem kommerziellem Erfolg vermarktet. Das Patent lief zwar im September 2000 aus, die Firma ist aber weiterhin erfolgreich im Kryptobereich tätig; sie hatte beispielsweise auch einen Kandidaten für AES entwickelt, der es immerhin bis in die Endrunde schaffte.

RSA ist übrigens identisch mit dem von laut GCHQ von COCKS vorgeschlagenen System. Die Beschreibung durch RIVEST, SHAMIR und ADLEMAN erschien 1978 unter dem Titel *A method for obtaining digital signatures and public-key cryptosystems* in Comm. ACM **21**, 120–126.

§ 2: Die Grundidee des RSA-Verfahrens

a) Allgemeine Vorüberlegungen

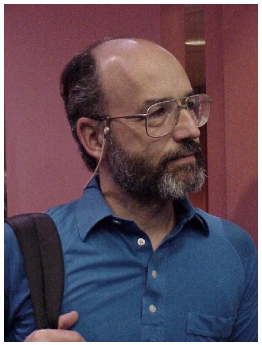
Die SHANNONschen Forderungen nach *Konfusion* und *Diffusion* müssen natürlich auch bei einer asymmetrischen Blockchiffre erfüllt sein. Bei den heute üblichen asymmetrischen Verfahren sind die verarbeiteten „Blöcke“ fast immer Zahlen aus $\mathbb{Z}/N = \{0, \ldots, N-1\}$ für eine hinreichend große natürliche Zahl N, und die Verschlüsselung ist ein Bijektion $f\colon \mathbb{Z}/N \to \mathbb{Z}/N$.

Von dieser Funktion erwarten wir drei Dinge:

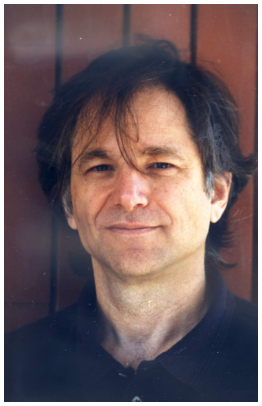
1. Sie muß einfach berechenbar sein



RONALD LINN RIVEST wurde 1947 in Schenectady im US-Bundesstaat New York geboren. Er studierte zunächst Mathematik an der Yale University, wo er 1969 seinen Bachelor bekam; danach studierte er in Stanford Informatik. Nach seiner Promotion 1974 wurde er Assistenzprofessor am Massachussetts Institute of Technology, wo er heute einen Lehrstuhl hat. Er arbeitet immer noch auf dem Gebiet der Kryptographie und entwickelte eine ganze Reihe weiterer Verfahren, auch symmetrische Verschlüsselungsalgorithmen und Hashverfahren. Er ist Koautor eines Lehrbuchs über Algorithmen. Seine home page ist //http://theory.lcs.mit.edu/∼rivest/ .



ADI SHAMIR wurde 1952 in Tel Aviv geboren. Er studierte zunächst Mathematik an der dortigen Universität; nach seinem Bachelor wechselte er ans Weizmann Institut, wo er 1975 seinen Master und 1977 die Promotion in Informatik erhielt. Nach einem Jahr als Postdoc an der Universität Warwick und drei Jahren am MIT kehrte er ans Weizmann Institut zurück, wo er bis heute Professor ist. Außer für RSA ist er bekannt sowohl für die Entwicklung weiterer Kryptoverfahren als auch für erfolgreiche Angriffe gegen Kryptoverfahren. Er schlug auch einen optischen Spezialrechner zur Faktorisierung großer Zahlen vor. Seine home page ist erreichbar unter http://www.wisdom.weizmann.ac.il/math/profile/scientists/shamir-profile.html



LEONARD ADLEMAN wurde 1945 in San Francisco geboren. Er studierte in Berkeley, wo er 1968 einen BS in Mathematik und 1976 einen PhD in Informatik erhielt. Thema seiner Dissertation waren zahlentheoretische Algorithmen und ihre Komplexität. Von 1976 bis 1980 war er an der mathematischen Fakultät des MIT; seit 1980 arbeitet er an der University of Southern California in Los Angelos. Seine Arbeiten beschäftigen sich mit Zahlentheorie, Kryptographie und Molekularbiologie. Er führte nicht nur 1994 die erste Berechnung mit einem „DNS-Computer“ durch, sondern arbeitete auch auf dem Gebiet der Aidsforschung. Heute hat er einen Lehrstuhl für Informatik und Molekularbiologie. http://www.usc.edu/dept/molecular-science/fm-adleman.htm

2. Ihre Umkehrfunktion muß einfach berechenbar sein.
3. Aus der bloßen Kenntnis von f darf man nicht auf die Umkehrfunktion schließen können.

Forderung 3 ist, mathematisch gesehen, natürlich unerfüllbar: f ist eine bijektive Abbildung zwischen endlichen Mengen, und damit ist ihre Umkehrfunktion eindeutig festgelegt. Andererseits muß man auch beim Entschlüsseln einer symmetrischen Blockchiffre im allgemeinen (z.B. wenn die Nachricht aus Klartext in einer natürlichen Sprache besteht) „nur“ alle Schlüssel durchprobieren. Wie so häufig in der Kryptographie müssen wir uns wieder einmal mit *praktischer* Sicherheit zufrieden geben, wobei *praktisch* nur bedeutet, daß wir kein Verfahren kennen, mit dem man die Funktion mit vertretbarem Aufwand umkehren könnte.

Für kleine Werte von N kann man sich die Umkehrfunktion von f einfach dadurch verschaffen, daß man zur Berechnung von $f^{-1}(y)$ für jedes $x \in \mathbb{Z}/N$ ausprobiert, ob $f(x) = y$ ist. Eine notwendige Bedingung für praktische Sicherheit ist daher, daß N hierfür zu groß sein muß. Wenn wir mit den Sicherheitsanforderungen an heutige symmetrische Blockchiffren vergleichen, heißt das konkret, daß $N \geq 2^{128}$ sein sollte. Tatsächlich müssen wir jedoch oft mit erheblich größeren Werten von N arbeiten, da f für die gängigen Verfahren eine einfache mathematische Struktur hat, so daß es bessere Ansätze zur Berechnung von f^{-1} gibt als das Durchprobieren aller potentieller Urbilder.

Für stetige, womöglich gar monotone Funktionen ist die Berechnung der Umkehrfunktion ziemlich problemlos; was wir benötigen ist also eine diskrete Funktion mit möglichst konfus aussehendem Graphen. Dazu bietet sich etwa die *modulo*-Funktion an: Für eine ganze Zahlen n und eine natürliche Zahl m ist bekanntlich n modulo m, in Zeichen n mod m, der Divisionsrest bei Division von n durch m; insbesondere ist also stets $0 \leq n \bmod m \leq m-1$. Fast alle asymmetrischen Kryptoverfahren hängen in der einen oder anderen Weise ab von dieser Funktion.

Generell gilt, daß asymmetrische Verfahren in erster Linie auf mathematischen Problemen und Sätzen beruhen, jedenfalls in viel stärkerem Maße als symmetrische. Etwa überspitzt beginnt NEAL KOBLITZ, einer

der Pioniere der Kryptographie mit elliptischen Kurven, einen Übersichtsartikel daher auch mit den Worten:

> During the first six thousand years – until the invention of public key in the 1970s – the mathematics used in cryptography was generally not very interesting. . . . Indeed, mathematicians looking at cryptography in those years might have found justification for Paul Halmos' infamous title "Applied Mathematics is Bad Mathematics". (NEAL KOBLITZ: *The Uneasy Relationship between Mathematics and Cryptography*, Notices of the American Mathematical Society, September 2007, S. 972–979; *siehe auch* `http://www.ams.org/notices/200708/index.html` .)

Im Umkehrschluß folgt, daß wir uns nun etwas mehr mit Mathematik beschäftigen müssen, zunächst vor allem mit einigen Grundlagen der Zahlentheorie.

b) Modulararithmetik

Die gängigen aus der Analysis bekannten bijektiven Funktionen haben allesamt Umkehrfunktionen, deren Berechnung einen ähnlichen Aufwand erfordert wie die der Funktion selbst; sie sind also nicht als Einwegfunktionen geeignet. Das liegt hauptsächlich daran, daß diese Funktionen stetig und über weite Teile auch monoton sind; eine echte Einwegfunktion sollte schon vom ersten Eindruck her deutlich „wilder" aussehen.

Die heute praktisch eingesetzten asymmetrischen Kryptoverfahren erreichen diese „Wildheit" allesamt durch den Übergang zu Divisionsresten: Ist $a \in \mathbb{Z}$ eine ganze und $N \in \mathbb{N}$ eine natürliche Zahl, so bezeichnen wir mit $a \bmod N \in \{0, 1, \ldots, N-1\}$ den Rest bei der Division von a durch N; falls a und b beide denselben Divisionsrest haben, schreiben wir kurz

$$a \equiv b \bmod N$$

und sagen, a sei kongruent b modulo N. Das ist offensichtlich genau dann der Fall, wenn die Differenz $b - a$ durch N teilbar ist.

Lemma: Der Übergang zu Divisionsresten ist verträglich mit der Addition, Subtraktion und Multiplikation; ist also $a \equiv b \bmod N$ und

$c \equiv d \bmod N$, so ist auch

$$a \pm c \equiv b \pm d \bmod N \quad \text{und} \quad a \cdot c \equiv b \cdot d \bmod N\,.$$

Für jede natürliche Zahl e ist außerdem $a^e \equiv b^e \bmod N$.

Beweis: Da $a \equiv b \bmod N$ ist, ist die Differenz $b - a$ durch N teilbar, läßt sich also in der Form $b - a = Ne$ schreiben mit einer ganzen Zahl e; entsprechend ist $d - c = Nf$ mit $f \in \mathbb{Z}$. Damit ist

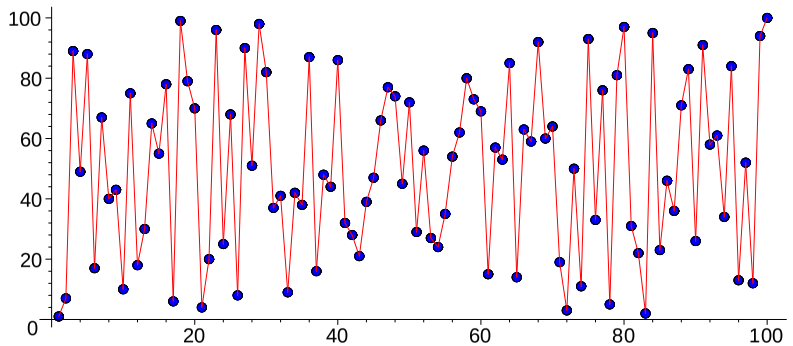
$$(b \pm d) - (a \pm b) = (a + Ne) \pm (b + Nf) - (a \pm b) = N(e \pm f)$$

durch N teilbar und genauso auch

$$bd - ac = (a + Ne)(b + Nf) - ab = N(eb + af) + N^2ef\,.$$

Dies beweist die ersten drei Behauptungen; die letzte folgt durch vollständige Induktion aus der Verträglichkeit der Kongruenz modulo N mit der Multiplikation. ∎

Die Funktion $x \mapsto x \bmod N$ ist natürlich nicht als Einwegfunktion geeignet; auf den Intervallen, auf denen sie bijektiv ist, ist sie stückweise linear und damit leicht umkehrbar. Wir können sie aber schachteln mit einer anderen Funktion, zum Beispiel einer Potenzfunktion $x \mapsto x^e$. Wie das Bild der Funktion $x \mapsto x^9 \bmod 101$ zeigt, können wir auf diese Weise zumindest recht wild aussehende Graphen bekommen.



Die Funktion $x \mapsto x^9 \bmod 101$

Es gibt allerdings keinen Grund, warum die Abbildung

$$f\colon \begin{cases} \mathbb{Z}/N \to \mathbb{Z}/N \\ \quad x \mapsto x^e \bmod N \end{cases}$$

für beliebige Werte von N und e bijektiv sein sollte. In der Tat ist die Abbildung etwa für $e = 2$ und ungerades $N > 1$ mit Sicherheit nicht injektiv, da dann stets

$f(N-x) = (N-x)^e \bmod N = (-x)^e \bmod N = (-1)^e x^e \bmod N = x^e \bmod N = f(x)$

ist. Wir müssen also einschränkende Bedingungen an die Zahlen N und e stellen.

c) Potenzfunktionen modulo einer Primzahl

Als erstes beschränken wir uns auf den Fall, daß $N = p$ eine Primzahl ist. Für $p = 2$ ist $\mathbb{Z}/2 = \{0, 1\}$ und φ ist für jeden Exponenten e einfach die Identität; daher ist offensichtlich nur der Fall einer ungeraden Primzahl interessant. Wie der folgende Satz zeigt, gibt es auch hier Exponenten, für die wir nur die identische Abbildung bekommen:

Kleiner Satz von Fermat: Für jedes $a \in \mathbb{Z}$ und jede Primzahl p ist

$$a^p \equiv a \bmod p\,;$$

ist a nicht durch p teilbar, gilt auch $a^{p-1} \equiv 1 \bmod p$.

Beweis: Wir betrachten zunächst nur nichtnegative Werte von a und beweisen die erste Aussage dafür durch vollständige Induktion:

Für $a = 0$ ist $0^p = 0$, also erst recht kongruent Null modulo p; genauso ist für $a = 1$ auch $a^p = 1$.

Für $a > 1$ schreiben wir

$$a^p = \big((a-1)+1\big)^p = \sum_{i=0}^{p} \binom{p}{i} (a-1)^i \quad \text{mit} \quad \binom{p}{i} = \frac{p!}{i!\,(p-i)!}\,.$$

Falls $1 \le i \le p-1$, ist der Nenner von $\binom{p}{i}$ nicht durch p teilbar, wohl aber der Zähler. Somit ist auch $\binom{p}{i}$ durch p teilbar, also kongruent Null modulo p. Damit ist

$$a^p \equiv \binom{p}{0}(a-1)^0 + \binom{p}{p}(a-1)^p = 1 + (a-1) = a \bmod p$$

nach Induktionsannahme.

Dies beweist die erste Aussage für $a \ge 0$. Für $a < 0$ ist im Falle $p = 2$ sowohl $-a \equiv a \bmod 2$ als auch $a^p = (-a)^p$; für ungerades p ist $(-a)^p = -(a^p)$, so daß die Behauptung in beiden Fällen folgt.

Zum Beweis der zweiten Behauptung beachten wir, daß

$$a^p - a = a\,(a^{p-1} - 1)$$

wie wir gerade bewiesen haben durch p teilbar ist. Falls a nicht durch p teilbar ist, muß also $a^{p-1} - 1$ durch p teilbar sein, und genau das ist die Behauptung. ∎



Der französische Mathematiker PIERRE DE FERMAT (1601–1665) wurde in Beaumont-de-Lomagne im Departement Tarn et Garonne geboren. Bekannt ist er heutzutage vor allem für seine 1994 von ANDREW WILES bewiesene Vermutung, wonach die Gleichung $x^n + y^n = z^n$ für $n \ge 3$ keine ganzzahlige Lösung mit $xyz \neq 0$ hat. Dieser „große“ Satzes von FERMAT, von dem FERMAT lediglich in einer Randnotiz behauptete, daß er ihn beweisen könne, erklärt den Namen der obigen Aussage. Obwohl FERMAT sich sein Leben lang sehr mit Mathematik beschäftigte und wesentliche Beiträge zur Zahlentheorie, Wahrscheinlichkeitstheorie und Analysis lieferte, war er hauptberuflich Jurist.

Der kleine Satz von FERMAT liefert auch einen Ansatz, wie wir gelegentlich eine Umkehrfunktion von $x \mapsto x^e \bmod p$ finden können: Offensichtlich ist für jede ganze Zahl k und jedes a auch

$$a^{1+k(p-1)} = a \cdot \left(a^{p-1}\right)^k \equiv a \bmod p\,,$$

denn ist a durch p teilbar, sind die rechte wie auch die linke Seite durch p teilbar, also kongruent Null modulo p, und andernfalls ist $a^{p-1} \equiv 1 \bmod p$, so daß der zweite Faktor des mittleren Terms modulo p eine Eins ist.

Falls wir also eine natürliche Zahl d finden können, für die gilt

$$de = 1 + k(p-1) \quad \text{mit} \quad k \in \mathbb{Z}\,,$$

so ist für alle $a \in \mathbb{Z}/p$

$$\left(a^e\right)^d = a^{de} \equiv a \bmod p\,,$$

die Abbildung $y \mapsto y^d \bmod p$ ist also invers zu $x \mapsto x^e \bmod p$.

Wenn wir eine solche Zahl d finden können, ist $1 = de - k(p-1)$; daher dürfen die Zahlen e und $p-1$ keinen gemeinsamen Teiler haben, denn der müßte sonst ja auch die Eins teilen.

Diese Bedingung ist bereits hinreichend: Für teilerfremde natürliche Zahlen e und $p-1$ gibt es stets solche Zahlen d und k. In der Tat gilt:

Satz: Zu zwei natürlichen Zahlen n, m gibt es stets natürliche Zahlen a, b derart, daß $an - bm$ gleich dem größten gemeinsamen Teiler t von n und m ist. Diese Zahlen (wie auch t selbst) lassen sich effizient mit Hilfe des erweiterten EUKLIDischen Algorithmus berechnen.

Da dieser Satz einigen Hörern bereits bekannt sein dürfte, ist sein Beweis zum besseren Überlesen separat im nächsten Abschnitt dargestellt.

Wenn wir diesen Satz annehmen, können wir als Fazit dieses Abschnitts zusammenfassen:

Ist p eine Primzahl und e eine zu $p-1$ teilerfremde natürliche Zahl, so gibt es eine einfach berechenbare natürliche Zahl d derart, daß die beiden Abbildungen

$$f\colon \begin{cases} \mathbb{Z}/p \to \mathbb{Z}/p \\ \quad x \mapsto x^e \bmod p \end{cases} \quad \text{und} \quad g\colon \begin{cases} \mathbb{Z}/p \to \mathbb{Z}/p \\ \quad x \mapsto x^d \bmod p \end{cases}$$

zueinander invers sind. Insbesondere sind f und g dann bijektiv; f kann als Verschlüsselungsfunktion verwendet werden und g als Entschlüsselungsfunktion. Die ersten beiden Forderungen, die wir in Abschnitt *a)* an eine asymmetrische Verschlüsselungsfunktion gestellt haben, sind damit erfüllt. Die dritte Forderung ist allerdings ganz eklatant verletzt: Wer f kennt, kennt insbesondere die Zahlen p und e, und daraus kann er das für die Entschlüsselung benötigte d nach obigem Satz effizient berechnet. Somit läßt sich f höchstens als symmetrische Verschlüsselungsfunktion benutzen, bei der die Parameter p und e als Schlüsselinformation geheimgehalten werden. Da eine Verschlüsselung nach Triple-DES oder gar AES deutlich schneller geht, wird dies kaum angewandt; lediglich für ganz spezielle Anwendungen wie Skat oder Poker per Telephon *bzw.*

Internet nutzt man aus, daß f im Gegensatz zu praktisch allen gängigen symmetrischen Kryptoverfahren ein Homomorphismus bezüglich der Multiplikation ist.

Auf der Suche nach einem asymmetrischen Kryptoverfahren mit Verschlüsselungsfunktion $x \mapsto x^e \bmod N$ können wir uns also nicht auf den Fall beschränken, daß N eine Primzahl ist. Im übernächsten Abschnitt werden wir uns überlegen, wie sich die Situation verändert, wenn N Produkt zweier Primzahlen ist. Zunächst folgt aber der angekündigte Anhang zu diesem Abschnitt.

d) Der erweiterte Euklidische Algorithmus

Hier geht es nur um den Beweis des letzten Satzes aus dem vorigen Abschnitt; wer damit vertraut ist, kann weiterblättern zum nächsten Abschnitt.

Beginnen wir mit dem einfachsten Fall, für den der Algorithmus schon als Proposition zwei im siebten Buch der Elemente EUKLIDs zu finden ist: Wir suchen den größten gemeinsamen Teiler zweier nichtnegativer ganzer Zahlen a und b, d.h. die größte ganze Zahl d, die sowohl a als auch b teilt. Für $a = b = 0$ gibt es kein *größtes* solches d; hier setzen wir $d = 0$. Wir schreiben kurz $d = \mathrm{ggT}(a, b)$.

Grundidee des EUKLIDischen Algorithmus ist die Anwendung der Division mit Rest: Für je zwei natürliche Zahlen x und y gibt es nichtnegative ganze Zahlen q und r, so daß $x = qy + r$ und $0 \le r < y$ ist. Alsdann ist $\mathrm{ggT}(x, y) = \mathrm{ggT}(y, r)$, denn wegen der beiden Gleichungen $x = qy + r$ und $r = x - qy$ teilt jeder gemeinsame Teiler von x und y auch r, und jeder gemeinsame Teiler von y und r teilt auch x. Da außerdem offensichtlich für alle $x \in \mathbb{N}_0$ der ggT von x und Null gleich x ist, können wir den ggT leicht rekursiv berechnen, indem wir die Regel $\mathrm{ggT}(x, y) = \mathrm{ggT}(y, r)$ so lange anwenden, bis $r = 0$ und damit der ggT gleich y ist. Wer Scheme/Racket oder einen anderen LISP-Dialekt kennt, kann den Algorithmus damit kurz und knapp als Einzeiler formulieren:

```
(define (ggT x y) (if (= y 0) x (ggT y (remainder x y))))
```

In mathematischer Sprechweise bedeutet das:

Schritt 0: Setze $r_0 = x$ und $r_1 = y$

Schritt i, $i \geq 1$**:** Falls $r_i = 0$ ist, endet der Algorithmus mit dem Ergebnis $\text{ggT}(x, y) = r_{i-1}$; andernfalls dividiere man r_{i-1} mit Rest durch r_i und bezeichne den Divisionsrest mit r_{i+1}.

Der Algorithmus bricht ab, da r_i (*bzw.* das zweite Argument y in der Scheme-Formulierung) in jedem Rekursionsschritt kleiner wird, aber stets eine nichtnegative ganze Zahl ist; nach endlich vielen Schritten muß es also Null sein, und der Algorithmus bricht ab. Die Korrektheit des Ergebnisses ist auch klar, denn aus der Gleichung

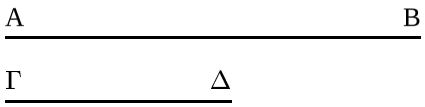
$$\text{ggT}(x, y) = \text{ggT}(y, x \bmod y)$$

folgt, daß in jedem Schritt $\text{ggT}(r_{i-1}, r_i) = \text{ggT}(x, y)$ ist.

Zum Vergleich sei hier noch EUKLIDs Beschreibung seines (wahrscheinlich schon mindestens 150 Jahre früher bereits den Pythagoräern bekannten) Algorithmus angegeben. In Proposition 2 des siebten Buchs seiner Elemente steht (in der Übersetzung von CLEMENS THAER für *Ostwalds Klassiker der exakten Wissenschaften*, Band 235):

Zu zwei gegebenen Zahlen, die nicht prim gegeneinander sind, ihr größtes gemeinsames Maß zu finden.

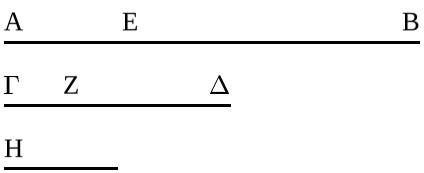
Die zwei gegebenen Zahlen, die nicht prim, gegeneinander sind, seien AB, ΓΔ. Man soll das größte gemeinsame Maß von AB, ΓΔ finden.



Wenn ΓΔ hier AB mißt – sich selbst mißt es auch – dann ist ΓΔ gemeinsames Maß von ΓΔ, AB. Und es ist klar, daß es auch das größte ist, denn keine Zahl größer ΓΔ kann ΓΔ messen.

Wenn ΓΔ aber AB nicht mißt, und man nimmt bei AB, ΓΔ abwechselnd immer das kleinere vom größeren weg, dann muß (schließlich) eine Zahl übrig bleiben, die die vorangehende mißt. Die Einheit kann nämlich nicht übrig bleiben; sonst müßten AB, ΓΔ gegeneinander prim sein, gegen die Voraussetzung. Also muß eine Zahl übrig bleiben, die die vorangehende

mißt. ΓΔ lasse, indem es BE mißt, EA, kleiner als sich selbst übrig; und EA lasse, indem es ΔZ mißt, ZΓ, kleiner als sich selbst übrig; und ΓZ messe AE.



Da ΓZ AE mißt und AE ΔZ, muß ΓZ auch ΔZ messen; es mißt aber auch sich selbst, muß also auch das Ganze ΓΔ messen. ΓΔ mißt aber BE; also mißt ΓZ auch BE; es mißt aber auch EA, muß also auch das Ganze BA messen. Und es mißt auch ΓΔ; ΓZ mißt also AB und ΓΔ; also ist ΓZ gemeinsames Maß von AB, ΓΔ. Ich behaupte, daß es auch das größte ist. Wäre nämlich ΓZ nicht das größte gemeinsame Maß von AB, ΓΔ, so müßte irgendeine Zahl größer ΓZ die Zahlen AB und ΓΔ messen. Dies geschehe; die Zahl sei H. Da H dann ΓΔ mäße und ΓΔ BE mißt, mäße H auch BE; es soll aber auch das Ganze BA messen, müßte also auch den Rest AE messen. AE mißt aber ΔZ; also müßte H auch ΔZ messen; es soll aber auch das Ganze ΔΓ messen, müßte also auch den Rest ΓZ messen, als größere Zahl die kleinere; dies ist unmöglich. Also kann keine Zahl größer ΓZ die Zahlen AB und ΓΔ messen; ΓZ ist also das größte gemeinsame Maß von AB, ΓΔ; dies hatte man beweisen sollen.



Es ist nicht ganz sicher, ob EUKLID wirklich gelebt hat; das nebenstehende Bild aus dem 18. Jahrhundert ist mit Sicherheit reine Phantasie. EUKLID ist vor allem bekannt als Autor der *Elemente*, in denen er die Geometrie seiner Zeit systematisch darstellte und (in gewisser Weise) auf wenige Definitionen sowie die berühmten fünf Postulate zurückführte. Diese Elemente entstanden um 300 v. Chr. und waren zwar nicht der erste, aber doch der erfolgreichste Versuch einer solchen Zusammenfassung. EUKLID arbeitete wohl am Museion in Alexandrien; außer den Elementen schrieb er auch ein Buch über Optik und weitere, teilweise verschollene Bücher.

Der *erweiterte* EUKLIDische Algorithmus war EUKLID selbst mit ziemlicher Sicherheit nicht bekannt; hier handelt es sich um eine auf dem

Grundalgorithmus beruhende und meist nach dem französischen Mathematiker ETIENNE BÉZOUT (1730–1783) benannte Identität, die dieser 1766 in einem Lehrbuch beschrieb (und auf Polynome verallgemeinerte). Für Zahlen ist diese Erweiterung jedoch bereits 1624 zu finden in der zweiten Auflage des Buchs *Problèmes plaisants et délectables qui se fonts par les nombres* von BACHET DE MÉZIRIAC.



CLAUDE GASPAR BACHET SIEUR DE MÉZIRIAC (1581-1638) verbrachte den größten Teil seines Lebens in seinem Geburtsort Bourg-en-Bresse. Er studierte zwar bei den Jesuiten in Lyon und Milano und trat 1601 in den Orden ein, trat aber bereits 1602 wegen Krankheit wieder aus und kehrte nach Bourg zurück. Sein Buch erschien erstmalig 1612, zuletzt 1959. Am bekanntesten ist BACHET für seine lateinische Übersetzung der *Arithmetika* von DIOPHANTOS. In einem Exemplar davon schrieb FERMAT seine Vermutung an den Rand. Auch Gedichte von BACHET sind erhalten. 1635 wurde er Mitglied der französischen Akademie der Wissenschaften.



ETIENNE BÉZOUT (1730-1783) wurde in Nemours in der Ile-de-France geboren, wo seine Vorfahren Magistrate waren. Er ging stattdessen an die Akademie der Wissenschaften; seine Hauptbeschäftigung war die Zusammenstellung von Lehrbüchern für die Militärausbildung. Im 1766 erschienenen dritten Band (von vier) seines *Cours de Mathématiques à l'usage des Gardes du Pavillon et de la Marine* ist die Identität von BÉZOUT dargestellt. Seine Bücher waren so erfolgreich, daß sie ins Englische übersetzt und z.B. in Harvard als Lehrbücher benutzt wurden. Heute ist er vor allem bekannt durch seinen Beweis, daß sich zwei Kurven der Grade n und m in höchstens nm Punkten schneiden können.

Die Gleichung $u = qv + r$ zur Division mit Rest läßt sich auch umschreiben als $r = u - qv$; der Divisionsrest ist also eine ganzzahlige Linearkombination des Dividenden u und des Divisors v. Falls sich diese wiederum als Linearkombination der beiden Ausgangszahlen x und y darstellen lassen, erhalten wir eine entsprechende Darstellung für r:

$$u = ax + by \quad \text{und} \quad v = cx + dy \quad \Longrightarrow r = (a - qc)x + (b - qd)z\,.$$

Wir können also ausgehen von den Darstellungen

$$x = 1 \cdot x + 0 \cdot y \quad \text{und} \quad y = 0 \cdot x + 1 \cdot y\,,$$

bei jeder Division im EUKLIDischen Algorithmus den Divisionsrest als ganzzahlige Linearkombination von x und y darstellen und damit auch den ggT als den letzten nichtverschwindenen solchen Rest.

Dies führt zu folgendem Algorithmus:

Schritt 0: Setze $r_0 = a$, $r_1 = b$, $\alpha_0 = \beta_1 = 1$ und $\alpha_1 = \beta_0 = 0$. Mit $i = 1$ ist dann

$$r_{i-1} = \alpha_{i-1}a + \beta_{i-1}b \quad \text{und} \quad r_i = \alpha_i a + \beta_i b\,.$$

Diese Relationen bleiben in jedem der folgenden Schritte erhalten:

Schritt i, $i \geq 1$**:** Falls $r_i = 0$ ist, endet der Algorithmus mit

$$\text{ggT}(a, b) = r_{i-1} = \alpha_{i-1}a + \beta_{i-1}b\,.$$

Andernfalls dividiere man r_{i-1} mit Rest durch r_i mit dem Ergebnis

$$r_{i-1} = q_i r_i + r_{i+1}\,.$$

Dann ist

$$\begin{aligned} r_{i+1} = -q_i r_i + r_{i-1} &= -q_i(\alpha_i a + \beta_i b) + (\alpha_{i-1}a + \beta_{i-1}b) \\ &= (\alpha_{i-1} - q_i\alpha_i)a + (\beta_{i-1} - q_i\beta_i)b\,; \end{aligned}$$

man setze also

$$\alpha_{i+1} = \alpha_{i-1} - q_i\alpha_i \quad \text{und} \quad \beta_{i+1} = \beta_{i-1} - q_i\beta_i\,.$$

Genau wie oben folgt, daß der Algorithmus für alle natürlichen Zahlen a und b endet und daß am Ende der richtige ggT berechnet wird; außerdem sind die α_i und β_i so definiert, daß in jedem Schritt $r_i = \alpha_i a + \beta_i b$ ist, insbesondere ist also im letzten Schritt der ggT als Linearkombination der Ausgangszahlen dargestellt. Da er kleiner ist als mindestens eine der beiden Zahlen, können nicht beide Koeffizienten positiv sein. Falls der erste positiv und der zweite negativ ist, haben wir den Satz aus dem vorigen Abschnitt bewiesen; andernfalls addieren wir so lange die Gleichung $ba - ab = 0$, bis dies der Fall ist.

Als Beispiel wollen wir den ggT von 200 und 148 als Linearkombination darstellen. Im nullten Schritt haben wir 200 und 148 als die trivialen Linearkombinationen

$$200 = 1 \cdot 200 + 0 \cdot 148 \quad \text{und} \quad 148 = 0 \cdot 200 + 1 \cdot 148\,.$$

Im ersten Schritt dividieren wir, da 148 nicht verschwindet, 200 mit Rest durch 148:

$$200 = 1 \cdot 148 + 52 \Longrightarrow 52 = 1 \cdot 200 - 1 \cdot 148$$

Da auch $52 \neq 0$, dividieren wir im zweiten Schritt 148 durch 52 mit Ergebnis $148 = 2 \cdot 52 + 44$, d.h.

$$44 = 148 - 2 \cdot (1 \cdot 200 - 1 \cdot 148) = 3 \cdot 148 - 2 \cdot 200$$

Auch $44 \neq 0$, wir dividieren also weiter: $52 = 1 \cdot 44 + 8$ und

$$\begin{aligned} 8 &= 52 - 44 = (1 \cdot 200 - 1 \cdot 148) - (3 \cdot 148 - 2 \cdot 200) \\ &= 3 \cdot 200 - 4 \cdot 148\,. \end{aligned}$$

Im nächsten Schritt erhalten wir $44 = 5 \cdot 8 + 4$ und

$$\begin{aligned} 4 &= 44 - 5 \cdot 8 = (3 \cdot 148 - 2 \cdot 200) - 5 \cdot (3 \cdot 200 - 4 \cdot 148) \\ &= 23 \cdot 148 - 17 \cdot 200\,. \end{aligned}$$

Bei der Division von acht durch vier schließlich erhalten wir Divisionsrest Null; damit ist vier der ggT von 148 und 200 und kann in der angegebenen Weise linear kombiniert werden.

e) Die RSA-Verschlüsselungsfunktion

Wie angekündigt, wollen wir nun als neuen Kandidaten für eine asymmetrische Verschlüsselungsfunktion die Funktion

$$f\colon \begin{cases} \mathbb{Z}/N \to \mathbb{Z}/N \\ \quad x \mapsto x^e \bmod N \end{cases} \qquad \text{mit} \quad N = pq$$

betrachten, wobei p und q zwei verschiedene Primzahlen sind.

Falls die ganze Zahl a teilerfremd zu N ist, kann sie weder ein Vielfaches von p noch eines von q sein; deshalb ist nach dem kleinen Satz von Fermat aus Abschnitt *c)*

$$a^{p-1} \equiv 1 \bmod p \quad \text{und} \quad a^{q-1} \equiv 1 \bmod q\,.$$

Bilden wir links die $(q-1)$-te und rechts die $(p-1)$-te Potenz, sehen wir, daß auch gilt

$$a^{(p-1)(q-1)} \equiv 1 \bmod p \quad \text{und} \quad a^{(p-1)(q-1)} \equiv 1 \bmod q\,.$$

Somit ist $a^{(p-1)(q-1)} - 1$ sowohl durch p als auch durch q teilbar, also auch durch $N = pq$. Daher ist

$$a^{(p-1)(q-1)} \equiv 1 \bmod N\,.$$

Wie in Abschnitt *c)* folgt daraus sofort, daß für jedes solche a auch gilt

$$a^{1+k(p-1)(q-1)} \equiv a \bmod N \quad \text{für alle} \quad k \in \mathbb{Z}\,.$$

Letztere Formel gilt tatsächlich sogar für alle $a \in \mathbb{Z}$, denn wie wir aus Abschnitt *c)* wissen, ist für beliebige ganze Zahlen $a, u, v \in \mathbb{Z}$

$$a^{1+u(p-1)} \equiv a \bmod p \quad \text{und} \quad a^{1+v(q-1)} \equiv a \bmod q\,.$$

Setzen wir speziell $u = k(q-1)$ und $v = k(p-1)$, ist also insbesondere

$$a^{1+k(p-1)(q-1)} \equiv a \bmod p \quad \text{und} \quad a^{1+k(p-1)(q-1)} \equiv a \bmod q\,.$$

Damit ist dann aber auch

$$a^{1+k(p-1)(q-1)} \equiv a \bmod N\,.$$

Nachdem wir das wissen, können wir genauso argumentieren wie in Abschnitt *c):* Wenn e teilerfremd ist zu $(p-1)(q-1)$, liefert uns der erweiterte EUKLIDische Algorithmus natürliche Zahlen $d, k \in \mathbb{N}$, so daß

$$1 = de - k(p-1)(q-1) \quad \text{und damit} \quad \left(a^e\right)^d \equiv a \bmod N\,.$$

Damit haben wir auch in diesem Fall eine einfach berechenbare Umkehrfunktion zu f, nämlich die entsprechende Exponentiation modulo N mit Exponent d, aber es gibt einen entscheidenden Unterschied zum Fall eines Primzahlexponenten: Für diesen mußten wir den erweiterten EUKLIDischen Algorithmus auf e und $p-1$ anwenden; jeder der in der Lage sein soll, die Verschlüsselungsfunktion f anzuwenden, muß diese Zahlen kennen und kann daher auch d berechnen.

Hier, für $N = pq$, wenden wir den erweiterten EUKLIDischen Algorithmus an auf e und $(p-1)(q-1)$. Wer verschlüsseln will, muß e und $N = pq$ kennen; die Primzahlen p, q muß er nicht kennen.

Wer diese nicht kennt, kann auch $(p-1)(q-1)$ nicht berechnen, denn die Kenntnis dieser Zahl ist äquivalent zu der von p und q:

$$(p-1)(q-1) = pq - p - q + 1 = (N+1) - (p+q)\,;$$

wer sowohl N als auch $(p-1)(q-1)$ kennt, kennt damit das Produkt $N = pq$ und die Summe $S = p+q$ von p und q. Daraus kann er die Primzahlen selbst problemlos berechnen als Lösungen der quadratischen Gleichung $x(S-x) = N$ oder $x^2 - Sx + N = 0$.

Falls es also keinen anderen Weg gibt, d aus N und e zu berechnen als den angegebenen, müßte jemand, der nur die Verschlüsselungsfunktion $f(x) = x^e \bmod N$ kennt, N in seine Primfaktoren zerlegen, was als schwierig gilt. Zumindest in den rund dreißig Jahren, in denen das RSA-Verfahren inzwischen angewandt wird, hatte niemand eine bessere Idee; deshalb ist dieser Ansatz das im Augenblick populärste asymmetrische Kryptoverfahren.

Zu seiner Anwendung wählt jeder Teilnehmer zwei Primzahlen p und q, die er streng geheimhält (und am besten vergißt, sobald er die folgenden Rechnungen durchgeführt hat). Daraus berechnet er $N = pq$ und wählt eine Zahl e, die teilerfremd zu $(p-1)(q-1)$ ist. Das Paar (N, e) veröffentlicht er als seinen *öffentlichen Schlüssel*. Damit kann jedermann Nachrichten an ihn *verschlüsseln*.

Der Teilnehmer, der auch p und q kennt, wendet außerdem noch den erweiterten EUKLIDischen Algorithmus an auf e und $(p-1)(q-1)$ und berechnet sich so als seinen *privaten Schlüssel* den Exponenten d. Damit kann er (und hoffentlich nur er) die Nachrichten auch *entschlüsseln*.

Für kleine Werte von e kann d auch ohne erweiterten EUKLIDischen Algorithmus berechnet werden: Wie wir wissen, gibt es stets natürliche Zahlen d und k, so daß

$$ed - k(p-1)(q-1) = 1 \quad \text{und} \quad d < (p-1)(q-1) \quad \text{und} \quad k < e$$

ist. Für kleine Werte von e kann man also auch einfach ausprobieren, für welche der Zahlen $k = 1, \ldots, e-1$ der Quotient

$$d = \frac{k(p-1)(q-1)}{e}$$

eine ganze Zahl ist; da e und $(p-1)(q-1)$ teilerfremd sind, gibt es genau ein solches k. Speziell im Fall des in der Praxis (leider) sehr populären Exponenten $e = 3$ muß man nur die Fälle $k = 1$ und $k = 2$ überprüfen.

§ 3: Praktische Anwendung von RSA

Natürlich ist RSA mit $p = 3$ und $q = 51$ kein sicheres Kryptoverfahren, und natürlich wollen wir in der Kryptographie meist keine natürlichen Zahlen übermitteln, sondern allgemeinere Nachrichten. Außerdem gehört das Rechnen modulo einer natürlichen Zahl nicht zu den Standardoperationen, die von jeder Programmiersprache als einfache Befehle bereitgestellt werden. In diesem Paragraphen soll kurz erläutert werden, wie man mit diesen Problemen umgeht.

a) Wie groß sollten die Primzahlen sein?

Ein treu sorgender Staat läßt seine Bürger bei einer derart wichtigen Frage natürlich nicht allein: Zwar gibt es noch keine oberste Bundesbehörde für Primzahlen, aber das Bundesamt für Sicherheit in der Informationstechnik (BSI) und die Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen erarbeiten jedes Jahr ein gemeinsames Dokument mit dem schönen Titel *Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung (Übersicht über geeignete Algorithmen)*.

Das Signaturgesetz und die dazu erlassene Signaturverordnung legen fest, daß elektronische Unterschriften in Deutschland grundsätzlich zulässig und rechtsgültig sind, sofern gewisse Bedingungen erfüllt sind. Zu diesen Bedingungen gehört unter anderem, daß das Verfahren und die Schlüssellänge gemeinsam einen „geeigneten Algorithmus“ im Sinne der jeweils gültigen Veröffentlichung der Bundesnetzagentur darstellen.

Da Rechner immer schneller und leistungsfähiger werden und auch auf der mathematisch-algorithmischen Seite fast jedes Jahr kleinere oder größere Fortschritte zu verzeichnen sind, gelten die jeweiligen Empfehlungen nur für etwa sechs Jahre. Für Dokumente, die länger gültig sein sollen, sind elektronische Unterschriften nicht vorgesehen.

Offiziell geht bei den Empfehlungen allgemein um geeignete Algorithmen für elektronische Unterschriften sowie deren Schlüssellängen, aber wie die Entwicklung der letzten Jahre zeigte, drehen sich die Diskussionen, die zu den jeweiligen Empfehlungen führen, tatsächlich fast ausschließlich um die jeweils notwendige Schlüssellänge für RSA.

Natürlich hat in einer Demokratie bei so einer wichtigen Frage auch die Bevölkerung ein Mitspracherecht; deshalb beginnt das BSI jeweils zunächst einen Entwurf, zu dem es um Kommentare bittet; erst einige Monate später wird die endgültige Empfehlung verkündet und im Bundesanzeiger veröffentlicht. Die jeweils aktuellen Versionen sind via häufig wechselnder Linkketten unter `www.bundesnetzagentur.de` zu finden; am schnellsten kommt man wohl mit Hilfe von Suchmaschinen auf die jeweils aktuelle Seite.

Die interessierte Öffentlichkeit, von der die Kommentare zu den Entwürfen kommen, besteht naturgemäß in erster Linie aus Anbietern von Hard- und Software zur Kryptographie, und als erfahrene Experten für Datensicherheit wissen diese, daß ein Verfahren nur dann wirklich geeignet sein kann, wenn es die eigene Firma im Angebot hat. (Am geeignetsten sind natürlich Verfahren, die kein Konkurrenzunternehmen anbietet.)

Hardware-Implementierungen von RSA unterstützen noch vor wenigen Jahren typischerweise nur Schlüssellängen von bis zu 1024 Bit; größere Schlüssel waren eher in *public domain* Software wie PGP zu finden. Dies erklärt, warum es in den letzten Jahren recht lebhafte Diskussionen gab:

Bis Ende 2000 galten 768 Bit als ausreichende Größe für das Produkt N der beiden Primzahlen. Schon in den Richtlinien für 1998 wurden 768 Bit jedoch ausdrücklich nur übergangsweise zugelassen; längerfristig, d.h. bei Gültigkeit über 2000 hinaus, waren mindestens 1024 Bit vorgeschrieben.

Die Richtlinien für 2000 erlaubten die 768 Bit ebenfalls noch bis zum Ende des Jahres; für Dokumente mit einer längeren Gültigkeit verlangten sie bis Mitte 2005 eine Mindestgröße von 1024 Bit, danach bis Ende 2005 sogar 2048 Bit.

Anbieterproteste führten dazu, daß nach den Richtlinien von 2001 eine Schlüssellänge von 1024 dann doch noch bis Ende 2006 sicher war;

die Schlüssellänge 2048 war nur noch „empfohlen“, also nicht mehr verbindlich.

Im April 2002 erschien der erste Entwurf für die 2002er Richtlinien; darin war für 2006 und 2007 nur eine Mindestlänge von 2048 Bit wirklich sicher. Einsprüche führten im September 2002 zu einem revidierten Entwurf, wonach 2006 doch noch 1024 Bit reichten, 2007 aber mindestens 1536 notwendig wurden. Die Mindestlänge von 2048 Bit wurde wieder zur „Empfehlung“ zurückgestuft.

Am 2. Januar 2003 erschienen endlich die offiziellen Richtlinien des Jahres 2002; veröffentlicht wurden sie am 11. März 2003 im Bundesanzeiger Nr. 48, S. 4202–4203. Danach reichten 1024 Bit auch noch bis Ende 2007, erst 2008 wurden 1280 Bit erforderlich. Die 2048 Bit blieben dringend empfohlen.

Nach diesem großen Kraftakt erschienen 2003 keine neuen Richtlinien mehr; erst für 2004 gab es am 2. Januar 2004 neue Empfehlungen (Bundesanzeiger Nr. 30 vom 13. Februar 2004, S. 2537–2538). Für den Zeitraum bis Ende 2008 wurden die alten Empfehlungen beibehalten, bis Ende 2009 aber 1536 Bit gefordert. Die nächsten Richtlinien für 2005 sahen in ihrem ersten Vorentwurf 2048 Bit bis Ende 2010 vor; nach Einsprüchen der Banken, daß das Betriebssystem SECCOS der heute üblichen Chipkarten nur mit maximal 1984 Bit-Schlüsseln umgehen kann, wurde die Länge im zweiten Entwurf auf 1984 gesenkt; in den endgültigen Richtlinien vom 2. Januar 2005 waren es schließlich nur noch 1728.

Die neuesten Richtlinien stammen vom 6. Januar 2010 (Bundesanzeiger Nr. 19 vom 4. Februar 2010, S. 426). Sie empfehlen grundsätzlich schon heute 2048 Bit, aber wirklich verbindlich sind bis Ende 2010 nur 1728 Bit, danach bis Ende 2016 aber 1976 Bit. (1976 unterscheidet sich nicht wesentlich von 2048; der minimal kleinere Wert wurde in Hinblick auf die oben erwähnten Probleme mit SECCOS gewählt. Mit Moduln der Länge 1976 läßt sich etwas effizienter arbeiten als mit den theoretisch noch implementierbaren 1984-Bit-Moduln.)

Die beiden Primfaktoren p, q sollen zufällig und unabhängig voneinan-

der erzeugt werden und aus einem Bereich stammen, in dem

$$\varepsilon_1 < |\log_2 p - \log_2 q| < \varepsilon_2$$

gilt. Als *Anhaltspunkte* werden dabei die Werte

$$\varepsilon_1 \approx 0{,}1 \quad \text{und} \quad \varepsilon_2 \approx 30$$

vorgeschlagen; ist p die kleinere der beiden Primzahlen, soll also gelten

$$1{,}071773463\,p \approx \sqrt[10]{2}\,p < q < 2^{30}p \approx 10^9 p\,.$$

Wenn wir der „dringenden Empfehlung“ der Bundesnetzagentur folgen, müssen wir somit Primzahlen mit ungefähr 1024 Bit oder 309 Dezimalstellen finden. Wie wir dabei vorgehen können ist Inhalt des nächsten Paragraphen.

b) Wie werden Nachrichten zu Zahlen?

RSA verschlüsselt Zahlen aus $\mathbb{Z}/N$; was wir übertragen wollen ist ein Text, eine elektronische Zahlungsanweisung oder ein Schlüssel für ein symmetrisches Kryptoverfahren. Diese Information muß irgendwie in eine Folge von Zahlen aus $\mathbb{Z}/N$ übersetzt werden.

Heute, da fast alle Information in Bit- oder eher Byte-Form vorliegt, gibt es dazu ein kanonisches Verfahren: Um etwa Bytes zu übertragen, nimmt man die größte Zahl n, für die $256^n < N$ ist, und faßt die Nachricht zusammen zu Blöcken aus jeweils n Bytes. Jedes dieser Bytes wird interpretiert als im Binärsystem geschriebene Zahl a_i; offensichtlich ist $0 \le a_i \le 255$. Die Zahlen $a_{n-1}, \ldots, a_0$ wiederum werden interpretiert als Zahl im System zur Basis 256, d.h. als die Zahl

$$m = \sum_{i=0}^{n-1} a_i 256^i\,.$$

Wenn es nur um die Übermittlung von Texten geht, kann man auch einfachere Verfahren zur Quellenkodierung verwenden: Als MARTIN GARDNER 1977 das RSA-Verfahren im *Scientific American* vorstellte, bekam er von RIVEST, SHAMIR und ADLEMAN als Beispiel-Modul die 129-stellige Zahl

$$114381625757888867669235779976146612010218296721242362562561842935\backslash$$
$$70693524573389783059712356395870505898907514759929002687954354 1$$

(seither bekannt als RSA-129), und dazu eine Zahl, die einer Nachricht entsprach, für deren Entschlüsselung die drei einen Preis von hundert Dollar ausgesetzt hatten. Sie schätzten, daß eine solche Entschlüsselung etwa vierzig Quadrillionen ($4 \cdot 10^{25}$) Jahre dauern würde. (Heute sagt RIVEST, daß dies auf einem Rechenfehler beruhte. Auch das ausgesetzte Preisgeld von \$100 spricht nicht gerade dafür, daß er diese Zahl sehr ernst nahm.) Tatsächlich wurde der Modul 1994 faktorisiert in einer gemeinsamen Anstrengung von 600 Freiwilligen, deren Computer immer dann, wenn sie nichts besseres zu tun hatten, daran arbeiteten. Nach acht Monaten war die Faktorisierung

$$\begin{aligned}&4905295108476509491478496199038981334177646384933878439908205 77\\ &\times 3276913299326670954996198819083446141317764296799294253979828853 3\end{aligned}$$

gefunden, und wie sich zeigte, hatten RIVEST, SHAMIR und ADLEMAN ihre Nachricht einfach mit dem Schema $A = 01$ bis $Z = 26$ und Zwischenraum gleich 00 quellkodiert; diese Zahlen wurden als Ziffern in einem Zahlensystem zur Basis 100 interpretiert, also (mit eventuellen führenden Nullen) einfach hintereinander geschrieben zu einer Zahl im Zehnersystem. Mit dieser Interpretation ist die Nachricht dann, wie jedermann auf dem aktuellen Übungsblatt selbst herausfinden kann, schnell entschlüsselt.

Beide Verfahren der Quellenkodierung haben allerdings einen entscheidenden Nachteil: Wie wir im letzten Abschnitt gesehen haben, müssen wir aus Sicherheitsgründen mit sehr großen RSA-Moduln arbeiten: Derzeit angebracht sind Moduln mit mindestens 2048 Bits, so daß wir bei byteweiser Verschlüsselung mindestens 128 Bytes pro Block übertragen können. Oft wollen wir allerdings deutlich weniger übertragen: Im Vergleich zum Aufwand für symmetrische Kryptoverfahren ist die RSA Ver- und Entschlüsselung mit einer Modullänge von 2048 Bit sehr viel aufwendiger, so daß RSA in der Praxis (abgesehen von kurzen Nachrichten, wie sie etwa im elektronischen Zahlungsverkehr anfallen) hauptsächlich dazu verwendet wird, um Schlüssel für ein symmetrisches Kryptoverfahren zu übertragen; bei den heute als sicher geltenden Verfahren sind das 128 oder höchstens 256 Bit. Die Nachricht ist also meist erheblich kürzer als die Blocklänge.

Bei einer guten symmetrischen Blockchiffre wäre das nicht weiter schlimm: Dort helfen uns gleich drei Sicherheitsfaktoren:

1. Der Lawineneffekt sorgt dafür, daß jede Änderung eines Bits der Nachricht rund die Hälfte der Chiffrebits beeinflußt.
2. Die Permutationen auf der Menge aller Blöcke, die durch die verschiedenen Schlüssel realisiert werden, wirken auf den Kryptanalytiker (fast) so, als seien sie zufallsverteilt.
3. Der Kryptanalytiker kann im allgemeinen nicht verifizieren, ob eine vermutete Entschlüsselung korrekt ist.

Bei asymmetrischen Chiffren können wir uns zumindest *a priori* auf keinen dieser drei Punkte verlassen:

Bei langen Nachrichten und großen RSA-Exponenten haben wir zwar einen durchaus beachtlichen Lawineneffekt, aber oftmals wird RSA zur Übertragung kurzer Nachrichten benutzt, und der populärste RSA-Exponent ist (trotz mehrerer bekannter Nachteile) die Drei: Die meisten Anwender suchen von vornherein nur nach Primzahlen p, q, für die weder $p-1$ noch $q-1$ durch drei teilbar sind, d.h. also nach Primzahlen kongruent zwei modulo drei, so daß dieser Exponent verwendet werden kann. Wird aber eine Nachricht, deren Bitlänge kleiner ist als ein Drittel der Bitlänge des RSA-Moduls N, durch ihre dritte Potenz modulo N ersetzt, so ist die übermittelte Chiffre einfach die in $\mathbb{N}_0$ berechnete dritte Potenz, und natürlich kann jeder Computer problemlos die Kubikwurzel einer natürlichen Zahl berechnen – mit einer Modifikation des aus der Schule bekannten Divisionsalgorithmus ist das sogar mit Bleistift und Papier nicht übermäßig schwierig.

Auch von einer zumindest für alle praktischen Zwecke zufälligen Verteilung der Permutationen können wir nicht ausgehen: Ein wesentliches Kriterium für die Beurteilung von DES war beispielsweise, daß die Menge aller dadurch realisierter Permutationen weit davon entfernt ist, eine Gruppe zu sein. Das ist bei RSA selbstverständlich nicht der Fall: Bei festgehaltenem Modul N bilden die Permutationen eine Gruppe, die ein homomorphes Bild der Gruppe der zu N teilerfremden Exponenten $e \in \mathbb{Z}$ ist. Schlimmer noch: Alle Permutationen $f\colon \mathbb{Z}/N \to \mathbb{Z}/N$, die durch RSA realisiert werden, haben die Eigenschaft, daß für beliebige

Nachrichten $m_1, m_2 \in \mathbb{Z}/n$ gilt $f(m_1 m_2) \equiv f(m_1) f(m_2) \bmod N$.

Die dritte Eigenschaft schließlich, daß ein Gegner nicht nachprüfen kann, ob eine vermutete Entschlüsselung korrekt ist, kann schon nach Definition eines asymmetrischen Kryptoverfahrens nicht gelten.

RSA ist also auf jeden Fall völlig unsicher, wenn kurze Nachrichten mit kurzen Exponenten übermittelt werden.

Leider werden kurze Nachrichten aber auch mit langen Exponenten nicht sicher verschlüsselt: Hier hilft dem Kryptanalytiker oft dieselbe Art von Angriff, mit der wir bereits den doppelten DES auf das (heute vernachlässigbare) Sicherheitsniveau des einfachen DES reduzieren konnten: die *meet in the middle attack*.

Ausgangspunkt dafür (wie auch später für andere Angriffe) ist die gerade erwähnte Multiplikativität der Verschlüsselungsfunktion:

Angenommen, wir wissen, daß die als Zahl betrachtete Nachricht m höchstens gleich M ist und daß sie (in $\mathbb{N}$) das Produkt zweier Zahlen m_1 und m_2 ist, die beide höchstens gleich einer Schranke S sind. Dann können wir die Nachricht m wie folgt aus dem Chiffretext $c = f(m)$ entschlüsseln: Wir berechnen für $k = 1, \ldots, S$ die Werte

$$f(k) = k^e \bmod N \quad \text{und} \quad c/f(k) \bmod N\,.$$

Die Werte $f(k)$ sortieren wird dann der Größe nach und schauen für jedes $c/f(k)$ nach, ob es in dieser Liste vorkommt. Falls wir ein Paar (m_1, m_2) gefunden haben mit $c/f(m_1) = f(m_2)$, ist $c = f(m_1 m_2)$; die Entschlüsselung von c ist also $m = m_1 m_2$.

Zur Division modulo N benutzen wir den erweiterten EUKLIDischen Algorithmus: Falls $\mathrm{ggT}\big(n, \varphi(k)\big) = 1$ ist, liefert dieser uns Zahlen a, b mit $a\varphi(k) + bN = 1$, d.h. $ca \cdot \varphi(k) \equiv c \bmod N$. Falls der ggT nicht gleich eins ist, kann er nur p oder q sein; in diesem Fall haben wir sogar den Modul N faktorisiert und können außer c auch noch jeden anderen Chiffretext entschlüsseln.

Bei der obigen Vorgehensweise müssen wir S RSA-Verschlüsselungen durchführen sowie S Divisionen modulo N. Außerdem müssen wir eine Liste mit S Werten sortieren und für bis zu S Werte nachschauen, ob und

gegebenenfalls wo sie in der sortierten Liste vorkommen. Der Aufwand dafür ist ein kleines Vielfaches von $S \log_2 S$, und dasselbe gilt somit auch für den Gesamtaufwand. Der Speicherbedarf liegt bei $2S$ RSA-Blöcken, kann aber auf Kosten eines leicht höheren Rechenaufwands verringert werden, wenn wir anstelle der Blöcke geeignete Hashwerte speichern.

Natürlich läßt sich nicht immer ein S finden, das wesentlich kleiner als M ist, so daß m Produkt zweier Faktoren der Größe höchstens S ist: Falls m etwa eine Primzahl ist, kann es kein solches $S < m$ geben. Andererseits gibt es aber auch Zahlen m, die sich als Produkt zweier fast gleich großer Faktoren schreiben lassen; selbst wenn man S nur geringfügig größer als $\sqrt{M}$ wählt, gibt es Werte von m, für die obige Attacke zum Erfolg führt.

> DAN BONEH, ANTOINE JOUX, PHONG Q. NGUYEN: Why Textbook ElGamal and RSA Encryption are Insecure, *AsiaCrypt '00, Lecture Notes in Computer Science* **1976** (2000), 30–44 *oder* `crypto.stanford.edu/~dabo/abstracts/ElGamalattack.html` ,

die diesen Angriff als erste vorgeschlagen haben, führten Experimente mit zufällig gewählten 64-Bit-Nachrichten durch. Für $S = 2^{32}$ hatten sie eine Erfolgsquote von 18%, bei $S = 2^{36}$ waren es 40%. Natürlich sind bereits deutlich kleinere Werte für ein sicheres Kryptoverfahren völlig inakzeptabel. **RSA in Reinform sollte daher nie verwendet werden.**

c) Probabilistische Verschlüsselung

Ein Ausweg, der auch beim Problem erratener Klartexte hilft, ist die 1984 von GOLDWASSER und MICALI vorgeschlagenen sogenannte *probabilistische Verschlüsselung*.

Um zu kurze Nachrichten zu verhindern, dürfen wir Nachrichten, die kürzer als die Blocklänge sind, nicht durch Nullen auf die volle Blocklänge ergänzen, sondern durch eine zufällige Folge von Null- und Einsbits. Wenn wir verlangen, daß jeder Block mindestens 128 solche Bits enthält, ist auch das Problem mit erratenen Klartexten vom Tisch, denn die Zufallsbits, so sie wirklich zufällig gewählt sind, lassen sich

weder erraten noch durchprobieren: Schließlich gehen wir auch bei symmetrischen Kryptoverfahren heute davon aus, daß das Durchprobieren von 2^{128} oder mehr Möglichkeiten nicht realistisch ist.

Bleibt noch das Problem, daß der Empfänger erkennen muß, welche Bits zur Nachricht gehören und welche nur zufällig gewählte Füllbits sind. Ein menschlicher Leser sollte damit zwar nur selten Probleme haben, aber erstens muß diese Entscheidung im allgemeinen ein Computer treffen, und zweitens können auch Zufallsbits gelegentlich einen Sinn ergeben, der selbst einen menschlichen Leser verwirren kann.

Die Art und Weise der Anwendung von Zufallsbits muß daher vorher vereinbart werden. Dazu gibt es Standards wie PKCS #1 (PKCS = *public key cryptography standard*), mit denen wir uns in §7 beschäftigen werden; insbesondere werden wir sehen, daß auch diese Standards gut überlegt sein müssen, da sonst ausgerechnet der Standard selbst neue Angriffsmöglichkeiten eröffnet.

d) Wie berechnet man die RSA-Funktion effizient?

Wer sich nach den Empfehlungen des Bundesamts für Sicherheit in der Informationstechnik sowie der Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen hält, sollte für RSA mit einem Modul N arbeiten, der mindestens eine Länge von 2 048 Bit hat.

Damit haben auch die zu übermittelnde Nachrichtenblöcke eine Länge von mindestens 2 048 Bit, also 256 Byte, und die e-te Potenz der entsprechenden Zahlen hat die e-fache Länge. Für die Verschlüsselung können wir einen kleinen Exponenten e wählen, für die Entschlüsselung allerdings wird der Exponent d mit an Sicherheit grenzender Wahrscheinlichkeit in der Größenordnung von N liegen, so daß m^d eine Bitlänge von etwa $(2\,048)^2$ Bit hat, also ein halbes Megabyte.

Dafür hat ein heutiger Computer natürlich mehr als genug Speicherplatz, aber er muß die Zahlen auch berechnen, und zumindest wenn man das in der dümmstmöglichen Weise durchführt, indem man sukzessive die Potenzen $m, m^2, m^3, \ldots$ berechnet, überfordern auch deutlich kleinere Exponenten selbst die besten heutigen Supercomputer um Größenordnungen.

Tatsächlich gibt es aber keinen Grund, die natürliche Zahl m^d wirklich zu berechnen: Wir brauchen schließlich nur $m^d \bmod N$. Außerdem käme hoffentlich auch kein Leser auf die dumme Idee, die Zahl 3^{32} durch 31-fache Multiplikation mit Drei zu berechnen: Da $32 = 2^5$ ist, läßt sich das Ergebnis viel schneller als

$$3^{32} = \left(\left(\left(\left(3^2\right)^2\right)^2\right)^2\right)^2$$

durch nur fünfmaliges Quadrieren berechnen.

Entsprechend können wir für jede gerade Zahl $n = 2m$ die Potenz x^n als Quadrat von x^m berechnen. Für einen ungeraden Exponenten e ist $e-1$ gerade, wenn wir also m^e als Produkt von m und m^{e-1} darstellen, können wir zumindest im nächsten Schritt wieder die Formel für gerade Exponenten verwenden. Da uns das Ergebnis nur modulo N interessiert, können wir zudem nach jeder Multiplikation und jeder Quadrierung das Ergebnis modulo N reduzieren; auf diese Weise entsteht nie ein Zwischenergebnis, das größer ist als N^2.

Dies führt auf folgenden rekursiven Algorithmus zur Berechnung von $m^e \bmod N$:

Falls $e = 2f$ gerade ist, berechne man zunächst $m^f \bmod N$ nach diesem Algorithmus und quadriere das Ergebnis modulo N; andernfalls gibt es im Falle $e = 1$ nichts zu tun, und für $e > 1$ berechne man zunächst $m^{e-1} \bmod N$ und multipliziere das Ergebnis modulo N mit m.

Falls e eine Zahl mit r Bit ist, erfordert dieser Algorithmus $r-1$ Quadrierungen und höchstens r, im Mittel rund $r/2$ Multiplikationen mit m. Für einen Exponenten mit 2 048 Bit brauchen wir also im Mittel rund 3 072 Multiplikationen, auf keinen Fall aber mehr als 4 096, und damit wird ein heutiger Computer problemlos fertig.

Bleibt noch die Frage: Wie multiplizieren wir zwei 2 048-Bit-Zahlen?

Für Taschenrechner wie auch 32- oder 64-Bit-Wörter eines Computers sind sie natürlich zu groß. Trotzdem ist die vielleicht erstaunliche Antwort auf obige Frage, daß wir genau so vorgehen können, wie wir es in der Schule gelernt haben: Zwar gibt es Multiplikationsalgorithmen,

die asymptotisch schneller sind als die Schulmethode, aber tatsächlich schneller werden sie erst, wenn die Zahlen eine Bitlänge haben, die eher bei Millionen liegt als bei bloßen Tausendern oder Hundertern

Einen Unterschied zur Schule sollten wir freilich machen: Während uns in der Grundschule das kleinen Einmaleins eingepaukt wird, also die Produkte der Zahlen von Eins bis Zehn untereinander, sind in den CPUs unserer Computer Algorithmen implementiert, die zwei 32-Bit-Zahlen zu einer 64-Bit-Zahl multiplizieren oder, falls der Computer hinreichend teuer war, zwei 64-Bit-Zahlen zu einer 128-Bit-Zahl. Wir sollten die Zahlen also nicht im Zehnersystem betrachten, sondern im Ziffernsystem mit Basis 2^{32} oder 2^{64}.

Nach jeder Multiplikation muß das Ergebnis modulo N reduziert werden; wir müssen also durch N dividieren. Auch dazu können wir *im Prinzip* genauso vorgehen wie in der Schule, haben dabei allerdings das Problem, daß das in der Schule gelehrte Divisionsverfahren kein Algorithmus ist: Wir müssen schließlich in jedem Schritt die nächste Ziffer des Quotienten *erraten* und sehen erst nach Multiplikation mit dem Divisor oder sogar erst nach Subtraktion dieses Produkts vom Dividenden, ob wir das korrekte Ergebnis haben.

Zum Glück läßt sich dieses „Erraten“ selbst für beliebige Basen des Ziffernsystems zumindest insoweit algorithmisch machen, daß das Ergebnis nie um mehr als zwei danebenliegt, und auch ein Fehler von zwei nur mit verschwindend geringer Wahrscheinlichkeit auftritt. Da es inzwischen viele Unterprogrammpakete und auch Programme gibt, in denen Algorithmen zum Rechnen mit Langzahlen implementiert sind, sei hier nicht auf Einzelheiten eingegangen; Interessenten finden diese zusammen mit allen Beweisen z.B. in Abschnitt 4.3 von

> DONALD E. KNUTH: The Art of Computer Programming, *vol. 2:* Seminumerical Algorithms, *Addison Wesley,*, ^{2}1981

e) Konkrete Implementierungen

Zumindest kurz sei erwähnt, wie diese Algorithmen in den Programmiersprachen oder Programmsystemen implementiert sind, mit denen die Hörer dieser Vorlesung wahrscheinlich am ehesten vertraut sind.

1.) Maple: Diejenigen, die keinerlei Erfahrung mit einer Programmiersprache haben, benutzen zumindest für Beispiele nur zur Demonstration am besten ein Computeralgebrasystem; die Beispiele in der Vorlesung wurden größtenteils in Maple programmiert.

Alle Computeralgebrasysteme können mit Langzahlen rechnen – zumindest bis zu Längen, die weit jenseits dessen liegen, was heute in der Kryptographie benutzt wird. Grundrechenarten werden einfach durch die üblichen Zeichen „+“, „–“, „*“ und „/“ bezeichnet, die Potenzierung mit „^“ und die Berechnung des Divisionsrests mit „mod“. Wenn wir allerdings `m^d mod N` eingeben, wird diese Formel von links nach rechts abgearbeitet, als erstes wird also m^d berechnet, was für realistische Beispiele aus der Kryptographie jenseits der Möglichkeiten heutiger Computer liegt. Um m^d mod N zu berechnen muß man daher die Auswertung des Operators „^“ zunächst verhindern; dafür sorgt ein vorangestelltes „&“. Der Ausdruck `m&^d mod N` veranlaßt, daß die Berechnung von m^d mod N nach dem Algorithmus aus dem vorigen Abschnitt erfolgt und damit für die Zahlen, mit denen wir es in der Kryptographie zu tun haben, ziemlich schnell geht.

Der EUKLIDische Algorithmus sowie seine Erweiterung sind Maple (wie auch jedem anderen Computeralgebrasystem) natürlich bekannt; die Funktion `igcd(x, y)` berechnet den ggT zweier ganzer Zahlen x, y, genauso auch `igcdex(x, y, 'a', 'b')`. Letztere Anweisung setzt als Nebeneffekt noch die Variablen a und b auf ganze Zahlen, für die $ax + by$ = ggT(x, y) ist. Falls man nur an a interessiert ist, kann man das Argument `'b'` auch weglassen.

2.) Maxima: Als kommerzielle Software ist Maple insbesondere in der Vollversion sehr teuer; auch der Preis der Studentenversion ist nicht zu vernachlässigen. Eine kostenlose Alternative ist das Computeralgebrasystem Maxima, dessen neueste Versionen jeweils unter `maxima.sourceforge.net` zu finden sind, speziell für Windows als ein einzelnes, sich selbst installierendes Programm. Es beruht auf Macsyma, einem der ersten Computeralgebrasysteme überhaupt, das ab 1968 am MIT entwickelt und unter anderem vom amerikanischen *Department of Energy (DOE)* gesponsort wurde. 1982 übergaben die MIT-Forscher

eine Version an das DOE, das wiederum 1998 einem seiner Entwickler erlaubte, das Programm als freie Software zu veröffentlichen. Diese Version ist unter dem Namen Maxima bekannt.

Am gewöhnungsbedürftigsten an Maxima ist der Doppelpunkt als Zuweisungsoperator: Um der Variablen x den Wert $2^{16} + 1$ zuzuweisen, muß man also `x : 2 ^ 16 + 1` eingeben. Die Operatoren für die Grundrechenarten sind die üblichen, potenziert wird wahlweise mit ^ oder mit **. Zur Berechnung von a^e mod N dient die Funktion `power_mod(a, e, N)`, den ggT berechnet zweier Zahlen oder Polynome a, b gerechnet `gcd(a, b)`. In der Variante `gcdex(a, b)` wird eine Liste `[c, d, e]` zurückgegeben mit dem ggT $e = ca + db$.

3.) Scheme/Racket: Scheme *bzw.* Racket ist eine funktionale Programmiersprache, die teilweise an Schulen in Rheinland-Pfalz gelehrt wird. Sie ist ein Dialekt der Programmiersprache LISP, in dem standardmäßig mit ganzen Zahl praktisch beliebiger Länge gerechnet werden kann; für Grundrechenarten sind also keine besonderen Befehle notwendig. Der Potenzierungsoperator `expt` kann aber zumindest für große Exponenten natürlich nicht verwendet werden: Er würde die Potenz als ganze Zahl berechnen. Ein Operator für die Potenzierung modulo einer natürlichen Zahl N ist standardmäßig nicht vorhanden, aber der Algorithmus aus dem vorigen Abschnitt läßt sich problemlos in Scheme übersetzen:

```
(define (modsquare x N) (remainder (* x x) N)

(define (modexp x e N)
   (cond ((= e 0) 1)
         ((even? e) ( modsquare (modexp x (/ e 2) N) ))
         ( else (remainder (* base (modexp x (– e 1))) N) )
   )
)
```

Mit dem EUKLIDischen Algorithmus einschließlich seiner Erweiterung gibt es auch keinerlei Schwierigkeiten: Der Grundalgorithmus ist einfach der Einzeiler

```
(define (ggT x y) (if (= y 0) x (ggT y (remainder y x))));
```

der erweiterte Algorithmus läßt sich beispielsweise folgendermaßen programmieren: Wir betrachten Sechstupel `(u v a b c d)` mit der Eigenschaft, daß stets gilt

$$\mathrm{ggT}(x,y) = \mathrm{ggT}(u,v), \quad u = ax + by \quad \text{und} \quad v = cx + dy\,.$$

Das Anfangs-Tupel mit dieser Eigenschaft ist `(x y 1 0 0 1)`. Wenn wir irgendein Sechstupel `(u v a b c d)` mit obigen Eigenschaften haben und zusätzlich $v = 0$ ist, wissen wir, daß $u = \mathrm{ggT}(x,y)$ ist, und nach Konstruktion der Sechstupel gilt

$$u = \mathrm{ggT}(x,y) = ax + by\,.$$

In diesem Fall können wir also die Liste `(u a b)` als Ergebnis zurückgeben.

Andernfalls dividieren wir u mit Rest durch v; wie die obige Rechnung zeigt, ist dann `(v r c d` $a - qc$ $b - qd$`)` ein neues Sechstupel mit den verlangten Eigenschaften. Seine zweite Komponente r ist echt kleiner als die zweite Komponente v des Ausgangstupels; somit muß der Algorithmus nach endlich vielen Schritten mit $v = 0$ abbrechen.

Eine vollständige Implementierung könnte somit etwa folgendermaßen aussehen:

```
(define (erwEuklid x y)
   (define (iteration u v a b c d)
      (if (= v 0) (list u a b)
       (let ((q (quotient u v)))
       (iteration v (remainder u v) c d
                  (- a (* q c)) (- b (* q d))))))
   (iteration x y 1 0 0 1))
```

4.) Java: Hier sind standardmäßig nur ganze Zahlen vorgesehen, die in ein Maschinenwort passen. In `java.math` gibt es jedoch eine Klasse BigInteger, die ganze Zahlen von (praktisch) beliebiger Länge bereitstellt. Sie ist leider gnadenlos objektorientiert, d.h. anstelle von $a + b$, $a - b$, $a \cdot b$, a/b oder a mod b muß man

```
a.add(b),  a.subtract(b),  a.multiply(b),
    a.divide(b)  oder  a.remainder(b)
```


schreiben, was längere Formeln schnell unübersichtlich werden läßt. Entsprechend braucht man zum Vergleich eine Methode `equals`, *usw.*

Erzeugt werden Langzahlen durch eine Vielzahl von Methoden; die wichtigsten sind `valueOf(x)` mit einer Zahl x vom Typ `long` und `BigInteger(`*string*`)`, wobei die Zeichenkette *string* aus den (beliebig vielen) Ziffern der Zahl besteht; umgekehrt gibt `toString()` die Zahl als Ziffernfolge aus. Auch verschiedene Algorithmen sind eingebaut; beispielsweise liefert die Methode `modPow(e, N)` die e-te Potenz des Objekts modulo N nach dem Algorithmus aus dem vorigen Abschnitt.

Der ggT zweier Langzahlen a und b kann als `a.gcd(b)` berechnet werden; der erweiterte EUKLIDische Algorithmus ist nur intern als private Methode der Klasse vorhanden, von außen ist er nicht ansprechbar. Wer ihn benutzen möchte, muß ihn also entweder selbst programmieren oder aber in einer Kopie der Klasse Änderungen vornehmen und dann mit dieser Kopie zu arbeiten.

5.) C, C++, *. . .* ***:*** Bei den meisten seriösen Anwendungen wird im Hintergrund irgendein Programm in C, C++ oder einer verwandten Sprache stehen; auch stellen die meisten anderen Programmiersprachen eine Schnittstelle bereit, über die sich C-Unterprogramme einbinden lassen.

Als systemnahe Sprache unterstützt C natürlich vor allem die Datentypen, die von der Hardware bereitgestellt werden, und dazu gehören – sofern man keine Spezialchips zur Signalverarbeitung in seinem Computer hat – keine Langzahlen.

Daß trotzdem (abgesehen von Java-basierten Internetanwendungen) wohl die meisten Kryptoalgorithmen in C oder C++ implementiert sein dürften, liegt daran, daß sich gerade etwas so hardwarenahes wie eine Langzahlarithmetik hiermit am besten effizient implementieren läßt – jedenfalls fast am besten. Für höchste Effizienz wird man nicht daran vorbeikommen, zumindest einen Teil der Rechnungen in Assembler zu programmieren: Wenn man beispielsweise zwei 32-Bit-Zahlen addiert und das Ergebnis länger als 32 Bit ist, meldet die Hardware einen Überlauf (neudeutsch *Overflow*). Dieser läßt zwar auch aus einem C-Programm abfragen, aber einen Additionsbefehl, der automatisch eins

addiert, wenn bei der letzten Addition ein Überlauf aufgetreten ist, gibt es zwar in der Maschinensprache wohl jedes heute üblichen Prozessors, von höheren Programmiersprachen aus ist dieser aber nicht ansprechbar. (Compiler verwenden traditionellerweise nur einen Bruchteil der zur Verfügung stehenden Assemblerbefehle.)

Als Beispiel einer Bibliothek für Langzahlarithmetik sei etwa GMP (**G**NU **M**ultiple **P**recision Arithmetic Library) genannt, zu finden bei `gmplib.org`, sowie PARI (`pari.math.u-bordeaux.fr`), wobei letzteres außer Langzahlarithmetik noch zahlreiche Funktionen aus der Zahlentheorie sowie zum Rechnen auf elliptischen Kurven zur Verfügung stellt. Mit dem zu PARI gehörenden Kommandointerpreter `gp` können die PARI-Funktionen auch ohne C-Programm in einer Art Taschenrechnermodus verwendet werden. Hier können Zahlen gleich als Elemente von $\mathbb{Z}/N$ definiert werden: Setzt man `x = Mod(a, N)`, so werden alle Rechenoperationen mit x automatisch modulo N ausgeführt. Der ggT von x und y wird durch `gcd(a, b)` berechnet, und `bezout(a, b)` berechnet einen Vektor $[c, d, e]$ mit $ca + db = e = \mathrm{ggT}(a, b)$. Gewöhnungsbedürftig bei der Benutzung von `gp` ist die Rolle des Strichpunkts: Ein Strichpunkt am Ende der Eingabezeile bedeutet, daß das Ergebnis nicht auf dem Bildschirm erscheint. Die Anweisung `x = 2^100;` weißt zwar der Variablen x den Wert von 2^{100} zu, bringt diesen im Gegensatz zur Anweisung ohne Strichpunkt aber nicht auf den Bildschirm.

Praktisch alle Pakete zur Langzahlarithmetik können sowohl mit C als auch mit C++ verwendet werden. C++ hat den großen praktischen Vorteil, daß man dort via *operator overloading* die entsprechenden Unterprogramme in der aus der Mathematik gewohnten Weise mit Infixoperatoren „+“, „-“, „*“ und „/“ aufzurufen kann. Bei der Potenzierung modulo N muß man natürlich darauf achten, daß man einen Operator verwendet, der in allen Rechenschritten modulo N reduziert.

§4: Was läßt sich mit RSA anfangen?

Mit symmetrischen Kryptoverfahren lassen sich Nachrichten verschlüsseln, und im wesentlichen ist das auch schon alles, was man damit tun kann. Asymmetrische Verfahren haben dagegen noch eine Reihe weitere

Einsatzmöglichkeiten, die in der Praxis oft erheblich größere Bedeutung haben als die Verschlüsselung. Einige der wichtigsten sollen hier kurz vorgestellt werden.

a) Identitätsnachweis

Hierzu läßt sich prinzipiell auch ein symmetrisches Kryptoverfahren nutzen, allerdings nur gegenüber dem Partner oder den Partnern, mit denen ein gemeinsamer Schlüssel vereinbart wurde, und es ist auch nicht möglich zwischen diesen zu unterscheiden: Wer in der Lage ist, einen vorgegebenen Chiffretext zu entschlüsseln, muß – falls man der Sicherheit des Verfahrens trauen kann – den Schlüssel kennen.

Bei asymmetrischen Kryptoverfahren wie RSA ist die Situation deutlich besser: Nur der Inhaber A des geheimen Schlüssels d kann zu einem gegebenen a eine Zahl b berechnen, für die $b^e \equiv a \bmod N$ ist, aber jeder, der den öffentlichen Schlüssel (N, e) kennt, kann nachprüfen, ob er diese Aufgabe wirklich gelöst hat.

Soll also A gegenüber B seine Identität nachweisen, verschafft sich B den öffentlichen Schlüssel (N, e) von A und schickt eine Zufallszahl $1 < x < N$ an A. Dieser berechnet $y = x^d \bmod N$ und schickt diese Zahl an B. Dieser prüft nach, ob $y^e \equiv x \bmod N$ ist. Bei sicher gewählten Parametern N und e kann niemand außer ihm ein derartiges y erzeugen.

Ähnliche Verfahren werden in Zugangskontrollsystemen tatsächlich angewandt; zumindest wenn man seine Identität gegenüber *jedermann* so etablieren möchte, dürfen dazu aber auf keinen Fall dieselben RSA-Parameter benutzt werden, die man zur Ver- und Entschlüsselung geheimer Nachrichten einsetzt:

Die offensichtlichste Sicherheitslücke ist hier, daß ein Gegner, der einen Chiffreblock c auffängt, vom Empfänger einen Identitätsnachweis verlangt, bei dem der „Zufallsblock“ x gleich c ist. Die Antwort wäre natürlich der Klartext zu c.

Optimisten können hoffen, daß niemand so dumm wäre, einen sinnvollen Klartext als Identitätsnachweis zurückzuschicken, aber erstens müssen wir angesichts der Größe der Zahlen, um die es hier geht, davon ausgehen, daß tatsächlich ein Computer oder (eher) eine Chipkarte auf die

Anfrage reagiert. Zweitens sollten wir, selbst wenn Menschen involviert sind, bei jedem praktisch eingesetzten Kryptosystem vorsichtshalber von der fast unbegrenzten Dummheit zumindest eines Teils der Anwender ausgehen. Drittens schließlich kann man selbst einen extrem vorsichtigen, vielleicht sogar paranoiden Anwender, der jeden Block vor dem Abschicken genau überprüft, leicht überlisten:

Ein Angreifer, der an der Entschlüsselung des Chiffreblocks c zum Klartext $m = c^d \bmod N$ interessiert ist, erzeugt eine Zufallszahl r und schickt $x = r^e c \bmod N$ als Herausforderung zum Identitätsnachweis an A. Dieser berechnet

$$y = x^d \bmod N = r^{ed} c^d \bmod N = rc^d \bmod N = rm \bmod N$$

und schickt diese Zahl zurück: Bei einem wirklich zufällig gewählten r kann er auch bei sorgfältigster Untersuchung den Zahlen x und y nichts ansehen. Der Angreifer, der r kennt, kann trotzdem problemlos m berechnen, indem er einfach modulo N durch r dividiert.

(Da $N = pq$ keine Primzahl ist, kann es natürlich vorkommen, daß r modulo N nicht invertierbar ist: Das ist genau dann der Fall, wenn r durch p oder q teilbar ist. Wenn wir p und q jeweils als 1024-Bit-Primzahlen wählen, liegt die Wahrscheinlichkeit für ein solches r bei etwa 2^{-1024}, ist also für alle praktischen Zwecke vernachlässigbar: Die Wahrscheinlichkeit, 43 Wochen hintereinander sechs Richtige im Lotto zu haben, ist etwa zehnmal so groß. Sollte dieser Fall trotzdem einmal eintreten, ist der ggT von r und N einer der beiden Primteiler von N; dann läßt sich also nicht nur der Chiffreblock c entschlüsseln, sondern sogar der private Exponent d von A berechnen und damit kann dessen gesamte künftige Kommunikation entschlüsselt werden.)

Weiterhin ist zu beachten, daß A mit diesem Verfahren zwar gegenüber seinem Herausforderer beweisen kann, daß er wirklich er selbst ist, daß aber letzterer nicht gegenüber einem Dritten beweisen kann, daß er wirklich mit A in Verbindung stand: Schließlich könnte der Herausforderer einfach eine Zufallszahl y erzeugen und dann behaupten, er habe y als Antwort auf die Herausforderung $x = y^e \bmod N$ erhalten. Praktische Bedeutung hat deshalb vor allem eine andere Variante für den Identitätsnachweis:

b) Elektronische Unterschriften

Hier geht es darum, daß der Empfänger erstens davon überzeugt wird, daß eine Nachricht tatsächlich vom behaupteten Absender stammt, und daß er dies zweitens auch einem Dritten gegenüber *beweisen* kann. (In Deutschland sind solche elektronischen Unterschriften, wie bereits erwähnt, genauso rechtsverbindlich wie klassische Unterschriften.)

Um einen Nachrichtenblock a mit $0 \le a < N$ zu unterschreiben, berechnet der Inhaber A des öffentlichen Schlüssels (N, e) mit seinem geheimen Schlüssel d die Zahl

$$u = a^d \bmod N$$

und sendet das Paar (a, u) an den Empfänger. Dieser überprüft, ob

$$u^e \equiv a \bmod N$$

ist; falls ja, akzeptiert er dies als unterschriebene Nachricht a. Da er ohne Kenntnis des geheimen Schlüssels d nicht in der Lage ist, den Block (a, u) zu erzeugen, kann er auch gegenüber einem Dritten beweisen, daß A die Nachricht a unterschrieben hat.

Für kurze Nachrichten ist dieses Verfahren in der vorgestellten Form praktikabel; in vielen Fällen kann man sogar auf die Übermittlung von a verzichten, da $u^e \bmod N$ für ein falsch berechnetes u mit an Sicherheit grenzender Wahrscheinlichkeit keine sinnvolle Nachricht ergibt.

Falls die übermittelte Nachricht geheimgehalten werden soll, müssen a und u natürlich noch vor der Übertragung mit dem öffentlichen Schlüssel des Empfängers oder nach irgendeinem anderen Kryptoverfahren verschlüsselt werden.

Bei langen Nachrichten ist die Verdoppelung der Nachrichtenlänge nicht mehr akzeptabel, und selbst, wenn man auf die Übertragung von a verzichten kann, ist das Unterschreiben jedes einzelnen Blocks sehr aufwendig. Deshalb wird man meist nicht die Nachricht selbst unterschrieben, sondern einen daraus berechneter Hashwert. Dieser Wert muß natürlich erstens von der gesamten Nachricht abhängen, und zweitens muß es für den Empfänger (praktisch) unmöglich sein, zwei Nachrichten zu

erzeugen, die zum gleichen Hashwert führen. Mit der Theorie und Praxis dieser sogenannten kollisionsfreien Hashfunktionen werden wir uns später beschäftigen.

c) Bankkarten mit Chip

In Deutschland und den meisten anderen Ländern hat eine Bankkarte einen Magnetstreifen, auf dem die wichtigsten Informationen wie Kontenname und -nummer, Bankleitzahl, Gültigkeitsdauer *usw.* gespeichert sind; dazu kommt mit Triple-DES verschlüsselte Information, die unter anderem die Geheimzahl enthält, aber auch von den oben genannten Daten abhängt.

Der Schlüssel, mit dem Triple-DES hier arbeitet, muß natürlich streng geheimgehalten werden: Wer ihn kennt, kann problemlos die Geheimzahlen fremder Karten ermitteln und selbst Karten mit Verfügungsgewalt über beliebige andere Konten erzeugen.

Um eine Karte zu überprüfen, muß daher eine Verbindung zu einem Zentralrechner aufgebaut werden, an den sowohl der Inhalt des Magnetstreifens als auch die vom Kunden eingetippte Geheimzahl übertragen werden; dieser wendet Triple-DES mit dem Systemschlüssel an und meldet dann, wie die Prüfung ausgefallen ist.

In Frankreich haben die entsprechenden Karten zusätzlich zum Magnetstreifen noch einen Chip, in dem ebenfalls die Kontendaten gespeichert sind sowie, in einem auslesesicheren Register, Informationen über die Geheimzahl. Dort wird die ins Lesegerät eingetippte Geheimzahl nicht an den Zentralrechner übertragen, sondern an den Chip, der sie überprüft und akzeptiert oder auch nicht.

Da frei programmierbare Chipkarten relativ billig sind, muß dafür Sorge getragen werden, daß ein solches System nicht durch einen sogenannten *Yes-Chip* unterlaufen werden kann, der ebenfalls die Konteninformationen enthält, ansonsten aber ein Programm, das ihn *jede* Geheimzahl akzeptieren läßt. Das Terminal muß also, bevor es überhaupt eine Geheimzahl anfordert, zunächst einmal den Chip authentisieren, d.h. sich davon überzeugen, daß es sich um einen vom Bankenkonsortium ausgegebenen Chip handelt.

Aus diesem Grund sind die Kontendaten auf dem Chip mit dem privaten RSA-Schlüssel des Konsortiums unterschrieben. Die Terminals kennen den öffentlichen Schlüssel dazu und können so die Unterschrift überprüfen.

Diese Einzelheiten und speziell deren technische Implementierung wurden vom Bankenkonsortium zunächst streng geheimgehalten. Trotzdem (KERCKHOFFS läßt grüßen) machte sich 1997 ein Elsässer Ingenieur namens SERGE HUMPICH daran, den Chip genauer zu untersuchen. Er verschaffte sich dazu ein (im freien Verkauf erhältliches) Terminal und untersuchte sowohl die Kommunikation zwischen Chip und Terminal als auch die Vorgänge innerhalb des Terminals mit Hilfe eines Logikanalysators. Damit gelang es ihm nach und nach, die Funktionsweise des Terminals zu entschlüsseln und in ein äquivalentes PC-Programm zu übersetzen. Durch dessen Analyse konnte er die Authentisierungsprozedur und die Prüflogik entschlüsseln und insbesondere auch feststellen, daß hier mit RSA gearbeitet wurde.

Blieb noch das Problem, den Modul zu faktorisieren. Dazu besorgte er sich ein japanisches Programm aus dem Internet, das zwar eigentlich für kleinere Zahlen gedacht war, aber eine Anpassung der Wortlänge ist natürlich auch für jemanden, der den Algorithmus hinter dem Programm nicht versteht, kein Problem. Nach sechs Wochen Laufzeit hatte sein PC damit den Modul faktorisiert:

$$\begin{aligned} & 213598703592091008239502270499962879705109534182\backslash \\ & 641740644252416500858395774644508840500943086599 9 \\ = \; & 111395432514882798792549017547702484407092284484 3 \\ \times \; & 191748170252450443937578626823086218069693418929 3 \end{aligned}$$

Als er seine Ergebnisse über einen Anwalt dem Bankenkonsortium mitteilte, zeigte sich, was dieses sich unter Sicherheitsstandards vorstellt: Es erreichte, daß HUMPICH wegen des Eindringens in ein DV-System zu zehn Monaten Haft auf Bewährung sowie einem Franc Schadenersatz plus Zinsen verurteilt wurde; dazu kamen 12 000 F Geldstrafe.

Seit November 1999 haben neu ausgegebene Bankkarten nun noch ein zusätzliches Feld mit einer Unterschrift, die im Gegensatz zum obigen

320-Bit-Modul einen 768-Bit-Modul verwendet. Natürlich kann es nur von neueren Terminals überprüft werden, so daß viele Transaktionen weiterhin nur über den 320-Bit-Modul mit inzwischen wohlbekannter Faktorisierung „geschützt“ sind.

Auch ein Modul der Länge 768 ist verglichen mit den Empfehlungen der Bundesnetzagentur lächerlich klein; wie wir weiter hinten sehen werden, wurde Ende 2009 die erste Faktorisierung eins 768-Bit RSA-Moduls publiziert, und die Geheimdienste reicher und technologisch entwickelter Staaten waren mit ziemlicher Sicherheit schon früher dazu in der Lage und hätten daher der französischen Wirtschaft unabsehbaren Schaden zufügen können.

d) Elektronisches Bargeld

Zahlungen im Internet erfolgen meist über Kreditkarten; die Kreditkartengesellschaften haben also einen recht guten Überblick über die Ausgaben ihrer Kunden und machen teilweise auch recht gute Geschäfte mit Kundenprofilen.

Digitales Bargeld will die Anonymität von Geldscheinen mit elektronischer Übertragbarkeit kombinieren und so ein anonymes Zahlungssystem z.B. für das Internet bieten.

Eine Idee zur Realisierung eines solchen Systems beruht auf dem in *a)* skizzierten Angriff auf RSA unter der Voraussetzung, daß derselbe Schlüssel sowohl zur Identitätsfeststellung als auch zur Verschlüsselung benutzt wird:

Eine Bank, die elektronisches Bargeld ausgeben will, erzeugt für jede angebotene Stückelung einen öffentlichen Schlüssel (N, e), der allen Teilnehmern des Zahlungssystems mitgeteilt wird. Eine elektronische Banknote ist eine mit dem zugehörigen geheimen Schlüssel unterschriebene Seriennummer.

Die Seriennummer kann natürlich nicht einfach *jede* Zahl sein; sonst wäre jede Zahl kleiner N eine Banknote. Andererseits dürfen die Seriennummern aber auch nicht von der Bank vergeben werden, denn sonst wüßte diese, welcher Kunde Scheine mit welchem Seriennummern hat. Als Ausweg wählt man Seriennummern einer sehr speziellen Form:

Ist $N > 10^{150}$, kann man etwa als Seriennummer eine 150-stellige Zahl wählen, deren Ziffern spiegelsymmetrisch zur Mitte sind, d.h. ab der 76. Ziffer werden die vorherigen Ziffern rückwärts wiederholt. Die Wahrscheinlichkeit, daß eine zufällige Zahl x nach Anwendung des öffentlichen Exponenten auf so eine Zahl führt, ist 10^{-75} und damit vernachlässigbar.

Seriennummern werden von den Kunden (unter Beachtung der Symmetrie) zufällig erzeugt. Für jede solche Seriennummer m erzeugt der Kunde eine Zufallszahl r, schickt mr^e mod N an die Bank und erhält (nach Belastung seines Kontos) eine Unterschrift u für diese Nachricht zurück. Wie in *a)* berechnet er daraus durch Multiplikation mit r^{-1} die Unterschrift $v = m^d$ mod N für die Seriennummer m. Mit dieser Zahl v kann er bezahlen.

Der Zahlungsempfänger berechnet v^e mod N; falls dies die Form einer gültigen Seriennummer hat, kann er *praktisch* sicher sein, einen von der Bank unterschriebenen Geldschein vor sich zu haben. Er kann allerdings noch nicht sicher sein, daß dieser Geldschein nicht schon einmal ausgegeben wurde.

Deshalb muß er die Seriennummer an die Bank melden, die mit ihrer Datenbank bereits ausbezahlter Seriennummern vergleicht. Falls die Zahl darin noch nicht vorkommt, wird sie eingetragen und der Händler bekommt sein Geld; andernfalls weigert sich die Bank zu zahlen.

Bei 10^{75} möglichen Nummern liegt die Wahrscheinlichkeit dafür, daß zwei Kunden, die eine (wirklich) zufällige Zahl wählen, dieselbe Nummer erzeugen, bei etwa $10^{-37{,}5}$. Die Wahrscheinlichkeit, mit jeweils einem Spielschein fünf Wochen lang hintereinander sechs Richtige im Lotto zu haben, liegt dagegen bei $\binom{49}{6}^{-5} \approx 5 \cdot 10^{-35}$, also etwa um den Faktor sechzig höher. Zwei gleiche Seriennummern sind also praktisch auszuschließen, wenn auch theoretisch möglich.

Falls wirklich einmal zufälligerweise zwei gleiche Seriennummern erzeugt worden sein sollten, kann das System nur funktionieren, wenn der zweite Geldschein mit derselben Seriennummer nicht anerkannt wird, so daß der zweite Kunde sein Geld verliert. Dies muß als eine

zusätzliche Gebühr gesehen werden, die mit an Sicherheit grenzender Wahrscheinlichkeit nie fällig wird, aber trotzdem nicht ausgeschlossen werden kann.

Da digitales Bargeld allerdings nur in kleinen Stückelungen sinnvoll ist (Geldscheinen im Millionenwert wären auf Grund ihrer Seltenheit nicht wirklich anonym und würden, wegen der damit verbundenen Möglichkeiten zur Geldwäsche, auch in keinem seriösen Wirtschaftssystem angeboten), wäre der theoretisch mögliche Verlust ohnehin nicht sehr groß.

Die hier skizzierte Idee für elektronisches Bargeld wurde von ihrem Erfinder DAVID CHAUM auch kommerziell realisiert mit einer Firma namens DigiCash. Er konnte mit mehreren Banken, darunter auch der Deutschen Bank, ins Geschäft kommen.

Die Deutsche Bank allerdings fand genau 26 Geschäftskunden, die bereit waren, Bezahlung in DigiCash zu akzeptieren, darunter etwa die Aktion „Brot für die Welt“, die auf digitale Spenden hoffte und nach mehreren Monaten Laufzeit auf knapp fünf Mark kam, oder die Universität Frankfurt, die Java-Applets vermarkten wollte. Ob dies an der Gebührenpolitik der Deutschen Bank lag oder daran, daß das Bedürfnis nach Bezahlung kleiner Beträge im Internet damals noch deutlich kleiner war als heute, läßt sich von außerhalb nur schwer beurteilen. Vielleicht lag der eigentliche Grund auch einfach darin, daß Kundenprofile für einige Banken und Kreditkartenunternehmen zu wertvoll sind, als daß sie ein echtes Interesse an anonymen Zahlungssystemen hätten.

CHAUMs Firma DigiCash beantragte 1998 Gläubigerschutz; die Deutsche Bank stellte ihren Versuch mit elektronischem Bargeld 2001 ein.

§5: Wie findet man Primzahlen für RSA?

Zur praktischen Anwendung von RSA brauchen wir zwei Primzahlen p und q, die nach derzeitigen Sicherheitsanforderungen etwa 1024 Bit, also 309 Dezimalstellen haben sollten. Die findet man natürlich nicht in Primzahltafeln.

a) Wie man es nicht machen sollte

Im Land der unbegrenzten Möglichkeiten gibt es trotzdem keine großen Probleme: Dort kann man Primzahlen, wie alles andere auch, einfach kaufen. Unter Sicherheitsgesichtspunkten ist das freilich nicht unbedingt die beste Strategie, denn erstens kann dann der Verkäufer der Primzahlen die gesamte verschlüsselte Korrespondenz des Käufers lesen und auch dessen elektronische Unterschrift nachmachen, und zweitens wird man nie ganz sicher sein können, ob Billiganbieter wie *Thrifty Primes* oder *Primes for a Buck* nicht gelegentlich dieselbe Primzahl an mehrere Kunden verkaufen, so daß ein Kunde versuchen kann, die öffentlich bekannten Moduln seiner Konkurrenten durch die gerade gekauften Primzahlen zu teilen und damit zumindest einen Teil davon zu faktorisieren. Auch wer selbst keine Primzahlen kauft, kann versuchen, die größten gemeinsamen Teiler der Moduln seiner Konkurrenten zu berechnen; sobald er bei zwei verschiedenen Moduln einen Wert ungleich eins erhält, kann er beide Moduln faktorisieren.

Sowohl aus Sicherheitsgründen als auch weil wir Mathematiker sind, sollten wir also unsere Primzahlen selbst erzeugen. Leider gibt es keine einfache Formel, die uns Primzahlen einer vorgegebenen Größe erzeugt, insbesondere keine möglichst zufälligen einer vorgegebenen Größe. So ist zwar spätestens seit EUKLID bekannt, daß es unendlich viele Primzahlen gibt: Gäbe es nämlich nur endlich viele Primzahlen $p_1, \ldots, p_r$, so wäre $p_1 \cdots p_r + 1$ weder eine Primzahl noch durch eine Primzahl teilbar, was offensichtlich unmöglich ist.

Das heißt nun aber nicht, daß auch Primzahlen beliebiger Länge bekannt wären; zwar hat die *Electronic Frontier Foundation*, die uns bereits beim DES-Cracker begegnet ist, Preise ausgeschrieben für die erste Primzahl mit mindestens einer Million, zehn Millionen, hundert Millionen *bzw.* einer Milliarde Dezimalstellen; bezahlen mußte sie bislang aber erst im Jahre 2000 die \$50 000 für eine Million Stellen und 2009 die \$100 000 für zehn Millionen Stellen.

Die größte derzeit bekannte Primzahl ist $2^{43\,112\,609} - 1$ mit 12 978 189 Dezimalstellen; sie wurde am 23. August 2008 im Rahmen der *Great Internet Mersenne Prime Search* (GIMPS) gefunden; siehe deren *home page* `www.mersenne.org`.

Wie bei allen Rekord-Primzahlen der letzten Jahre ist sie eine sogenannte MERSENNEsche Primzahl, d.h. eine Primzahl der Form $2^n - 1$. Nur zwischen 1989 und 1992 war eine andere Zahl ($391\,581 \cdot 2^{216\,193} - 1$) größte bekannte Primzahl; ansonsten war der Rekordhalter seit 1952 stets eine MERSENNE-Zahl.

$2^n - 1$ kann höchstens dann eine Primzahl sein, wenn auch n prim ist: Läßt sich n nämlich als Produkt uv zweier Zahlen $u, v > 1$ schreiben, so ist $2^u \equiv 1 \bmod 2^u - 1$, d.h.

$$2^n = 2^{uv} \equiv 1^v = 1 \bmod 2^u - 1 \quad \text{und} \quad 2^n - 1 \equiv 0 \bmod 2^u - 1$$

ist durch $2^u - 1$ teilbar.

Die Zelle des französischen Mönchs MARIN MERSENNE (1588–1648) war ein Treffpunkt für Mathematiker wie FERMAT, PASCAL und andere. MERSENNE selbst beschäftigte sich außer mit seinen Primzahlen auch mit Mechanik, wo er als erster GALILEIs Ideen außerhalb Italiens bekannt machte. Außerdem schrieb er ein Buch über Musik, Musikinstrumente und Akustik.

Für RSA sind MERSENNEsche Primzahlen freilich ungeeignet: Erstens sind im Augenblick nur 47 solche Zahlen bekannt, und zweitens sind fast alle entweder viel zu groß oder viel zu klein um als Faktoren sicherer und praktikabler RSA-Moduln in Frage zu kommen. Da das Beispiel der MERSENNE-Zahlen aber zeigt, daß man selbst bei Millionen von Dezimalstellen entscheiden kann, ob eine Zahl prim ist, können wir guter Hoffnung sein, daß es bei den für RSA benötigten Primzahlen mit gerade einmal ein paar hundert Dezimalstellen keine allzu großen Schwierigkeiten geben sollte.

b) Wie man es idealerweise machen sollte

Wie auch bei den klassischen Kryptoverfahren wird ein Gegner, der RSA knacken will, unter anderem versuchen, statistische Inhomogenitäten auszunutzen. Damit kann er Erfolg haben, wenn wir Verfahren benutzen die bestimmte Primzahlen (im Extremfall etwa nur MERSENNEsche Primzahlen) bevorzugen. Idealerweise sollte also jede Primzahl exakt dieselbe Chance haben.

Das einzige Verfahren, daß dies garantiert, besteht darin, daß man solange *echte* Zufallszahlen erzeugt, bis man eine Primzahl gefunden hat.

Wie wir noch sehen werden, ist die Erzeugung solcher Zahlen ohne Spezialhardware ziemlich aufwendig; man begnügt sich daher, wie so oft in der Kryptographie, meist mit Zahlen, die sich bezüglich der als realistisch erachteten Rechenmöglichkeiten der zu erwartenden Gegner wie wirklich zufällige Zahlen verhalten.

Algorithmisch erzeugte Zahlen, die sich bezüglich gewisser meist statistischer Kriterien wie echte Zufallszahlen verhalten, bezeichnet man als Pseudo-Zufallszahlen; alle Computeralgebrasysteme sowie die Unterprogrammbibliotheken der Betriebssysteme enthalten entsprechende Routinen. Diese erzeugen Folgen $(x_i)_{i\in\mathbb{N}}$ meist einfach durch Iteration einer Funktion f, d.h. $x_{i+1} = f(x_i)$. In Maple beispielsweise ist $f(x) = ax \bmod p$ mit

$$a = 427419669081 \quad \text{und} \quad p = 10^{12} - 11 = 999999999989\,;$$

p ist die größte zwölfstellige Primzahl. Die entstehende Folge ist natürlich periodisch (da nur Zahlen zwischen 0 und $p-1$ auftreten, ist das unvermeidlich), man kann aber zeigen, daß (abgesehen vom Startwert $x = 0$, der auf lauter Nullen führt) die Periode gleich $p-1$ ist, was für kryptographische Anwendungen mehr als ausreicht.

Der Generator liefert uns grundsätzlich nur Zufallszahlen kleiner p; aber auch das stört nicht weiter, denn wenn wir Zufallszahlen mit größerer Stellenzahl brauchen, können wir beispielsweise einfach Summen der Form $\sum_{i=0}^{n} x_{k+i} p^i$ betrachten. (Maple berechnet stattdessen die Summe $\sum_{i=0}^{n} x_{k+i} 10^{12(n-i)}$ und reduziert diese anschließend auf das Intervall, in dem das Ergebnis liegen soll.)

Das große Problem bei dieser Vorgehensweise ist, daß es nur $p-1$ Startwerte x_0 gibt. Daher gibt es auch, selbst wenn wir dreihundertstellige Zufallszahlen erzeugen, nur knapp $10^{12} \approx 2^{40}$ Möglichkeiten. Die kann ein entschlossener Gegner mit vertretbarem Aufwand durchprobieren.

Ein Zufallsgenerator, der für RSA-Primzahlen benutzt wird, muß also mehr verschiedene Startwerte zulassen als ein Gegner durchprobieren kann. Wenn wir, wie bei symmetrischen Blockchiffren, davon ausgehen, daß 2^{128} Möglichkeiten auch den entschlossensten Gegner überfordern, brauchen wir also einen Startwert mit mindestens 128 Bit und natürlich

auch einen kryptographisch guten, d.h. schwer durchschaubaren Generator. Im Kapitel über Hashfunktionen werden wir solche Generatoren kennenlernen und uns auch kurz mit der Erzeugung „echter“ Zufallszahlen befassen.

Im Augenblick gehen wir einfach davon aus, daß wir uns irgendwie Zufallszahlen oder Pseudozufallszahlen der gewünschten Größenordnung verschaffen können; unser nächstes Problem ist dann, wie man herausfindet, welche davon Primzahlen sind.

Damit wir realistisch abschätzen können, wie groß der Aufwand zur Primzahlsuche ist, wollen wir uns aber zunächst mit einer anderen Frage beschäftigen:

c) Wie dicht liegen die Primzahlen?

Dies ist ein sehr altes Problem, das immer noch nicht vollständig gelöst ist; die bekannte Antwort ist aber für praktische Zwecke gut genug. Die Mathematik, die im Beweis der folgenden Aussagen steckt, ist leider jenseits des zeitlichen Rahmens dieser Vorlesung – obwohl einige der Beweise inzwischen vergleichsweise sehr elementar geworden sind. Wer einigermaßen mit Funktionentheorie vertraut ist, sollte in der Lage sein, die Darstellung in

HELMUT KOCH: Einführung in die klassische Mathematik I, *Akademie-Verlag* und (in Lizenz) *Springer-Verlag*, 1986, §27

zu lesen; alle notwendigen Voraussetzungen aus Funktionentheorie, Zahlentheorie *usw.* sind im Buch selbst zu finden.

Wir bezeichnen die Anzahl der Primzahlen, die kleiner oder gleich einer Zahl x sind, mit $\pi(x)$. Demnach ist also $\pi(2) = 1$ und $\pi(3) = \pi(4) = 2$ und so weiter. Der englische Mathematiker SYLVESTER bewies 1892 folgende Verschärfung einer etwa vierzig Jahre älteren Ungleichung von TSCHEBYSCHEFF:

$$0{,}95695\frac{x}{\ln x} < \pi(x) < 1{,}04423\frac{x}{\ln x}\,,$$

$\pi(x)$ verhält sich also asymptotisch ungefähr wie $x/\ln x$. (Ein sehr elementarer Beweis einer schwächeren Ungleichung, bei der links und

rechts anstelle konkreter Zahlen nur irgendwelche Konstanten c_1, c_2 stehen, wird im Skriptum der Zahlentheorievorlesung des kommenden Semesters zu finden sein.)

Nach dem Primzahlsatz, den GAUSS vermutete und den HADAMARD und DE LA VALLÉE POUSSIN 1896 bewiesen, ist die Asymptotik sogar exakt, d.h.

$$\lim_{x\to\infty} \frac{\pi(x)}{x/\ln x} = 1\,.$$

Numerisch besser ist die (ebenfalls auf GAUSS zurückgehende) Approximation durch den Integrallogarithmus:

$$\pi(x) = \int_2^x \frac{d\xi}{\ln \xi} + O(xe^{-c\sqrt{\ln x}})$$

mit einer (im Prinzip berechenbaren) Konstanten $c > 0$.

Die angegebene Fehlerschranke ist wahrscheinlich zu pessimistisch; falls die RIEMANNsche Vermutung über die Nullstellen der sogenannten Zeta-Funktion $\zeta(s)$ wahr ist, erhält man eine deutlich bessere Schranke. Diese Vermutung ist allerdings bislang immer noch offen; sie ist eines der sieben Millennium Problems, für deren Lösung das Clay Mathematics Institute einen Preis von jeweils einer Million Dollar ausgesetzt hat.

Für uns wichtig ist diese Folgerung: Unter den Zahlen der Größenordnung N haben der Primzahlen die Dichte $1/\ln N$, d.h. der Abstand zwischen zwei Primzahlen liegt im Mittel bei etwa $\ln N$. Für $N = 10^n$ ist dies $\ln 10^n = n \ln 10 \approx 2{,}3n$; bei hundertstelligen Zahlen ist also etwa jede 230ste prim und bei 1024-Bit-Zahlen ungefähr jede 710. Allerdings sind dies natürlich nur grobe Anhaltspunkte, denn die tatsächliche Verteilung der Primzahlen zeigt enorme Schwankungen: So hat man bislang in allen untersuchten Größenordnungen sowohl Primzahlzwillinge gefunden, d.h. Paare $(p, p+2)$ von Primzahlen, als auch primzahlenfreie Intervalle, die deutlich länger sind als $\ln N$: Am anderen Ende sagt uns der Satz von BERTRAND nur, daß es zwischen $N > 1$ und $2N$ stets mindestens eine Primzahl gibt; neuere Verschärfungen zeigen, daß es

für hinreichend große N mehr geben muß, aber viel mehr wissen wir nicht.

Wenn wir so sicherheitsbewußt sind, echte 1024-Bit-Zufallszahlen auf Primalität zu testen, sagt uns die obige Abschätzung also auch, daß wir im Mittel 710 Zahlen testen müssen, bevor wir die erste Primzahl gefunden haben.

d) Das Sieb des Eratosthenes

Das wohl älteste Verfahren, das Primzahlen effizienter als durch Probedivisionen liefert, ist das von ERATOSTHENES angegebene Siebverfahren. In seiner klassischen Form dient es dazu, alle Primzahlen unterhalb einer Schranke N zu bestimmen. Dazu schreibt man die Zahlen von Eins bis N (oder auch nur die ungeraden darunter) in eine Reihe, streicht die Nicht-Primzahl Eins und sodann, solange die kleinste noch nicht gestrichene Zahl nicht größer als $\sqrt{N}$ ist, deren sämtliche Vielfache. Was am Ende übrigbleibt, sind genau die Primzahlen aus der Liste.



ERATOSTHENES (Ερατοσθένες) wurde 276 v.Chr. in Cyrene im heutigen Libyen geboren, wo er zunächst von Schülern des Stoikers ZENO ausgebildet wurde. Danach studierte er noch einige Jahre in Athen, bis ihn 245 der Pharao PTOLEMAIOS III als Tutor seines Sohns nach Alexandrien holte. 240 wurde er dort Bibliothekar der berühmten Bibliothek im Museion.
Heute ist er außer durch sein Sieb vor allem durch seine Bestimmung des Erdumfangs bekannt. Er berechnete aber auch die Abstände der Erde von Sonne und Mond und entwickelte einen Kalender, der Schaltjahre enthielt. 194 starb er in Alexandrien, nach einigen Überlieferungen, indem er sich, nachdem er blind geworden war, zu Tode hungerte.

In dieser Form ist das Sieb für uns nutzlos: Wenn wir eine dreihundertstellige Primzahl brauchen, können wir unmöglich zunächst eine Liste aller Primzahlen unterhalb von 10^{150} erzeugen. Trotzdem kann es uns auch da eine Hilfe sein: Wenn die Liste anstelle der ersten N natürlichen Zahlen die Zahlen aus einem Suchintervall $[N, N+\ell]$ enthält, können wir immer noch die Vielfachen kleiner Primzahlen aussieben; wir müssen nur für jede Primzahl p, mit der wir sieben, ihr erstes Vielfaches im

Suchintervall bestimmen. Dieses ist offensichtlich $N + p - (N \bmod p)$, und ab dort wird jeder p-te Eintrag in der Liste gestrichen.

Natürlich müssen hier die Primzahlen p aus einer separaten Liste kommen, und p wird um Größenordnungen kleiner sein als $\sqrt{N+\ell}$. Somit können wir nicht erwarten, daß alle nicht ausgestrichenen Listenelemente Primzahlen sind, und müssen diese Zahlen weiteren Tests unterziehen. Da diese, wie wir sehen werden, erheblich aufwendiger sind als das Sieb des ERATOSTHENES, lohnt es sich aber trotzdem, mit dem Sieb zu beginnen.

Die Feinverteilung der Primzahlen ist sehr inhomogen; daher sollte man bei der Wahl der Intervallänge ℓ eine gewisse Sicherheitsreserve einplanen und ℓ so wählen, daß im Durchschnitt etwa drei bis zehn Primzahlen in $[N, N+\ell]$ zu erwarten sind, d.h. $\ell \approx a \ln \ell$ mit einem a zwischen drei und zehn. Der Intervallanfang N muß natürlich zufällig gewählt werden.

Die Primzahlen, die wir bei einem solchen Verfahren erwarten können, sind *nicht* gleichverteilt: Ist p eine Primzahl und q die größte Primzahl echt kleiner p, so gibt es offenbar genau $p - q$ Anfangswerte N, die zu p als erster Primzahl in $[N, N+\ell]$ führen. Um jeder Primzahl dieselbe Chance zu geben, müßte man Zufallszahlen auf Primalität testen und, sofern eine Zahl den Test nicht besteht, zur nächsten Zufallszahl übergehen. Der Aufwand hierfür ist erheblich größer als der für das Sieb, da wir im Mittel $\ln p$ Zahlen testen müssen, bevor wir eine Primzahl p finden. Da bislang keine Verfahren bekannt sind, wie ein Gegner die bei Verwendung des Siebs resultierenden Abweichungen von einer Gleichverteilung ausnutzen kann, wird dieser Nachteil angesichts der gewaltigen Zeitersparnis oft in Kauf genommen.

e) Der Fermat-Test

Unabhängig davon, ob wir das Sieb des ERATOSTHENES benutzen oder nicht, brauchen wir auf jeden Fall noch weitere Primzahltest. Der kleine Satz von FERMAT gibt uns sofort eine Aussage darüber, wann eine Zahl p *nicht* prim ist:

Falls für eine natürliche Zahl $1 \le a \le p-1$ *gilt* $a^{p-1} \not\equiv 1 \bmod p$*, kann* p *keine Primzahl sein.*

Beispiel: Ist $F_{20} = 2^{2^{20}} + 1$ eine Primzahl? Falls ja, ist nach dem kleinen Satz von FERMAT insbesondere

$$3^{F_{20}-1} = 1 \bmod F_{20}\,.$$

Nachrechnen zeigt, daß

$$3^{(F_{20}-1)/2} \not\equiv \pm 1 \bmod F_{20}\,,$$

die Zahl ist also nicht prim. (Das „Nachrechnen“ ist bei dieser 315653-stelligen Zahl natürlich keine Übungsaufgabe für Taschenrechner: 1988 brauchte eine Cray X-MP dazu 82 Stunden, eine Cray-2 immerhin noch zehn; siehe *Math. Comp.* **50** (1988), 261–263. Die anscheinend etwas weltabgewandt lebenden Autoren meinten, das sei die teuerste bislang produzierte 1-Bit-Information.)

Die Umkehrung der obigen Aussage gilt leider nicht: Es gibt, wie man inzwischen weiß, unendlich viele Nichtprimzahlen n, für die trotzdem $a^{n-1} \equiv 1 \bmod n$ ist für jedes zu n teilerfremde a; das sind die sogenannten CARMICHAEL-Zahlen. Trotzdem wird es für große Zahlen zunehmend unwahrscheinlich, daß eine Zahl p für auch nur ein a den obigen Test besteht, ohne Primzahl zu sein. Rechnungen von

SU HEE KIM, CARL POMERANCE: The probability that a Random Probable Prime is Composite, *Math. Comp.* **53** (1989), 721–741

geben folgende obere Schranke für die Fehlerwahrscheinlichkeit ε:

$p \approx 10^{60}$	10^{70}	10^{80}	10^{90}	10^{100}
$\varepsilon \le 7{,}16 \cdot 10^{-2}$	$2{,}87 \cdot 10^{-3}$	$8{,}46 \cdot 10^{-5}$	$1{,}70 \cdot 10^{-6}$	$2{,}77 \cdot 10^{-8}$
$p \approx 10^{120}$	10^{140}	10^{160}	10^{180}	10^{200}
$\varepsilon \le 5{,}28 \cdot 10^{-12}$	$1{,}08 \cdot 10^{-15}$	$1{,}81 \cdot 10^{-19}$	$2{,}76 \cdot 10^{-23}$	$3{,}85 \cdot 10^{-27}$
$p \approx 10^{300}$	10^{400}	10^{500}	10^{600}	10^{700}
$\varepsilon \le 5{,}8 \cdot 10^{-29}$	$5{,}7 \cdot 10^{-42}$	$2{,}3 \cdot 10^{-55}$	$1{,}7 \cdot 10^{-68}$	$1{,}8 \cdot 10^{-82}$
$p \approx 10^{800}$	10^{900}	10^{1000}	10^{2000}	10^{3000}
$\varepsilon \le 5{,}4 \cdot 10^{-96}$	$1{,}0 \cdot 10^{-109}$	$1{,}2 \cdot 10^{-123}$	$8{,}6 \cdot 10^{-262}$	$3{,}8 \cdot 10^{-397}$

(Sie geben natürlich auch eine allgemeine Formel an, jedoch ist diese zu grausam zum Abtippen.)

Selbst wenn wir noch mit 512-Bit-Moduln arbeiteten und somit knapp achtzigstellige Primzahlen bräuchten, läge also die Fehlerwahrscheinlichkeit bei nur etwa 10^{-5}; um sie weiter zu erniedrigen, müssten wir einfach mit mehreren zufällig gewählten Basen testen und hätten dann beispielsweise be drei verschiedenen Basen eine Irrtumswahrscheinlichkeit von höchstens etwa 10^{-15}, daß alle drei Tests das falsche Ergebnis liefern. (Dieses Argument gilt strenggenommen nur unter der Voraussetzung, daß es sich bei fehlgeschlagenen FERMAT-Tests mit verschiedenen Primzahlen um unabhängige Ereignisse handelt, was wahrscheinlich nicht der Fall ist. Die allgemeine experimentelle Erfahrung mit Primzahlen zeigt jedoch, daß sie sich zumindest in den überprüften Bereichen oft wie zufallsverteilt verhalten.)

Bei den etwa 155-stelligen Zahlen, die wir für 1024-Bit-Moduln brauchen, hat schon ein einziger Test eine geringere Fehlerwahrscheinlichkeit als 10^{-15}, so daß es sich nur selten lohnt, einen größeren Aufwand zu treiben. Die Bundesnetzagentur empfiehlt allerdings, bei probabilistischen Primzahltests eine Irrtumswahrscheinlichkeit von höchstens $2^{-80} \approx 8{,}27 \cdot 10^{-25}$ zuzulassen, ab 2010 sogar nur noch $2^{-100} \approx 7{,}89 \cdot 10^{-31}$. Bei den für 2048-Bit-Moduln notwendigen über dreihundertstelligen Primzahlen wird selbst letzterer Wert schon bei einem einzigen FERMAT-Test unterschritten.

Gelegentlich werden Zahlen, die einen FERMAT-Test bestanden haben, als „wahrscheinliche Primzahlen“ bezeichnet. Das ist natürlich Unsinn: Eine Zahl ist entweder *sicher* prim oder *sicher* zusammengesetzt; für Wahrscheinlichkeiten gibt es hier keinen Spielraum. Besser ist der gelegentlich zu hörende Ausdruck „industrial grade primes“, also „Industrieprimzahlen“, der ausdrücken soll, daß wir als Mathematiker zwar nicht entscheiden können, ob die Zahl wirklich prim ist, daß sie aber für industrielle Anwendungen gut genug ist.

Man kann das FERMAT-Verfahren ohne großen Aufwand noch etwas verbessern zu einem Test, den erstmalig ARTJUHOV 1966/67 vorschlug. Die Grundidee ist folgende: Falls p eine Primzahl ist, ist $\mathbb{Z}/p$ ein Körper.

Ist dort $a^n = 1$ für eine gerade Zahl $n = 2m$, so erfüllt $x = a^m$ die Gleichung $x^2 = 1$. Da ein quadratisches Polynom über einem Körper höchstens zwei Nullstellen haben kann, muß also $x = \pm 1$ sein. (Falls $\mathbb{Z}/p$ kein Körper ist, p also keine Primzahl, gilt dies nicht: Wäre etwa p das Produkt zweier ungerader Primzahlen, so gäbe es vier Lösungen der Gleichung $x^2 = 1$ in $\mathbb{Z}/p$, für $p = 15$ beispielsweise $x = 1, 4, 11$ und 14.)

Dies läßt sich folgendermaßen ausnutzen: Für eine zu testende ungerade Zahl p schreiben wir $p - 1 = 2^r u$ mit einer ungeraden Zahl u. Sodann wählen wir eine Basis a zwischen 2 und $p-2$ und berechnen $a^u \bmod p$. Ist diese Zahl gleich eins, so ist erst recht $a^{p-1} \equiv 1 \bmod p$, und der Test ist bestanden. Dasselbe gilt für das Ergebnis $p - 1 \equiv -1 \bmod p$, denn $r \geq 1$ für eine ungerade Zahl p.

Andernfalls quadriere man das Ergebnis höchstens $r - 1$ mal modulo p; dies liefert die Potenzen $a^{2^s u} \bmod p$ für $s = 1, \ldots, r - 1$. Sobald ein Ergebnis gleich $p - 1$ wird, bricht der Test ab mit dem Ergebnis *bestanden;* offensichtlich ist dann $a^{p-1} \equiv 1 \bmod p$. Falls keines der Ergebnisse gleich $p - 1$ ist, kann p keine Primzahl sein, denn dann ist entweder $a^{p-1} \not\equiv 1 \bmod p$, oder aber wir haben eine Zahl gefunden, die von ± 1 verschieden ist, aber trotzdem Quadrat Eins hat, was in einem Körper nicht möglich ist.

Wie MONIER und RABIN 1980 gezeigt haben, ist die Anzahl der Basen a, die ein *falsches* Ergebnis liefern, für die eine zusammengesetzte Zahl p also den Test besteht, kleiner als $p/4$; so etwas wie CARMICHAEL-Zahlen kann es für diesen verschärften Test daher nicht geben.

Natürlich kennt die Mathematik auch Verfahren, um exakt zu entscheiden, ob eine Zahl prim ist oder nicht; das einfachste besteht darin, alle potentiellen Primteiler einfach auszuprobieren. Wie man im Zahlentheorieskriptum des letzten Semesters im Kapitel über Primzahltests nachlesen kann, läßt sich auch der FERMAT-Test zu einem solchen Verfahren ausbauen, allerdings nur falls man die Zahl $p - 1$ in ihre Primfaktoren zerlegen kann, was für RSA-Primzahlen nur selten der Fall sein dürfte. Dort wird auch ein Verfahren beschrieben, das MANINDRA AGRAWAL, NEERAJ KAYAL und NITIN SAXENA im August 2002 vorstellten: Sie

entwickelten auf der Grundlage von FERMAT einen Test, der zumindest asymptotisch schneller ist als alle anderen bislang bekannten Verfahren. Für die Praxis von RSA freilich ist dieses Verfahren bedeutungslos, da gängige, asymptotisch langsamere Alternativen, bei den hier benötigten Größenordnungen deutlich schneller sind.

Diese anderen Algorithmen benutzen anspruchsvollere Mathematik als FERMAT; die meisten arbeiten mit Charaktersummen und/oder elliptischen Kurven. Da sich FERMAT, wie wir gesehen haben, nur selten irrt, sei auf ihre Behandlung verzichtet.

§6: Sicherheit und Sparsamkeit

Nicht erst seit McKinsey & Co sind Industrieunternehmen sehr kostenbewußt. Angesichts der hohen Schäden, die durch Industriespionage entstehen können, sollte man daher meinen, daß sie lieber die deutlich niedrigeren Kosten für kryptographische und sonstige Sicherheitsstandards aufwenden. Tatsächlich geschieht das aber oft erst nach dem ersten erfolgreichen Angriff, denn bis dahin sind eben Sicherheitsausgaben ein laufender Bilanzposten, Schäden aber nur eine vage Möglichkeit, von der man hoffentlich verschont bleiben wird. Von daher ist zu verstehen, daß auch beim Einsatz von Kryptographie gespart werden muß wo man nur kann. Gerade bei RSA zeigt sich allerdings, daß fast jede Sparmöglichkeit gleichzeitig ein Sicherheitsproblem ist. Bei zu kleinen Moduln ist das klar; in diesem Paragraphen soll diskutiert werden, wo sonst noch Probleme auftauchen können.

a) Primzahlen sind Wegwerfartikel

Die Suche nach einer Primzahl mit 1024 Bit kann auf einem nicht sonderlich leistungsfähigen PC rund eine Minute dauern; wenn man viele Schlüssel erzeugen muß, bietet sich also an, mit den teuren Primzahlen sparsam umzugehen.

Genau das darf man aber natürlich, wie bereits erwähnt, auf keinen Fall tun: Wenn jemals zwei unterschiedliche Moduln M, N dieselbe Primzahl p enthalten, kann p leicht als ggT von M und N berechnet

werden, so daß beide Moduln faktorisiert sind. Der EUKLIDische Algorithmus erfordert auch für 2048-Bit-Zahlen auf einem Standard-PC einen Aufwand, der höchstens im unteren Sekundenbereich liegt; es ist also problemlos möglich, auch bei einer Liste von mehreren Tausend Moduln für jedes Paar den ggT zu berechnen.

Der sichere Umgang mit Primzahlen besteht darin, die Primzahlen sofort nach Berechnung des öffentlichen und des privaten Schlüssels zu vernichten. Die Wahrscheinlichkeit, daß später wieder einmal eine dieser Primzahlen als Zufallsprimzahl auftaucht, liegt bei vernünftiger Implementierung des „Zufalls“ deutlich unter 10^{-100} und kann daher für alle praktischen Zwecke ignoriert werden.

b) Jeder braucht seinen eigenen RSA-Modul

Da die Erzeugung von Primzahlen teuer ist, könnte ein sparsames Unternehmen versucht sein, einen gemeinsamen Modul N für alle Mitarbeiter zu erzeugen und jedem Mitarbeiter einen individuellen öffentlichen Exponenten e zusammen dem zugehörigen privaten Exponenten d zuzuordnen. Die Verteilung könnte etwa so realisiert sein, daß nur der Sicherheitschef, der ohnehin alles lesen darf, die Faktorisierung von N kennt; er erzeugt dann für jeden neuen Mitarbeiter einen geeigneten Exponenten e und berechnet dazu den privaten Exponenten d.

Zumindest im Bankenbereich, wo ein Großteil der Nachrichten elektronische Zahlungsanweisungen sind, muß es eine Instanz geben, die alles lesen und kontrollieren kann; für sich allein betrachtet sind die Befugnisse des „Sicherheitschefs“ also nicht unbedingt ein Nachteil.

Tatsächlich kennt in diesem Modell aber nicht nur der Sicherheitschef die Faktorisierung von N, sondern auch jeder Mitarbeiter mit Interesse an der Zahlentheorie oder einem kompetenten Bekannten. Insbesondere kann also zumindest prinzipiell jeder Mitarbeiter die Post eines jeden anderen lesen und sich auch für diesen ausgeben. Die gleichen Möglichkeiten hätte ein Außenstehender, der nur einen einzigen privaten Exponenten d kaufen oder ausspionieren kann.

Der Grund dafür ist, daß die Kenntnis des öffentlichen *und* des privaten Exponenten zur Faktorisierung von N führt:

$N = pq$ sei Produkt zweier Primzahlen, e sei der öffentliche Exponent und d der private. Dann ist für alle $a \in \mathbb{Z}$

$$(a^e)^d = a^{ed} \equiv a \bmod N\,;$$

für ein zu N teilerfremdes a ist also

$$a^{de-1} \equiv 1 \bmod N\,.$$

Wir schreiben $de - 1 = 2^r \cdot u$ mit einer ungeraden Zahl u und betrachten die Folge der Zahlen

$$a^u \bmod N, \quad a^{2u} \bmod N, \quad \ldots, \quad a^{2^r u} \bmod N\,.$$

Die letzte dieser Zahlen ist stets gleich eins; wenn wir Pech haben, ist für das gerade betrachtete a auch schon die erste gleich eins. Wenn nicht, gibt es einen kleinsten Exponenten s, so daß

$$a^{2^s u} \not\equiv 1 \bmod N \quad \text{und} \quad a^{2^{s+1} u} \equiv 1 \bmod N\,.$$

Es könnte sein, daß dann $a^{2^s u} \equiv -1 \bmod N$ ist, aber da N eine zusammengesetzte Zahl ist, muß das nicht der Fall sein: Modulo 15 ist beispielsweise auch vier eine Zahl mit Quadrat eins.

Modulo einer Primzahl hat eins natürlich nur die beiden Quadratwurzeln ± 1, denn die natürlichen Zahlen modulo einer Primzahl bilden einen Körper, und in einem Körper kann das quadratische Polynom $x^2 - 1$ höchstens zwei Nullstellen haben.

Ist aber $x^2 \equiv 1 \bmod N = pq$, so ist auch $x^2 \equiv 1 \bmod p$ und $\bmod q$, also $x \equiv \pm 1 \bmod p$ und $x \equiv \pm 1 \bmod q$, wobei nicht in beiden Fällen das gleiche Vorzeichen stehen muß. In der Hälfte aller Fälle werden beide Vorzeichen gleich sein; dann ist $x \equiv \pm 1 \bmod N$ mit demselben Vorzeichen.

Ist aber etwa $x \equiv 1 \bmod p$ und $x \equiv -1 \bmod q$, so ist

$$p = \mathrm{ggT}(x - 1, N) \quad \text{und} \quad q = \mathrm{ggT}(x + 1, N)\,;$$

sobald wir eine solche Zahl kennen, haben wir N also faktorisiert.

Wenn wir mit einem zufällig gewählten a so wie oben verfahren, werden wir in etwa der Hälfte aller Fälle eine von ± 1 verschiedene Quadratwurzel der Eins erhalten. Mit nur wenigen Werten für a bekommen wir daher praktisch sicher eine Faktorisierung von N.

c) Der chinesische Restesatz

Das Argument im vorigen Abschnitt kann auch zu einer schnelleren Durchführung der RSA Ver- und Entschlüsselung zu schnelleren elektronischen Unterschriften führen: Die Berechnung von $m^e \bmod N$ *bzw.* $m^d \bmod N$ besteht aus Quadrierungen und Multiplikationen modulo N; der Aufwand dafür steigt quadratisch mit der Ziffernlänge von N. Kennt man also die Faktorisierung $N = pq$, kann man die beiden Werte $a^d \bmod p$ und $m^d \bmod q$ in jeweils einem Viertel der Zeit für $m^d \bmod N$ berechnen, beide zusammen also in der halben Zeit. Das Gesamtergebnis ist dadurch eindeutig bestimmt, denn da p und q teilerfremd sind, ist jede Zahl die modulo p und modulo q bekannt ist, auch modulo N eindeutig bestimmt.

Dies läßt sich leicht konstruktiv durchführen: Mit dem erweiterten Euklidischen Algorithmus findet man Zahlen α, β, so daß $\alpha p + \beta q = 1$ ist; dann ist

$$\beta q \equiv 1 \bmod p \quad \text{und} \quad \alpha p \equiv 1 \bmod q\,.$$

Für zwei beliebig vorgegebene Zahlen a, b ist entsprechend

$$a\beta q \equiv a \bmod p \quad \text{und} \quad b\alpha p \equiv b \bmod p$$

eine Lösung der Kongruenz

$$x \equiv a \bmod p \quad \text{und} \quad x \equiv b \pmod q\,.$$

Da α und β nur einmal für p und q berechnet werden müssen, ist der Aufwand für das Zusammensetzen der Restklassen modulo p und modulo q zu einer Restklasse modulo N sehr gering.

Mit diesem Verfahren läßt sich der Aufwand für eine elektronische Unterschrift also praktisch halbieren, was vor allem dann von Bedeutung ist, wenn die Unterschrift mit einer Smartcard geleistet wird.

Leider ist das Verfahren aber mit einem potentiell katastrophalen Risiko verbunden: Falls bei einer der beiden Rechnungen

$$a \leftarrow m^d \bmod p \quad \text{und} \quad b \leftarrow m^d \bmod q$$

ein Fehler auftritt, ist das Ergebnis für $m^d \bmod N$ korrekt modulo der einen, nicht aber modulo der anderen Primzahl. Nehmen wir etwa an,

das Ergebnis modulo q sei falsch; dann wird also eine Unterschrift u berechnet, die modulo p kongruent ist zu m^d, nicht aber modulo q.

Zum Überprüfen der Unterschrift berechnet der Empfänger $u^e \bmod N$ und vergleicht dies mit m; im vorliegenden Beispiel stimmen die beiden Zahlen nur modulo p überein, nicht aber modulo q, sie sind also insbesondere verschieden, so daß die Unterschrift nicht akzeptiert wird.

Da $u^e - m$ aber durch p teilbar ist und nicht durch q, kann der Prüfer nun p als den ggT von $u^e - m$ und N berechnen und somit N faktorisieren; er kann also künftig die Unterschrift des anderen nachmachen.

Hardwarefehler, die bei der Berechnung von einem der beiden Teilergebnisse zu einem falschen Ergebnis führen sind sicherlich sehr seltene Ereignisse, allerdings kann man dem nachhelfen: Durch Magnetfelder, Mikrowelle, Hitze, mechanische Belastung und ähnliches läßt sich die Karte durchaus so beeinflussen, daß Fehler wahrscheinlich, aber nicht zu wahrscheinlich werden. Dann besteht eine realistische Chance, daß genau eines der beiden Zwischenergebnisse falsch berechnet wird und die Faktorisierung von N gelingt.

Sicherer ist es also auf jeden Fall, trotz des rund doppelt so hohen Aufwands direkt modulo N zu rechnen; die oben aufgestellte Regel, wonach man die Primzahlen so schnell wie möglich vergessen sollte, behält auch hier ihren Sinn.

Nicht vergessen sollten wir allerdings die Methode, eine Zahl modulo N aus ihren Restklassen modulo gewisser Teiler von N zu bestimmen, denn sie hat noch viele andere, auch kryptographisch wichtige Anwendungen. Daher möchte ich den dahinter stehenden sogenannten *Chinesischen Restesatz* hier allgemein formulieren. Er wurde angeblich früher von chinesischen Generälen benutzt, um Truppenstärken zu berechnen, indem sie die Soldaten in Reihen verschiedener Breite antreten ließen und dabei jeweils nur die Anzahl der Soldaten in der letzten Reihe zählten.

Chinesischer Restesatz: Die natürlichen Zahlen $d_1, \ldots, d_r$ seien paarweise teilerfremd und $N = d_1 \cdots d_r$ sei ihr Produkt. Dann hat das

Gleichungssystem

$$x \equiv a_1 \bmod d_1, \quad \ldots, \quad x \equiv a_r \bmod d_r$$

für jede Wahl der a_i eine modulo N eindeutig bestimmte Lösung; diese kann mit Hilfe des erweiterten EUKLIDischen Algorithmus berechnet werden.

Beweis: Für nur zwei Zahlen $d_1 = p$ und $d_2 = q$ haben wir das oben nachgerechnet, wobei offensichtlich nur eine Rolle spielte, daß p und q teilerfremd sind, nicht aber, daß sie Primzahlen sind. Der allgemeine Fall folgt durch vollständige Induktion, denn auch $d_1 \cdots d_s$ und $d_1 \cdots d_s d_{s+1}$ sind teilerfremd. ∎

d) Kleine öffentliche Exponenten und Kettenbriefe

Da der Verschlüsselungsaufwand bei RSA proportional zur Ziffernzahl des Exponenten ansteigt, sind kleine Verschlüsselungsexponenten e sehr beliebt; besonders populär sind $e = 3$ und $e = 2^{16} + 1$.

Wie wir bereits gesehen haben, ist so etwas katastrophal, wenn wir einen kleinen Block mit $e = 3$ verschlüsseln; aber natürlich wissen wir bereits seit langem, daß man (auch aus anderen Gründen) jeden Block vor der Verschlüsselung durch Zufallsbits auffüllen muß. Bei der Betrachtung von Normen für elektronische Unterschriften werden wir sehen, daß es bei unsachgemäßer Handhabung auch noch weitere Probleme insbesondere mit $e = 3$ geben kann.

Hier wollen wir einen Fall betrachten, in dem es Probleme geben muß: Wenn nämlich dieselbe Nachricht (oder derselbe Nachrichtenteil, wie z.B. eine Anlage oder ein Block ASCII-Kunst am Ende) an mehrere Empfänger geht.

Nehmen wir an, die Nachricht m werde an drei Empfänger geschickt, deren öffentliche Schlüssel $(N_1, 3)$, $(N_2, 3)$ und $(N_3, 3)$ seien. Verschickt werden also die drei Blöcke

$$m^3 \bmod N_1, \quad m^3 \bmod N_2 \quad \text{und} \quad m^3 \bmod N_3 \,.$$

Ein Gegner, der alle drei abfängt, kann dann nach dem chinesischen Restesatz $m^3 \bmod N_1 N_2 N_3$ berechnen, und da m kleiner als jedes N_i sein muß, kennt er damit m^3. Die Berechnung der Kubikwurzel auch einer sehr großen Zahl ist vom Aufwand her mit einer Division vergleichbar, liegt also höchstens im unteren Sekundenbereich.

(Falls N_1, N_2, N_3 nicht paarweise teilerfremd sein sollten, merkt man das bei der Anwendung des erweiterten EUKLIDischen Algorithmus und hat dann sogar eine Faktorisierung von mindestens zwei Moduln, was dann über die privaten Exponenten insbesondere auf die Nachricht führt.)

e) Kleine private Exponenten

Da elektronische Unterschriften häufig mit Smartcards oder in Zukunft vielleicht auch Mobiltelephonen und ähnlichen Geräte mit vergleichsweise schwacher Rechenleistung erzeugt werden, bietet sich an, nicht den öffentlichen, sondern den privaten Exponenten möglichst klein zu wählen. Ein privater Exponent drei wäre natürlich unmöglich, denn der private Exponent ist schließlich geheim und darf nicht durch systematisches Durchprobieren kleiner Zahlen gefunden werden.

Systematisches Durchprobieren ist aber, wie wir bei der Diskussion symmetrischer Kryptoverfahren gesehen haben, mit heutiger Technologie nur bis zu etwa 2^{80} Fällen möglich; 2^{128} Möglichkeiten gelten nach Ansicht praktisch aller öffentlich publizierender Experten heute als sicher. Ein privater Exponent mit 128 statt 2048 Bit führt zu einer Reduktion des Rechenaufwands um den Faktor 16, was gerade bei Smartcards spürbar sein sollte.

Leider gilt aber auch hier wieder, daß Rechenerleichterungen zu Sicherheitsmängeln führen; ein privater Exponent mit 128 Bit würde bei einem Modul von 1024 oder 2048 Bit innerhalb von Sekunden zu dessen Primfaktorzerlegung führen.

Der Grund ist folgender: Der öffentliche Exponent e und der private Exponent d erfüllen die Gleichung

$$de - k(p-1)(q-1) = 1$$

mit einer natürlichen Zahl k. Division durch $d(p-1)(q-1)$ macht daraus

$$\frac{e}{(p-1)(q-1)} - \frac{k}{d} = \frac{1}{d(p-1)(q-1)} \,.$$

Der linke Bruch hat einen Nenner in der ungefähren Größenordnung des Moduls $N = pq$; davon subtrahiert wird ein Bruch mit Nenner d, und die rechte Seite der Gleichung sagt uns, daß die Differenz sehr klein ist.

Ist also der private Exponent d klein, so kann der linksstehende Bruch durch einen Bruch mit sehr viel kleinerem Nenner sehr gut approximiert werden. Damit kann ein Gegner noch nichts anfangen, denn er kennt den Nenner $(p-1)(q-1)$ nicht; andererseits kennt er $N = pq$, und die Differenz ändert sich nicht sehr, falls man den Nenner durch N ersetzt:

$$\begin{aligned}\left|\frac{e}{N}-\frac{k}{d}\right| &= \left|\frac{e}{N}-\frac{e}{(p-1)(q-1)}+\frac{e}{(p-1)(q-1)}-\frac{k}{d}\right| \\ &\le \left|\frac{e(p-1)(q-1)-epq}{N(p-1)(q-1)}\right|+\frac{1}{d(p-1)(q-1)} \\ &= \frac{e(p+q-1)}{N(p-1)(q-1)}+\frac{1}{d(p-1)(q-1)}\,.\end{aligned}$$

Da p und q in der Größenordnung von $\sqrt{N}$ liegen, ist auch das noch eine recht kleine Zahl.

Bei kleinem d kann sich das ein Gegner mittels des folgenden Satzes zunutze machen:

Satz: Für die reelle Zahl $x > 0$ gebe es teilerfremde natürliche Zahlen a, b derart, daß

$$\left|x-\frac{a}{b}\right| < \frac{1}{2b^2}\,.$$

Dann ist a/b eine Konvergente der Kettenbruchentwicklung von x.

Um mit diesem Satz etwas anfangen zu können, müssen wir zunächst wissen, was die Kettenbruchentwicklung einer reellen Zahl ist. Diese berechnet sich nach folgendem Algorithmus, in dem $[x]$ für eine reelle Zahl x stets die größte ganze Zahl $n \le x$ bezeichnet:

1. Schritt: Setze $x_0 = x$ und $a_0 = [x]$.

n-ter Schritt, $n \ge 1$: Falls $x_{n-1} = a_{n-1}$ ist, bricht der Algorithmus an dieser Stelle ab; andernfalls setze

$$x_n = \frac{1}{x_{n-1}-a_{n-1}} \quad \text{und} \quad a_n = [x_n]\,.$$

Die n-te Konvergente dieser Kettenbruchentwicklung ist die rationale Zahl

$$a_0+\cfrac{1}{a_1+\cfrac{1}{a_2+\cfrac{1}{a_3+\cfrac{\cdots}{a_{n-1}+\cfrac{1}{a_n}}}}}\,.$$

Der Beweis des obigen Satzes ist elementar, aber langwierig; Interessenten finden ihn im Zahlentheorieskriptum des kommenden Semesters.

Falls man diesen Satz anwenden kann, läßt sich d also bestimmen, indem man die Nenner der Konvergenten der Kettenbruchentwicklung von e/N bestimmt und jeweils durch Ausprobieren nachprüft, ob für einen Zufallsblock a die gewünschte Beziehung $a^{de} \equiv a \bmod N$ gilt.

Eine einfache Abschätzung zeigt, daß er für p und q von etwa gleicher Größe anwendbar ist, sofern d höchstens die Größenordnung von etwa $\sqrt[4]{N}$ hat; neuere, etwas aufwendigere Untersuchungen zeigen, daß auch man d auch noch für $d < N^{0{,}289}$ rekonstruieren kann. Fachleute erwarten, daß möglicherweise sogar alle $d < \sqrt{N}$ unsicher sind.

Private Exponenten müssen also immer groß sein. Falls man von einem vorgegebenen öffentlichen Exponenten ausgeht, ist das für realistische N mit an Sicherheit grenzender Wahrscheinlichkeit erfüllt; Vorsicht ist nur geboten, wenn man mit dem privaten Exponenten startet.

§ 7: RSA im wirklichen Leben

Wie bereits zu Beginn der Vorlesung erwähnt, ist es oft einfacher, ein Kryptoverfahren nicht direkt anzugreifen, sondern über sein Umfeld. Bei RSA (wie auch bei praktisch allen anderen asymmetrischen Kryptoverfahren) gibt es dazu eine offensichtliche Methode: Wenn es C schafft, A davon zu überzeugen, daß (N, e) der öffentliche Schlüssel von B ist, wird A seine Nachrichten m an B als $c = m^e \bmod N$ verschlüsseln, und nur C wird in der Lage sein, diese Nachricht zu entschlüsseln. Genauso

wird A glauben, jede Unterschrift u mit $u^e \equiv m \bmod N$ sei die Unterschrift von B unter die Nachricht m. Zur Anwendung von RSA und ähnlichen Systemen im wirklichen Leben muß also durch eine geeignete Infrastruktur sichergestellt werden, daß A, der eine Nachricht an B schicken möchte, sich den korrekten Schlüssel von B verschaffen kann.

a) Allgemeine Struktur einer public key infrastracture

Asymmetrische Kryptoverfahren bieten im Unterschied zu den symmetrischen auch die Möglichkeit einer elektronischen Unterschrift. Dadurch wird es möglich einen öffentlichen Schlüssel durch Unterschrift zu bestätigen – vorausgesetzt man bekam den öffentliche Schlüssel zur Unterschrift aus vertrauenswürdiger Quelle. Dazu gibt es im wesentlichen zwei Vorgehensweisen:

1.) Hierarchische Modelle: In diesem Modell gibt es Zertifizierungsstellen, bei denen jemand (gegen Vorlage von Personalausweis, Gewerbeschein, Handelsbucheintrag, ...) seinen öffentlichen Schlüssel zertifizieren lassen kann, d.h. die Zertifizierungsstelle unterschreibt eine Nachricht, die die Identität des Antragstellers beschreibt und dessen öffentlichen Schlüssel enthält. Gleichzeitig legt sie einen Datensatz vor, in dem die nächsthöhere Zertifizierungsstelle auf dieselbe Weise den öffentlichen Schlüssel der unteren Stelle bekanntgibt und gleichzeitig bestätigt, daß es sich hier um eine Zertifizierungsstelle handelt.

Das Verfahren muß natürlich irgendwo enden; hier in Deutschland ist die Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen oberste Zertifizierungsstelle. Ihr öffentlicher Schlüssel ist nicht nur auf ihrer *home page* zu finden, sondern dürfte wohl auch in eine ganze Reihe sicherheitsrelevanter Software eingebaut sein. Sicherheitsbewußte Unternehmen, die wissen, wie einfach es ist, jemandem eine Webseite oder ein Programm zu unterschieben, werden sicherlich noch auf andere Weise überprüfen, daß sie *diesen* Schlüssel definitiv im Original haben.

2.) Grassroot Modelle: Zertifizierungsstelle sind keine Wohltätigkeitsorganisationen; sie können nur überleben, wenn sie sich ihre Arbeit bezahlen lassen. Die geforderten Preise sind nicht billig: Schon vor

mehreren Jahren las ich, daß sie bei bis zu 300$ pro Jahr lagen. Dies ist selbst Universitäten wie Mannheim oder Karlsruhe zuviel; für Privatpersonen, die einfach abhörsichere E-Mails an ihre Freunde schicken möchten, ist es (meist) unerschwinglich.

Speziell für die Kommunikation unter Privatleuten entwickelte PHILIP R. ZIMMERMANN das Programm PGP = *Pretty Good Privacy*. Der typische Anwender, für den er dieses Programm schrieb, möchte weder solche Summen ausgeben noch traut er Zertifizierungsstellen, die letztlich von einer Regierungsinstitution abhängen. Sein Sicherheitsmodell war daher ein völlig anderes: Jedermann traut seinen engsten Freunden, etwas weniger seinen entfernteren Freunden, und wenn es zu Freunden von Freunden geht, nimmt das Vertrauen naturgemäß ab.

Bei PGP läßt jeder Teilnehmer seinen öffentlichen Schlüssel von seinen Freunden zertifizieren. Diese wiederum sind von *ihren* Freunden zertifiziert. Wenn A mit B Kontakt aufnehmen will, sucht er nach dem öffentlichen Schlüssel von B. Er weiß natürlich, daß dieser gefälscht sein kann; er traut ihm aber, wenn sein bester Freund ihn unterschrieben hat, und er hat auch ein bißchen Zutrauen, wenn ihn einer seiner entfernteren Freunde unterschrieben hat.

Es kann natürlich auch vorkommen, daß er eine Nachricht bekommt von jemandem, der ihm völlig unbekannt ist. Wenn er Glück hat, ist der öffentliche Schlüssel des Absenders aber von einem seiner Freunde unterschrieben. Falls nicht, hat er vielleicht die Unterschrift von einem weiteren Unbekannten, dessen öffentlicher Schlüssel von jemandem unterschrieben ist, dem er traut, und so weiter. Anhand dieser Informationen kann er dann entscheiden, wieviel Vertrauen er dem Schlüssel entgegenbringen kann.

b) SSL, TLS & Co

Ein typischer Fall für die Kommunikation zwischen zwei einander unbekannten Partnern ist ein Kauf via Internet. Spätestens die Übermittlung der Informationen zur Abwicklung der Bezahlung müssen kryptographisch geschützt werden; auch muß der Kunde sicher sein können, daß er diese Informationen wirklich an die Firma schickt, von der er etwas kaufen möchte und nicht an jemanden, der nur abkassieren will.

Zur Realisierung dieser Ziele wurde der Standard SSL *(Secure Sockets Layer)* sowie sein Nachfolger TLS *(Transport Layer Security)* entwickelt. Sie wurden zwar in erster Linie für `https` konzipiert, die gleichen Ideen werden jedoch auch bei `ftps`, `ssh` und ähnlichen Diensten angewandt.

Allen Verbindungen gemeinsam ist, daß sich die Partner zunächst auf Verschlüsselungsverfahren einigen müssen. Hier machen sich immer noch alte amerikanische Exportbeschränkungen bemerkbar: Bis September 1998 galt alle Kryptographie als Munition die nur mit Einzelgenehmigung ins Ausland verkauft werden durfte. Diese Genehmigung wurde in der Praxis nur erteilt, wenn bei symmetrischer Kryptographie die Schlüssellänge höchstens gleich vierzig war und bei asymmetrischer Kryptographie zumindest für die Verschlüsselung mit ähnlich schwachen Algorithmen gearbeitet wurde. (Für reine Authentisierung durften auch starke Algorithmen exportiert werden.)

Da die beiden damals am meisten verbreiteten Browser, Netscape und Windows Explorer beide aus USA kamen, unterstützten diese zumindest in ihren Exportversionen daher nur schwache Kryptographie. Auch bei den Servern von Netscape waren Versionen mit starker Kryptographie (die natürlich nur innerhalb der USA verkauft werden durften) deutlich teurer als die Exportversionen, so daß viele Unternehmen noch heute nur Server mit schwacher Kryptographie unterhalten.

Nimmt ein Client Kontakt auf zu einem Server, teilt er ihm daher zunächst einmal mit, welche Kryptoverfahren er kennt. Dazu gibt es eine Liste von genormten Namen für die verschiedenen symmetrischen, asymmetrischen und Unterschriftsverfahren; für TLS etwa ist diese in den Anhängen zu RFC 2246 zu finden. Zulässig ist bei den meisten Protokollen auch jeweils der Wert „Keines“, was zur Folge hat, daß keine entsprechende Verschlüsselung stattfindet.

Der Server vergleicht die erhaltene Liste mit den Kryptoverfahren, die er beherrscht, und wählt dann eines aus – oder aber beendet die Verbindung, falls es kein von beiden beherrschtes *bzw.* akzeptiertes Verfahren gibt.

Der entsprechende Austausch findet selbstverständlich im Klartext statt und ist daher ein möglicher Angriffspunkt für einen Gegner: Indem er

die Liste des Clients abfängt und alle starken Verfahren daraus streicht, kann er die Verwendung schwacher Kryptographie erzwingen.

Im nächsten Schritt identifiziert sich der Server und schickt seinen öffentlichen Schlüssel. Da auch hier ein Angreifer sich als Server ausgeben könnte, sollte dieser Schritt mit einer Authentisierung verbunden sein, d.h. der Server legt dem Client ein unterschriebenes Zertifikat vor, das sowohl seine Identität als auch seinen öffentlichen Schlüssel und das dazugehörige Verfahren enthält.

Dadurch ist das Problem natürlich nur um eine Stufe verschoben, denn ein Angreifer könnte sich auch als Zertifizierungsbehörde ausgeben. Theoretisch ist dies dadurch gelöst, daß die öffentlichen Schlüssel der (relativ wenigen) anerkannten Zertifizierungsinstitutionen im Programmcode der Browser enthalten sind. Wer sich freilich seinen Browser einfach von irgendeiner Internetseite holt, hat keine Garantie, daß dort nicht auch zusätzliche Schlüssel eines Angreifer stehen.

Ein weiteres Problem besteht darin, daß Zertifizierungsbehörden relativ hohe Preise verlangen; ein einziges Zertifikat kann bereits über 300$ kosten. Für `amazon.com` und ähnliche Unternehmen sind das *peanuts;* kleinere Betriebe oder auch Institutionen wie etwa die Universität Mannheim aber schrecken vor diesen Kosten zurück und stellen für ihr Subnetz eigene Zertifikate aus. Grundsätzlich gibt es auch die Möglichkeit, daß sich solche selbsternannte Zertifizierungsstellen von einer offiziell anerkannten zertifizieren lassen mit einem Zertifikat, das ihnen (im Gegensatz zu den Inhabern „üblicher“ Zertifikate) auch das Recht einräumt, selbst in einem gewissen Namensraum zu zertifizieren. Eine solche Lizenz natürlich noch teurer und wird daher nicht oft vorgelegt. In einem solchen Fall, wenn die Identität des Servers nicht zweifelsfrei festgestellt werden kann, wird üblicherweise der Benutzer gefragt, ob er trotzdem weitermachen will und ob er gegebenenfalls die Unterschrift der dem Browser unbekannten Zertifizierungsstelle künftig anerkennen will.

Bei `ssh`-Verbindungen dürfte es wohl die Regel sein, daß der Server kein Zertifikat vorlegen kann; hier speichert der Client den Schlüssel *bzw.* einen Fingerabdruck davon, nachdem der Benutzer beim ersten

Mal gefragt wurde, ob er sich sicher sei, mit dem richtigen Rechner verbunden zu sein. Künftig werden dann Verbindungen zu diesem Server nur noch aufgebaut, wenn der Server den richtigen Schlüssel schickt.

Wenn der Client den öffentlichen Schlüssel des Servers kennt, kann er nun zufällig einen Sitzungsschlüssel für das vom Server ausgewählte symmetrische Verfahren erzeugen und diesen mit dem asymmetrischen Verfahren verschlüsselt an den Server schicken. Die weitere Kommunikation erfolgt dann symmetrisch verschlüsselt, wobei gegebenenfalls noch zusätzlich eine Prüfsumme zur Sicherung der Nachrichtenintegrität übertragen wird. (Wie man solche Prüfsummen kryptographisch sicher erzeugt, werden wir weiter hinten sehen.)

c) PKCS #1v1.5

Wie so ziemlich alles im Internet müssen natürlich auch die Nachrichten, die Client und Server austauschen, in einem standardisierten Format sein Bei Verwendung von RSA als asymmetrischem Verfahren legt der von RSA Data Security Inc. entwickelte Standard PKCS #1 fest, in welcher Form der Schlüssel für das symmetrische Verfahren übermittelt wird. In seiner alten Version „v1.5“, die auf Grund einer im nächsten Abschnitt beschriebenen Schwäche inzwischen nicht mehr empfohlen wird und durch eine Alternative ersetzt ist, geht man folgendermaßen vor:

Wird RSA mit einem m-Bit-Modul N verwendet, so sei zunächst $k = [m/8]$ der bei ganzzahliger Division mit ignoriertem Rest entstehende Quotient. Gesendet werden jeweils Blöcke aus $k + 1$ Bytes.

Da nicht alle durch $k + 1$ Bytes darstellbare natürliche Zahlen kleiner als N sind, läßt sich das erste Byte nicht wirklich nutzen, denn die Verschlüsselung soll selbstverständlich injektiv sein. Daher wird dieses Byte stets auf 00 gesetzt.

Das nächste Byte gibt an, worum es sich bei dem zu übermittelnden Block handelt. Im Falle einer RSA-verschlüsselten Nachricht wird es auf 02 gesetzt, während beispielsweise 01 für eine elektronische Unterschrift steht.

Danach folgt ein Block von mindestens acht Zufallsbytes, die alle einen von Null verschiedenen Wert haben müssen; dies realisiert die in §3*)* geforderte probabilistische Verschlüsselung. Das Ende dieses Blocks wird durch ein angehängtes Nullbyte angezeigt; die restlichen Bytes sind für die eigentliche Nachricht vorgesehen.

d) Der Angriff von Bleichenbacher

Bei Verwendung eines Kodierungsverfahrens wie dem gerade beschriebenen Standard entspricht nicht mehr jede Zahl $0 \leq a \leq N - 1$ einer Nachricht. Damit kann es vorkommen, daß z.B. durch Übertragungsfehler ein Empfänger Nachrichten enthält, deren Entschlüsselung sich nicht sinnvoll entsprechend der Norm interpretieren läßt.

Ein menschlicher Empfänger wird solche Nachrichten, insbesondere wenn sie gehäuft auftreten, wohl einfach ignorieren oder vielleicht auch beim ersten Mal noch eine Nachricht an den Absender schicken, daß er dessen Nachricht nicht lesen kann; ein Server, der solche Nachrichten im Rahmen eines SSL-Verbindungsaufbaus erhält, wird jedes Mal genau das tun, was der Programmierer für diesen Fall vorgesehen hat. Früher war dies die kanonische Reaktion, die man in solchen Fällen erwartet: eine Fehlermeldung.

Eine solche Fehlermeldung kann ein Angreifer als eine Art *Orakel* benutzen: Wenn er eine Zahl $0 \leq c \leq N - 1$ an den Server schickt, interpretiert dieser dies als eine verschlüsselte Nachricht $c = a^e \bmod N$ und entschlüsselt sie als $a = c^d \bmod N$, wobei (N, e) der öffentliche und d der private Schlüssel des Servers ist. Falls a nicht die erwartete Form hat, kann der Server die Nachricht nicht interpretieren und schickt eine Fehlermeldung zurück. Da Computer sehr geduldig sind, wird er dies nicht nur einmal, sondern gegebenenfalls auch mehrere Millionen Mal für denselben Absender tun, so daß dieser beliebig viele Zahlen c testen kann.

Hier setzt der Angriff von BLEICHENBACHER an: Man kann zeigen, daß bei RSA jedes einzelne Bit so sicher ist wie der gesamte Block; mit anderen Worten: Falls jemand ein Verfahren hat, mit dem er ein festes Bit der Nachricht, z.B. das letzte oder das dritte, berechnen kann, so kann er daraus ein Verfahren machen, um die gesamte Nachricht zu entschlüsseln. Die wesentliche Idee des Beweis besteht darin, daß die

RSA-Verschlüsselung $a \mapsto a^e \bmod M$ ein Gruppenhomomorphismus ist, was wir ja bereits bei den blinden Unterschriften und beim elektronischen Bargeld ausgenutzt hatten.

PKCS #1v1.5 liefert einem Angreifer, der den Server als Orakel nutzen kann, so etwas ähnliches wie die Entschlüsselung gewisser Bits: Er kann für gewisse Zahlen $0 \le c \le N-1$ erfahren, daß sie mit den beiden Bytes 00 und 02 beginnen. Er weiß dann also, daß $c^d \bmod N$ in einem gewissen Intervall $[B_1, B_2]$ liegt, wobei wir etwa

$$B_1 = 2^{k+1} + 2^k \quad \text{und} \quad B_2 = 3 \cdot 2^k - 1$$

setzen können. Falls bekannt ist, wie viele Zufallsbytes verwendet werden, kann man eventuell auch B_2 noch etwas schärfer abschätzen, andererseits bringt das nicht sonderlich viel. BLEICHENBACHER begnügt sich sogar bei der unteren Grenze einfach mit $B_1 = 2^{k+1}$.

Ziel der Attacke von BLEICHENBACHER ist es, zu einer vorgegebenen Zahl $0 \le c \le N-1$ die Zahl $c^d \bmod N$ zu bestimmen, um entweder eine abgefangene Chiffretextnachricht c wie den Sitzungsschlüssel für eine SSL-Verbindung zu entschlüsseln oder aber die Unterschrift des Servers für eine konstruierte Nachricht zu fälschen.

Falls c ein abgefangener Chiffretext ist, muß $c^d \bmod N$ in $[B_1, B_2]$ liegen. Andernfalls sucht der Angreifer nach einer Zahl s derart, daß für $c_0 = cs^e \bmod N$ die Entschlüsselung

$$c_0^d \bmod N = \left(cs^e\right)^d \bmod N = c^d s^{ed} \bmod N = c^d s \bmod N$$

vom Server akzeptiert wird.

Die Wahrscheinlichkeit dafür, daß dies für ein zufällig gewähltes s der Fall ist, läßt sich einigermaßen abschätzen: Wie können davon ausgehen, daß für zufälliges s auch die Zahlen $c_0^d \bmod N$ zufallsverteilt sind im Intervall $[0, N-1]$. Die Wahrscheinlichkeit dafür, daß eine solche Zahl im Intervall $[B_1, B_2]$ liegt ist daher das Verhältnis der Intervallängen, also ungefähr $N/2^k$. Je nach Kongruenzklasse der Bitanzahl von N modulo acht liegt dieser Wert zwischen $2^{-16} = 1 : 65536$ und $2^{-8} = 1 : 256$; da die Bitlängen von RSA-Moduln meist Vielfache von acht sind, dürfte sie sich eher in der Nähe der unteren Grenze bewegen.

Dazu kommt noch, daß auf die beiden Bytes 00 und 02 mindestens acht von Null verschiedene Bytes folgen müssen und dann irgendwann ein Nullbyte, was die Wahrscheinlichkeit der Akzeptanz noch etwas weiter verringert, wenn auch um keinen sonderlich großen Faktor: Falls wir etwa mit 2048 Bit-Modul arbeiten, besteht ein Block aus 256 Bytes; wir können also mit einer ziemlich hohen Wahrscheinlichkeit davon ausgehen, daß irgendeines davon zwischen den Positionen elf und 256 das Nullbyte ist.

Bei einem automatisierten Angriff kann man somit in relativ kurzer Zeit eine Zahl s finden, so daß $c_0 = cs^e \bmod N$ vom Server akzeptiert wird. Falls man dann $a_0 = c_0^d \bmod N$ bestimmen kann, läßt sich leicht auch

$$a = c^d \bmod N = s^{-1} a_0 \bmod N$$

berechnen. Wir können uns daher im folgenden auf das Problem beschränken, zu einem c, das einer korrekt verschlüsselten Nachricht a entspricht, deren Entschlüsselung $a = c^d \bmod N$ zu ermitteln.

Dazu bestimmt BLEICHENBACHER, wieder durch Probieren so lange, bis der Server keine Fehlermeldung mehr schickt, eine Folge von Zahlen

$$0 < s_1 < s_2 < \cdots < N$$

derart, daß $c_i = cs_i^e \bmod N$ vom Server akzeptiert wird. Dann ist

$$a_i = c_i^d \bmod N = as_i \bmod N \in [B_1, B_2]\,,$$

es gibt also eine Zahl r_i derart, daß

$$as_i - r_i N \in [B_1, B_2] \quad \text{oder} \quad a \in \left[\frac{B_1 + r_i N}{s_i}, \frac{B_2 + r_i N}{s_i}\right]$$

Damit liegt a auch im Durchschnitt eines dieser Intervalle mit $[B_1, B_2]$, so daß wir a weiter eingegrenzt haben.

Im Hinblick auf die weiteren Schritte wollen wir annehmen, wir wüßten bereits, daß a in einem Intervall $[u, v]$ liegt. Dann können wir nun genauer sagen, daß a sogar im Durchschnitt von $[u, v]$ mit der Vereinigung der obigen Intervalle liegt, d.h. in der Vereinigung

$$\bigcup_{r \in \mathbb{Z}} \left(\left[\frac{B_1 + rN}{s_i}, \frac{B_2 + rN}{s_i}\right] \cap [u, v] \right)\,.$$

Tatsächlich sind natürlich fast alle diese Durchschnitte leer; ein nichtleerer Durchschnitt ist nur möglich, wenn

$$\frac{B_1 + rN}{s_i} \le v \quad \text{und} \quad \frac{B_2 + rN}{s_i} \ge u\,,$$

also

$$\frac{s_i u - B_2}{N} \le r \le \frac{s_i v - B_1}{N}$$

ist.

Damit ist BLEICHENBACHERs Vorgehensweise zumindest im Prinzip klar: Wir betrachten eine Menge L von Intervallen derart, daß a in einem Intervall aus der Liste sein muß; zu Beginn besteht L genau aus dem Intervall $[B_1, B_2]$. Außerdem setzen wir $s_0 = [N/B_2]$; man überzeugt sich leicht, daß sa für $s \le s_0$ höchstens gleich N aber natürlich größer als B_2 ist, so daß as dann unmöglich akzeptiert werden kann.

Im i-ten Schritt für $i \ge 1$ wird durch Serveranfragen eine Zahl $s_i > s_{i-1}$ ermittelt derart, daß as_i in $[B_1, B_2]$ liegt; sodann wird L ersetzt durch die Menge aller Intervalle der Form

$$\left[\frac{B_1 + rN}{s_i}, \frac{B_2 + rN}{s_i}\right] \cap [u, v]$$
$$\text{mit} \quad [u, v] \in L \quad \text{und} \quad \frac{s_i u - B_2}{N} \le r \le \frac{s_i v - B_1}{N}\,.$$

Dieses Verfahren wird so lange fortgesetzt, bis L nur noch ein Intervall der Länge eins enthält, das dann notwendigerweise gleich $[a, a]$ ist.

Tatsächlich optimiert BLEICHENBACHER noch etwas: Durch geschickte Wahl der s_i kann man nämlich r noch etwas genauer unter Kontrolle bekommen. Mit einer solchen Strategie kann er zeigen, daß im Schnitt etwa 2^{20}, also rund eine Million, Serveranfragen genügen.

e) Elektronische Unterschriften nach PKCS#1

RSA-Verschlüsselungen langer Texte sind teuer, elektronische Unterschriften eher noch teurer, da kurze öffentliche Exponenten bei RSA zwar relativ problemlos sind, kurze private Exponenten aber, wie wir

gesehen haben, katastrophal. Ein längerer Text wird daher praktisch nie blockweise unterschrieben.

Stattdessen bildet man nach Verfahren, mit denen wir uns in einem eigenen Kapitel beschäftigen werden, einen kryptographisch sicheren Hashwert und unterschreibt diesen. Die heute üblichen Verfahren zur Berechnung solcher Hashwerte liefern Ergebnisse einer Länge von 160, 256, 382 oder 512 Bit; verglichen mit der Länge eines auch nur einigermaßen sicheren RSA-Blocks ist das recht kurz.

Damit ist auch hier Auffüllen unvermeidbar, allerdings gibt es einen bedeutenden Unterschied zum Fall der Nachrichten: Bei einer Nachricht bestimmt der Absender, was sie enthalten soll; je weniger man ihn dabei einschränkt und je undurchschaubarer er arbeitet, desto weniger Ansatzpunkte hat ein Gegner zur unbefugten Entschlüsselung. Von daher sind vom Absender festzulegende Zufallsbits hier die beste Methode zum Auffüllen.

Bei einer elektronischen Unterschrift dagegen muß der Empfänger die Korrektheit überprüfen, indem er eine öffentlich bekannte Funktion anwendet und das Ergebnis mit einem erwarteten Wert vergleicht. Hier würde das Auffüllen mit Zufallsbits einem Fälscher die Arbeit erleichtern, denn Zufallsbits kann der Empfänger natürlich nicht verifizieren.

Nehmen wir beispielsweise an, der zu unterschreibende Hashwert h aus k Byte sei ungerade – durch Probieren mit minimalen Veränderungen am Dokument läßt sich dies ziemlich schnell erreichen. Außerdem nehmen wir an, daß der zu unterschreibende RSA-Block Platz für mindestens $3k + 3$ Byte bietet – das ist bei gängigen Kombinationen heute üblicher Verfahren meist automatisch der Fall. Schließlich wollen wir noch annehmen, daß der öffentliche Exponent zur Verifikation der Unterschrift u gleich drei sei – auch das ist ein leider auch heute noch immer recht häufiger Standard.

Ein Angreifer kann dann folgendermaßen eine Unterschrift u unter den Hashwert h fälschen: Er berechnet eine Zahl $u < 2^{8(k+1)}$ mit

$$u^3 \equiv h \bmod 2^{8(k+1)}\,.$$

Wie das folgende Lemma zeigt, ist dies stets möglich, und der Beweis gibt auch ein Verfahren, mit dem u effizient konstruiert werden kann:

Lemma: Für jedes $n \in \mathbb{N}$ und jedes ungerade $a < 2^n$ gibt es ein $x < 2^n$, so daß $x^3 \equiv a \bmod 2^n$ ist.

Beweis: Für $n = 1$ ist notwendigerweise $a = 1$, und die Lösung ist $x = 1$. Für $n > 1$ können wir induktiv annehmen, daß wir bereits ein $z < 2^{n-1}$ gefunden haben, für das $z^3 \equiv a \bmod 2^{n-1}$ ist. Die Zahl $z^3 - a$ ist dann durch 2^{n-1} teilbar, es gibt also ein $b \in \mathbb{Z}$, so daß $z^3 = a + b \cdot 2^{n-1}$ ist. Zur Konstruktion von x machen wir den Ansatz $x = z + 2^{n-1}y$; dann ist

$$\begin{aligned} x^3 &= z^3 + 3 \cdot 2^{n-1} \cdot y + 3 \cdot 2^{2(n-1)} \cdot y^2 + 2^{3(n-1)} \\ &\equiv z^3 + 3y \cdot 2^{n-1} = a + (b + 3y) \cdot 2^{n-1} \bmod 2^n \,. \end{aligned}$$

Falls $b + 3y$ gerade ist, folgt also $x^3 \equiv a \bmod 2^n$. Das können wir aber immer erreichen: Für gerades b setzen wir beispielsweise $y = 0$, für ungerades b nehmen wir $y = 1$. Wegen $z < 2^{n-1}$ ist in beiden Fällen $x = z + 2^{n-1}y < 2^n$, das Lemma ist also bewiesen. ∎

Wir können also zu einer ungeraden Zahl h stets eine Zahl $u < 2^{8(k+1)}$ finden mit $u^3 \equiv h \bmod 2^{8(k+1)}$. Falls N, wie angenommen, mindestens $8(k + 1)$ Byte hat, ist $N > u^3$, also $u^3 \bmod N = u^3$. Die letzten $(k + 1)$ Byte von u^3 bestehen wegen obiger Kongruenz aus einem Nullbyte gefolgt von den k Byte von h. Damit ist u eine gültige Unterschrift unter h.

Um so etwas zu verhindern, darf der Unterschreibende keine Kontrolle über die Bits zum Auffüllen haben. Der Standard PKCS#1 setzt fest, daß genau das folgende Wort zu unterschreiben ist:

Links steht ein (eventuell unvollständiges) Nullbyte, darauf folgt ein Byte 01 um anzuzeigen, daß es sich um eine elektronische Unterschrift handelt, sodann Füllbytes, die aus lauter binären Einsen bestehen, d.h. sie haben den hexadezimalen Wert FF und den Dezimalwert 255. Dann folgt zunächst ein Nullbyte, danach der Name des verwendeten Hashverfahrens gemäß der Norm ASN.1 sowie der Hashwert selbst, dessen Länge durch das Hashverfahren bestimmt ist.

Hier ist offensichtlich alles festgelegt, und die Wahrscheinlichkeit dafür, daß die dritte Potenz einer natürlichen Zahl entsteht, liegt bei praktisch null. Falls etwa N genau 2 048 Bit hat und die Folge aus Nullbyte, Algorithmenname und Hashwert aus 288 Bytes besteht (wie es beim immer noch populären SHA-1 der Fall ist), dann liegt der zu unterschreibende Wert zwischen $2^{2041} - 2^{289}$ und $2^{2041} - 2^{288}$. In diesem Intervall gibt es keine einzige Kubikzahl.

f) Bleichenbachers Angriff dagegen

Trotzdem konnte BLEICHENBACHER auch hier eine Angriffsstrategie finden; sie funktioniert allerdings nicht immer und außerdem höchstens dann, wenn die Verifikation der Unterschrift schlampig programmiert ist – was leider in einer ganzen Reihe von Browsern der Fall ist. Ein auf Effizienz bedachter Programmierer könnte die Verifikation einer PKCS#1-Unterschrift folgendermaßen implementieren: Er wendet die öffentliche Verschlüsselungsfunktion auf die Unterschrift an und überprüft zunächst, ob das erste Byte des dabei erhaltenen Blocks den Wert 00 und das zweite der Wert 01 hat. Danach ignoriert er alle Bytes mit Wert FF und verifiziert, daß das erste davon verschiedene Byte den Wert 00 hat. Die darauf folgenden Bytes versucht er als Name eines Hashverfahrens zu interpretieren; falls dies möglich ist, liest er nach Ende des Namens die Anzahl von Bytes, die der entsprechende Algorithmus produziert und interpretiert sie als einen Hashwert h'. Nun bearbeitet er den angeblich unterschriebenen Klartext mit dem angegebenen Hashverfahren und berechnet den Hashwert h. Falls $h = h'$, akzeptiert er die Unterschrift. Bei Daten, die aus vertrauenswürdiger Quelle kommen und bei denen man sicher sein kann, daß sie der Spezifikation entsprechen, mag so eine Vorgehensweise vielleicht gerade noch angehen, obwohl man bei einer realistischen Sichtweise der heute vorherrschenden Softwarequalität auch da lieber einen Test zuviel als einen zuwenig machen sollte. In der Kryptologie allerdings *müssen* wir jedem ankommenden Text mißtrauen – wenn wir allen trauen könnten, bräuchten wir schließlich keine Kryptographie. Spätestens unter diesem Gesichtspunkt hat die gerade skizzierte Vorgehensweise einen ganz gravierenden Nachteil: Sie überprüft nicht, ob der Block wirklich mit dem Hashwert endet, oder ob danach noch weitere Bytes folgen.

Dadurch hat ein Fälscher plötzlich wieder Manipulationsmöglichkeiten, da nun *er* festlegen kann, wie viele Füllbytes FF verwenden will und er im übrigen völlige Freiheit hat bezüglich der Bytes, die er *hinter* dem Hashwert platziert.

BLEICHENBACHER gab auf der Crypto'2006 in einer Abenddiskussion eine mögliche Strategie an, wie man dies in manchen Fällen zur Fälschung von Unterschriften ausnutzen kann; er hat allerdings anscheinend bislang noch nichts veröffentlicht. In Diskussionslisten zur Kryptologie sind allerdings Hinweise auf seinen Ansatz zu finden. Danach geht er aus von der Formel

$$(2^n - x)^3 = 2^{3n} - 3 \cdot 2^{2n} \cdot x + 3 \cdot 2^n \cdot x^2 - x^3 \,.$$

Ist h der zu unterschreibende Hashwert (einschließlich dem führenden Nullbyte und dem Namen des Hashverfahrens), und hat h eine Länge von r Bit (wobei r natürlich ein Vielfaches von acht sein muß), so setzt er hier x auf $y/3$ mit $y = 2^r - h$. Natürlich gibt es keinen Grund, warum y durch drei teilbar sein sollte, aber wieder ist es kein Problem, eine sinngleiche Modifikation der Nachricht zu konstruieren, für deren Hashwert dies der Fall ist. Dann ist

$$(2^n - x)^3 = 2^{3n} - 2^{2n}y + 2^n\frac{y^2}{3} - \frac{y^3}{27} = 2^{3n} - 2^{2n+r} + 2^n h + 2^n\frac{y^2}{3} - \frac{y^3}{27} \,.$$

Für $n \geq 2r$ ist $y^2 < 2^n$ und $y^3 < 2^{2n}$; falls uns also nur die Bits bis zur Position von 2^{2n} interessieren, können wir die letzten beiden Summanden vergessen. $2^{3n} - 2^{2n+r}$ ist eine Zahl, die im Binärsystem mit $n - r + 1$ Einsen beginnt, darauf folgen $2n + r$ Nullen, und $2^n h$ ist der Wert von h um $2n$ nach links verschoben. Wer diese Verschiebung nicht bemerkt, wird $2^n - x$ als Unterschrift unter h akzeptieren. Dies funktioniert natürlich nur, wenn der RSA-Modul N eine Bytelänge r hat mit $r \equiv 2 \bmod 3$, aber erstens wird so etwas immer wieder vorkommen, und zweitens ist der gerade skizzierte Angriff, den BLEICHENBACHER in einer Abenddiskussion auf der Crypto'2006 skizziert hat, sicherlich nicht die einzige Möglichkeit, eine Kubikzahl kleiner N zu produzieren, die im Binärsystem mit lauter Einsen beginnt, dann nach einem Nullbyte einen vorgegebenen Wert h enthält und danach beliebige Bits enthalten darf. Ein möglicher Schutz vor diesem Angriff besteht natürlich darin,

daß man keinen Browser und auch kein sonstiges Programm verwenden sollte, das bei der Verifikation einer Unterschrift nicht überprüft, ob der Hashwert wirklich rechtsbündig steht. Angesichts der Vielzahl heute erhältlicher Browser und der Tatsache, daß kaum ein Benutzer feststellen kann, wie seiner eine Unterschrift überprüft, ist das aber leider nicht sonderlich realistisch. Besser wäre es, wenn die „Unterschreiber" keine öffentlichen Schlüssel mit $e = 3$ verwenden würden; da aber meist nicht sie, sondern ihre Kunden einen etwaigen Schaden tragen müssen, ist das leider fast noch unrealistischer.

§8: Faktorisierungsverfahren

Der offensichtliche Angriff auf RSA ist die Faktorisierung der öffentlich bekannten Zahl N; sobald man diese in ihre beiden Primfaktoren p und q zerlegt hat, ist das Verfahren gebrochen. Wir wollen daher in diesem Paragraphen sehen, welche Möglichkeiten es gibt, N in seine Primfaktoren zu zerlegen.

a) Mögliche Ansätze zur Faktorisierung

Grundsätzlich gibt es zwei Klassen von Verfahren, mit denen man einen Teiler einer natürlichen Zahl N finden kann: Einmal Verfahren, deren erwartete Laufzeit von der Länge des Faktors abhängt, zum anderen solche, deren Laufzeit nur von N abhängt.

Bei der Anwendung von RSA wird man, um Verfahren der ersten Kategorie auszubremsen, p und q ungefähr gleich groß wählen, so daß diese Verfahren im schlechtestmöglichen Fall arbeiten müssen.

Die einfachste Art der Faktorisierung ist das Abdividieren von Primzahlen, vor allem für kleine Primfaktoren. Mindestens bis etwa 2^{16} ist dies auch die schnellste und effizienteste Methode, da die anderen Verfahren Schwierigkeiten haben, Produkte kleiner Primzahlen zu trennen.

Für etwas größere Faktoren bis zu etwa acht Dezimalstellen ist die POLLARDsche Monte-Carlo-Methode oder ρ-Methode sehr gut geeignet: Man erzeugt mit einem quadratischen Generator (populär ist z.B. $x_{i+1} = x_i^2 + c \bmod N$) Zufallszahlen und berechnet deren ggT mit der zu

faktorisierenden Zahl. Da der EUKLIDische Algorithmus im Vergleich zur Erzeugung der Zufallszahlen relativ teuer ist, empfiehlt es sich, die erzeugten Zufallszahlen zunächst modulo N miteinander zu multiplizieren und dann erst in etwa jedem hundertsten Schritt den ggT von N mit diesem Produkt zu berechnen. (Dies setzt voraus, daß alle sehr kleinen Faktoren bereits abdividiert sind; sonst ist die Gefahr zu groß, daß im Produkt von hundert Zufallszahlen mehr als ein Primfaktor steckt.) Bei Faktoren mit mehr als acht Dezimalstellen wird die Methode schnell langsamer, so daß man dann zu alternativen Verfahren übergehen sollte.

Die nächste Klasse von Verfahren beruht auf gruppentheoretischen Überlegungen, im wesentlichen dem kleinen Satz von FERMAT im Falle zyklischer Gruppen und Verallgemeinerungen auf weitere Gruppen wie die multiplikative Gruppe eines Körpers $\mathbb{F}_{p^2}$ oder einer elliptischen Kurve. Diese Verfahren sind sehr effizient, wenn die Gruppenordnung nur relativ kleine Primteiler hat. Aus diesem Grund wurde früher häufig empfohlen, daß für die Primteiler p eines RSA-Moduls N sowohl $p-1$ als auch $p+1$ jeweils mindestens einen „großen“ Primfaktor haben sollen; auch heute ist diese Empfehlung noch in einigen Büchern zu finden. Da die genannten Verfahren ihre Stärke jedoch bei Faktoren mit einer Länge von bis etwa 30 oder 35 Dezimalstellen haben und ein N mit 70 Dezimalstellen für RSA heute natürlich völlig unsicher ist, hat die Empfehlung heute ihre Berechtigung verloren und wir müssen uns insbesondere auch nicht näher mit den Faktorisierungsverfahren beschäftigen, vor denen sie schützen sollte.

Umso interessanter ist dagegen ein Verfahren, dessen Stärke bei nahe beieinander liegenden Primfaktoren liegt. Es wurde von PIERRE DE FERMAT vorgeschlagen und beruht auf der Formel

$$x^2 - y^2 = (x+y)(x-y)\,.$$

Ist $N = pq$ Produkt zweier ungerader Primzahlen, so ist

$$N = (x+y)(x-y) \quad \text{mit} \quad x = \frac{p+q}{2} \quad \text{und} \quad y = \frac{p-q}{2}\,;$$

zusammen mit obiger Formel folgt, daß dann $N + y^2 = x^2$ ist. FERMAT berechnet daher für $y = 0, 1, 2, \ldots$ die Zahlen $N + y^2$; falls er auf ein

Quadrat x^2 stößt, berechnet er

$$\text{ggT}(N, x+y) \quad \text{und} \quad \text{ggT}(N, x-y)\,.$$

Wenn er Pech hat, sind dies die beiden Zahlen eins und N, wenn er Glück hat, sind es p und q. Für zufällig gewählte Paare (x, y) kommt beides jeweils mit fünfzigprozentiger Wahrscheinlichkeit vor.

Falls p und q nahe beieinander liegen, führt schon ein kleines y zur korrekten Faktorisierung; bei der Wahl der Primzahlen muß also darauf geachtet werden, daß sie zwar die gleiche *Größenordnung* haben, aber nicht zu weit beieinander liegen. Liegt etwa p in der Größenordnung von $2q$, hat die Differenz die gleiche Größenordnung wie p, und FERMATS Verfahren bräuchte etwa p Rechenschritte, wird also vom Aufwand her vergleichbar mit der Faktorisierung durch Abdividieren. Somit kann man sich auch gegen diese Attacke recht gut schützen.

Wirklich gefährlich sind eine Klasse von Siebverfahren, die auf FERMATS Methode aufbauen. Diese Verfahren sind die schnellsten derzeit bekannten zur Faktorisierung von RSA-Moduln; die Wahl einer sicheren Ziffernlänge hängt also davon ab, welche Zahlen diese Verfahren faktorisieren können.

b) Das quadratische Sieb

Das quadratische Sieb ist der Grundalgorithmus der ganzen Klasse; es ist logisch einfacher, allerdings vor allem für große Zahlen auch deutlich langsamer als die Variationen, mit denen wir uns im nächsten Abschnitt kurz beschäftigen werden.

Bei allen diesen Verfahren geht es darum, Zahlenpaare (x, y) zu finden, für die $x^2 \equiv y^2 \bmod N$ ist. Für diese erwarten wir, daß in etwa der Hälfte aller Fälle

$$\text{ggT}(x+y, N) \quad \text{und} \quad \text{ggT}(x-y, N)$$

nichttriviale Teiler von N sind.

Beim quadratischen Sieb betrachten wir dazu das Polynom

$$f(x) = \left(x + \left\lceil \sqrt{N} \right\rceil\right)^2 - N\,.$$

Offensichtlich ist für jedes x

$$f(x) \equiv \left(x + \left\lceil\sqrt{N}\right\rceil\right)^2 \bmod N\,,$$

allerdings stehen links und rechts verschiedene Zahlen. Insbesondere steht links im allgemeinen keine Quadratzahl.

Falls wir allerdings Werte $x_1, x_2, \ldots, x_r$ finden können, für die das Produkt der $f(x_i)$ eine Quadratzahl ist, dann ist

$$\prod_{i=1}^{r} f(x_i) \equiv \prod_{i=1}^{r} \left(x + \left\lceil\sqrt{N}\right\rceil\right)^2 \bmod N$$

eine Relation der gesuchten Art.

Um die x_i zu finden, betrachten wir eine Menge $\mathcal{B}$ von Primzahlen, die sogenannte Faktorbasis. Typischerweise enthält $\mathcal{B}$ für die Faktorisierung einer etwa hundertstelligen Zahl etwa 100–120 Tausend Primzahlen, deren größte somit, wie die folgende Tabelle zeigt, im einstelligen Millionenbereich liegt.

n	n-te Primzahl	n	n-te Primzahl
100 000	1 299 709	600 000	8 960 453
200 000	2 750 159	700 000	10 570 841
300 000	4 256 233	800 000	12 195 257
400 000	5 800 079	900 000	13 834 103
500 000	7 368 787	1 000 000	15 485 863

Beim quadratischen Sieb interessieren nur x-Werte, für die $f(x)$ als Produkt von Primzahlen aus $\mathcal{B}$ (und eventuell auch Potenzen davon) darstellbar ist.

Natürlich wäre es viel zu aufwendig, für jedes x durch Abdividieren festzustellen, ob $f(x)$ als Produkt von Primzahlen aus B geschrieben werden kann: $f(x)$ ist eine Zahl in der Größenordnung von N, und wenn auch die Primzahlen aus $\mathcal{B}$ verhältnismäßig klein sind, kostet doch jede Division ihre Zeit. Wenn wir eine ungefähr hundertstellige Zahl faktorisieren, müssen wir außerdem davon ausgehen, daß nur etwa einer aus einer Milliarde Funktionswerten $f(x_i)$ über $\mathcal{B}$ vollständig faktorisiert

werden kann; wir müssen also sehr viele Funktionswerte testen. Dazu dient die Siebkomponente des Algorithmus:

Der Funktionswert $f(x)$ ist genau dann durch p teilbar, wenn

$$f(x) \equiv 0 \bmod p$$

ist. Da für $x, y, a, b \in \mathbb{Z}$ und mit $x \equiv y \bmod p$ und $a \equiv b \bmod p$ gilt

$$a + x \equiv b + y \bmod p \quad \text{und} \quad a \cdot x \equiv b \cdot y \bmod p$$

und für $n \in \mathbb{N}$ auch

$$x^n \equiv y^n \bmod p\,,$$

ist für jedes Polynom f mit ganzzahligen Koeffizienten

$$f(x) \equiv f(y) \bmod p\,.$$

Ist also insbesondere $f(x) \equiv 0 \bmod p$, so ist auch

$$f(x + kp) \equiv 0 \bmod p \quad \text{für alle } k \in \mathbb{Z}\,.$$

Es genügt daher, im Bereich $0 \le x < p - 1$ nach Werten zu suchen, für die $f(x)$ durch p teilbar ist.

Dazu kann man f auch als Polynom über dem Körper $\mathbb{F}_p$ mit p Elementen betrachten und nach Nullstellen in diesem Körper suchen. Für Polynome großen Grades und große Werte von p kann dies recht aufwendig sein; hier, bei einem quadratischen Polynom, müssen wir natürlich einfach eine quadratische Gleichung lösen: In $\mathbb{F}_p$ wie in jedem anderen Körper auch gilt

$$f(x) = \left(x - \left\lceil\sqrt{N}\right\rceil\right)^2 - N = 0 \Longleftrightarrow \left(x - \left\lceil\sqrt{N}\right\rceil\right)^2 = N\,,$$

und diese Gleichung ist genau dann lösbar, wenn es ein Element $w \in \mathbb{F}_p$ gibt mit Quadrat N, wenn also in $\mathbb{Z}$ die Kongruenz $w^2 \equiv N \bmod p$ eine Lösung hat. Für $p > 2$ hat $f(x) = 0$ in $\mathbb{F}_p$ dann die beiden Nullstellen

$$x = \left\lceil\sqrt{N}\right\rceil \pm w\,;$$

andernfalls gibt es keine Lösung. Im Falle $p = 2$ ist jedes Element von $\mathbb{F}_2 = \{0, 1\}$ sein eigenes Quadrat; hier ist $x = N + \left\lceil\sqrt{N}\right\rceil \bmod 2$ die einzige Lösung.

Insbesondere kann also $f(x)$ nur dann durch p teilbar sein, wenn N modulo p ein Quadrat ist; dies ist für etwa die Hälfte aller Primzahlen der Fall. Offensichtlich sind alle anderen Primzahlen nutzlos, und wir können sie aus der Faktorbasis streichen.

Für die verbleibenden p können wir die beiden Lösungen der Gleichung $f(x) = 0$ in $\mathbb{F}_p$ berechnen: Im Vergleich zum sonstigen Aufwand der Faktorisierung ist die Nullstellensuche durch Probieren durchaus vertretbar, allerdings kann man Quadratwurzeln modulo p mit etwas besseren Zahlentheoriekenntnissen auch sehr viel schneller berechnen *bzw.* zeigen, daß sie nicht existieren. Als Beispiel möchte ich nur den einfachsten Fall betrachten:

Falls es für $p \equiv 3 \bmod 4$ ein $w \in \mathbb{Z}$ gibt mit $w^2 \equiv N \bmod p$, sagt uns der kleine Satz von FERMAT, daß $w^{p+1} = w^{p-1} \cdot w^2 \equiv N \bmod p$ ist. Modulo p läßt sich die linke Seite auch schreiben als $N^{(p+1)/2} \bmod p$, wobei der Exponent $(p+1)/2$ wegen der Voraussetzung $p \equiv 3 \bmod 4$ immer noch eine gerade Zahl ist. Somit können wir auch mit $(p+1)/4$ potenzieren, und $N^{(p+1)/4} \equiv \pm w \bmod p$. Damit ist eine Quadratwurzel von N modulo p als Potenz von N dargestellt und somit berechenbar. Wenn wir nicht wissen, ob die Gleichung $x^2 \equiv N \bmod p$ eine Lösung hat, können wir auch das leicht entscheiden: Wir berechnen $w = N^{(p+1)/4} \bmod N$ und testen, ob $w^2 \equiv N \bmod p$. Falls ja, ist die Gleichung lösbar, und wir haben auch gleich eine Lösung gefunden. Ist aber $w^2 \not\equiv N \bmod p$, so kann es keine Lösung geben, denn gäbe es eine, müßte – wie wir uns gerade überlegt haben – auch $N^{(p+1)/4} \bmod N$ eine sein.

Für $p \equiv 1 \bmod 4$, gibt es aufwendigere, aber durchaus handhabbare Verfahren sowohl für die Entscheidung, ob die Kongruenz $x^2 \equiv N \bmod p$ lösbar ist, als auch zur Berechnung der Quadratwurzel; für Einzelheiten sei auf die Zahlentheorievorlesung verwiesen.

Da die Primzahlen in der Faktorbasis üblicherweise höchstens in der Größenordnung einer Million sind, führt aber auch das einfachste und offensichtlichste Verfahren relativ schnell zum Erfolg: Man teste einfach die Quadrate der Zahlen von Eins bis $(p-1)/2$ modulo p. Falls eines davon kongruent N ist, haben wir eine Wurzel gefunden; andernfalls gibt es keine, denn die Zahlen von $(p-1)/2+1$ bis $p-1$ sind einfach

die Negativen der Zahlen von Eins bis $(p-1)/2$ und haben somit die gleichen Quadrate.

Sobald eine Wurzel von N modulo p gefunden ist, können wir die beiden Nullstellen x_1, x_2 von f modulo p bestimmen und wissen, daß $f(x)$ genau dann durch p teilbar ist, wenn $x \equiv x_1 \bmod p$ oder $x \equiv x_2 \bmod p$.

Wir legen ein Siebintervall fest; dieses kann beispielsweise alle natürlichen Zahlen von Eins bis zu einer gewissen Grenze M enthalten oder aber alle ganzen Zahlen von $-M$ bis M. Falls N keine Quadratzahl ist, kann $f(x)$ nicht verschwinden; somit existiert für jedes x aus dem Siebintervall der Logarithmus von $|f(x)|$. Diese Logarithmen L_x (zu irgendeiner festen Basis) berechnen wir näherungsweise und speichern sie. Aus Effizienzgründen arbeitet man hier zweckmäßigerweise mit Festkommaarithmetik; oft beschränkt man sich einfach auf einen ganzzahligen Näherungswert für den Logarithmus zur Basis zwei.

Nun betrachten wir nacheinander die Primzahlen p aus der Faktorbasis $\mathcal{B}$, berechnen jeweils die beiden Lösungen x_1 und x_2 der Kongruenz $f(x) \equiv 0 \bmod p$ und ersetzen ausgehend von L_{x_1} und von L_{x_2} jedes p-te L_x durch $L_x - \log p$.

Falls $f(x)$ ein Produkt von Primzahlen aus $\mathcal{B}$ ist, sollte L_x nach Ende des Siebens bis auf Rundungsfehler gleich null sein; um keine Fehler zu machen, untersuchen wir daher für alle L_x mit Betrag unterhalb einer gewissen Grenze durch Abdividieren, ob das zugehörige $f(x)$ über $\mathcal{B}$ wirklich komplett faktorisiert und bestimmen auf diese Weise auch noch, *wie* es faktorisiert. Wenn wir mit einem Intervall der Form $[-M, M]$ arbeiten, kann $f(x)$ auch negative Werte annehmen; um dies zu berücksichtigen, betrachten wir dann $p = -1$ bei den Primzerlegungen als zusätzliches Element der Faktorbasis $\mathcal{B}$.

Für $f(x_i) = \prod_{p \in \mathcal{B}} p^{e_{ip}}$ ist $\prod_{i=1}^{r} f(x_i)^{\varepsilon_i} = \prod_{p \in \mathcal{B}} p^{\sum_{i=1}^{r} \varepsilon_i e_{ip}}$ genau dann ein Quadrat, wenn $\sum_{i=1}^{r} \varepsilon_i e_{ip}$ für alle $p \in \mathcal{B}$ gerade ist. Dies hängt natürlich nur ab von den $\varepsilon_i \bmod 2$ und den $e_{ip} \bmod 2$; wir können ε_i und e_{ip} daher als Elemente des Körpers mit zwei Elementen auffassen und bekommen

dann über $\mathbb{F}_2$ das Gleichungssystem

$$\sum_{i=1}^{r} \varepsilon_i e_{ip} = 0 \quad \text{für alle } p \in \mathcal{B}\,.$$

Betrachten wir die ε_i als Variablen, ist dies ein homogenes lineares Gleichungssystem in r Variablen mit soviel Gleichungen, wie es Primzahlen in der Faktorbasis gibt. Dieses Gleichungssystem hat nichttriviale Lösungen, falls die Anzahl der Variablen die der Gleichungen übersteigt, falls es also mehr Zahlen x_i gibt, für die $f(x_i)$ über der Faktorbasis faktorisiert werden kann, als Primzahlen in der Faktorbasis.

Für jede nichttriviale Lösung $(\varepsilon_1, \ldots, \varepsilon_r)$ ist

$$\prod_{i=1}^{r} f(x_i)^{\varepsilon_i} \equiv \prod_{i=1}^{r} \left(x + \left\lceil \sqrt{N} \right\rceil\right)^{2\varepsilon_i} \bmod N$$

eine Relation der Form $x^2 \equiv y^2 \bmod N$, die mit einer Wahrscheinlichkeit von etwa ein halb zu einer Faktorisierung von N führt. Falls wir zehn linear unabhängige Lösungen des Gleichungssystems betrachten, führt also mit einer Wahrscheinlichkeit von etwa 99,9% mindestens eine davon zu einer Faktorisierung.

Dazu brauchen wir allerdings nicht die Quadrate x^2 und y^2, sondern der Wurzeln

$$x = \prod_{p \in \mathcal{B}} p^{\frac{1}{2} \sum_{i=1}^{r} \varepsilon_i e_{ip}} \quad \text{und} \quad y = \prod_{i=1}^{r} \left(x + \left\lceil \sqrt{N} \right\rceil\right)^{\varepsilon_i}\,.$$

wobei beide Produkte nur modulo N berechnet werden müssen. Falls wir Glück haben, sind ggT$(x \pm y, N)$ echte Faktoren von N; andernfalls müssen wir anhand einer anderen Lösung des Gleichungssystems neue Kandidaten x und y bestimmen.

Zum besseren Verständnis des Verfahrens wollen wir versuchen, damit die Zahl 5 352 499 zu faktorisieren. Dies ist zwar eine sehr untypische Anwendung, da das quadratische Sieb üblicherweise erst für mindestens etwa vierzigstellige Zahlen angewandt wird, aber zumindest das Prinzip sollte auch damit klar werden.

Als Faktorbasis $\mathcal{B}$ verwenden wir die Menge aller Primzahlen $p < 100$, modulo derer p ein Quadrat ist; als Siebintervall nehmen wir die natürlichen Zahlen von 1 bis 20 000. Da die Quadratwurzel von N ungefähr 2313,546844 ist, betrachten wir das Polynom

$$f(x) = (x - 2\,313)^2 - 5\,352\,499\,.$$

Als erstes müssen wir seine Nullstellen modulo p bestimmen. Nachrechnen zeigt, daß N für 14 der 25 Primzahlen kleiner 100 ein Quadrat ist; die Nullstellen von $f(x)$ mod p sind in der folgenden Tabelle zu finden:

$p =$	3	5	11	13	17	19	23
$x_{1/2} =$	1, 2	0, 4	0, 5	4, 11	6, 9	2, 8	0, 20
$p =$	31	41	43	53	59	83	89
$x_{1/2} =$	27, 28	3, 4	26, 35	39, 52	2, 33	35, 70	23, 68

Damit könne wir sieben; von den 20 000 Werten aus dem Siebintervall bleiben 18 übrig, für die $f(x)$ über der Faktorbasis zerfällt:

i	x_i	$f(x_i)$ = *Faktorisierung*
1	23	$104397 = 3 \cdot 17 \cdot 23 \cdot 89$
2	121	$571857 = 3 \cdot 11 \cdot 13 \cdot 31 \cdot 43$
3	533	$2747217 = 3 \cdot 11 \cdot 17 \cdot 59 \cdot 83$
4	635	$3338205 = 3 \cdot 5 \cdot 13 \cdot 17 \cdot 19 \cdot 53$
5	741	$3974417 = 31 \cdot 41 \cdot 53 \cdot 59$
6	895	$4938765 = 3 \cdot 5 \cdot 13 \cdot 19 \cdot 31 \cdot 43$
7	2013	$13361777 = 11 \cdot 13 \cdot 41 \cdot 43 \cdot 53$
8	2185	$14879505 = 3 \cdot 5 \cdot 17 \cdot 23 \cdot 43 \cdot 59$
9	2477	$17591601 = 3 \cdot 31 \cdot 43 \cdot 53 \cdot 83$
10	2649	$19268945 = 5 \cdot 19 \cdot 43 \cdot 53 \cdot 89$
11	4163	$36586077 = 3 \cdot 11 \cdot 19 \cdot 23 \cdot 43 \cdot 59$
12	4801	$45256497 = 3 \cdot 11 \cdot 13 \cdot 31 \cdot 41 \cdot 83$
13	5497	$55643601 = 3 \cdot 13 \cdot 17 \cdot 23 \cdot 41 \cdot 89$
14	6253	$68023857 = 3 \cdot 11 \cdot 19 \cdot 23 \cdot 53 \cdot 89$
15	10991	$171643917 = 3 \cdot 17 \cdot 23 \cdot 41 \cdot 43 \cdot 83$
16	11275	$179281245 = 3 \cdot 5 \cdot 11 \cdot 13 \cdot 19 \cdot 53 \cdot 83$
17	14575	$279852045 = 3 \cdot 5 \cdot 11 \cdot 17 \cdot 19 \cdot 59 \cdot 89$
18	18535	$429286605 = 3 \cdot 5 \cdot 11 \cdot 23 \cdot 31 \cdot 41 \cdot 89$

Wir suchen Produkte der Zahlen $f(x_i)$, die Quadratzahlen sind, in denen also alle vierzehn Primzahlen mit geraden Exponenten vorkommen. Dazu müssen wir über dem Körper mit zwei Elementen das folgende lineare Gleichungssystem lösen

$$
\begin{array}{lcccccccccccccccccc}
p{=}3\colon & \varepsilon_1{+} & \varepsilon_2{+} & \varepsilon_3{+} & \varepsilon_4{+} & & \varepsilon_6{+} & & \varepsilon_8{+} & \varepsilon_9{+} & & \varepsilon_{11}{+} & \varepsilon_{12}{+} & \varepsilon_{13}{+} & \varepsilon_{14}{+} & \varepsilon_{15}{+} & \varepsilon_{16}{+} & \varepsilon_{17}{+} & \varepsilon_{18} = 0 \\
p{=}5\colon & & & & \varepsilon_4{+} & & \varepsilon_6{+} & & \varepsilon_8{+} & & \varepsilon_{10}{+} & & & & & & & \varepsilon_{16}{+} & \varepsilon_{17}{+} & \varepsilon_{18} = 0 \\
p{=}11\colon & & \varepsilon_2{+} & \varepsilon_3{+} & & & & \varepsilon_7{+} & & & & \varepsilon_{11}{+} & \varepsilon_{12}{+} & & \varepsilon_{14}{+} & & \varepsilon_{16}{+} & \varepsilon_{17}{+} & \varepsilon_{18} = 0 \\
p{=}13\colon & & \varepsilon_2{+} & & \varepsilon_4{+} & & \varepsilon_6{+} & \varepsilon_7{+} & & & & & \varepsilon_{12}{+} & \varepsilon_{13}{+} & & & \varepsilon_{16} & & & = 0 \\
p{=}17\colon & \varepsilon_1{+} & & \varepsilon_3{+} & \varepsilon_4{+} & & & & \varepsilon_8{+} & & & & & \varepsilon_{13}{+} & & \varepsilon_{15}{+} & & \varepsilon_{17} & & = 0 \\
p{=}19\colon & & & & \varepsilon_4{+} & & \varepsilon_6{+} & & & & \varepsilon_{10}{+} & \varepsilon_{11}{+} & & & \varepsilon_{14}{+} & & \varepsilon_{16}{+} & \varepsilon_{17} & & = 0 \\
p{=}23\colon & \varepsilon_1{+} & & & & & & & \varepsilon_8{+} & & & \varepsilon_{11}{+} & & \varepsilon_{13}{+} & \varepsilon_{14}{+} & \varepsilon_{15}{+} & & & & \varepsilon_{18} = 0 \\
p{=}31\colon & & \varepsilon_2{+} & & & \varepsilon_5{+} & \varepsilon_6{+} & & & \varepsilon_9{+} & & & \varepsilon_{12}{+} & & & & & & & \varepsilon_{18} = 0 \\
p{=}41\colon & & & & & \varepsilon_5{+} & & \varepsilon_7{+} & & & & & \varepsilon_{12}{+} & \varepsilon_{13}{+} & & \varepsilon_{15}{+} & & & \varepsilon_{18} = 0 \\
p{=}43\colon & & \varepsilon_2{+} & & & & \varepsilon_6{+} & \varepsilon_7{+} & \varepsilon_8{+} & \varepsilon_9{+} & \varepsilon_{10}{+} & \varepsilon_{11}{+} & & & & \varepsilon_{15} & & & & = 0 \\
p{=}53\colon & & & & \varepsilon_4{+} & \varepsilon_5{+} & & \varepsilon_7{+} & & \varepsilon_9{+} & \varepsilon_{10}{+} & & & & \varepsilon_{14}{+} & & \varepsilon_{16} & & & = 0 \\
p{=}59\colon & & & \varepsilon_3{+} & & \varepsilon_5{+} & & & \varepsilon_8{+} & & & \varepsilon_{11}{+} & & & & & & & \varepsilon_{17} & & = 0 \\
p{=}83\colon & & & \varepsilon_3{+} & & & & & & \varepsilon_9{+} & & & \varepsilon_{12}{+} & & & \varepsilon_{15}{+} & \varepsilon_{16} & & & = 0 \\
p{=}89\colon & \varepsilon_1{+} & & & & & & & & & \varepsilon_{10}{+} & & & \varepsilon_{13}{+} & \varepsilon_{14}{+} & & & & \varepsilon_{17}{+} & \varepsilon_{18} = 0
\end{array}
$$

Dieses System können wir nach dem GAUSS-Algorithmus lösen; da wir über dem Körper mit zwei Elementen arbeiten, ist das auch bei dieser Größe leicht mit Bleistift und Papier möglich, denn Elimination einer Variablen bedeutet hier ja einfach, daß wir eine andere Gleichung, in der dieselbe Variable vorkommt, addieren. Beim vorliegenden System können wir zum Beispiel die Gleichung für $p = 3$ zu denen für $p = 17, 23$ und 89 addieren; danach kommt die Variable ε_1 nur noch in der ersten Gleichung vor, und so weiter. In der Endgestalt lassen sich die Variablen $\varepsilon_{12}, \varepsilon_{14}, \varepsilon_{15}, \varepsilon_{16}, \varepsilon_{17}$ und ε_{18} frei wählen; wir erhalten also einen sechsdimensionalen Lösungsraum. Er besteht aus allen Vektoren der Form

(b, e+c, e+c, e+c, b+c+a, b+d, c+f, b, e+d+f, b+e+a+d, b+e+a, f, d+b, e, d, c, b, a)

mit $a, b, c, d, e, f \in \mathbb{F}_2$. Setzen wir hier beispielsweise $a = b = f = 1$ und $c = d = e = 0$, führt dies auf den Vektor

$$(1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0)\,;$$

wir müssen also das Produkt x der x_i und –nach der obigen Formel – die Wurzel y des Produkts der $f(x_i)$ mit $i \le 8$, sowie $i = 12, 13, 14, 18$ berechnen. Modulo N erhalten wir $x = 3\,827\,016$ und $y = 1\,525\,483$; leider ist $x + y = N$ und $x - y$ ist teilerfremd zu N, so daß wir mit dieser Lösung nichts anfangen können.

Setzen wir stattdessen $c = d = 0$ und $a = b = e = f = 1$, erhalten wir den Vektor $(1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0)$, der uns auf $x = 4\,093\,611$ und $y = 1\,020\,903$ führt. Hier hat $x - y = 3\,072\,708$ den ggT 1 237 mit N, wir haben also einen Faktor gefunden. Der andere ist $N/1\,237 = 4\,327$, damit ist die Faktorisierung $5\,352\,499 = 1\,237 \cdot 4\,327$ gefunden.

Bei realistischen Anwendungen des quadratischen Siebs kommen wir natürlich nicht auf ein lineares Gleichungssystem mit nur vierzehn Gleichungen und achtzehn Unbekannten; da bewegen sich beide Anzahlen mindestens im sechsstelligen Bereich, bei neueren Rekordfaktorisierungen sogar im neunstelligen. Früher konnten solche Gleichungssysteme nur auf Supercomputern gelöst werden (während das Sieben natürlich problemlos auch mit einfachen PCs möglich ist); heute kann auch dieser Schritt bei geschickter Parallelisierung auf PCs ausgeführt werden.

c) Varianten des quadratischen Siebs

Der Rechenaufwand beim quadratischen Sieb entfällt größtenteils auf das Sieben: Nur ein verschwindend kleiner Teil aller Zahlen zerfällt über der gewählten Faktorbasis, und je größer x wird, desto weniger dicht liegen diese Zahlen. Bei einer Zahl um 10^{100} und einer Faktorbasis aus 10 000 Primzahlen etwa kann man für $x \le 10^{10}$ etwa fünf vollständig faktorisierbare Werte von $f(x)$ erwarten, im neunmal so großen Intervall $[10^{10}, 10^{11}]$ nur noch etwa 23 und so weiter.

Zumindest qualitativ ist dies klar, denn je größer die Zahlen werden, desto größer wird die Wahrscheinlichkeit großer Primfaktoren. Eine Abschätzung mit (teils nur heuristischen) Formeln über die Verteilung von Primfaktoren zeigt, daß man in einem solchen Fall ein Intervall sieben muß, das bis über 10^{14} hinausreicht. Verbesserungen des quadratischen Siebs konzentrieren sich daher darauf, die Siebphase zu optimieren um so in kürzerer Zeit mehr Relationen zu finden.

1.) Die Multipolynomialversion: Die Multipolynomialversion des quadratischen Siebs optimiert dieses an zwei Stellen: Einmal betrachtet sie außer dem Polynom $\left(x - [\sqrt{N}]\right)^2 - N$ noch weitere Polynome, um Relationen zu bekommen, so daß man jedes dieser Polynome nur über ein kürzeres Intervall sieben muß; zum andern betrachtet sie auch negative Werte von x, so daß – bei geschickt gewählten Polynomen f – der Betrag von $f(x)$ für ein längeres Intervall klein bleibt.

Als Polynome betrachtet man quadratische Polynome der Form

$$f(x) = ax^2 + 2bx + c \text{ mit } 0 \le b < a \text{ und } b^2 - ac = N\,.$$

Für diese ist $af(x) = (ax+b)^2 - b^2 + ac = (ax+b)^2 - N$, so daß $af(x)$ zwar kongruent $(ax + b)^2$ ist, aber nicht gleich. Auch diese Polynome liefern also die Art von Relationen, die wir brauchen, und sie können genauso gesiebt werden wie das spezielle Polynom aus dem letzten Abschnitt.

Ein gewisser Nachteil dabei ist, daß man vor dem Sieben für jedes neue Polynom f und jede Primzahl p neu die Nullstellen von f modulo p ausrechnen muß. Da aber alle Polynome quadratisch sind mit Diskriminante N, steht in der Lösungsformel für die quadratische Gleichungen stets N unter der Wurzel, so daß man nur einmal die Wurzeln von N modulo p berechnen muß; danach lassen sich *alle* quadratischen Gleichungen mit wenigen Rechenoperationen lösen. Verglichen mit der Siebzeit fällt dies praktisch nicht ins Gewicht. Daher verwendet man typischerweise sehr viele Polynome und dafür relativ kurve Siebintervalle.

Zur Konstruktion von Polynomen f wählt man zunächst eine Zahl a so, daß das Polynom in einem Intervall $[-M, M]$ möglichst beschränkt bleibt. Falls a, b, c deutlich kleiner sind als M, liegt der Maximalwert von $f(x)$ an den Intervallenden, liegt das Maximum bei

$$f(-M) \approx \frac{1}{a}\left(a^2M^2 - N\right);$$

das Minimum wird bei $x = -b/a$ angenommen und ist

$$f\left(-\frac{b}{a}\right) = \frac{b^2}{a} - \frac{2b^2}{a} + c = \frac{-b^2 + ac}{a} = -\frac{N}{a}\,.$$

Für $a \approx \sqrt{2N}/M$ haben beide Zahlen ungefähr denselben Betrag, aber entgegengesetzte Vorzeichen; also wählen wir a in dieser Größenordnung.

Da $b^2 - 4ac = N$ werden muß, kommen für b nur Werte in Frage, für die $b^2 \equiv N \bmod a$ ist, uns sobald ein solches b gewählt ist, liegt auch c eindeutig fest.

Die Anzahl der Polynome, die Polynome selbst und die Zahl M sollten idealerweise so gewählt werden, daß die Rechenzeit minimal wird. Deren Abschätzung hängt ab von einer ganzen Reihe von Größen, die teils nur mit großem Aufwand berechnet werden können, teils nur modulo unbewiesener Vermutungen wie etwa der RIEMANN-Vermutung bekannt sind und für die teils sogar nur rein heuristische Formeln existieren. Ich möchte auf die damit verbundenen Probleme nicht eingehen, sondern nur das Ergebnis angeben, wonach der Rechenaufwand zum Faktorisieren einer Zahl N mit der Multipolynomialvariante des quadratischen Siebs proportional ist zu $e^{c\sqrt{\ln N \ln\ln N}}$ mit einer Konstanten c, die von der Wahl der verschiedenen Parameter abhängt.

2.) Das Zahlkörpersieb: Die derzeit schnellste Verbesserung des quadratischen Siebs ist das *Zahlkörpersieb*, das nicht mehr mit quadratischen Polynomen arbeitet, sondern mit Polynomen beliebigen Grades.

Der Grad d dieser Polynome wird in Abhängigkeit der zu faktorisierenden Zahl N festgelegt, sodann wählt man führende Koeffizienten a_d und eine natürliche Zahl

$$m \approx \sqrt[d]{\frac{N}{a_d}}\,.$$

Alle Polynome sind homogen, haben also die Form

$$F(x,y) = a_d x^d + a_{d-1}x^{d-1}y + \cdots + a_1 x y^{d-1} + a_0 y^d\,,$$

wobei die a_i mit $i < d$ höchstens Betrag $m/2$ haben.

Hinzu kommt das homogene lineare Polynom $G(x,y) = x - my$; das Sieb sucht nach Zahlenpaaren (x,y), für die sowohl $F(x,y)$ als auch $G(x,y)$ über der Faktorbasis zerfallen. Da die Polynome von zwei Variablen abhängen, muß man entweder jeweils eine Variable festhalten

und über die andere sieben, oder aber ein zweidimensionales Siebverfahren anwenden; üblich ist eine Kombination beider Methoden.

Gesucht sind Paare (x, y) teilerfremder ganzer Zahlen, für die

$$F(x, y) \quad \text{und} \quad G(x, y)$$

beide über der gewählten Faktorbasis zerfallen, und das Ziel ist, wie beim quadratischen Sieb, eine Relation der Form

$$\prod_{(x,y)\in\mathcal{M}} F(x, y) \equiv \prod_{(x,y)\in\mathcal{M}} G(x, y) \mod N\,,$$

in der rechts und links Quadrate stehen.

Die besten derzeit bekannten Strategien zur Polynomauswahl *usw.* führen auf eine Laufzeitabschätzung proportional

$$e^{c(\ln N)^{\frac{1}{3}}(\ln\ln N)^{\frac{2}{3}}} \quad \text{mit} \quad c = \sqrt[3]{\frac{64}{9}} \approx 1{,}923$$

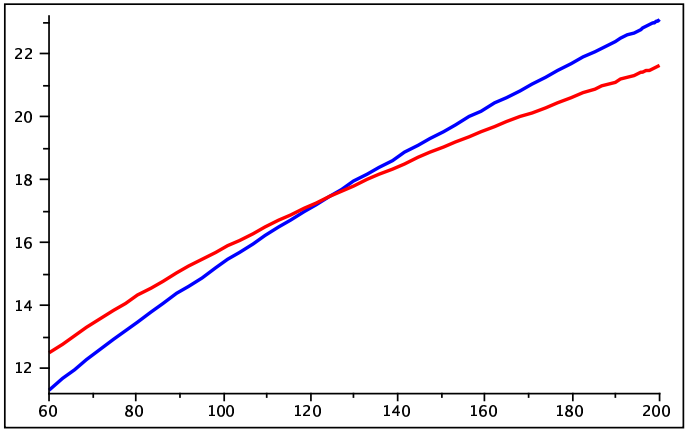
für die Faktorisierung einer Zahl N nach dieser Methode.

Für hinreichend große N ist diese Methode offensichtlich schneller als das quadratische Sieb, bei dem $\ln N$ als Quadratwurzel im Exponenten steht; in der Abbildung, wo der Aufwand für die Faktorisierung einer Zahl 10^x aufgetragen ist, sieht man, daß das Zahlkörpersieb (rote Linie) ab etwa 125-stelligen Zahlen dem quadratischen Sieb (blaue Linie) überlegen ist.

d) Faktorisierungsrekorde

Neue Faktorisierungsverfahren werden meist vorgestellt mit einem Faktorisierungsbeispiel, das den bisherigen Verfahren getrotzt hat. Berühmt sind dabei die sogenannten *ten most wanted factorisations* des *Cunningham-Projekts*, wo es vor allem um Zahlen der Form $b^n \pm a$ für kleine Werte von a und b. Mit diesen Problemen befaßten sich die algorithmischen Zahlentheoretiker schon lange vor der praktischen Bedeutung von Faktorisierungen im Zusammenhang mit dem RSA-Verfahren.

Im Zusammenhang mit der Kryptographie interessanter ist ein Liste von *challenges*, die *RSA Computer Security Incorperated* früher regelmäßig



Zeitaufwand fü quadratisches Sieb und Zahlkörpersieb

zusammenstellte, denn hier geht es um Zahlen, die sorgfältig unter kryptographischen Gesichtspunkten ausgewählt wurden. Die größte im Rahmen der *challenge* erfolgreich faktorisierte Zahl war RSA-200, eine 200-stellige Dezimalzahl (entsprechend 663 Bit), deren Faktorisierung am 8. Mail 2005 beendet war; zwei Jahre später beendete *RSA Computer Security Incorperated* den Wettbewerb.

Die Zahlen sind allerdings weiterhin im Netz zu finden, und am 3. Dezember 2009 wurde mit RSA-768 die bislang größte davon faktorisiert, die 232-stellige Dezimalzahl

123018668453011775513049495838496272077285356959533479219732245215172640050726365751874520219978646938995647494277406384592519255732630345373154826850791702612214291346167042921431160222124047927473779408066535141959745985690214341 3

mit den beiden Faktoren

p = 33478071698956898786044169848212690817704794983713768568912431388982883793878002287614711652531743087737814467999489

$$q = 367460436667995904282446337996279526322791581643430876426760322838157396665112792333734171433968102700927987363089 17$$

Die dreizehn an der Faktorisierung beteiligten Autoren kommen von Universitäten in Amsterdam, Bonn, Lausanne, Nancy und Tokyo und brauchten dazu etwa zweieinhalb Jahre. Sie verwendeten das Zahlkörpersieb, wobei sie das erste halbe Jahr damit verbrachten, achtzig Prozessoren nach geeigneten Polynomen suchen zu lassen. Der Hauptteil der Arbeit, das Sieben wurde auf „viele hundert“ Computer verteilt und dauerte zwei Jahre; für die restlichen Aufgaben reichten wenige Tage und eine zweistellige Anzahl von Prozessoren. Interessierte Leser finden einen genaueren Bericht unter `eprint.iacr.org/2010/006.pdf` oder in den Proceedings der Konferenz Crypto 2010, veröffentlicht in den *Lecture Notes in Computer Science* Band 6223, Springer Verlag, 2010, auf den Seiten 333-350.

Wer zurückblättert zu §3*a)* wird dort finden, daß RSA-Moduln mit 768 Bit nach den in Deutschland geltenden Standards bis Ende 2000 als hinreichend sicher angesehen wurden – wenn auch schon 1998 festgestellt wurde, daß dies nur übergangsweise und definitiv nicht über Ende 2000 hinaus gelte.

Dies zeigt wieder einmal deutlich, daß kryptographische Sicherheit zeitabhängig ist und sollte uns warnen, daß auch die heute als sicher angesehenen Parameterwerte höchstwahrscheinlich in einigen Jahren geknackt werden können. Die nächste kritische Länge von RSA-Moduln sind die 1024 Bit, die bis Ende 2008 zulässig waren. Die Autoren der RSA-768-Faktorisierung sind sich ziemlich sicher, daß sie mit ihren Methoden in den nächsten fünf Jahren nicht in der Lage sein werden, den *challenge*-Modul RSA-1024 zu faktorisieren; danach, sagen sie, sei alles offen.

Wie lange die heute als sicher geltenden 2048-Bit-Moduln wirklich sicher sind, kann natürlich niemand vorhersagen; ein plötzlicher Durchbruch etwa bei den am Ende der Vorlesung betrachteten Quantencomputern könnte nicht nur sie, sondern das gesamte RSA-Verfahren ziemlich schnell unbrauchbar machen. Rein spekulativ können wir allerdings

die Ergebnisse der letzten Jahre extrapolieren und so zu einer vagen Abschätzung kommen, wann RSA-Moduln welcher Bitlänge möglicherweise faktorisiert werden können.

Am 22. August 1999 wurde die RSA *challenge* Zahl RSA-155 mit 512 Bit faktorisiert. Die 17 Autoren verglichen in ihrem Bericht (EUROCRYPT 2000, *Lecture Notes in Computer Science* **1807** (2000), S. 1–18) Faktorisierungsrekorde der bis dahin vergangenen dreißig Jahre, angefangen von der 1970 faktorisierten 39-stelligen FERMAT-Zahl $2^{2^7}+1$, die ein heutiges Computeralgebrasystem auf einem handelsüblichen Computer in weniger als zehn Sekunden faktorisiert, bis hin zum damaligen Rekord RSA-155. Sie fanden, daß sich das Jahr, in dem erstmalig eine (schwierige) d-stellige Zahl faktorisiert wurde, näherungsweise berechnen läßt als

$$13{,}24\sqrt[3]{d} + 1928{,}6\,.$$

Setzen wir einige der heute und in naher Zukunft oder Vergangenheit interessanten Bitlängen in diese Formel ein, erhalten wir folgende Tabelle:

Bit:	768	1024	1280	1536	2048	2560	3072	4096
Jahr:	2010	2018	2025	2031	2041	2050	2057	2070

Der einzige Wert, den wir überprüfen können, ist der für 768 Bit; hier hat die zehn Jahre alte Formel den Termin sehr genau vorhergesagt. Trotzdem kann sie uns natürlich nicht garantieren, daß die heute ratsamen 2048-Bit-Moduln nicht doch schon deutlich vor 2041 faktorisiert werden. Fortschritte bei der Faktorisierung kamen zumindest bisher zu ungefähr gleichen Teilen aus drei Entwicklungen: Neue mathematische Algorithmen, schnellere Computer und bessere Implementierungen. Auch in Zukunft wird es wohl auf allen drei Gebieten Fortschritte geben, auch wenn bei den mathematischen Algorithmen das Zahlkörpersieb nun schon seit ungewöhnlich langen zwanzig Jahren der beste bekannte Algorithmus ist,

e) Faktorisierung mit Spezialhardware

Faktorisierungen mit dem quadratischen oder Zahlkörpersieb benötigen zwar zumindest für einige Schritte wie die Lösung des linearen

Gleichungssystems leistungsfähige Rechner mit viel Speicher, für die Hauptarbeit, das Sieben, genügen aber einfachste Rechner, von denen dann allerdings zumindest bei Rekordfaktorisierungen sehr viele eine sehr lange Zeit rechnen müssen.

1999 schlug ADI SHAMIR, einer der Erfinder des RSA-Verfahrens, ein optoelektronisches Gerät vor, mit dem er das Sieben ungefähr um den Faktor Tausend beschleunigen wollte; er nannte es TWINKLE: **T**he **W**eizman **IN**stitute **K**ey **L**ocating **E**ngine.

Das Gerät sitzt in einer schwarzen Röhre mit etwa 15 cm Durchmesser und 25 cm Länge, deren wesentlicher Chip im Innern etwa eine Million LEDs enthält. Jede dieser LEDs steht für eine Primzahl p aus der Faktorbasis und hat eine Leuchtkraft proportional $\log_2 p$, was etwa über ihre Größe oder (einfacher) mittels einer Abdeckfolie mit kontinuierlichem Grauschleier realisiert werden kann.

Hierin liegt der wesentliche Unterschied zu Software-Implementierungen der Siebe: Dort werden die Primzahlen nacheinander behandelt, während den x-Werten Speicherzellen entsprechen. Bei TWINKLE werden die x-Werte auf die Zeitachse abgebildet; da die x-Werte, für die $f(x)$ durch p teilbar ist, von der Form $x_{1/2} + kp$ sind, muß also jede LED periodisch aufleuchten, was nicht schwer zu realisieren ist. Die Taktrate, mit der das Gerät arbeitet, soll bei 10 GHz liegen, ein Wert, der in optischen Hochgeschwindigkeitsnetzen heute durchaus normal ist.

In jedem Takt mißt ein den LEDs gegenüberliegender Sensor die Gesamtlichtstärke. Da ein Takt nur eine Länge von 10^{-10} Sekunden hat, läßt sich die Ausbreitungsgeschwindigkeit des Lichts hier nicht vernachlässigen; bei einer Geschwindigkeit von etwa 300 000 km/sec legt es pro Takt etwa drei Zentimeter zurück. Die Laufwegunterschiede der Lichtstrahlen zwischen den verschiedenen Dioden und der Meßzelle müssen also deutlich kleiner als drei Zentimeter sein. Dies wird dadurch erreicht, daß alle LEDs auf einem einzigen Wafer sitzen.

Die Meßgenauigkeit der Zelle muß nicht übermäßig hoch sein: Beim klassischen Sieb arbeitet man schließlich auch nur mit ganzzahligen Approximationen der $\log_2 p$. Eine Zahl x ist uninteressant, wenn zum entsprechenden Zeitpunkt eine Lichtstärke gemessen wird, die kleiner

ist als $\log_2 f(x)$ minus einem Sicherheitsabstand; ansonsten wird sie an konventionelle Elektronik weitergereicht und dort bearbeitet. Wie wir oben gesehen haben, ist dies ein sehr seltenes Ereignis, das im Schnitt höchstens einmal pro einer Milliarde x-Werte eintritt, also etwa zehnmal pro Sekunde. Mit diesen Datenraten können auch einfache Computer leicht fertig werden.

Das Hauptproblem ist der Bau das Chips mit den Dioden und deren Steuerelektronik; SHAMIR meint, daß dies mit der heute existierende GaAs-Technologie gerade möglich sein sollte, und möglicherweise stehen inzwischen schon solche oder ähnliche Maschinen in Labors von NSA und ähnlichen Organisationen. In der offenen Literatur ist nicht über die Existenz solcher Maschinen bekannt und auch nichts über Pläne welche zu bauen. Der Entwicklungsaufwand dürfte sicherlich in die Hunderttausende oder gar Millionen gehen, aber SHAMIR schätzt, daß das Gerät dann mit Stückkosten von etwa 5 000 $ hergestellt werden kann.

2000 stellte SHAMIR zusammen mit A. LENSTRA, einem der Erfinder des Zahlkörpersiebs, eine verbesserte Version vor; die beiden Autoren schätzen, daß diese Version zusammen mit 15 PCs eine 512-Bit-Zahl in einem halben Jahr faktorisieren kann. Wegen der sehr guten Parallelisierbarkeit des Zahlkörpersiebs könnte man mit mehr TWINKLEs und PCs natürlich auf deutlich kürzere Zeiten kommen.

Für 768-Bit-Faktorisierungen schätzen sie den Aufwand auf fünf Tausend TWINKLEs, unterstützt von achtzig Tausend PCs, bei einem Zeitbedarf von insgesamt neun Monaten.

Ob TWINKLE oder ein ähnliches Gerät je gebaut wurde, ist unbekannt; in der offenen Literatur ist jedenfalls nichts zu finden.

2003 schlugen SHAMIR und ERAN TROMER ein neues Gerät vor namens TWIRL, **T**he **W**eizman **I**nstitute **R**elation **L**ocator. Im Gegensatz zu TWINKLE arbeitet es rein elektronisch: Anstelle einer Diode ist jeder Primzahl ein Addierer zugeordnet der, so er aktiviert wird, einen Näherungswert für den Logarithmus dieser Primzahl subtrahiert.

Es ist klar, daß nicht alle aktivierten Addierer gleichzeitig mit derselben Zahl x rechnen können, deshalb sind die Addierer (ähnlich wie bei

einem Vektorrechner) in einer Pipeline realisiert: Während sich der erste Addierer (falls aktiviert) mit der Zahl x beschäftigt, ist der zweite für $x-1$ zuständig, der dritte für $x-3$ *usw.* Bei N Addierern dauert es also N Zeittakte, bis eine Zahl x vollständig verarbeitet ist, jedoch wird in jedem Zeittakt durch jeden Addierer eine Zahl geschickt. Je nachdem, ob dieser von der Steuerungselektronik für diesen Zeittakt aktiviert ist, subtrahiert er seine voreingestellte Zahl oder leitet seine Eingabe einfach weiter an den nächsten Addierer.

Zur zusätzlichen Beschleunigung gibt es für jede Primzahl aus der Faktorbasis nicht nur einen Addierer, sondern eine feste Anzahl m. Auf diese Weise läßt sich das Siebintervall in m Teilintervalle aufspalten und, wenn r deren Länge bezeichnet, werden im t-ten Zeittakt parallel die Zahlen $t, t+r, \ldots, t+(m-1)r$ in die Pipeline geschickt.

SHAMIR und TROMER rechnen damit, daß man mit so einem Gerät bei einem Kostenaufwand von zehn Millionen Dollar pro Jahr einen RSA-Schlüssel mit 1024 Bit faktorisieren könnte.

Auch im Falle von TWIRL gibt es keinen Hinweis, daß je so ein Gerät gebaut wurde; wenn man allerdings bedenkt, daß bei NSA Ingenieure sitzen, die sehr viel mehr Erfahrung mit dem Bau elektronischer Schaltungen haben als Wissenschaftler an einer Universität, spricht schon einiges dafür, daß es dort irgendwelche Hardware gibt, die 1024 Bit Zahlen faktorisieren kann – zehn Millionen Dollar sind beim Budget der NSA schließlich kein Problem, wenn es um die Entschlüsselung wirklich wichtiger Daten geht.

§9: Literatur

RSA ist immer noch das gebräuchlichste asymmetrische Kryptoverfahren; es gibt daher kaum ein nach etwa 1980 erschienenes Lehrbuch der Kryptologie, das nichts darüber enthält. Insbesondere wird RSA natürlich auch in den für die gesamte Vorlesung empfohlenen Büchern ausführlich behandelt.

Noch mehr Informationen findet man beispielsweise bei

WENBO MAO: *Modern Cryptography – Theory & Practice*, Prentice Hall, 2004,

wo insbesondere auch auf praktische Aspekte eingegangen wird, oder bei

SONG Y. YAN: *Cryptanalytic Attacks on RSA*, Springer 2008.

Dazu kommen eine ganze Reihe von Lehrbüchern der algorithmischen Zahlentheorie, die RSA behandeln, dabei aber ihr Hauptaugenmerk auf Primzahlen und auch Faktorisierung legen, beispielsweise

RICHARD CRANDALL, CARL POMERANCE: *Prime numbers – A Computational Perspective*, Springer, 2001

SAMUEL WAGSTAFF: *Cryptanalysis of Number Theoretic Ciphers*, Chapman & Hall/CRC, ^{2}2003

Mit Implementierungsfragen beschäftigt sich

MICHAEL WELSCHENBACH: *Kryptographie in C und C++*, Springer, 1998

§6 folgt weitgehend dem Artikel

DAN BONEH: *Twenty years of Attacks on the RSA Cryptosystem*, Notices of the AMS, February 1999; auch online verfügbar unter `www.ams.org/notices/199902/boneh.pdf`.

Kapitel 5
Verfahren mit diskreten Logarithmen

Die Sicherheit des RSA-Verfahrens hängt zusammen mit der Schwierigkeit der Zerlegung großer Zahlen in ihre Primfaktoren; direkt geht es allerdings um das Problem, aus $x^e \bmod N$ auf den Wert von x zu schließen, also die e-te Wurzel modulo N aus einer Zahl zu ziehen. Für eine Primzahl N ist das für die Exponenten, die wir bei RSA verwenden, problemlos möglich; für eine zusammengesetzte Zahl aber geht es anscheinend nur mit Kenntnis von deren Faktorisierung.

Eine zweite vor allem für elektronische Unterschriften populäre Gruppe von Verfahren baut stattdessen auf die Schwierigkeit, aus der Kenntnis von $a^x \bmod N$ bei bekanntem a auf den Wert von x zu schließen; hier geht es also um die Umkehrung einer modularen Exponentialfunktionen deren Umkehrfunktion bezeichnet man in Analogie zur Umkehrfunktion einer reellen Exponentialfunktion als diskreten Logarithmus (oder Index). Effiziente Verfahren, um ihn auch für große N sind selbst für große Primzahlen N nicht bekannt; da man diskrete Logarithmen modulo einer zusammengesetzten Zahl N leicht nach dem chinesischen Restesatz über die diskreten Logarithmen modulo der Primteiler von N berechnen kann, wählt man bei Kryptoverfahren auf der Basis diskreter Logarithmen N stets als eine Primzahl p.

Wir beginnen mit dem ältesten Beispiel eines solchen Verfahrens:

§ 1: Schlüsselaustausch nach Diffie und Hellman

Wie wir im letzten Kapitel gesehen haben, waren DIFFIE und HELLMAN mit ihrer Arbeit *New directions in cryptography* die Initiatoren der

asymmetrischen Kryptographie in der akademischen Welt; kurz nach dieser Arbeit entwickelten sie auch ein entsprechendes Verfahren, das zwar nicht zur Verschlüsselung dienen konnte, aber dafür die bis heute wichtigste Aufgabe der Kryptographie mit öffentlichen Schlüssel lösen konnte: Die Vereinbarung eines Schlüssels über eine unsichere Leitung.

Im Gegensatz zum RSA-Verfahren brauchen sie dazu nicht einmal öffentliche Schlüssel: Die beiden Teilnehmer können miteinander sicher kommunizieren ohne zuvor irgendwelche öffentlichen oder privaten Schlüssel zu kennen. Damit ist dieses Verfahren vor allem interessant im privaten Bereich, wo zertifizierte öffentliche Schlüssel oder gelegentlich auch überhaupt die Speicherung von Schlüsseln zu aufwendig wäre.

a) Das Verfahren

Die beiden Teilnehmer einigen sie sich zunächst (über die unsichere Leitung) auf eine Primzahl p und eine natürliche Zahl a derart, daß die Potenzfunktion $x \mapsto a^x$ möglichst viele Werte annimmt. Als nächstes wählt Teilnehmer A eine Zufallszahl $x < p$ und B entsprechend ein $y < p$. A schickt $u = a^x \bmod p$ an B und erhält dafür $y = a^y \bmod p$ von diesem. Sodann berechnet A die Zahl

$$v^x \bmod p = \left(a^y\right)^x \bmod p = a^{xy} \bmod ap$$

und B entsprechend

$$u^y \bmod p = \left(a^x\right)^y \bmod p = a^{xy} \bmod p\,;$$

beide haben also auf verschiedene Weise dieselbe Zahl berechnet, die sie zum Beispiel verwenden können, um daraus einen Schlüssel für ein symmetrisches Kryptosystem zu bestimmen. Verfahren dazu gibt es mehr als genug: Sie könnten etwa die letzten oder sonst irgendwelche Bits dieser Zahl verwenden, aber auch einen irgendwie definierten Hashwert.

Ein Gegner, der den Datenaustausch abgehört hat, kennt die Zahlen p, a, u und v; er kann also problemlos alle möglichen Zahlen modulo p der Art $a^{\alpha x+\beta y} = u^\alpha \cdot v^\beta$ berechnen. Es fällt aber schwer, sich eine Art und Weise vorzustellen, wie er $a^{xy} \bmod p$ finden kann, ohne den diskreten Logarithmus von u oder v zu berechnen. (Bewiesen ist hier, wie üblich, natürlich nichts.)

b) Die *man in the middle attack*

Da der Gegner die Wahl der Mittel hat, muß er nicht notwendigerweise die mathematische Seite des Verfahrens angreifen: Er kann angreifen, was immer er für eine vielversprechende Schwachstelle hält.

Nehmen wir etwa an, der Gegner habe eine gewisse Kontrolle über das Netz, in dem der Datenaustausch stattfindet – beispielsweise, weil er Systemverwalter eines für die betreffende Verbindung unbedingt notwendigen Knotenrechners ist. Dann kann er eine sogenannte *man in the middle attack* durchführen: Er fängt alle Datenpakete zwischen A und B ab und ersetzt sie durch selbstfabrizierte eigene Pakete.

Damit kann er sich gegenüber A als B auszugeben und umgekehrt: Alles, was A an B zu schicken glaubt, geht tatsächlich an den Gegner G, und alles was B von A zu erhalten glaubt, kommt tatsächlich von G. In Gegenrichtung ist es natürlich genauso.

Im einzelnen läuft der Angriff folgendermaßen ab:

Falls die Zahlen a und p nicht ohnehin Konstanten eines Verbunds sind, dem A und B angehören, läßt G die Kommunikation, die zu deren Vereinbarung führt, ungehindert zu: In diesem Stadium beschränkt er sich auf reines Abhören.

Als nächstes wählen A und B ihre Zufallszahlen $x < p$ und $y < p$; gleichzeitig wählt G eine Zufallszahl $z < p$ oder vielleicht auch zwei verschiedene solche Zahlen z_A und z_B für die beiden Teilnehmer.

Wenn A die Zahl $u = a^x \bmod p$ an B schickt, fängt G diese Nachricht ab und ersetzt sie durch $w_B = a^{z_B} \bmod p$; entsprechend fängt er Bs Nachricht $y = a^y \bmod p$ ab und schickt stattdessen $w_A = a^{z_A}$ an A. Dies führt dazu, daß am Ende A und G einen gemeinsamen Schlüssel s_A haben und B und G einen gemeinsamen Schlüssel s_B. Sowohl A als auch B glauben, der ihnen bekannte Schlüssel s_A *bzw.* s_B sei aus $a^{xy} \bmod p$ abgeleitet und senden nun damit verschlüsselte Nachrichten an ihren Partner. Diese Nachrichten fängt G ab, entschlüsselt sie mit dem Schlüssel, den er mit dem Absender gemeinsam hat, und verschlüsselt sie anschließend, gegebenenfalls nach einer seinen Interessen entsprechenden Modifikation, mit dem Schlüssel, den er mit dem Empfänger gemeinsam hat. Auf

diese Weise hat er die gesamte Konversation unter Kontrolle, ohne daß A und B etwas merken.

Die Möglichkeit für diese Attacke kommt natürlich daher, daß sich A und B nicht sicher sein können, den jeweils anderen am anderen Ende der Leitung zu haben. Die kryptographisch einwandfreie Modifikation, die das Verfahren gegen diese Art von Angriff sicher macht, bestünde beispielsweise darin, daß A und B ihre Nachrichten x und y vor dem Versenden unterschreiben – aber dann verschwindet auch wieder der Vorteil, daß sie ohne Kenntnis irgendeines Schlüssels miteinander kommunizieren können: Zur Verifikation einer Unterschrift braucht man schließlich den öffentlichen Schlüssel des Unterschreibenden.

Falls sich A und B hinreichend gut kennen, um die Stimme des jeweils anderen am Telephon einigermaßen sicher zu erkennen, können sie diese Art von Attacke auch dadurch erschweren, daß sie nach dem Austausch von u und v per Telephon über diese Zahlen (z.B. die 317. bis 320. Ziffer) und gegebenenfalls auch noch über Schwänke aus ihrer gemeinsamen Jugendzeit reden; dann müßte der Angreifer zusätzlich noch ein begabter, kundiger und reaktionsschneller Stimmenimitator sein, der auch die Telephonverbindung als *man in the middle* so angreifen kann, daß weder A noch B etwas merkt. Bei Videokonferenzen könnte man auch die Zahlen langsam über den Bildschirm des jeweils anderen laufen lassen. Die volle Sicherheit einer Schlüsselvereinbarung via RSA wird aber nicht erreicht, und da oft zumindest einer der Teilnehmer ein Unternehmen ist, das sich einen zertifizierten RSA-Schlüssel leisten kann, werden Schlüssel für symmetrische Kryptoverfahren in der Praxis sehr viel häufiger via RSA vereinbart als via DIFFIE-HELLMAN.

Im *electronic banking* wird die Idee eines zweiten Kommunikationskanals trotzdem häufig angewandt, hier im allgemeinen dadurch, daß ein Teil des Protokolls via SMS abläuft. Auch die können zwar selbstverständlich manipuliert werden, aber der Aufwand eines Angreifers steigt ganz beträchtlich, wenn er *gleichzeitig* zwei verschiedene Verbindungen manipulieren muß: Die meisten *phishing*-Attacken arbeiten schließlich mit gefälschten Webseiten im Ausland, und selbst im Inland ist es nicht so einfach, Mobilfunkverbindungen in einem größeren Gebiet zu überwachen und zu manipulieren.

§ 2: Verschlüsselung und elektronische Unterschriften

Zwischen RSA und den Verfahren mit diskreten Logarithmen gibt es einen ganz wesentlichen Unterschied: Wer die Faktorisierung des RSA-Moduls N kennt, kann die sonst schwer zugängliche Umkehrfunktion von $x \mapsto x^e \bmod N$ leicht berechnen, so daß Potenzieren mit e direkt als Verschlüsselung benutzt werden kann.

Bei der modularen „Exponentialfunktion“ $x \mapsto a^x \bmod p$ sind keine speziellen Wahlen von a und p bekannt, die vermöge einer geheimen Information zu einer einfachen Umkehrfunktion führen – diskrete Logarithmen sind für alle gleich schwer zu berechnen.

Die geheime Information bei einem asymmetrischen Verfahren auf der Basis diskreter Logarithmen kann daher nur in der Kenntnis *einzelner* diskreter Logarithmen bestehen: Wer für einen speziellen Wert x die Potenz $u = a^x \bmod p$ berechnet hat, weiß anschließend, daß x der diskrete Logarithmus von u modulo p zur Basis a ist.

Bei diesen sehr viel spezielleren „Geheimnissen“ ist klar, daß Kryptoverfahren auf der Basis von diskreten Logarithmen anders aussehen müssen als RSA.

a) Verschlüsselung nach Elgamal

Im Prinzip könnte man die Schlüsselvereinbarung nach DIFFIE und HELLMAN direkt zu einem Verschlüsselungsverfahren erweitern: Nachdem das gemeinsame Geheimnis $\gamma = a^{xy} \bmod p$ vereinbart ist, können Nachrichtenblöcke m_i mit $0 \le m_i < p-1$ in beide Richtungen verschlüsselt werden als $c_i = \gamma m_i \bmod p$. Da beide Partner den Wert von γ kennen, können sie leicht nach dem erweiterten EUKLIDischen Algorithmus ein δ berechnen, so daß $\gamma\delta \equiv 1 \bmod p$, und die verschlüsselte Information kann einfach entschlüsselt werden als $m_i = \delta c_i \bmod p$.

Solange nur ein einzelner Block m übertragen werden soll, ist dagegen nichts einzuwenden. Sobald aber mehrere Blöcke zu übertragen sind, wird dieses Verfahren verwundbar gegen Angriffe mit bekanntem Klartext: Falls ein Gegner für einen einzigen Chiffreblock c_i den Klartextblock m_i kennt (oder errät), kann er $\delta = m_i/c_i \bmod p$ berechnen und damit den gesamten Klartext entschlüsseln. Um das Verfahren

sicher zu machen, müßte man daher für jeden Block ein eigenes γ vereinbaren und dazu jedes Mal das gesamte DIFFIE-HELLMAN-Protokoll durchlaufen, was sehr aufwendig wäre.

Das Verfahren von ELGAMAL umgeht dieses Problem, indem es exakt dieselbe Mathematik mit einem leicht modifizierten Protokoll zu einem asymmetrischen Kryptoverfahren macht:

Die Parameter a und p sind entweder allgemein bekannte Systemparameter, oder jeder Teilnehmer A wählt sie selbst als Teil seines öffentlichen Schlüssels. Zusätzlich wählt er sich eine geheime Zufallszahl x und veröffentlicht $u = a^x \bmod p$.

Wer immer eine Nachricht $m_1, \ldots, m_r$ an A schicken möchte, erzeugt für jeden Block m_i eine Zufallszahl y_i berechnet daraus $v_i = a^{y_i} \bmod p$ und $c_i = u^{y_i} m_i$. Dann schickt er die Folge der Paare (v_i, c_i) an A. Der Chiffretext ist damit doppelt so lang wie der Klartext, was das Verfahren insbesondere für lange Texte nicht sonderlich attraktiv macht.

A muß zur Entschlüsselung den Multiplikator u^{y_i} kennen; dann kann er m_i als $c_i u^{-y_i}$ berechnen. Da $u^{y_i} \equiv a^{x y_i} \equiv \left(a^{y_i}\right)^x \equiv {v_i}^x \bmod p$ ist, hat er damit keine Probleme.



TAHER ELGAMAL wurde 1955 in Ägypten geboren. Er studierte zunächst Elektrotechnik an der Universität Kairo; nachdem er dort seinen BSc bekommen hatte, setzte er seine Studien fort an den Information Systems Laboratories der Stanford University. In seiner Masterarbeit ging es hauptsächlich um Systemtheorie, jedoch hörte er parallel auch freiwillig viele Mathematikvorlesungen und kam auf diesem Weg zur Kryptographie, die zum Thema seiner Doktorarbeit wurde. Nach dem Studium arbeitete er für eine ganze Reihe von Unternehmen, beispielsweise war er von 1995–1998 als Chefwissenschaftler von Netscape maßgeblich an der Entwicklung von SSL beteiligt. Zeitweise arbeitete er auch in selbst gegründeten Firmen. 2006 wurde er Chief Technology Officer der Tumbleweed Communications Corporation; seitdem diese 2008 von Axway übernommen wurde, ist er deren Chief Security Officer sowie Berater einer Reihe weiterer Unternehmen. Sein Name wird in der Literatur oft auch EL GAMAL oder ELGAMAL geschrieben; die obige Schreibweise ist die, die er selbst im Englischen benutzt.

Der offensichtliche Angriff eines Gegners besteht darin, aus u und a den diskreten Logarithmus x zu ermitteln, was nach derzeitigem Stand der Dinge schwierig erscheint. Ob andere Angriffe zum Erfolg führen könnten, ist (wie üblich) unbekannt – hoffentlich auch unseren Gegnern.

Genau wie bei RSA gibt es aber natürlich auch hier eine ganze Reihe von Möglichkeiten, das Verfahren durch schlechte Parameterwahl oder unsachgemäßen Gebrauch unsicher zu machen; einige davon sind in der am Ende von Kap. 4, §3*b)* zitierten Arbeit zu finden. Insbesondere muß auch hier die Nachricht mit Zufallsbits auf volle Blocklänge gebracht werden; ansonsten hat der Gegner Ansätze zur Entschlüsselung *ohne* Berechnung diskreter Logarithmen.

b) Das Verfahren von Massey-Omura

Bei diesem Verfahren geht es, wie bei der Schlüsselvereinbarung nach DIFFIE-HELLMAN, um Nachrichtenaustausch zwischen zwei Partnern A und B, die über keinerlei gemeinsame Schlüsselinformation verfügen; es gibt auch keine öffentlichen Schlüssel.

Das Verfahren läßt sich am einfachsten verstehen, wenn wir mit einem nichtmathematischen Analogon beginnen: Angenommen, A möchte einen Container mit wichtigen Unterlagen an B schicken, traut aber dem Transporteur nicht. Wenn er B vorher treffen kann, kauft er einfach ein gutes Vorhängeschloß und gibt B einen der beiden Schlüssel. Später kann er dann den Container mit dem Schloß und seinem Schlüssel verschließen, und B kann mit seinem Schlüssel das Schloß wieder entfernen, um den Container zu öffnen.

Wenn A und B keine Möglichkeit zu einem vorherigen Treffen haben, müssen sie umständlicher vorgehen: Jetzt kauft sich jeder der beiden ein Schloß, dessen Schlüssel dann nur er hat. A verschließt den Container mit seinem Schloß und schickt ihn an B. Der kann ihn natürlich nicht öffnen und schickt ihn deshalb ungeöffnet wieder zurück, verschließt ihn aber vorher noch zusätzlich mit *seinem* Schloß. A kann nun *sein* Schloß entfernen und schickt ihn, nun nur noch mit Bs Schloß gesichert, an B. Dieser kann *sein* Schloß entfernen und dann den Container öffnen.

In der digitalen Welten sieht das ganze so aus:

A und B einigen sich auf eine Primzahl p (die auch Konstante eines ganzen Netzwerks sein kann), und jeder erzeugt sich einen (geheimzuhaltenden) Exponenten e_A *bzw.* e_B, der prim ist zu $p-1$. Dazu berechnet er nach dem erweiterten EUKLIDischen Algorithmus ein (ebenfalls geheimzuhaltendes) Inverses modulo $p-1$; diese Inversen seien d_A und d_B. Nach dem kleinen Satz von FERMAT ist somit für jedes $m \in \mathbb{Z}$

$$m^{e_A d_A} \equiv m^{e_B d_B} \equiv m \bmod p\,.$$

Will nun B eine Nachricht m verschlüsselt an A schicken, so schickt er $c_1 = m^{e_B} \bmod p$. Damit kann natürlich weder A noch ein etwaiger Lauscher etwas anfangen: Da niemand außer B die beiden Exponenten e_B und d_B kennt, ist das einfach *irgendeine* Potenz zu *irgendeiner* Basis. Selbst ein BAYESscher Gegner, der alle Kombinationen (M, e) mit $M^e \equiv m^{e_B} \bmod p$ durchprobieren kann, wird dort für große p eine Fülle von potentiellen Klartexten finden, die alle ungefähr gleich wahrscheinlich sind.

A schickt die Nachricht daher gleich wieder zurück, potenziert sie aber vorher mit seinem Exponenten e_A. Was B erhält, ist also $c_2 = m^{e_B e_A}$, eine Nachricht die niemand entschlüsseln kann.

B potenziert diese Nachricht mit seinem Exponenten d_B; dies liefert

$$\left(m^{e_B e_A}\right)^{d_B} = m^{e_B e_B d_B} = m^{e_B d_B e_A} = \left(m^{e_B d_B}\right)^{e_A} \equiv m^{e_A} \bmod p\,.$$

Diese Nachricht schickt er an A, der nun mit seinem Exponenten d_A leicht den Klartext ermitteln kann.

Auch die Sicherheit dieses Verfahrens hängt an diskreten Logarithmen: Ein etwaiger Lauscher kennt die Zahlen

$$m^{e_B} \bmod p, \quad m^{e_B e_A} = \left(m^{e_B}\right)^{e_A} \quad \text{und} \quad m^{e_A} = \left(m^{e_B e_A}\right)^{d_B};$$

falls er in der Lage ist, diskrete Logarithmen modulo p zu berechnen, kann er e_A bestimmen als den diskreten Logarithmus von $m^{e_B e_A}$ zur Basis m^{e_B} und d_B als diskreten Logarithmus von $\left(m^{e_B e_A}\right)^{d_B}$ zur Basis $m^{e_B e_A}$. Damit kann auch er m berechnen, indem er beispielsweise m^{e_B} modulo p mit d_B potenziert. Die Primzahl p muß also auch bei diesem Verfahren so groß sein, daß die Berechnung diskreter Logarithmen modulo p zumindest praktisch undurchführbar ist. Ein *man in the*

middle-Angriff ist hier, im Gegensatz zu ELGAMAL, natürlich in genau der gleichen Weise wie beim Verfahren von DIFFIE-HELLMAN möglich.

JAMES L. MASSEY wurde 1934 in Wauseon, Ohio geboren. Er studierte Elektrotechnik an der University of Notre Dame und am MIT, wo er sich vor allem auf Informations- und Kodierungstheorie konzentrierte. Nach dem Studium war er 14 Jahre lang Professor in Notre Dame, dann kurz am MIT und an der University of California, Los Angelos (UCLA), bis er 180 einem Ruf an die ETH Zürich folgte, wo er bis zu seiner Emeritierung zum 1. April 1999 einen Lehrstuhl für Signal- und Informationsverarbeitung hatte.

JIM K. OMURA studierte in Stanford Elektrotechnik und war dann 15 Jahre lang Professor an der UCLA. Danach gründete er eine eigene Firma namens Cylink (inzwischen von Safenet übernommen) und arbeitete als Berater für verschiedene Firmen und Stiftungen.

c) DSA

Der Zusatzaufwand gegenüber RSA macht sowohl ELGAMAL aus auch MASSEY-OMURA für praktische Anwendungen eher uninteressant; hinzu kommt, daß zumindest bei MASSEY-OMURA ohne einen zusätzlichen Kanal keiner der beiden Partner sicher sein kann, daß er wirklich mit dem anderen kommuniziert. Dasselbe gilt natürlich zumindest für den Empfänger auch bei RSA; dort allerdings liefert das Verfahren selbst eine Möglichkeit für elektronische Unterschriften, die dieses Problem löst. Auch das Verfahren von ELGAMAL kann so modifiziert werden, daß es elektronische Unterschriften realisiert, mit diskreten Logarithmen kann man aber auch noch weiter gehen und sogar relativ kurze, aber trotzdem sichere Unterschriften produzieren.

Der Rechenaufwand pro Byte ist bei asymmetrischer Kryptoverfahren deutlich höher als bei den gängigen symmetrischen Verfahren; andererseits sind zu unterzeichnende Texte oft sehr lang, weil sie beispielsweise von Juristen unter Berücksichtigung aller Eventualitäten abgefaßt wurden. Deshalb wird meist nicht die gesamte Nachricht unterzeichnet, sondern nur ein sogenannten Hashwert. Dabei handelt es sich um eine kurze Bitfolge, die nach Art einer Prüfziffer von der ganzen Nachricht abhängt und diese auch charakterisiert.

Wir werden uns im übernächsten Kapitel genauer mit solchen Hashverfahren beschäftigen; dabei werden wir auch sehen, daß Hashwerte bislang meist 160 Bit hatten und daß man gerade dabei ist, diese zunehmend problematische Länge auf 224 oder besser noch 256 Bit zu erhöhen.

Wenn wir so einen Hashwert mit RSA unterzeichnen, hat die Unterschrift nach derzeitigem Sicherheitsstandard eine Länge von 2048 Bit. Verglichen mit der Länge des zu unterzeichnenden Hashwerts ist das offensichtlich weit übertrieben. Andererseits wäre eine Unterschrift, die auf diskreten Logarithmen in einem Körper mir nur etwa 2^{256} Elementen beruht, ohne großen Aufwand fälschbar.

Der *Digital Signature Algorithm* DSA bietet einen Ausweg aus diesem Dilemma, indem er zwar in einer großen Gruppe rechnet, dabei aber kurze Unterschriften aus einer deutlich kleineren Untergruppe liefert. Dieser Algorithmus wurde im *Digital Signature Standard* DSS der USA spezifiziert und zählt neben RSA auch zu den von der Bundesnetzagentur festgelegten „Geeigneten Algorithmen“.

Als Ordnung der Untergruppe wählt man eine Primzahl q, für die nach den derzeitigen Empfehlungen der Bundesnetzagentur seit Anfang 2010 eine Länge von mindestens 224 Bit notwendig ist; ab Anfang 2016 erhöht sich die Länge auf mindestens 256 Bit. Diese Längen hängen in erster Linie ab von den verwendeten (und zulässigen) Hashverfahren, nicht so sehr von Sicherheitsanforderungen.

Die Sicherheit wird gewährleistet (soweit dies möglich ist) durch eine zweite Primzahl p, die so gewählt wird, daß $p \equiv 1 \bmod q$ ist; für ihre Größe sind mindestens 2048 Bit vorgeschrieben.

Primzahlen $p \equiv 1 \bmod q$ sind nicht schwerer zu finden als beliebige Primzahlen: Falls man bei der Primzahlsuche wirklich auf Nummer sicher geht und Zufallszahlen auf Primalität testet, nimmt man hier einfach Zufallszahlen k und testet $kq + 1$ auf Primalität. Falls man mit ERATOSTHENES arbeitet, kann man das Sieben leicht so modifizieren, daß nur Zahlen der Form $kq + 1$ gesiebt werden. An den Erfolgschancen ändert dies in beiden Fällen nichts: Nach einem Satz von DIRICHLET über Primzahlen in arithmetischen Folgen ist die Dichte der Primzahlen der Form $kq + i$ für jedes i mit $0 < i < q$ dieselbe; in der Größenordnung n ist also weiterhin im Mittel jede $\ln n$-te solche Zahl eine Primzahl. (Tatsächlich sind es sogar geringfügig mehr, denn außer q selbst gibt es

natürlich keine Primzahl der Form $p = kq$. Bei den Größenordnungen von q mit denen wir arbeiten, geht aber der Unterschied zwischen q und $q-1$ definitiv im „Rauschen“ der im Kleinen sehr unregelmäßigen Primzahlverteilung unter.)

Als nächstes muß ein Element g gefunden werden, dessen Potenzen im Körper $\mathbb{F}_p$ eine Gruppe der Ordnung q bilden. Auch das ist einfach: Man starte mit irgendeinem Element $g_0 \in \mathbb{F}_p \setminus \{0\}$ und berechne seine $(p-1)/q$-te Potenz. Falls diese ungleich eins ist, muß sie wegen $g_0^{p-1} = 1$ die Ordnung q haben; andernfalls muß ein neues g_0 betrachtet werden.

Die so bestimmten Zahlen q, p und g werden veröffentlicht und können auch in einem ganzen Netzwerk global eingesetzt werden. Geheimer Schlüssel jedes Teilnehmers ist eine Zahl x zwischen eins und $q-1$; der zugehörige öffentliche Schlüssel ist $u = g^x \bmod p$.

Unterschreiben lassen sich mit diesem Verfahren Nachrichtenblöcke m mit $0 \le m < q$; im allgemeinen wird es sich dabei um Hashwerte der eigentlich zu unterschreibenden Nachricht handeln. Dazu wählt man für jede Nachricht eine Zufallszahl k mit $0 < k < q$ und berechnet

$$r = (g^k \bmod p) \bmod q\,.$$

Da q eine Primzahl ist, hat k ein multiplikatives Inverses modulo q; man kann also modulo q durch k dividieren und erhält eine Zahl s, für die

$$sk \equiv m + xr \bmod q$$

ist; die Unterschrift unter die Nachricht m besteht dann aus den beiden Zahlen r und s zwischen 0 und $q-1$. Sie kann nur erzeugt werden von jemanden, der den geheimen Schlüssel x kennt.

Überprüfen kann die Unterschrift allerdings jeder: Ist t das multiplikative Inverse zu s modulo q, so ist $k \equiv tsk \equiv tm + xtr \bmod q$, also, da g die Ordnung q hat, $g^k \bmod p = g^{tm} g^{xtr} \bmod p = g^{tm} u^{tr} \bmod p$. Modulo q ist die linke Seite gleich r, und auf der rechten Seite können sowohl g^{tm} als auch u^{tr} aus öffentlicher Information und der Unterschrift berechnet werden. Modulo q kann diese Gleichung somit überprüft werden; die Unterschrift wird anerkannt, wenn

$$r \equiv (g^{tm} u^{tr} \bmod p) \bmod q$$

ist. (Die beiden Potenzen und ihr Produkt müssen natürlich zunächst modulo p berechnet werden: Zwei modulo p kongruente Zahlen sind praktisch nie auch kongruent modulo q.)

Ein Angreifer müßte sich nach allem was wir wissen x aus u verschaffen, müßte also ein diskretes Logarithmenproblem modulo der großen Primzahl p lösen, so daß der Sicherheitsstandard dem des diskreten Logarithmenproblems modulo p entsprechen sollte, obwohl die Unterschriften deutlich kürzer sind.

§3: Strategien zur Berechnung diskreter Logarithmen

Genau wie es zahlreiche Ansätze gibt, ganze Zahlen auf mehr oder weniger effiziente Weise zu faktorisieren, gibt es auch die verschiedensten Methoden, diskrete Logarithmen zu berechnen. Obwohl es keinen klaren theoretischen Zusammenhang zwischen den beiden Problemen gibt, zeigt die Erfahrung der letzten Jahren eine erstaunliche Parallelität zwischen den entsprechenden Algorithmen. Da das Interesse an Faktorisierungsalgorithmen zumindest bislang deutlich größer ist, kamen neue Entwicklungen in der letzten Zeit immer von dort; erstaunlicherweise stellte sich aber immer ziemlich schnell heraus, daß ein ähnliches Verfahren mit praktisch demselben Aufwand auch diskrete Logarithmen berechnen kann. Daher benötigen wir, um vergleichbare Sicherheit zu erreichen, bei der Kryptographie mit diskreten Logarithmen Moduln zumindest derzeit dieselben Längen wie bei RSA.

a) Probieren

Am einfachsten und langwierigsten ist das Probieren: Um den diskreten Logarithmus modulo p von a zur Basis g zu bestimmen, berechnet man (analog zur Faktorisierung durch Abdividieren) systematisch alle Potenzen von g, bis man a erhält. Dies erfordert im Mittel $p/2$ Versuche.

b) Gruppentheoretische Formulierung des Problems

Für bessere Verfahren müssen wir das Problem zunächst mathematisch aufbereiten. Dazu dient die Gruppentheorie. Zur Erinnerung seien die wesentlichen Definitionen noch einmal wiederholt:

Definition: Eine Gruppe ist eine Menge G zusammen mit einer Verknüpfung $\star\colon G \times G \to G$, so daß gilt:

- Für alle $g, h, k \in G$ ist $(g \star h) \star k = g \star (h \star k)$ *(Assoziativgesetz)*
- Es gibt ein Element $e \in G$, genannt *Neutralelement*, so daß für alle $g \in G$ gilt: $g \star e = e \star g = g$.
- Zu jedem $g \in G$ gibt es ein *inverses Element* $h \in G$, für das gilt: $g \star h = h \star g = e$.
- Die Gruppe heißt *abelsch*, wenn zusätzlich gilt: $g \star h = h \star g$ für alle $g, h \in G$. *(Kommutativgesetz)*.
- Die Gruppe heißt *endlich*, wenn die Menge G nur endlich viele Elemente hat.
- Eine endliche Gruppe heißt *zyklisch*, wenn es ein Element $g \in G$ gibt, so daß sich jedes $h \in G$ schreiben läßt als $h = \underbrace{g \star \cdots \star g}_{n \text{ mal}} \stackrel{\text{def}}{=} g^n$ mit einer natürlichen Zahl $n \in \mathbb{N}$.

Das inverse Element h zu g schreiben wir kurz als g^{-1}, und für $n \in \mathbb{N}$ soll $g^{-n} = \left(g^{-1}\right)^n$ sein. g^0 bezeichne für jedes $g \in G$ das Neutralelement.

Als erstes müssen wir klären, welche Werte die Potenzen $g^x \bmod p$ überhaupt annehmen können. Klar ist

Lemma: Für jede natürliche Zahl N ist die Menge aller $g^x \bmod N$ mit $x \in \mathbb{N}$ eine zyklische Gruppe bezüglich der Multiplikation modulo N.

Beweis: Das Assoziativgesetz folgt sofort aus dem für die Addition natürlicher Zahlen. Da es nur endlich viele Restklassen modulo N gibt, muß es außerdem zwei Zahlen $r > s$ geben mit $g^r \equiv g^s \bmod N$; mit $m = r - s$ ist daher $g^m \equiv 1 \bmod N$ als Potenz von g darstellbar, und das ist das neutrale Element. Da $g^{x+m} \equiv g^x \bmod N$ für alle x, läßt sich jede Restklasse in der Form $g^x \bmod N$ mit $1 \leq x \leq m$ darstellen; das Inverse dazu ist dann $g^{m-x} \bmod N$. Damit sind alle Gruppenaxiome nachgewiesen, und zyklisch ist die Gruppe nach Konstruktion. ∎

Definition: Die kleinste natürliche Zahl m, für die $g^m \equiv 1 \bmod N$ ist, heißt *Ordnung von g modulo N*.

Das Knacken eines Kryptosystems auf der Basis diskreter Logarithmen ist umso einfacher, je kleiner die Ordnung der Basis g ist. Daher müssen

wir Elemente möglichst großer Ordnung finden. Dazu betrachten wir das Problem gruppentheoretisch:

Definition: Die Ordnung eines Elements a einer (multiplikativ geschriebenen) Gruppe G ist die kleinste natürliche Zahl r, für die a^r gleich dem Neutralelement ist. Falls es keine solche Zahl r gibt, sagen wir, a habe unendliche Ordnung. Die Ordnung einer endlichen Gruppe ist deren Elementanzahl.

Lemma (LAGRANGE): In einer endlichen Gruppe teilt die Ordnung r eines jeden Elements g die Gruppenordnung.

Beweis: Wir führen auf der Gruppe G eine Äquivalenzrelation ein durch die Vorschrift $u \sim v$, falls es ein $s \in \mathbb{N}$ gibt mit $u = vg^s$. Offensichtlich besteht die Äquivalenzklasse eines jeden Elements $u \in G$ aus genau r Elementen, nämlich $u, ug, \ldots, ug^{r-1}$. Da G die Vereinigung aller Äquivalenzklassen ist, muß die Gruppenordnung somit ein Vielfaches von r sein. ∎



JOSEPH-LOUIS LAGRANGE (1736–1813) wurde als GIUSEPPE LODOVICO LAGRANGIA in Turin geboren und studierte dort zunächst Latein. Erst eine alte Arbeit von HALLEY über algebraische Methoden in der Optik weckte sein Interesse an der Mathematik, woraus ein ausgedehnter Briefwechsel mit EULER entstand. In einem Brief vom 12. August 1755 berichtete er diesem unter anderem über seine Methode zur Berechnung von Maxima und Minima; 1756 wurde er, auf EULERs Vorschlag, Mitglied der Berliner Akademie; zehn Jahre später zog er nach Berlin und wurde dort EULERs Nachfolger als mathematischer Direktor der Akademie. 1787 wechselte er an die Pariser Académie des Sciences, wo er bis zu seinem Tod blieb und unter anderem an der Einführung des metrischen Systems beteiligt war. Seine Arbeiten umspannen weite Teile der Analysis, Algebra und Geometrie.

Für eine Primzahl p bilden die Zahlen modulo p bekanntlich einen Körper; wie uns das folgende Lemma zeigt, können wir dann Elemente der größtmöglichen Ordnung $p - 1$ finden:

Lemma: Ist k ein endlicher Körper, so bilden die Elemente von $k \setminus \{0\}$ bezüglich der Multiplikation eine zyklische Gruppe.

Beweis: Da die multiplikative Gruppe eines Körpers mit q Elementen aus allen Körperelementen außer der Null besteht, hat sie die Ordnung $q-1$. Nach LAGRANGE ist daher die Ordnung eines jeden Elements ein Teiler von $q-1$. Wir müssen zeigen, daß es mindestens ein Element gibt, dessen Ordnung *genau* $q-1$ ist.

Für jeden Primteiler p_i von $q-1$ hat die Polynomgleichung

$$x^{(q-1)/p_i} = 1$$

höchstens $(q-1)/p_i$ Lösungen im Körper k; es gibt also zu jedem p_i ein Körperelement a_i mit $a_i^{(q-1)/p_i} \neq 1$.

q_i sei die größte Potenz von p_i, die $q-1$ teilt, und $g_i = a_i^{(q-1)/q_i}$ die $(q-1)/q_i$-te Potenz von a_i. Dann ist

$$g_i^{q_i} = a_i^{q-1} = 1 \quad \text{und} \quad g_i^{\frac{q_i}{p_i}} = a_i^{\frac{q-1}{p_i}} \neq 1\,;$$

g_i hat also die Ordnung q_i. Da die verschiedenen q_i Potenzen verschiedener Primzahlen p_i sind, hat daher das Produkt g aller g_i das Produkt aller q_i als Ordnung, also $q-1$. Damit ist die multiplikative Gruppe des Körpers zyklisch. ∎

In unserer Situation bedeutet dies, daß es mindestens eine Zahl g gibt, so daß die Abbildung

$$\varphi\colon \begin{cases} \mathbb{Z}/(p-1) \to \mathbb{Z}/p \setminus \{0\} \\ \qquad x \mapsto g^x \end{cases}$$

bijektiv ist. Solche Zahlen g bezeichnet man als *primitive Wurzeln* modulo p. Kryptoverfahren auf der Basis diskreter Logarithmen sind daher dann am sichersten, wenn die Basis g eine primitive Wurzel modulo p ist; in diesem Fall gibt es die meisten verschiedenen Potenzen $g^x \bmod p$, nämlich $p-1$ Stück.

Nach LAGRANGE ist die Ordnung m eines jeden Elements g ein Teiler m von $p-1$. Ist m ein echter Teiler, so gibt es mindestens einen Primteiler q von $p-1$, so daß m sogar ein Teiler von $(p-1)/q$ ist; wenn wir entscheiden wollen, ob eine gegebene Basis g eine primitive Wurzel ist, genügt es also, für alle Primteiler q von $p-1$ die Potenzen $g^{(p-1)/q}$ zu

berechnen; falls keine davon gleich eins modulo p ist, haben wir eine primitive Wurzel gefunden.

In der Praxis wird dies freilich meist daran scheitern, daß wir $p-1$ nicht faktorisieren können – es sei denn, wir haben p geeignet gewählt. Falls wir beispielsweise eine Primzahl p der Form $p = 2p' + 1$ mit primem p' wählen, ist $p-1 = 2p'$, und g ist genau dann primitive Wurzel modulo p, wenn weder g^2 noch $g^{p'}$ kongruent eins modulo p ist. Da g^2 nur für $g \equiv \pm 1 \bmod p$ kongruent eins ist, ist somit ein g mit $1 < g < p-1$ genau dann primitive Wurzel, wenn $g^{p'} \not\equiv 1 \bmod p$ ist; andernfalls hat es die (kryptographisch fast genauso sichere) Ordnung p'.

c) Anwendung des chinesischen Restesatzes

Die Basis g habe die Ordnung m modulo N und ihre Primfaktorzerlegung sei $m = \prod_{i=1}^{r} q_i^{e_i}$. Wir wollen das diskrete Logarithmenproblem $g^x \equiv a \bmod N$ lösen. Um es auf diskrete Logarithmenprobleme in Gruppen der Ordnungen $q_i^{e_i}$ zurückzuführen, setzen wir $n_i = m/q_i^{e_i}$; für $g^x \equiv a \bmod N$ ist dann auch $g^{n_i x} \equiv a^{n_i} \bmod N$, und $g_i = g^{n_i}$ hat modulo N nur die Ordnung $q_i^{e_i}$.

Falls wir die r diskrete Logarithmenprobleme $g_i^{x_i} \equiv a^{n_i} \bmod N$ lösen können, lassen sich die Lösungen x_i leicht zu einer Lösung des ursprünglichen Problems zusammensetzen: Da die n_i keinen gemeinsamen Primteiler haben, ist ihr ggT gleich eins; es gibt also ganze Zahlen α_i mit $\sum_{i=1}^{r} \alpha_i n_i = 1$, die wir uns mit dem erweiterten EUKLIDischen Algorithmus leicht verschaffen können. Mit $x = \sum_{i=1}^{r} \alpha_i n_i x_i$ ist dann

$$g^x = \prod_{i=1}^{r} g^{n_i x_i \alpha_i} \equiv \prod_{i=1}^{r} a^{n_i \alpha_i} = a^{\sum_{i=1}^{r} \alpha_i n_i} = a \bmod N\,.$$

d) Das Verfahren von Pohlig und Hellman

Tatsächlich reicht es sogar, wenn wir das diskrete Logarithmenproblem statt in Gruppen der Ordnung $q_i^{e_i}$ in Gruppen der Ordnung q_i lösen können. Dazu gehen wir folgendermaßen vor:

Der Einfachheit halber beschränken wir uns auf eine zyklische Gruppe G von Primzahlpotenzordnung q^e und wählen dort ein erzeugendes Ele-

ment g. Gesucht ist der diskrete Logarithmus eines weiteren Elements $a \in G$ zur Basis g.

Diese gesuchte Zahl x liegt zwischen null und $q^e - 1$ (tatsächlich ist sie sogar höchstens $q^e - q^{e-1}$); wenn wir sie in Ziffern zur Basis p schreiben, ist also

$$x = x_0 + x_1 q + x_2 q^2 + \cdots + x_{e-1} q^{e-1} \quad \text{mit} \quad 0 \le x_i \le q\,.$$

Wir wollen uns überlegen, daß wir die „Ziffern“ x_i als diskrete Logarithmen in einer Untergruppe der Ordnung q berechnen können.

Aus $a = g^x$ folgt die Beziehung

$$a^{q^j} = g^{x p^j} = g^{x_0 q^j} \cdot g^{x_1 q^{j+1}} \cdot g^{x_2 q^{j+2}} \cdots g^{x_{e-1} q^{j+e-1}}\,.$$

Da aber $g^{q^e} \equiv 1 \bmod N$ ist, sind modulo N alle Terme, in denen q einen größeren Exponenten als e hat, gleich eins; tatsächlich ist also

$$a^{q^j} \equiv q^{p^j x} \equiv g^{x_0 q^j} \cdot g^{x_1 q^{j+1}} \cdot g^{x_2 q^{j+2}} \cdots g^{x_{e-j-1} q^{e-1}} \bmod N\,.$$

Insbesondere folgt für $j = e - 1$, daß $a^{q^{e-1}} \equiv g^{x_0 q^{e-1}} \bmod N$ ist, x_0 ist also die Lösung eines diskreten Logarithmenproblems in der von $g^{q^{e-1}}$ erzeugten Untergruppe der Ordnung q.

Angenommen, wir haben die „Ziffern“ von x_0 bis x_r bereits bestimmt. Dann schreiben wir

$$a \cdot g^{-x_0 - x_1 q - \cdots - x_r q^r} = g^{x_{r+1} q^{r+1}} \cdots g^{x_{e-1} q^{e-1}}$$

und potenzieren diese Gleichung mit q^{e-2-r}. Modulo N können wir rechts alle Terme außer dem ersten streichen; wir erhalten also die Gleichung

$$a^{q^{e-2-r}} \cdot g^{-q^{e-2-r}(x_0 + x_1 q + \cdots + x_r q^r)} \equiv q^{x_{r+1} q^{e-1}} \bmod N\,,$$

die uns x_{r+1} als diskreten Logarithmus der (bekannten) linken Seite liefert, und zwar wieder in der von $g^{q^{e-1}}$ erzeugten Untergruppe der Ordnung q.

Auf diese Weise können wir nacheinander die sämtlichen x_i berechnen und damit auch x.

Zusammen mit dem oben diskutierten Ansatz über den chinesischen Restesatz ist dies das Verfahren von POHLIG und HELLMAN zur Berechnung diskreter Logarithmen in einer zyklischen Gruppe G der Ordnung n: Sie kann zurückgeführt werden auf die Berechnung diskreter Logarithmen in Untergruppen, deren Ordnungen die Primteiler von n sind.

e) Folgerung für die Sicherheit von Kryptosystemen

Die Diskussion in den beiden vorigen Abschnitten zeigt, daß die Schwierigkeit der Berechnung einen diskreten Logarithmus in einer zyklischen Gruppe der Ordnung m relativ einfach zurückgeführt werden kann auf die Berechnung diskreter Logarithmen in Untergruppen, deren Ordnungen die Primteiler von m sind. Als Faustregel können wir daher festhalten, daß die Sicherheit diskreter Logarithmen in einer zyklischen Gruppe der Ordnung m im wesentlichen nur gleich der Sicherheit in einer Untergruppe der Ordnung q ist, wobei q der größten Primteiler von n ist.

Idealerweise sollte daher die Gruppenordnung m selbst eine Primzahl sein, was allerdings zumindest für die multiplikative Gruppe modulo p nur im kryptographisch völlig uninteressanten Fall $p = 3$ der Fall ist.

Hier empfiehlt sich, eine Primzahl p der Form $2p' + 1$ zu wählen, wobei auch p' eine Primzahl ist. Die multiplikative Gruppe modulo p hat dann die Ordnung $2p'$, und die Sicherheit entspricht der in einer Gruppe der Ordnung p'.

f) Baby step und giant step

Bei der Faktorisierung ganzer Zahlen konnten wir den Aufwand gegenüber dem naiven Abdividieren durch die Monte-Carlo-Methode auf ungefähr die Quadratwurzel der zu faktorisierenden Zahl reduzieren. Auch für die Bestimmung diskreter Logarithmen gibt es entsprechende Verfahren, zum Beispiel den *baby step – giant step* Algorithmus von SHANKS, das auf Kosten von mehr Speicherplatz die Rechenzeit gegenüber reinem Probieren deutlich reduziert: Ist n die Ordnung von g, so ist der Aufwand nicht mehr proportional zu n, sondern nur noch zu $\sqrt{n}$.

DANIEL SHANKS (1917–1996) wurde in Chicago geboren, wo er zur Schule ging und 1937 einen Bachelorgrad in Physik der University of Chicago erwarb. Er arbeitete bis 1950 in verschiedenen Positionen als Physiker, danach als Mathematiker. 1949 begann er ein *graduate* Studium der Mathematik an der University of Maryland, zu dessen Beginn er der erstaunten Fakultät als erstes eine fertige Doktorarbeit vorlegte. Da zu einem *graduate* Studium auch Vorlesungen und Prüfungen gehören, wurde diese noch nicht angenommen, und da er während seines Studiums Vollzeit arbeitete, dauerte es noch bis 1954, bevor er alle Voraussetzungen erfüllte; dann wurde die Arbeit in praktisch unveränderter Form akzeptiert. Erst 1977 entschloß er sich, eine Stelle an einer Universität anzunehmen und war dann bis zu seinem Tod Professor an der University of Maryland.

SHANKS wählt eine natürliche Zahl m die ungefähr gleich $\sqrt{n}\cdot\log_2 n$ ist; falls man n nicht so genau kennt, kann zwar die Effizienz des Verfahrens unter einer schlechten Wahl von m geringfügig leiden, aber solange die Größenordnungen einigermaßen stimmen, ist das nicht so dramatisch. Wichtig ist nur, daß $m > \sqrt{n}$ ist, aber nicht dramatisch größer.

Danach berechnet er die sämtlichen Potenzen g^i von g mit Exponenten $i \le m$; das sind m sogenannte *baby steps*.

Bei den dann folgenden *giant steps* berechnet er, um den diskreten Logarithmus von a zu erhalten, die Elemente $a \cdot g^{-mj}$ für $j = 1, 2, \ldots$ und vergleicht sie mit den Vielfachen aus dem ersten Teil. Ein solcher Vergleich kann etwa über eine binäre Suche oder eine *hash*-Tabelle implementiert werden und hat einen Aufwand proportional $\log_2 n$.

Sobald ein Wert $a \cdot g^{-mj}$ gefunden ist, der mit einer der in den *baby steps* berechneten Potenzen g^i übereinstimmt, gilt $a \cdot g^{-mj} = g^i$ oder $a = g^{mj+i}$; der diskrete Logarithmus von a zur Basis g ist also $mj + i$. Die notwendige Anzahl von *giant steps* liegt im schlimmsten Fall bei $n/m \approx \sqrt{n}$; im Durchschnitt ist sie halb so groß.

g) Zahme und wilde Kängurus

In den Jahren um 1975 entwickelte der britische Mathematiker JOHN M. POLLARD mehrere recht einfache Algorithmen zur Faktorisierung ganzer Zahlen sowie zur Berechnung diskreter Logarithmen, die auch heute noch (teils in verbesserter Form) zu den Standardwerkzeugen der algorithmischen Zahlentheorie gehören. Eine seiner Methoden verwendet eine Strategie zur Jagd auf Kängurus.

JOHN M. POLLARD ist ein britischer Mathematiker, der hauptsächlich bei British Telecom arbeitete. Er veröffentlichte zwischen 1971 und 2000 rund zwanzig mathematische Arbeiten, größtenteils auf dem Gebiet der algorithmischen Zahlentheorie. Bekannt ist er auch für seine Beiträge zur Kryptographie, für die er 1999 den RSA Award erhielt. Außer dem hier vorgestellten Algorithmus entwickelte er unter anderem auch das im letzten Kapitel erwähnte Zahlkörpersieb, dessen Weiterentwicklungen die derzeit schnellsten Faktorisierungsalgorithmen für große Zahlen sind. Seine home page, um die er sich auch jetzt im Ruhestand noch kümmert, ist `jmptidcott.googlepages.com/index.html`.

Da POLLARD kein Australier ist, sondern Brite, sind natürlich auch seine Kängurus britisch. Das geht zwar nicht so weit, daß diese Schlangen an Bushaltestellen bilden, sie springen aber im Gegensatz zu ihren australischen Artgenossen auch nicht völlig ungeordnet durch die Gegend: Sie springen immer geradeaus, und die Sprunglängen sind natürliche Zahlen aus einer endlichen Teilmenge $S \subset \mathbb{N}$, die durch den Startpunkt des Sprungs eindeutig festgelegt sind. Die Position eines Kängurus kann daher durch eine natürliche Zahl $u \in \mathbb{N}$ beschrieben werden, und wenn es von dort aus abspringt, springt es zum Punkt $u + f(u)$, wobei $f\colon \mathbb{N} \to S$ eine bekannte Funktion ist, die wohl in erster Linie von der Bodenbeschaffenheit in den einzelnen Punkten $u \in \mathbb{N}$ abhängen dürfte. Da die Landschaft in Großbritannien sehr variabel ist, können wir in erster Näherung annehmen, daß sich f wie eine Zufallsfunktion verhält; ihr Erwartungswert sei m, das arithmetische Mittel der Elemente von S.

Um ein wildes Känguru zu fangen, überredet POLLARD ein zahmes Känguru, von einem Startpunkt u_0 aus loszuspringen und n Sprünge zu machen. Nach jedem Sprung soll es auf seiner jeweiligen Position ein Loch graben und dieses gut mit Zweigen oder ähnlichem kaschieren. Die Positionen u_i, bei denen es Löcher gräbt, sind rekursiv berechenbar durch die Vorschrift $u_i = u_{i-1} + f(u_{i-1})$.

Falls nun ein wildes Känguru auf derselben Strecke unterwegs ist, kennen wir dessen Startpunkts v_0 natürlich nicht; da es britisch ist, wissen wir aber, wie es springt: Nach i Sprüngen ist es auf einer Position v_i, die über die Rekursion $v_i = v_{i-1} + f(v_{i-1})$ gegeben ist.

Da sich f gemäß unserer Annahme wie eine Zufallsfunktion verhält, fällt das wilde Känguru bei jedem Schritt mit einer Wahrscheinlichkeit von ungefähr $1/m$ in ein Loch und endet dann als Kängurubraten; seine

Chance, bei $r < n$ Sprüngen alle Löcher zu vermeiden, liegt also etwa bei $(1 - 1/m)^r$. Setzen wir $r = am$, so können wir dies auch schreiben als

$$\left(1 - \frac{1}{m}\right)^r = \left(1 - \frac{1}{m}\right)^{ma} \approx e^{-a}$$

für hinreichend große m. Schon für $a = 3$ beträgt diese Chance nur noch knapp 5%, für $a = 6$ ist sie ungefähr $1 : 400$, für $a = 7$ weniger als $1 : 1000$. Das Känguru hat also kaum eine Chance, dem Kochtopf zu entgehen.

Nun gibt es zwar sicherlich sowohl Zahlentheoretiker als auch Kryptanalytiker, die gerne Kängurubraten essen; während der Arbeitszeit interessieren sie sich aber mehr für Fragen wie die Faktorisierung ganzer Zahlen oder die Berechnung diskreter Logarithmen.

Zur Berechnung des diskreten Logarithmus einer Zahl u zur Basis g modulo p können Kängurujäger wie folgt vorgehen: Sie wählen zunächst ein Intervall (A, B), in dem der diskrete Logarithmus liegt. Wenn sie keinerlei spezielle Informationen haben, wird dies zwangsläufig das Intervall $(1, N)$ sein müssen; in Situationen wie beim DSA mit großer Primzahl p und kleiner Primzahl q weiß man aber, daß es bereits eine Lösung im viel kleineren Intervall $(1, q)$ gibt.

Das zahme Känguru startet mit $v_0 = g^A \bmod p$ und springt dann nacheinander die Positionen $v_i = v_{i-1} g^{f(v_{i-1})}$ an, bis es das Suchintervall (A, B) verlassen hat.

Das wilde Känguru startet an der Position $u_0 = u = g^x \bmod p$, wobei x den zu berechnenden diskreten Logarithmus bezeichnet; seine Landepositionen sind die Punkte u_i mit $u_i = u_{i-1} g^{f(u_{i-1})}$, bis es eventuell in ein vom zahmen Känguru gegrabenes Loch fällt. Wenn es alle Fallen vermeidet, war der Ansatz erfolglos; andernfalls haben wir haben wir zwei Indizes i, j mit $u_i = v_j$.

Die Rekursionen für u_i und v_j lassen sich leicht auflösen; wir erhalten die Gleichung

$$g^{B+f(v_0)+f(v_1)+\cdots+f(v_{i-1})} = g^{x+f(u_0)+f(u_1)+\cdots+f(u_{j-1})}$$

und somit den gesuchten diskrete Logarithmus von u als

$$x = B + f(v_0) + f(v_1) + \cdots + f(v_{i-1}) - f(u_0) - f(u_1) - \cdots - f(u_{j-1}) .$$

Um Aufwand und Erfolgschancen der Jagd abzuschätzen, gehen wir davon aus, daß beide sich die Sprungpositionen beider Kängurus wie Zufallsfolgen verhalten. Ist m das arithmetische Mittel von S, braucht das zahme Känguru ungefähr $\alpha = (B - A)/m$ Sprünge, um das Intervall zu durchqueren; da das wilde irgendwo mitten im Intervall startet, können es da erheblich weniger sein. Auf jeden Fall haben wir aber in einem Intervall der Länge $B - A$ mehr als α zufällige Werte; nach dem Geburtstagsparadoxon (mit dem wir uns im Kapitel über Hashfunktionen noch genauer beschäftigen werden) steigt die Chance auf eine Koinzidenz sehr schnell in die Nähe der Eins, sobald $\alpha = (B-A)/m > \sqrt{B - A}$ ist, also $m < \sqrt{B - A}$. Mit m etwas kleiner als $\sqrt{B - A}$ haben wir etwas mehr als $\sqrt{B - A}$ Sprünge des zahmen Kängurus und im Mittel etwa halb so viele für das wilde; der Aufwand ist also in der Größenordnung von $\sqrt{B - A}$ mit einer zwar recht hohen Erfolgswahrscheinlichkeit, aber ohne Erfolgsgarantie.

h) Indexkalkül

Die derzeit besten Faktorisierungsalgorithmen beruhen auf dem quadratischen und dem Zahlkörpersieb; für beide wurden bald nach ihrer Einführung ähnliche Siebalgorithmen gefunden, die zur Berechnung diskreter Logarithmen führen. Wir beschränken uns hier, wie auch schon bei der Faktorisierung, auf das quadratische Sieb, dessen Variante für diskrete Logarithmen als *Indexkalkül* bezeichnet wird nach der in der Zahlentheorie ebenfalls gebräuchlichen Sprechweise Index für den diskreten Logarithmus. Wir beschränken uns auf die einfachste Variante speziell für Primkörper $\mathbb{F}_p$.

Wie beim quadratischen Sieb wird eine Schranke B festgelegt und damit eine Faktorbasis $\mathcal{B}$ definiert; diese besteht hier aus *allen* Primzahlen $q \leq B$. Der Algorithmus besteht aus zwei Teilen:

Im ersten Teil berechnet man die diskreten Logarithmen aller Primzahlen q aus der Faktorbasis zur gegebenen Basis a modulo p. Dies mag auf den ersten Blick unsinnig erscheinen, denn schließlich suchen wir

den diskreten Logarithmus *einer* Zahl und beginnen dazu mit der Berechnung der diskreten Logarithmen vieler Zahlen. Die Logarithmen der Primzahlen lassen sich aber simultan wie folgt berechnen: Man berechne viele Potenzen $a^y \bmod p$ und suche diejenigen, die eine Primfaktorzerlegung mit lauter Faktoren aus $\mathcal{B}$ haben. Ist

$$a^y \bmod p = q_1^{e_1} \cdots q_r^{e_r} ,$$

so ist

$$y \equiv e_1 \log_a q_1 + \cdots + e_r \log_a q_r \bmod m ,$$

wobei m die kleinste natürliche Zahl ist mit $a^m \equiv 1 \bmod p$. Für eine primitive Wurzel a modulo p ist $m = p - 1$, ansonsten kann m auch ein echter Teiler davon sein.

Mit genügend vielen Gleichungen dieser Form hat man ein lineares Gleichungssystem für die Logarithmen der $q \in \mathcal{B}$, allerdings leider nicht über einem Körper, sondern modulo der im allgemeinen zusammengesetzten Zahl m. Falls m Produkt von Primzahlen ist, löst man das Gleichungssystem modulo jeder dieser Primzahlen und setzt die Lösungen nach dem chinesischen Restesatz zusammen; wenn auch echte Primzahlpotenzen P^s in m stecken, schreibt man die e_i und die linken Seiten y im Zahlensystem zur Basis P und erhält dann für jede Ziffer ein lineares Gleichungssystem über dem Körper mit P Elementen, aus denen man die Lösung modulo P^s zusammensetzen kann. Dieser erste Schritt ist offensichtlich völlig unabhängig vom Element x, dessen Logarithmus wir suchen; er kann für eine gegebene Basis a und Primzahl p ein für allemal im voraus durchgeführt werden.

Im zweiten Schritt betrachtet man für zufällig gewählte Exponenten y die Elemente $a^y x \bmod p$, bis man eines findet, das nur durch Primzahlen aus der Faktorbasis teilbar ist. Falls etwa $a^y x \bmod p = q_1^{f_1} \cdots q_s^{f_s}$ ist, erhalten wir

$$\log_a x = f_1 \log_a q_1 + \cdots + f_s \log_a q_s - y \bmod m .$$

Leider gibt es kein Siebverfahren, mit dem sich feststellen läßt, welche Werte $a^y \bmod p$ durch eine gegebene Primzahl q teilbar sind; hier muß man also explizit faktorisieren – zumindest so lange, bis alle Faktoren aus $\mathcal{B}$ gefunden sind. Egal ob man hier mit Probedivisionen arbeitet

oder etwa mit POLLARDs ρ-Methode oder mit elliptischen Kurven: Das Verfahren ist für große p deutlich schneller als beispielsweise *baby step – giant step*, und der Aufwand steigt langsamer als exponentiell in der Ziffernzahl von p.

§4: Diskrete Logarithmen in anderen Gruppen

2048 Bit-Zahlen sind bereits ziemlich unhandlich, und in Zukunft wird die Mindestlänge sicherer Moduln garantiert noch weiter wachsen. Daher suchen Kryptologen bereits seit langer Zeit nach Alternativen. Die derzeit interessantesten (und zunehmend bereits in der Praxis eingesetzten) Verfahren beruhen auf einer Verallgemeinerung diskreter Logarithmen:

a) Die abstrakte Situation

Ist G irgendeine Gruppe und $g \in G$, so können wir die Abbildung

$$\varphi_g \colon \mathbb{Z} \to G; \quad n \mapsto g^n$$

betrachten. Falls man in der Gruppe G überhaupt konkret rechnen kann, lassen sich die Werte $\varphi_g(n) = g^n$ nach dem üblichen Verfahren durch Quadrieren und Multiplizieren mit einem Aufwand in der Größenordnung $\log_2 n$ berechnen.

Die Umkehrfunktion $\varphi_g^{-1} \colon \text{Bild}\, \varphi_g \to \mathbb{Z}/\,\text{Kern}\, \varphi_g$ bezeichnen wir auch hier als einen diskreten Logarithmus; falls er hinreichend schwer zu berechnen ist, eignet er sich als Grundlage für Kryptosysteme.

Das Bild von φ_g besteht offensichtlich genau aus den Potenzen von g, ist also eine zyklische Gruppe. Damit können wir uns ohne Beschränkung der Allgemeinheit auf zyklische Gruppen beschränken.

Für die Berechnung diskreter Logarithmen können wir die meisten Verfahren aus dem letzten Paragraphen ohne nennenswerte Änderungen auf die abstrakte Situation verallgemeinern: Für den chinesischen Restesatz sowie die Methode von POHLIG und HELLMAN brauchen wir nur irgendeine zyklische Gruppe, und auch den zahmen und wilden Kängurus ist es gleichgültig, durch welche Gruppe wir sie springen lassen:

Die Sprungweiten beziehen sich schließlich auf den stets ganzzahligen Exponenten.

Anders ist es beim Indexkalkül: Hier haben wir ganz wesentlich benutzt, daß sich die Elemente, deren diskrete Logarithmen wir suchen, als natürliche Zahlen darstellen lassen, so daß wir mit deren Primzerlegung arbeiten können. Die sollte nicht in allen Gruppen funktionieren.

b) Multiplikative Gruppen beliebiger endlicher Körper

Tatsächlich gibt es nicht nur für jede Primzahl p einen Körper mit p Elementen, sondern für jede Primzahl*potenz* $q = p^n$. Vor allem für den Fall $q = 2^8 = 256$ werden wir dies im nächsten Kapitel genauer betrachten.

Wie wir aus §3b wissen, bilden die von Null verschiedenen Elemente eines endlichen Körpers bezüglich der Multiplikation eine zyklische Gruppe; falls ihre Ordnung $q - 1 = p^n - 1$ prim ist oder einen großen Primteiler hat, lassen sich auch so sichere Kryptosysteme aufbauen.

In der Praxis kryptographischen Praxis spielen Potenzen ungerader Primzahlen allerdings kaum eine Rolle: Das Rechnen in den entsprechenden Körpern ist aufwendiger als das in einem vergleichbar großen Körper von Primzahlordnung, ohne daß dies zu einem Sicherheitsgewinn führen würde, da eine Variante des Indexkalküls bei beliebigen endlichen Körpern funktioniert.

Anders steht es mit Körpern von Zweipotenzordnung: Da Computer ohnehin im Zweiersystem rechnen, können sie damit sehr effizient umgehen. Es gibt sogar eine Reihe von Exponenten, für die $2^n - 1$ prim ist; für $n \leq 2\,500$ sind dies 2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279, 2203 und 2281.

Grundsätzlich dürfte wohl die Sicherheit diskreter Logarithmen in einem Körper mit 2^n Elementen vergleichbar sein mit der eines Körpers, dessen Elementanzahl eine Primzahl derselben Größenordnung ist, allerdings gibt es sehr viel weniger Zweierpotenzen als Primzahlen, und nicht für alle n hat $2^n - 1$ einen großen Primteiler. Dadurch besteht die Gefahr, daß sich kriminelle Energie (sowie auch Nachrichtendienste) auf die wenigen interessanten Werte von n stürzen und dort mit Spezialhardware

arbeiten, was die Sicherheitssituation zu ihren Gunsten verschiebt. Somit dürften im allgemeinen Primzahlen vorzuziehen sein; hinzu kommt, daß zumindest DSA ohnehin nur für diesen Fall definiert ist.

c) Elliptische Kurven

In der Praxis ist für Kryptoverfahren auf der Basis diskreter Logarithmen abgesehen von den multiplikativen Gruppe der Körper von Primzahlpotenzordnung vor allem noch eine andere Art von Gruppe wichtig: die Gruppe der rationalen Punkte einer elliptischen Kurve über einem endlichen Körper. In bislang eher noch experimentellen Systemen arbeitet man auch mit höherdimensionalen Verallgemeinerungen davon, hauptsächlich den JACOBIschen hyperelliptischer Kurven. Da elliptische Kurven Gegenstand einer eigenen Vorlesung sind, soll hier nur kurz erläutert werden, um welche Art von Gruppe es sich bei einer elliptischen Kurven handelt.

Elliptische Kurven sind keine Ellipsen; sie haben ihren Namen davon, daß bei der Berechnung der Bogenlänge einer Ellipse algebraische Integrale auftreten, deren Integranden solche Kurven definieren.

Eine elliptische Kurve über einem Körper k ist eine ebene Kurve vom Grad drei; sie ist also gegeben durch ein Polynom f vom Grad drei mit Koeffizienten aus k in zwei Variablen x und y. Wir verlangen zusätzlich, daß sich das Polynom f auch über Erweiterungskörpern von k nicht als Produkt eines linearen und eines quadratischen Polynoms schreiben läßt, daß es mindestens eine Nullstelle $(x, y) \in k^2$ hat und daß die partiellen Ableitungen f_x und f_y für keine Lösung (x, y) der Gleichung $f(x, y) = 0$ über irgendeinem Erweiterungskörper von k simultan verschwinden.

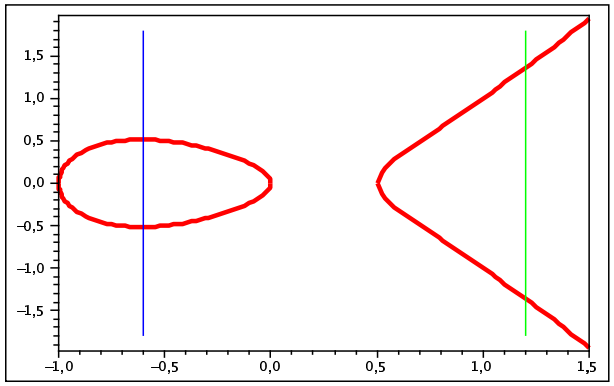
Geometrisch bedeuten diese Forderungen, daß die durch $f(x, y) = 0$ definierte Kurve nicht als Vereinigung einer Geraden und einer anderen Kurve geschrieben werden kann, daß sie mindestens einen Punkt mit Koordinaten aus k hat und daß es in jedem Punkt eine wohldefinierte Tangente gibt.

Schneiden wir eine solche Kurve mit einer Geraden, so erlaubt uns die Geradengleichung die Elimination einer der beiden Variablen x und y; was übrig bleibt ist eine höchstens kubische Gleichung in der anderen.

Damit ist klar, daß eine Gerade eine elliptische Kurve in höchstens drei Punkten schneidet.

Um besser zu sehen, was hier möglich ist, beschränken wir uns auf Kurven mit einer Gleichung der speziellen Form $y^2 = f(x)$, wobei $f(x)$ ein Polynom dritten Grades in x ist und nehmen außerdem an, daß der Körper k nicht den Körper $\mathbb{F}_2$ enthält. (Über einem solchen Körper läßt sich sogar für *jede* elliptische Kurve ein Koordinatensystem finden, in dem sie wie oben geschrieben werden kann; unsere Annahme ist also keine echte Einschränkung.) Die Bedingung, daß die partiellen Ableitungen nach x und y in keinem Kurvenpunkt *beide* verschwinden dürfen, besagt hier einfach, daß $f(x)$ keine mehrfache Nullstelle hat.

Falls $f(x_0)$ für ein $x_0 \in k$ eine Quadratwurzel $y_0 \in k$ hat, ist auch $-y_0$ eine; wenn $f(x_0)$ nicht verschwindet, gibt es also genau zwei Punkte mit x-Koordinate x_0. Die Verbindungsgerade dieser beiden Punkte ist natürlich $x = x_0$, und offensichtlich gibt es keinen dritten Schnittpunkt mit der Kurve.



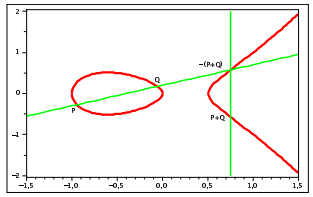
Um dieses Problem zu umgehen, ergänzen wir die Kurve durch einen weiteren Punkt O, der auf jeder Gerade der $x = x_0$ liegen soll; wir bezeichnen O als den „unendlich fernen“ Punkt in Richtung der y-Achse.

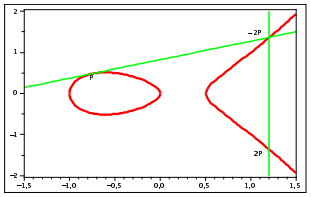
Im Fall zweier Punkte (x_1, y_1) und (x_2, y_2) mit $x_1 \neq x_2$ überzeugt man sich leicht, daß die Verbindungsgerade $y = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) + y_1$

die Kurve in (mit Vielfachheit gezählt) drei Punkten schneidet, denn schreiben wir die Geradengleichung in der Form $y = mx + b$ und setzen in die Kurvengleichung ein, erhalten wir die Gleichung dritten Grades $f(x) - (mx + b)^2 = 0$. Sie hat x_1 und x_2 als Nullstellen, und nach Division durch $(x - x_1)(x - x_2)$ bleibt eine lineare Gleichung für die dritte Lösung x_3 übrig.

Damit könnte man versuchen, eine Verknüpfung zu definieren, indem je zwei Punkten der dritte Schnittpunkt ihrer Verbindungsgeraden mit der Kurve zugeordnet wird. Diese Verknüpfung erfüllt aber leider abgesehen vom Kommutativgesetz keines der Gruppenaxiome.

Eine leichte Modifikation führt aber zu einer Gruppenstruktur: Wenn P, Q und R auf einer Geraden liegen, soll das nicht bedeuten, daß $R = P + Q$ ist, sondern daß $P + Q + R$ gleich dem Neutralelement O ist; somit ist also $R = -(P + Q)$. Da der unendlich ferne Punkt O so gewählt wurde, daß (x, y), $(-x, y)$ und O auf einer Geraden liegen, ist für $P = (x, y)$ das inverse Element einfach gleich $(-x, y)$; zur Berechnung von $P + Q$ muß also der dritte Schnittpunkt der Geraden durch P und Q noch an der x-Achse gespiegelt werden. (Im Falle $P = Q$ ist unter der Geraden durch P und Q natürlich die Tangente im Punkt P zu verstehen.)





Alle Gruppenaxiome mit Ausnahme des Assoziativgesetzes lassen sich leicht überprüfen; für letzteres ist jedoch einiges an zusätzlicher mathematischer Theorie notwendig, was den Rahmen dieser Vorlesung sprengen würde.

Die Punkte einer elliptischen Kurve können offensichtlich nicht mit ganzen Zahlen identifiziert werden, und von ihrer Primzerlegung können wir auch nicht reden. Damit fällt der Indexkalkül als Strategie zur Berechnung diskreter Logarithmen auf elliptischen Kurven weg.

Natürlich gibt es auch spezielle Methoden für die Berechnung diskreter Logarithmen auf elliptische Kurven; in machen Fällen kann man sie sogar zurückführen auf die Berechnung diskreter Logarithmen in der multiplikativen Gruppe eines nicht sehr viel größeren Körpers. Wer mit den theoretischen Grundlagen der Kryptographie mit elliptischen Kurven vertraut ist, kann aber zumindest die bekannten Angriffsmethoden so erschweren, daß der entsprechende Erweiterungskörper schon für relativ kleine Grundkörper so groß ist, daß dort nach heutigem Kenntnisstand auch klassische diskrete Logarithmenprobleme nicht effizient gelöst werden können. Die Empfehlungen der Bundesnetzagentur sehen hier daher für die Kryptographie mit elliptischen Kurven deutlich kleinere Primzahlen vor als im klassischen Fall. Wie dort gibt es zwei Primzahlen p und q: Die elliptische Kurve ist definiert über einem Körper mit p Elementen, und die Unterschrift liegt in einer Untergruppe der Ordnung q der elliptischen Kurve. Wie beim DSA muß q eine Länge von derzeit mindestens 224 Bit haben, ab 2016 sind es 250. Für die Primzahl p dagegen gibt es keinerlei Einschränkung, außer daß $p \neq q$ sein muß und es auf natürlich auch eine elliptische Kurve über $\mathbb{F}_p$ geben muß, auf der ein Punkt der Ordnung q liegt. Zur Erschwerung der oben erwähnten Angriffsmethode wird außerdem gefordert, daß es kein $r \leq 10^4$ geben darf, so daß q ein Teiler von $p^r - 1$ ist; die Gruppe, in der die Unterschriften liegen, soll sich also nicht in die multiplikative Gruppe einer „kleinen“ Erweiterung des Grundkörpers einbetten lassen. Außerdem soll noch die Hauptordnung, die zum Endomorphismenring der Kurve gehört, eine Klassenzahl von mindestens 200 haben – wie man sieht, steckt in der Kryptographie mit elliptischen Kurve einiges mehr an Mathematik als in den klassischen Verfahren, so daß Einzelheiten im Rahmen dieser Vorlesung nicht behandelt werden können.

§5: Literatur

Diskrete Logarithmensysteme über endlichen Körper werden in denselben Büchern behandelt wie RSA; hierfür sei daher auf die Literaturangaben zum vorigen Kapitel verwiesen. Bücher, die auch Verfahren auf der Grundlage elliptischer und (teilweise) hyperelliptischer Kurven betrachten sind

NEAL KOBLITZ: A Course in Number Theory and Cryptography, *Graduate Texts in Mathematics* **114**, *Springer*, ^{2}1994

und

NEAL KOBLITZ: Algebraic Aspects of Cryptography, *Springer*, 1998

Als ersten Überblick über elliptische Kurven kann man etwa das Buch

ANNETTE WERNER: Elliptische Kurven in der Kryptographie, *Springer*, 2002,

konsultieren, das eine elementare Einführung in die Theorie elliptischer Kurven unter dem Gesichtspunkt der Kryptographie geht. Beweise sind nicht immer vollständig, und anspruchsvollere Algorithmen werden nur sehr kurz oder gar nicht behandelt.

Deutlich anspruchsvoller und vollständiger sind

DARREL HANKERSON, ALFRED MENEZES, SCOTT VANSTONE: Guide to Elliptic Curve Cryptography, *Springer*, 2004

LAWRENCE C. WASHINGTON: Elliptic Curves – Number Theory and Cryptography, *Chapman & Hall/CRC*, 2003

IAN BLAKE, GADIEL SEROUSSI, NIGEL SMART: Elliptic Curves in Cryptography, *London Mathematical Society Lecture Notes Series* **265**, *Cambridge University Press*, 1999

IAN BLAKE, GADIEL SEROUSSI, NIGEL SMART [HRSG.:] Advances in Elliptic Curve Cryptography, *London Mathematical Society Lecture Notes Series* **317**, *Cambridge University Press*, 2005

Die vollständigste und ausführlichste Information bietet derzeit wohl

HENRI COHEN, GERHARD FREY, ROBERTO AVANZI, CHRISTOPHE DOCHE, TANJA LANGE, KIM NGUYENM FREDERIK VERCAUTEREN: Handbook of Elliptic and Hyperelliptic Curve Cryptography, *Chapman & Hall/CRC*, 2006

Speziell mit Implementierungsfragen beschäftigt sich

MICHAEL ROSING: Implementing Elliptic Curve Cryptography, *Manning*, 1999

Kapitel 6
Der Advanced Encryption Standard Rijndael

§ 1: Geschichte und Auswahlkriterien

DES wurde in Zusammenarbeit mit der National Security Agency der Vereinigten Staaten von IBM entwickelt und dann als amerikanischer Standard verkündet. Diese Vorgehensweise weckte von Anfang an den Verdacht, daß möglicherweise eine „Falltür“ eingebaut sei, insbesondere da zumindest ursprünglich nicht alle Design-Kriterien publiziert wurden.

Nachdem dreißig Jahre intensiver Kryptanalyse keine Falltür gefunden haben, müßte diese – so vorhanden – zumindest sehr gut versteckt sein; aber egal ob mit oder ohne Falltür: Wie wir im letzten Kapitel gesehen haben, erfüllt DES mit nur 56 Bit Schlüssellänge nicht mehr die heutigen Sicherheitsanforderungen. Ursprünglich war er ohnehin nur für eine Laufzeit von zehn Jahren vorgesehen und wurde nur immer wieder verlängert, weil die meisten Anwender von Kryptographie äußerst träge sind und sich nicht um Fortschritte der Kryptanalyse kümmern.

Obwohl es die Internationale Standardisierungsorganisation ISO ablehnt, ein Kryptoverfahren zu standardisieren (Ein Grund dafür ist die dann befürchtete Bündelung krimineller Energie auf dieses Verfahren), hat das amerikanische Handelsministerium die Suche nach einem Nachfolgealgorithmus für DES am 2. Januar 1997 international ausgeschrieben; der vorläufige Name des Algorithmus war AES *(Advanced Encryption Standard).* Federführend für die Auswahl war das *National Institute of Standards and Technology* (NIST) in Gaithersburg, Maryland.

Dieses ging von Anfang an davon aus, daß der zu wählende Algorithmus *stärker* sein müsse als Triple DES; er sollte zwanzig bis dreißig Jahre lang anwendbar sein und dementsprechende Sicherheit bieten. Nach einer internationalen Konferenz über die Auswahlkriterien am 15. April 1997 veröffentlichte es am 12. September 1997 die endgültige Ausschreibung.

Minimalanforderung an die einzureichenden Algorithmen waren danach, daß es sich um symmetrische Blockchiffren handeln muß, die mindestens eine Blocklänge von 128 Bit bei Schlüssellängen von 128 Bit, 192 Bit und 256 Bit vorsieht.

Als Kriterien für die Wahl zwischen den einzelnen Algorithmen wurden die folgenden Aspekte genannt:

1. **Sicherheit:** Wie sicher ist der Algorithmus im Vergleich zu den anderen Kandidaten? Inwieweit ist seine Ausgabe ununterscheidbar von der einer Zufallspermutation? Wie gut ist die mathematische Basis für die Sicherheit des Algorithmus begründet? (Im Gegensatz zu DES sollten dieses Mal alle Kriterien publiziert werden.)
2. **Kosten:** Welche Lizenzgebühr werden fällig? Wie effizient geht der Algorithmus mit Rechenzeit und Speicherplatz um?
3. **Flexibilität:** Ein Algorithmus, der auf einer Vielzahl von Plattformen implementierbar ist (PCs, 8-Bit-Prozessoren, ATM-Netze, HDTV, . . .) ist vorzuziehen, genauso einer, der auch als Stromchiffre, kryptographisch sichere Hash-Funktion oder ähnliches verwendet werden kann.
4. **Software:** Der Algorithmus *muß* auch softwaremäßig effizient implementierbar sein.
5. **Einfachheit:** Der Aufbau des Algorithmus soll möglichst einfach sein.

Nicht nur die Art der Suche unterschied sich also beträchtlich von der DES-Entwicklung hinter verschlossenen Türen, auch die Auswahlkriterien hatten sich geändert: DES war noch in erster Linie für Hardware entworfen; manche Aspekte wie etwa die für die kryptographische Sicherheit völlig irrelevante Anfangspermutation hatten wohl keinen anderen Sinn als die Verlangsamung von Software-Implementierungen.

Inzwischen hat freilich die Erfahrung mit der Kryptographie in der Geschäftswelt gezeigt, daß man dort den Kostenaufwand für Spezialhardware scheut und sich lieber mit den dubiosen Verfahren begnügt, die in der ohnehin vorhandenen Office-Software eingebaut ist (Preis für das Knacken durch spezialisierte Unternehmen je nach Programm zwischen 40 $ und 250 $), wohingegen sich die Hacker innerhalb und außerhalb der Geheimdienste durch einen hohen Aufwand nur wenig beeindrucken lassen.

Die Ausschreibung führte bis zum Abgabeschluß am 15. Juni 1998 zu fünfzehn Vorschlägen aus aller Welt. Diese wurden auf einer ersten AES-Konferenz vom 20.–22. August 1998 in Ventura, Kalifornien vorgestellt und der Fachwelt zur Analyse und Kommentierung empfohlen. Tatsächlich wurden vierzehn der fünfzehn Algorithmen schon vor der Konferenz veröffentlicht; lediglich der Algorithmus *Magenta* der Deutschen Telekom wurde erst am 20. August auf der Konferenz selbst bekanntgegeben. Er war auch der einzige Algorithmus, der bereits während der Konferenz geknackt wurde in einer auf den 20. August 1998 datierten Arbeit von E. BIHAM, A. BIRYUKOV, N. FERGUSON, L. KNUDSEN, B. SCHNEIER und A. SHAMIR, die noch auf der Konferenz verteilt wurde.

Die zweite AES-Konferenz am 22. und 23. April 1999 in Rom führte zu einer ersten Diskussion der Ergebnisse und Empfehlungen, welche der fünfzehn Algorithmen weiter betrachtet werden sollten. Die endgültige Entscheidung des NIST wurde am 9. August 1999 bekanntgegeben:

Bei fünf der eingereichten Algorithmen hatte sich herausgestellt, daß sie entweder mit einem Aufwand zu knacken sind, der deutlich unter der Durchsuchung des gesamten Schlüsselraums liegt (darunter natürlich auch Magenta) oder aber, daß es zu viele „schwache“ Schlüssel gibt. Da diese fünf Algorithmen auch mit die langsamsten Kandidaten waren, sprach nichts dafür, sie noch weiter zu betrachten.

Fünf weitere Kandidaten zeigten ebenfalls Schwächen, die zwar für sich allein betrachtet nicht so schwerwiegend waren, daß man die Verfahren eliminieren müßte; da diese fünf Kandidaten andererseits nirgends entscheidende Vorteile boten, wurden auch sie eliminiert, so daß noch fünf

Finalisten übrig blieben: Drei aus USA (MARS von IBM, RC6 von RSA und Twofish von Counterpane), Serpent von drei Kryptologen aus England, Israel und Norwegen, und Rijndael von zwei Kryptologen, die damals an der Katholischen Universität Leuven in Belgien arbeiteten.

Bei der dritten AES-Konferenz am 13. und 14. April 2000 in New York bekam Rijndael 86 Stimmen, Serpent 59, Twofish 31, RC6 23 und MARS 13, so daß es nicht weiter verwunderte, daß das NIST am 2. Oktober 2000 Rijndael für den vorgeschlagenen Standard nominierte. Am 6. Dezember 2001 verkündete ihn dann das amerikanische Handelsministerium offiziell als Federal Information Processing Standard FIPS-197.

Rijndael hat seinen Namen von seinen beiden Autoren JOAN DAEMEN (*1965) und VINCENT RIJMEN (*1970), die 1995 beziehungsweise 1997 an der Elektrotechnischen Fakultät der Katholischen Universität Leuven über Themen aus der Kryptographie promoviert hatten; RIJMEN hat inzwischen einen Lehrstuhl fÃ…r Kryptographie an der Technischen Universität Graz, DAEMAN arbeitet bei dem Halbleiterhersteller STMicroelectronics. Aufgrund der Wahl von Rijndael zum AES wurden sie als flämische Persönlichkeiten des Jahres 2000 gewählt.

Als Aussprachehilfe für Personen, die kein Niederländisch, Flämisch, Surinamer oder Afrikaans sprechen, geben die Autoren folgende englische Approximationen des Wortes „Rijndael“: „Reign Dahl“, „Rain Doll“ und „Rhine Dahl“.

§ 2: Algebraische Vorbereitungen

Alle Operationen von Rijndael sind algebraisch definiert. Es gibt einmal Operationen auf Byte-Ebene, die durch die Grundrechenarten im Körper mit 256 Elementen realisiert sind; dazu kommen Operationen auf 4-Byte-Wörtern, die im Polynomring über diesem Körper definiert sind. Als erstes müssen wir daher mit diesem Körper vertraut werden.

a) Euklidische Ringe

Für den EUKLIDischen Algorithmus im ersten Abschnitt waren im wesentlichen zwei Dinge wesentlich: Ersten mußten wir wissen, wann eine

Zahl Teiler einer anderen ist, und zweitens brauchten wir eine Divison mit Rest, für die der Rest in irgendeiner Weise kleiner als der Divisor ist, wobei jede Folge immer kleiner werdender Reste nach endlich vielen Schritten abbrechen muß. Diese beiden Eigenschaften werden im Begriff des EUKLIDischen Rings formalisiert:

Definition: *a)* Ein Ring ist eine Menge R zusammen mit zwei Verknüpfungen $+, \cdot\colon R \times R \to R$, für die gilt:

1.) $(R, +)$ ist eine abelsche Gruppe

2.) $a(bc) = (ab)c$ für alle $A, b, c \in R$

3.) $a(b + c) = ab + ac$ für alle $a, b, c \in R$.

b) R heißt *kommutativer Ring mit Eins*, falls zusätzlich gilt

4.) Es gibt ein Element $1 \in R$, so daß $1 \cdot a = a \cdot 1 = a$ für alle $a \in R$

5.) $a \cdot b = b \cdot a$ für alle $a, b \in R$.

c) Ein kommutativer Ring mit Eins heißt *Integritätsbereich*, wenn gilt:

6.) Für $a, b \in R \setminus \{0\}$ ist auch $ab \neq 0$.

d) Ein Element t eines Integritätsbereichs R heißt Teiler von $r \in R$, in Zeichen $t|r$, wenn es ein $q \in R$ gibt mit $r = qt$.

e) $d \in R$ heißt größter gemeinsamer Teiler von $r, s \in R$, wenn $d|r, s$ und wenn für jeden weiteren gemeinsamen Teiler t von r und s gilt: $t|d$.

f) Ein EUKLIDischer Ring ist ein Integritätsbereich R zusammen mit einer Abbildung $\nu\colon R \setminus \{0\} \to \mathbb{N}_0$, für die gilt:

7.) Zu je zwei Elementen $a, b \in R \setminus \{0\}$ gibt es $q, r \in R$ mit

$$a = bq + r \quad \text{und} \quad \nu(r) < \nu(b) \ \text{ falls } r \neq 0$$

und für jeden Teiler b von a gilt $\nu(b) \leq \nu(a)$.

Offensichtlich ist der Ring $\mathbb{Z}$ der ganzen Zahlen ein EUKLIDischer Ring; hier können wir einfach $\nu(a) = |a|$ setzen. Als Übung kann man auch leicht zeigen, daß der Ring

$$\Gamma = \mathbb{Z} \oplus \mathbb{Z}i = \left\{ a + bi \in \mathbb{C} \;\middle|\; a, b \in \mathbb{Z} \right\}$$

der GAUSSschen Zahlen EUKLIDisch ist mit

$$\nu(a + ib) = a^2 + b^2 \,.$$

Interessanter für uns ist, daß für jeden Körper k der Polynomring

$$k[X] = \left\{ \sum_{\ell=0}^{n} a_\ell X^\ell \;\middle|\; n \in \mathbb{N}_0 \ \text{ und } \ a_i \in k \right\}$$

ein EUKLIDischer Ring ist, wobei die Funktion ν hier einfach jedem Polynom seinen Grad zuordnet, also den Exponenten der höchsten vorkommenden X-Potenz. Hier zeigt die übliche Polynomdivison mit Rest, daß es in der Tat zu je zwei Polynomen f, g mit Koeffizienten aus k Polynome q, r gibt, so daß

$$f = gq + r \quad \text{und} \quad r = 0 \ \text{ oder } \ \deg r < \deg g \,.$$

Man beachte, daß die oben definierten größten gemeinsamen Teiler nicht eindeutig bestimmt sein müssen: In $\mathbb{Z}$ beispielsweise ist sowohl drei als auch minus drei ein größter gemeinsamer Teiler von sechs und minus neun. Allgemein gilt:

Lemma: In einem EUKLIDischen Ring R gibt es zu je zwei Elementen r, s, die nicht beide gleich null sind, einen größten gemeinsamen Teiler d. Dieser kann nach dem EUKLIDischen Algorithmus berechnet werden und läßt sich als Linearkombination

$$d = \alpha r + \beta s \quad \text{mit} \quad \alpha, \beta \in R$$

darstellen. Sind d und d' zwei größte gemeinsame Teiler von r und s, so gibt es eine Einheit e mit $d' = ed$.

Beweis: Die Existenz eines größten gemeinsamen Teilers folgt genau wie im Fall der natürlichen Zahlen: Ist $a = bq + r$, so ist ein Element $t \in R$ genau dann gemeinsamer Teiler von a und b, wenn es gemeinsamer Teiler von b und r ist. Daher gibt es genau dann einen größten gemeinsamen Teiler von a und b, wenn es einen größten gemeinsamen Teiler von b und r gibt, und dieser ist dann gleichzeitig größter gemeinsamer Teiler von a und b.

Der EUKLIDische Algorithmus funktioniert in einem beliebigen EUKLIDischen Ring genau wie im Ring der ganzen Zahlen, und genau wie dort muß er auch enden mit einem Divisionsrest null, denn die Folge der Zahlen $\nu(r_i) \in \mathbb{N}_0$ ist strikt fallend. Im Falle $r_n = 0$ ist

ggT$(r_{n-2}, r_{n-1}) = r_{n-1}$, da r_{n-1} dann r_{n-2} ohne Rest teilt; induktiv folgt, daß das auch ein ggT von a und b ist.

Sind d und d' zwei größte gemeinsame Teiler von a und b, so sind beide insbesondere gemeinsame Teiler von a und b; nach Definition eines größten gemeinsamen Teilers muß daher d Teiler von d' sein und umgekehrt. Es gibt somit ein $e \in R$ mit $d' = ed$ und ein $e' \in R$ mit $d = e'd'$. Also ist

$$d = e'd' = e'ed \Longrightarrow (1 - e'e)d = 0 \Longrightarrow 1 - e'e = 0 \Longrightarrow e'e = 1\,,$$

da R nach Voraussetzung ein Integritätsbereich ist. Die letzte Gleichung rechts zeigt, daß e eine Einheit ist.

Schließlich müssen wir noch zeigen, daß sich *jeder* größte gemeinsame Teiler d von a und b als Linearkombination von a und b schreiben läßt. Der erweiterte EUKLIDische Algorithmus liefert eine solche Darstellung für *einen* größten gemeinsamen Teiler

$$d' = \alpha a + \beta b\,;$$

sind e, e' die oben betrachtete Einheit zu d und d', so folgt

$$d = e'd' = (\alpha e')a + (\beta e')b\,,$$

wie behauptet. ∎

Als Beispiel wollen wir für $k = \mathbb{Q}$ den größten gemeinsamen Teiler der beiden Polynome

$$P = X^8 + X^6 - 3X^4 - 3X^3 + 8X^2 + 2X - 5$$

und

$$Q = 3X^6 + 5X^4 - 4X^2 - 9X + 21$$

berechnen: Division von P durch Q führt auf den Quotienten $X^2/3 - 2/9$ und Divisionsrest

$$R_2 = -\frac{5}{9}X^4 + \frac{1}{9}X^2 - \frac{1}{3}\,.$$

Division von Q durch R_2 ergibt

$$R_3 = -\frac{117}{25}X^2 - 9X + \frac{441}{25}\,,$$

bei der Division von R_2 durch R_3 bleibt Rest

$$R_4 = \frac{233150}{6591}X - \frac{102500}{2197}\,,$$

und bei der letzten Divison verbleibt als Rest der ggT

$$R_5 = \frac{1288744821}{543589225}\,.$$

Da beide Ausgangspolynome ganzzahlige Koeffizienten haben, erscheint ein ggT mit einem so großen Nenner seltsam. Wir wissen aber, daß „der“ größte gemeinsame Teiler nur bis auf Einheiten bestimmt ist, und im Polynomring über einem Körper sind alle von null verschiedenen Konstanten Einheiten. Der größte gemeinsame Teiler ist daher nur eindeutig bis auf Multiplikation mit einer nichtverschwindenden Konstanten; diese Konstante kann nach Belieben gewählt werden und wird meist so gewählt, daß das Ergebnis in irgendeinem Sinne einfach wird.

Auf das obige Beispiel angewendet heißt das, daß mit

$$R_5 = \frac{1288744821}{543589225}$$

auch eins ein ggT von A und B ist und man daher im allgemeinen sagen würde, „der“ ggT von A und B sei eins. Es ist ein wohlbekanntes (und umgehbares) Problem der Computeralgebra, daß der EUKLIDische Algorithmus diese einfache Lösung in einer so komplizierten Form liefert; da wir aber vor allem Polynome über endlichen Körpern benötigen, braucht uns das nicht weiter zu kümmern.

b) Endliche Körper von Primzahlpotenzordnung

Beim Beweis, daß die ganzen Zahlen modulo einer Primzahl einen Körper bilden, gab es nur einen nichttrivialen Schritt: die Existenz des multiplikativen Inversen, die wir aus der linearen Kombinierbarkeit des ggT folgerten und daraus, daß der ggT einer Zahl mit einer Primzahl gleich eins ist, falls die Zahl kein Vielfaches der Primzahl ist.

Genauso wollen wir jetzt Körper definieren, indem wir Polynome über einem festen Körper k modulo einem vorgegebenen Polynom P betrachten: Für ein beliebiges Polynom A über k ist A mod P gleich dem Rest bei der Division von A durch P.

Falls A kleineren Grad als P hat, ist natürlich einfach $A \bmod P = A$; zum konkreten Rechnen können wir daher ausgehen vom Vektorraum V aller Polynome vom Grad höchstens d, wobei $d+1$ der Grad von P ist. Die Addition ist die gewöhnliche Addition von Polynomen, das Nullpolynom ist Neutralelement, und $-A$ ist invers zu A.

Das Produkt AB zweier Polynome $A, B \in V$ kann größeren Grad als d haben; wir setzen daher

$$A \odot B = AB \bmod P\,;$$

dies ist ein Polynom vom Grad höchstens d, und es ist klar, daß die so definierte Multiplikation kommutativ und assoziativ ist und das Distributivgesetz erfüllt. Das konstante Polynom 1 ist Neutralelement auch bezüglich dieser Multiplikation. Algebraisch gesehen identifizieren wir V also mit dem Faktorring $k[X]/(P)$.

Ein inverses Polynom zu A ist ein Polynom B, für das $A \odot B = 1$ ist, d.h.

$$AB = 1 + CP \quad \text{oder} \quad AB + CP = 1$$

für ein geeignetes Polynom C. Zu vorgegebenen Polynomen A und P gibt es solche Polynome B und C genau dann, wenn der ggT von A und P gleich eins ist; alsdann lassen sich B und C nach dem erweiterten EUKLIDischen Algorithmus berechnen.

Wenn wir möchten, daß jedes Polynom A, dessen Grad kleiner als deg P ist, ein Inverses hat, müssen wir sicherstellen, daß A und P immer teilerfremd sind; dies ist offensichtlich genau dann der Fall, wenn P keinen nichttrivialen Teiler hat, keinen Teiler also, dessen Grad größer als null und kleiner als deg P ist. Ein solches Polynom heißt *irreduzibel*.

Falls es ein irreduzibles Polynom P vom Grad n mit Koeffizienten aus k gibt, läßt sich der Vektorraum k^n also zu einem Körper machen, indem wir ein n-tupel $(a_0, \ldots, a_{n-1})$ mit dem Polynom

$$a_{n-1}X^{n-1} + a_{n-2}X^{n-2} + \ldots + a_1X + a_0$$

identifizieren und die Multiplikation als Multiplikation von Polynomen modulo P erklären.

Bekanntestes Beispiel ist $k = \mathbb{R}$: Für $n = 2$ gibt es irreduzible Polynome vom Grad n, beispielsweise das Polynom $P = X^2 + 1$. Da

$$\begin{aligned}(a_1X + a_0)(b_1X + b_0) &= a_1b_1X^2 + (a_0b_1 + a_1b_0)X + a_0b_0 \\ &\equiv (a_0b_1 + a_1b_0)X + (a_0b_0 - a_1b_1) \bmod X^2 + 1\end{aligned}$$

ist, folgt

$$(a_0, a_1) \odot (b_0, b_1) = (a_0b_0 - a_1b_1, a_0b_1 + a_1b_0)\,,$$

wir erhalten also den Körper der komplexen Zahlen. Weitere Beispiele über $\mathbb{R}$ gibt es nicht, denn ein irreduzibles reelles Polynom muß entweder Grad eins oder Grad zwei haben, und da jedes irreduzible quadratische Polynom zwei konjugiert komplexe Nullstellen hat, entstehen dabei immer die komplexen Zahlen – lediglich die Basis über $\mathbb{R}$ ändert sich.

Über endlichen Körpern ist die Situation etwas komplizierter: Hier gibt es für *jedes* n mindestens ein irreduzibles Polynom vom Grad n, allerdings gilt auch hier, daß zwei irreduzible Polynome desselben Grads auf den gleichen Körper führen. Eine kurze Beweisskizze ist unten angedeutet; einen vollständigen Beweis findet man in jedem Lehrbuch der Algebra.

Es gibt keinen einfachen Ausdruck für ein irreduzibles Polynom vom Grad n über einem vorgegebenen endlichen Körper k; in der Computeralgebra behilft man sich meist damit, daß man so lange zufällige Polynome vom Grad n erzeugt, bis man ein irreduzibles gefunden hat – ein Algorithmus, der sich in der Praxis als deutlich effizienter erweist als manch ein deterministischer Algorithmus zur Lösung von Standardproblemen.

Es gibt allerdings auch eine Alternative, die zumindest für kleine Körper durchaus anwendbar ist: Ist $k = \mathbb{F}_q$ ein endlicher Körper mit q Elementen, so ist $k \setminus \{0\}$ eine multiplikative Gruppe der Ordnung $q-1$. Da die Ordnung eines jeden Elements einer Gruppe Teiler der Gruppenordnung ist, folgt

$$x^{q-1} = 1 \quad \text{für alle } x \in \mathbb{F}_q \setminus \{0\}\,.$$

Dies gilt insbesondere für das Element $X \bmod P$, das $\mathbb{F}_q$ über $\mathbb{F}_p$ erzeugt, also ist P ein Teiler von $X^{q-1} - 1$. Die möglichen Polynome P

zur Erzeugung von $\mathbb{F}_{p^n}$ über $\mathbb{F}_p$ sind also genau die irreduziblen Teiler vom Grad n des Polynoms $X^{p^n} - 1$. Die Computeralgebra stellt gerade für Polynome über endlichen Körpern effiziente Faktorisierungsalgorithmen zur Verfügung, so daß man diese Teiler noch für recht große Werte von p^n relativ schnell berechnen kann.

Als weitere Konsequenz aus obiger Formel kommt man auch zu einer neuen Interpretation des Körpers mit $q = p^n$ Elementen: Da alle q-1 Elemente von $\mathbb{F}_q \setminus \{0\}$ Nullstellen des Polynoms $x^{q-1}-1$ sind und dieses Polynom den Grad $q-1$ hat, handelt es sich hier um *alle* Nullstellen von $x^{q-1}-1$; in der Sprache der Algebra ist $\mathbb{F}_q$ daher der Zerfällungskörper des Polynoms $x^{q-1}-1$ über $\mathbb{F}_p$. Daraus folgt wegen der Eindeutigkeit des Zerfällungskörpers, daß alle Körper mit q Elementen isomorph sind.

c) Der Körper mit 256 Elementen

Für AES ist der Körper $\mathbb{F}_{256}$ von zentraler Bedeutung; da $256 = 2^8$ ist, handelt es sich hier um den Vektorraum $\mathbb{F}_2^8$. Die Addition ist sehr einfach: Da die Addition in $\mathbb{F}_2$ mit dem exklusiven Oder übereinstimmt, ist die Addition in $\mathbb{F}_{256}$ das bitweise exklusive Oder, eine Operation, die nicht nur in den gängigen CPUs, sondern auch in vielen Prozessoren für spezielle Anwendungen in der Signalverarbeitung als Grundoperation implementiert und somit sehr schnell ist.

Um eine Multiplikation zu definieren, brauchen wir ein irreduzibles Polynom vom Grad acht über $\mathbb{F}_2$. Da $\mathbb{F}_2$ ein sehr kleiner Körper und acht eine ziemlich kleine Zahl ist, hat zumindest ein Computer keinerlei Schwierigkeiten, alle diese Polynome zu bestimmen: Wie im vorigen Abschnitt erwähnt, sind das genau die irreduziblen Faktoren vom Grad acht in der Faktorisierung des Polynoms $X^{255} - 1$ über $\mathbb{F}_2$. Faktorisierung von Polynomen über Körpern von Primzahlordnung ist einer der Grundalgorithmen der Computeralgebra und geht auch noch bei sehr viel komplizierteren Polynomen sehr schnell; hier zeigt das Ergebnis, daß es dreißig Faktoren vom Grad acht gibt. Diese führen zwar alle auf denselben Körper, aber das praktische Rechnen in diesem Körper hängt natürlich stark von der Wahl des Polynoms ab. Insbesondere wird die Geschwindigkeit umso höher, je weniger Terme das Polynom hat.

Dreizehn der dreißig Polynome bestehen aus sieben nichtverschwindenden Termen, die restlichen siebzehn aus fünf; Wir wählen natürlich eines der letzteren. Alle diese Polynome haben, wie jedes Polynom vom Grad acht über $\mathbb{F}_2$, den führenden Term X^8; danach folgen vier weitere Terme. Bei der Reduktion modulo einem solchen Polynom $P = X^8 + \mathit{Rest}$ benutzt man, daß dann

$$X^8 \equiv \mathit{Rest}, \qquad X^9 \equiv X \cdot \mathit{Rest}, \quad \ldots$$

ist; dies wird umso häufiger mehrfach angewandt werden müssen, je höheren Grad die Terme in *Rest* haben. Am effizientesten kann man also rechnen, wenn das Polynom *Rest* den kleinstmöglichen Grad hat, und wenn zudem auch noch die hinteren Terme von *Rest* möglichst kleinen Grad haben. Inspektion der siebzehn Polynome mit fünf Termen zeigt, daß das Polynom

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

in dieser Hinsicht optimal ist.

Die genaue Festlegung für das Rechnen in $\mathbb{F}_{256} = \mathbb{F}_2^8$ für die Zwecke von AES ist folgende: Wir schreiben ein Byte als $(a_7, a_6, \ldots, a_0)$ und identifizieren es mit dem Polynom

$$a_7X^7 + a_6X^6 + \cdots + a_1X + a_0\,;$$

das Byte 0000 0010 entspricht also X.

Der Einfachheit halber schreiben wir Bytes meist als zweiziffrige Hexadezimalzahlen: Im betrachteten Beispiel wäre das 02_{hex}, und das Byte $\text{A5}_{\text{hex}} = 1010\,0101$ entspricht dem Polynom $X^7 + X^5 + X^2 + 1$.

Man beachte, daß trotz dieser Schreibweise die Addition und Multiplikation in $\mathbb{F}_{256}$ natürlich nichts mit der Addition und Multiplikation von Hexadezimalzahlen zu tun haben. Zwar ist $01_{\text{hex}} + 02_{\text{hex}} = 03_{\text{hex}}$, aber $01_{\text{hex}} + 01_{\text{hex}} = 00_{\text{hex}}$ und $05_{\text{hex}} + 04_{\text{hex}} = 01_{\text{hex}}$.

Zur Berechnung von $\text{A5}_{\text{hex}} \odot 01_{\text{hex}}$ müssen wir das Polynom

$$(X^7 + X^5 + X^2 + 1) \cdot X = X^8 + X^6 + X^3 + X$$

berechnen und modulo $m(X)$ reduzieren. Da

$$X^8 \bmod m(X) = X^4 + X^3 + X^2 + 1$$

ist (in $\mathbb{F}_2$ ist $-1 = 1$), ist dies

$$(X^4 + X^3 + X^2 + 1) + (X^6 + X^3 + X) = X^6 + X^4 + X^2 + X + 1\,.$$

Somit ist $A5_{hex} \odot 01_{hex} = 56_{hex}$.

Trotz der Wahl des optimalen Polynoms ist die Multiplikation also immer noch erheblich aufwendiger als die Addition.

§3: Spezifikation von Rijndael

a) Terminologie und Bezeichnungen

Rijndael arbeitet mit verschiedenen Blocklängen sowie auch verschiedenen Schlüssellängen: Beide können (unabhängig voneinander) alle durch 32 teilbaren Werte zwischen 128 und 256 annehmen. Wir schreiben die Blocklänge als $32N_b$ und die Schlüssellänge als $32N_k$; die Zahlen N_b und N_k liegen also jeweils zwischen vier und acht.

Der offizielle FIPS-Standard AES ist nicht wirklich *gleich* Rijndael; für AES ist nur die eine Blocklänge 128 normiert und nur die drei Schlüssellängen $128, 196$ und 256; hier ist also stets $N_b = 4$ und N_k kann die drei Werte $4, 6, 8$ annehmen.

Rijndael verschlüsselt somit einen Block von $32N_b$ Bit oder $4N_b$ Bytes, und genau wie bei DES geschieht dies sukzessive in mehreren Runden. In der Terminologie von Rijndael wird der Block in seinem jeweiligen Bearbeitungsstand als „Zustand“ bezeichnet; die einzelnen Runden finden also jeweils einen bestimmten Zustand vor und verändern diesen.

Die Anzahl der Runden wird als N_r bezeichnet; sie hängt von N_b und N_k ab gemäß der Gleichung $N_r = \max(N_b, N_k) + 6$.

b) Die Grundoperationen

Auch wenn Rijndael nicht mehr nach dem Prinzip eines FEISTEL-Netzwerks arbeitet, sind in den einzelnen Runden immer noch die klassischen SHANNONschen Prinzipien von Konfusion und Diffusion realisiert.

Für die Konfusion sind bei DES die verschiedenen S-Boxen zuständig; bei Rijndael gibt es nur eine einzige Funktion zu diesem Zweck; diese ist

definiert als Hintereinanderausführung der Bildung des (multiplikativen) Inversen in $\mathbb{F}_{256}$ mit einer über $\mathbb{F}_2$ affinen Abbildung von $\mathbb{F}_{256}$ nach $\mathbb{F}_{256}$.

Die Inversenbildung ist natürlich nur für $\mathbb{F}_{256} \setminus \{0\}$ erklärt; um trotzdem eine bijektive Abbildung von $\mathbb{F}_{256}$ nach $\mathbb{F}_{256}$ zu bekommen, nehmen wir die einzig mögliche Fortsetzung, bilden die Null also auf sich selbst ab. Auch wenn es algebraisch kompletter Unsinn ist, wollen wir zur Vermeidung von Fallunterscheidungen für die Definition von Rijndael die Kurzschreibweise $0^{-1} = 0$ verwenden mit der Interpretation, daß $x \mapsto x^{-1}$ die Fortsetzung der Inversenbildung zu einer bijektiven Abbildung von $\mathbb{F}_{256}$ nach $\mathbb{F}_{256}$ sein soll.

Diese Abbildung ist weder über $\mathbb{F}_{256}$ noch über $\mathbb{F}_2$ linear, und sie ist das einzige nichtlineare Element von Rijndael. Rechnerisch ist die Bestimmung von x^{-1} aufwendig, allerdings gibt es nur 256 mögliche Werte für x, die man vorausberechnen und in 256 Byte abspeichern kann; dieser Speicherbedarf dürfte selbst für Smartcards problemlos sein.

Diffusion wird durch eine Reihe von $\mathbb{F}_{256}$-linearen Abbildungen erzielt, die Operationen sind also im Gegensatz zu den Bit-Permutationen von DES allesamt auf Byte-Niveau definiert. Die Multiplikation von $\mathbb{F}_{256}$ sorgt dafür, daß trotzdem auch eine beträchtliche Diffusion innerhalb der Bytes stattfindet. Durch Tabellen gegebene Permutationen gibt es in AES überhaupt nicht; alle Diffusion wird durch Shift-Operationen sowie durch eine algebraische Operation realisiert.

Letztere arbeitet mit Wörtern von vier Byte, die wiederum mit Polynomen identifiziert werden, jetzt aber nicht, wie bei der Definition der Multiplikation in $\mathbb{F}_{256}$, mit Polynomen über $\mathbb{F}_2$, sondern solchen über $\mathbb{F}_{256}$. Wir schreiben die neuen Polynome daher zur besseren Unterscheidung in einer neuen Variablen Y und identifizieren $\mathbb{F}_{256}^4$ somit mit dem Vektorraum aller Polynome vom Grad höchstens drei in Y mit Koeffizienten aus $\mathbb{F}_{256}$, indem wir das Quadrupel (b_0, b_1, b_2, b_3) auffassen als Polynom

$$b_3Y^3 + b_2Y^2 + b_1Y + b_0 \quad \text{mit} \quad b_i \in \mathbb{F}_{256}\,.$$

Diese Polynome multiplizieren wir modulo dem Polynom $M = Y^4 + 1$.

Man beachte, daß dieses Polynom über $\mathbb{F}_2$ (und erst recht über $\mathbb{F}_{256}$) nicht irreduzibel ist: Da alle Binomialkoeffizienten außer dem ersten und dem letzten gerade sind, ist

$$(Y+1)^4 = Y^4 + 1 \bmod 2\,.$$

Mit der hier definierten Multiplikation wird $\mathbb{F}_{256}^4$ also nicht zu einem Körper. Rijndael verwendet diese Multiplikation allerdings nur für eine einzige lineare Abbildung von $\mathbb{F}_{256}^4$ nach $\mathbb{F}_{256}^4$, und diese ist gegeben durch Multiplikation mit dem Polynom

$$C = 03_{\text{hex}}Y^3 + Y^2 + Y + 02_{\text{hex}}\,.$$

An der Stelle $Y = 1$ ist

$$C = 03_{\text{hex}} + 1 + 1 + 02_{\text{hex}} = 03_{\text{hex}} + 02_{\text{hex}} = 1\,,$$

das Polynom ist also nicht durch $Y + 1$ teilbar (zur Erinnerung: über $\mathbb{F}_2$ wie auch $\mathbb{F}_{256}$ ist $Y + 1 = Y - 1$), und damit auch teilerfremd zu $Y^4 + 1 = (Y + 1)^4$. Damit hat zumindest dieses Polynom ein multiplikatives Inverses, das man mit dem erweiterten EUKLIDischen Algorithmus über $\mathbb{F}_{256}$ berechnen kann – auch wenn das von Hand ziemlich aufwendig ist. Im Maple-*worksheet* zu diesem Paragraphen findet man die detaillierte Rechnung mit Unterprogrammen für die Rechenoperationen von $\mathbb{F}_{256}$; als Ergebnis findet man dort das inverse Polynom

$$C' = 0\text{B}_{\text{hex}}Y^3 + 0\text{D}_{\text{hex}}Y^2 + 09_{\text{hex}}Y + 0\text{E}_{\text{hex}}\,.$$

Wer will, kann nachrechnen, daß $C \cdot C'$ modulo M gleich eins ist, allerdings erfordert bereits das recht viele Multiplikationen in $\mathbb{F}_{256}$.

c) Der Aufbau der Runden

Nach diesen Vorbereitungen können wir uns mit dem Aufbau der einzelnen Runden beschäftigen. Abgesehen von einer leichten Modifikation bei der letzten Runde besteht jede Runde aus denselben vier Schritten

1. Bytesubstitution
2. Zeilenshift
3. Spaltenmix
4. Addition des Rundenschlüssels

1.) Die Bytesubstitution: Dies ist der Konfusionsschritt; er operiert auf Bytes und wird auf jedes einzelne Byte des Zustands in derselben Weise angewendet; mit geeigneter Hardware kann dies also auch parallel erfolgen. Da es nur 256 mögliche Bit gibt, wird man diese Operation im allgemeinen über eine Tabelle implementieren; der Speicherbedarf von 256 Bit sollte auch auf einer Smartcard im allgemeinen problemlos sein,

Der erste Schritt ist, wie bereits erwähnt, die Inversenbildung im Körper $\mathbb{F}_{256}$; danach folgt eine Diffusion auf Bitniveau, indem $\mathbb{F}_{256}$ als affiner Raum $\mathbb{F}_2^8$ interpretiert wird und die affine Abbildung

$$\vec{x} \mapsto M\vec{x} + \vec{b}$$

mit

$$M = \begin{pmatrix} 1&0&0&0&1&1&1&1\\ 1&1&0&0&0&1&1&1\\ 1&1&1&0&0&0&1&1\\ 1&1&1&1&0&0&0&1\\ 1&1&1&1&1&0&0&0\\ 0&1&1&1&1&1&0&0\\ 0&0&1&1&1&1&1&1\\ 0&0&0&1&1&1&1&1 \end{pmatrix} \quad \text{und} \quad \vec{b} = \begin{pmatrix} 1\\1\\0\\0\\0\\1\\1\\0 \end{pmatrix}$$

angewandt wird. Insgesamt führt die Bytesubstitution ein Byte x also über in $Mx^{-1} + \vec{b}$.

Betrachtet man $\mathbb{F}_{256}$ nicht als affinen Raum, sondern als Raum der Polynome vom Grad höchstens sieben, kann man die affine Abbildung auch interpretieren als

$$P \mapsto (X^7 + X^6 + X^5 + X^4 + 1)P + (X^7 + X^6 + X^2 + X) \bmod X^8 + 1\,;$$

da $X^7 + X^6 + X^5 + X^4 + 1$ an der Stelle eins den Wert eins annimmt, ist dieses Polynom teilerfremd zu $X^8 + 1 = (X + 1)^8$, die Abbildung und damit die Matrix M sind also invertierbar – was sie natürlich auch sein müssen, damit die Chiffre entschlüsselbar ist.

Als dritte Alternative kann man die Abbildung auch durch ein Polynom über $\mathbb{F}_{256}$ beschreiben, denn da der Körper endlich ist findet sich zu *jeder*

Abbildung $\mathbb{F}_{256} \to \mathbb{F}_{256}$ ein Interpolationspolynom, das sie beschreibt. Im vorliegenden Fall ist dies das Polynom

$$\begin{aligned} &63_{\text{hex}} + 05_{\text{hex}} Z \quad + 09_{\text{hex}} Z^2 \; + \text{F9}_{\text{hex}} Z^4 \; + 25_{\text{hex}} Z^8 \\ &\quad + \text{F4}_{\text{hex}} Z^{16} + 01_{\text{hex}} Z^{32} + \text{B5}_{\text{hex}} Z^{64} + 8\text{F}_{\text{hex}} Z^{128} \,. \end{aligned}$$

Diese Abbildung wird angewandt auf das multiplikative Inverse eines Elements von $\mathbb{F}_{256}$. Im Körper $\mathbb{F}_{256}$ ist für jedes Element $z \neq 0$

$$z^{255} = 1 \quad \text{und} \quad (z^{-1})^n = z^{-n} = z^{255-n} \,,$$

wobei letztere Gleichung wegen unserer Konvention $0^{-1} = 0$ für *alle* z gilt. Damit können wir die Bytesubstitution insgesamt auch beschreiben durch das Polynom

$$\begin{aligned} &63 + 05_{\text{hex}} Z^{254} + 09_{\text{hex}} Z^{253} + \text{F9}_{\text{hex}} Z^{251} + 25_{\text{hex}} Z^{247} \\ &\quad + \text{F4}_{\text{hex}} Z^{239} + 01_{\text{hex}} Z^{223} + \text{B5}_{\text{hex}} Z^{191} + 8\text{F}_{\text{hex}} Z^{127} \,. \end{aligned}$$

Die Umkehrabbildung der Bytesubstitution hat als affine Abbildung die Form

$$\vec{y} \mapsto M^{-1}\vec{y} + M^{-1}\vec{b} \,;$$

dabei ist

$$M^{-1} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \quad \text{und} \quad M^{-1}\vec{b} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} .$$

Die Inversenabbildung ist natürlich (auch mit der Konvention, daß die Null auf sich selbst abgebildet wird) zu sich selbst invers; die Bytesubstitution wird also rückgängig gemacht, indem man ein Byte $\vec{y}$ zunächst auf $A^{-1}\vec{y} + A^{-1}\vec{b}$ abbildet und dann die erweiterte Inversenabbildung von $\mathbb{F}_{256}$ anwendet.

2.) Die Zeilenshifts: Der Diffusionsschritt ist zweigeteilt: Der zu verschlüsselnde Block ist ein Vektor aus $4N_b$ Bytes; er wird umgeschrieben in eine Byte-Matrix mit vier Zeilen und N_b-Spalten, die spaltenweise aufgefüllt wird. Wird die Matrix als $A = (a_{ij})$ bezeichnet, wobei der Zeilenindex i von 0 bis 3 und der Spaltenindex j von 0 bis $N_b - 1$ geht, ist der ursprüngliche Vektor also (in Zeilenschreibweise)

$$(a_{00}, a_{10}, a_{20}, a_{30}, a_{01}, a_{11}, \ldots, a_{N_b-2,3}, a_{N_b-1,0}, \ldots, a_{N_b-1,3}) \,.$$

Indiziert man die Komponenten des Vektors durch einen Index n linear von 0 bis $4N_b - 1$, ist also

$$i = n \bmod 4, \quad j = \left[\frac{n}{4}\right] \quad \text{und} \quad n = i + 4j \,.$$

Der erste Diffusionsschritt ist eine zyklische Verschiebung der Zeilen von A: Die 0-te Zeile wird überhaupt nicht verschoben und die erste um eine Stelle; die Richtung der Verschiebung ist, wie auch stets im folgenden, nach links. Bei den beiden unteren Reihen hängt die Weite der Verschiebung von N_b ab: Für $N_b \neq 8$ wird die zweite Zeile um zwei Stellen verschoben, für $N_b = 8$ und drei. Die dritte Zeile wird für $N_b \leq 6$ um drei Stellen, für $N_b \leq 7$ um drei Stellen verschoben. Mit entsprechender Hardware können die Verschiebungen der einzelnen Zeilen natürlich parallel durchgeführt werden.

3.) Der Spaltenmix: Auf Spaltenebene ist die Diffusion aufwendiger: Ein Spaltenvektor ist ein Element von $\mathbb{F}_{256}^4$. Diesen Vektorraum hatten wir oben mit den kubischen Polynomen über $\mathbb{F}_{256}$ identifiziert und dort eine Multiplikation eingeführt über die Polynommultiplikation modulo Y^4+1. Genau dies verwendet der Spaltenmix, indem jeder Spaltenvektor mit dem Polynom

$$C = 03_{\text{hex}} Y^3 + Y^2 + Y + 02_{\text{hex}}$$

multipliziert wird. Wegen der einfachen Struktur des Polynoms $Y^4 + 1$ ist dies eine sehr einfache lineare Abbildung mit Matrix

$$\begin{pmatrix} 02_{\text{hex}} & 03_{\text{hex}} & 01_{\text{hex}} & 01_{\text{hex}} \\ 01_{\text{hex}} & 02_{\text{hex}} & 03_{\text{hex}} & 01_{\text{hex}} \\ 01_{\text{hex}} & 01_{\text{hex}} & 02_{\text{hex}} & 03_{\text{hex}} \\ 03_{\text{hex}} & 01_{\text{hex}} & 01_{\text{hex}} & 02_{\text{hex}} \end{pmatrix}$$

bezüglich der Basis $\{1, Y, Y^2, Y^3\}$. Diese kann für die N_b Spalten parallel durchgeführt werden.

Im Vergleich zum Zeilenshift ist diese Operation relativ aufwendig, da mehrere Multiplikationen in $\mathbb{F}_{256}$ durchgeführt werden müssen. In der letzten Runde, in der dieser Diffusionsschritt keiner anschließenden Konfusion mehr unterworfen wird und somit keinen großen Sicherheitsgewinn mehr bietet, wird daher auf diesen Schritt verzichtet.

Wie wir bereits gesehen haben, ist C modulo $Y^4 + 1$ invertierbar mit inversem Polynom $C' = 0\mathrm{B}_{\mathrm{hex}}Y^3 + 0\mathrm{D}_{\mathrm{hex}}Y^2 + 09_{\mathrm{hex}}Y + 0\mathrm{E}_{\mathrm{hex}}$; Multiplikation mit C' ist als lineare Abbildung gegeben durch die Matrix

$$\begin{pmatrix} 0\mathrm{E}_{\mathrm{hex}} & 0\mathrm{B}_{\mathrm{hex}} & 0\mathrm{D}_{\mathrm{hex}} & 09_{\mathrm{hex}} \\ 09_{\mathrm{hex}} & 0\mathrm{E}_{\mathrm{hex}} & 0\mathrm{B}_{\mathrm{hex}} & 0\mathrm{D}_{\mathrm{hex}} \\ 0\mathrm{D}_{\mathrm{hex}} & 09_{\mathrm{hex}} & 0\mathrm{E}_{\mathrm{hex}} & 0\mathrm{B}_{\mathrm{hex}} \\ 0\mathrm{B}_{\mathrm{hex}} & 0\mathrm{D}_{\mathrm{hex}} & 09_{\mathrm{hex}} & 0\mathrm{E}_{\mathrm{hex}} \end{pmatrix};$$

als Übung für das Rechnen in $\mathbb{F}_{256}$ sollte man zumindest für einige Einträge nachrechnen, daß das Produkt dieser beiden Matrizen über $\mathbb{F}_{256}$ gleich der Einheitsmatrix ist.

4.) Schlüsselexpansion und Rundenschlüssel: Die bisherigen drei Schritte sorgten für Konfusion und Diffusion; sie sind aber für jeden, der das grundsätzliche Verfahren kennt, leicht rückgängig zu machen. Das ist nicht weiter verwunderlich, denn bislang haben wir ja den *Schlüssel* noch nicht ins Spiel gebracht, an dem bei einem normierten Standard wie AES die ganze Sicherheit des Verfahrens hängt.

Der vierte Schritt jeder Runde ist genau wie bei DES eine einfache Addition des Rundenschlüssels; die Addition ist dabei die gewöhnliche Vektoraddition in $\mathbb{F}_2^{32N_n}$, also das logische XOR. Sicherheitsrelevant ist, genau wie bei DES, die Berechnung der Rundenschlüssel aus dem vorgegebenen Schlüssel; der Schlüssel mit seinen $4N_k$ Bytes muß so aufgebläht werden auf $N_r \times 4N_b$ Schlüsselbytes, daß ein Kryptanalytiker aus einem oder auch einigen irgendwie ermittelten Rundenschlüsseln nicht auf den Gesamtschlüssel schließen kann.

Rijndael verwendet dazu im wesentlichen dieselben Techniken wie bei den Verschlüsselungsschritten: Das erweiterte Schlüsselfeld wird auf-

gefaßt als eine Folge von Wörtern W_i mit einer Länge von vier Byte oder 32 Bit. Die ersten Wörter W_0 bis W_{N_k-1} sind der eigentliche Schlüssel des Verfahrens, der Rest wird rekursiv daraus berechnet. Da die ersten N_b Worte nicht als Rundenschlüssel verwendet werden, sondern vor der ersten Runde zum Klartext addiert werden, hat das gesamte Schlüsselfeld eine Länge von $N_b(N_r + 1)$ Worten.

Für $i \geq N_k$ wird W_i sukzessive aus seinem unmittelbaren Vorgänger W_{i-1} und dem N_k Wörter zurückliegenden Vorgänger W_{i-N_k} berechnet:

- Falls i nicht durch N_k teilbar ist und im Falle $N_k > 6$ auch nicht durch vier, ist $W_i = W_{i-N_k} \oplus W_{i-1}$, wobei $\oplus$ die Vektorraumaddition in $\mathbb{F}_2^{32} = \mathbb{F}_{256}^8$ bezeichnet, also das logische XOR für 32-Bit-Wörter.
- Falls i durch N_k teilbar ist, wird W_{i-1} zunächst zyklisch um eins nach links verschoben. Dann wird auf jedes Byte des so entstandenen Worts die Bytesubstitution aus Absatz *1)* angewendet, und das Ergebnis wird mit $\oplus$ zu W_{i-N_k} addiert. Dazu wird noch eine Rundenkonstante addiert, deren erstes Byte dasjenige Element von $\mathbb{F}_{256}$ ist, das der Potenz X^{i/N_k} modulo dem die Multiplikation definierenden Polynom entspricht und dessen weitere drei Bytes alle Null sind.
- Falls $N_k > 6$ und $i \equiv 4 \bmod N_k$, wird die Bytesubstitution auf W_{i-1} angewendet und das Ergebnis zu W_{i-N_k} addiert.

d) Gesamtablauf von Rijndael

Nachdem nun alle Einzelheiten spezifiziert sind, kann der Gesamtablauf von Rijndael leicht angegeben werden:

1. Die ersten N_b Worte des Schlüsselfelds werden zum Klartext addiert.
2. $N_r - 1$ Runden werden gemäß obiger Beschreibung durchgeführt.
3. Die N_r-te Runde wird entsprechend ausgeführt, aber ohne den Spaltenmix.

e) Geschwindigkeitsoptimierung

So wie Rijndael spezifiziert ist, ist vor allem die Bytesubstitution sehr

langsam; aber wie bereits erwähnt, läßt sie sich leicht über eine Tabelle implementieren.

Ein weiterer rechnerisch aufwendiger Bestandteil von Rijndael sind die Multiplikationen im Körper $\mathbb{F}_{256}$. Eine vorausberechnete Multiplikationstabelle würde $256 \times 256 = 64 \times 1024$ Byte oder 64 Kilobyte in Anspruch nehmen, was erstens etwas viel und zweitens auch nicht optimal ist: Die Multiplikation von $\mathbb{F}_{256}$ wird nur beim Spaltenmix benötigt, und da das hierzu verwendete Polynom C nur 01_{hex}, 02_{hex} und 03_{hex} als Koeffizienten hat, reicht es für die Verschlüsselung, wenn man die Produkte mit 02_{hex} und 03_{hex} bilden kann. Der jeweils andere Faktor des Produkts ist stets das Ergebnis einer Bytesubstitution, man spart also Rechenzeit, wenn man für jedes Byte das Ergebnis der Bytesubstitution sowie dessen Produkt mit 02_{hex} und mit 03_{hex} abspeichert; mit 768 Byte für Tabellen kann man also zur Laufzeit auf alle Multiplikationen und Divisionen in $\mathbb{F}_{256}$ verzichten.

Mit vier Kilobyte Tabellenplatz läßt sich sogar die gesamte Rundentransformation auf $4N_b$ XOR-Operationen auf 32-Bit-Wörtern zurückführen: Man speichere für jedes Byte $a \in \mathbb{F}_{256}$ die vier 32-Bit-Wörter

$$T_0(a) = \begin{pmatrix} 02_{\text{hex}} \odot S(a) \\ S(a) \\ S(a) \\ 03_{\text{hex}} \odot S(a) \end{pmatrix}, \quad T_1(a) = \begin{pmatrix} 03_{\text{hex}} \odot S(a) \\ 02_{\text{hex}} \odot S(a) \\ S(a) \\ S(a) \end{pmatrix},$$

$$T_2(a) = \begin{pmatrix} S(a) \\ 03_{\text{hex}} \odot S(a) \\ 02_{\text{hex}} \odot S(a) \\ S(a) \end{pmatrix} \quad \text{und} \quad T_3(a) = \begin{pmatrix} S(a) \\ S(a) \\ 03_{\text{hex}} \odot S(a) \\ 02_{\text{hex}} \odot S(a) \end{pmatrix},$$

wobei $S\colon \mathbb{F}_{256} \to F_{256}$ die Bytesubstitution ist.

Ist dann (bei Matrizenschreibweise) $\vec{b}_j$ der j-te Spaltenvektor des Zustands zu Beginn einer Runde, $\vec{e}_j$ der entsprechende Vektor am Ende der Runde und $\vec{k}_j$ der j-te Spaltenvektor des Rundenschlüssels, so ist

$$\vec{e}_j = T_0(b_{0j}) \oplus T_1(b_{1,j\ominus 1}) \oplus T_2(b_{2,j\ominus c_2}) \oplus T_3(b_{3,j\ominus c_3}) \oplus \vec{k}_j\ ;$$

dabei stehen c_2 und c_3 die Beträge, um die die zweite und dritte Zeile

verschoben werden, d.h.

$$c_2 = \begin{cases} 2 & \text{für } N_b < 8 \\ 3 & \text{für } N_b = 8 \end{cases} \quad \text{und} \quad c_3 = \begin{cases} 2 & \text{für } N_b < 7 \\ 3 & \text{für } N_b \geq 7 \end{cases},$$

und $\ominus$ steht für die Subtraktion modulo N_b, wie man sie für die Zeilenshifts braucht. Man beachte, daß auch diese Rechnung bei entsprechender Hardware für alle N_b Spalten parallel ausgeführt werden kann.

Da sich auch die sämtlichen Rundenschlüssel mit einem Speicheraufwand von weniger als einem halben Kilobyte vorausberechnen lassen, läßt sich die Verschlüsselung mittels Rijndael also auch auf einem Standard-PC sehr schnell durchführen.

Bei anderen Implementierungen, bei denen es Probleme mit dem Speicherplatz gibt, kann man auf Kosten einer höheren Rechenzeit mit sehr viel weniger Speicher auskommen. Eine relativ billige Art und Weise, wie man bei 32-Bit-Prozessoren drei Kilobyte einsparen kann, folgt beispielsweise aus der Beobachtung, daß die verschiedenen $T_i(a)$ durch zyklische Verschiebung auseinander entstehen. Somit reicht es, die $T_0(a)$ abzuspeichern, und indem man analog zum HORNER-Schema rechnet, liegt der zusätzliche Rechenaufwand nur bei drei zyklischen Vertauschungen pro Spalte und pro Runde: Ist $\mathcal{Z}$ die zyklische Linksverschiebung um ein Byte, so ist

$$\vec{e}_j = \vec{k}_j \oplus T_0(b_{0j}) \oplus \mathcal{Z}\Big(T_0(b_{1,j-1}) \oplus \mathcal{Z}\Big(T_0(b_{2,j-c_2}) \oplus \mathcal{Z}\big(T_0(b_{3,j-c_3})\big)\Big)\Big)\ .$$

Für die Entschlüsselung braucht man natürlich entsprechende Tabellen; hier werden die Produkte mit $0B_{\text{hex}}$, $0D_{\text{hex}}$, 09_{hex} und $0E_{\text{hex}}$ benötigt.

Das Arbeiten mit Tabellen kann übrigens unter Sicherheitsaspekten vorteilhaft sein gegenüber direktem Rechnen: Ein Gegner, der den Stromverbrauch der Rechnung kontinuierlich messen kann (z.B. weil eine Smartcard ihren Strom aus seinem Lesegerät bezieht), kann daraus eventuell Rückschlüsse auf Klartext und/oder Schlüssel ziehen, da etwa eine Ziffer eins in einer Multiplikation mehr Aufwand erfordert als eine Ziffer null. Bei Multiplikation via Tabellen ist der Stromverbrauch jedoch unabhängig von den Faktoren, da stets nur ein Tabellenwert gelesen und weiterverwendet wird.

§4: Angriffe auf Rijndael

Wie jede andere Blockchiffre bietet auch Rijndael keinerlei Sicherheit gegen einen BAYESschen Gegner – zumindest dann nicht, wenn man (wie praktisch immer) redundante Information verschlüsselt. Da allerdings bereits die einfachste Variante von Rijndael mit einer Schlüssellänge von 128 arbeitet, ist das Durchprobieren aller Schlüssel für real existierende Gegner mit heutiger Technologie unrealistisch: Gegenüber DES mit seiner Schlüssellänge 56 steigt der Aufwand immerhin um den Faktor

$$2^{128-56} = 2^{72} = 4\,722\,366\,482\,869\,645\,213\,696\,,$$

also um mehr als zwanzig Größenordnungen. Ein Gegner muß daher, um erfolgreich zu sein, Methoden finden, die mit deutlich geringerem Aufwand auskommen. Da er in der Wahl seiner Methoden frei ist, können wir nie wirklich wissen, wie er arbeitet; alle Sicherheitsaussagen beruhen nur darauf, daß keine realistische Attacke *bekannt* ist, obwohl im Verlauf des Begutachtungsprozesses international führende Experten mehrere Jahre lang danach gesucht haben. Natürlich geht die Suche auch jetzt noch weiter, und in der Tat sind inzwischen auch neue Ansätze aufgetaucht, die bei der Wahl von Rijndael noch nicht bekannt waren.

Natürlich ist das Design von Rijndael so gewählt, daß der Algorithmus resistent ist gegen differentielle und lineare Kryptanalyse; beides war schließlich zum Zeitpunkt seines Entwurfs wohlbekannt und gut verstanden. Auch ist die grundsätzliche Struktur von Rijndael nicht neu: Es ist zwar kein FEISTEL-Netzwerk mehr, aber er kommt aus einer Familie von Kryptoverfahren um den Algorithmus Square, der bereits seit einiger Zeit kryptanalysiert worden war. Insbesondere hatten DAEMEN und RIJMEN die sogenannte Square attack entwickelt, die mit speziell gewählten Klartexten den letzten Rundenschlüssel angreift: Verschlüsselt werden 256 Blöcke, die in allen Bytes mit einer Ausnahme übereinstimmen; das variable Byte nimmt alle 256 möglichen Werte an.

Verfolgt man diese Blöcke geschickt durch den Algorithmus, läßt sich mit hinreichend vielen Gruppen solcher Blöcke auch ein auf sechs Runden reduzierter Rijndael kryptanalysieren; da dies den beiden Designern bewußt war, ist die Rundenzahl entsprechend größer gewählt.

Ein Kritikpunkt an Rijndael war seine relativ einfache algebraische Struktur, die zwar die Implementierung erleichtert und beschleunigt, aber eventuell auch Sicherheitsprobleme aufwerfen könnte.

Im Mai 2001 gelang es NIELS FERGUSON, RICHARD SCHROEPPEL und DOUG WHITING, die allesamt in der Wirtschaft (bei verschiedenen Unternehmen) über Sicherheitsfragen arbeiten, Rijndael in einer geschlossenen Formel darzustellen; für die 128 Bit Version hat diese Formel $2^{50} \approx 10^{15}$ Terme, für 256 Bit sind es $2^{70} \approx 10^{21}$. Obwohl die Formel natürlich hoch strukturiert ist, ist allerdings völlig unklar, ob dieser Ansatz je zu einer Bedrohung für Rijndael werden kann; schließlich ist eine Formel nur selten die beste Möglichkeit für den Umgang mit einer Funktion. Die Originalarbeit ist zu finden unter
`www.xs4all.nl/~vorpal/pubs/rdalgeq.html` .

Potentiell gefährlicher ist ein Ansatz von NICOLAS COURTOIS und JOSEF PIEPRZYK, deren XSL-Attacke (`eprint.iacr.org/2002/044/`) das Knacken von Rijndael übersetzt in die Lösung eines überbestimmten nichtlinearen Gleichungssystems. Ob dieser Ansatz langfristig zu einem Angriff führt, der schneller ist als das Durchprobieren aller Schlüssel, ist im Augenblick noch nicht abzuschätzen.

§5: Literatur

Als viel benutzter Standard ist AES natürlich in allen neueren Lehrbüchern der Kryptologie zu finden, insbesondere auch in dem für die gesamte Vorlesung angegebenen Buch von BUCHMANN. Speziell mit AES beschäft sich das Buch der beiden Entwickler von AES

JOAN DAEMEN, VINCENT RIJMEN: *The Design of Rijndael: AES – the Advanced Encryption Standard*, Springer, 2002

Eine neuere Darstellung, die auch auf seither diskutierte Angriffsmöglichkeiten eingeht, ist

CARLOS CID, SEAN MURPHY, MATTHEW ROBSHAW: *Algebraic Aspects of the Advanced Encryption Standard*, Springer, 2006

Kapitel 7
Kryptographisch sichere Hashverfahren

§1: Nochmals elektronische Unterschriften

Elektronische Unterschriften, so wie wir sie bislang kennen, sind ungefähr so aufwendig wie die Verschlüsselung eines Dokuments mit einem asymmetrischen Verfahren. Da niemand ein längeres Dokument blockweise mit einem asymmetrischen Verfahren verschlüsselt, sollte klar sein, daß es auch niemand mit einem solchen Verfahren unterzeichnet. Genauso, wie man bei der Verschlüsselung das asymmetrische Verfahren nur zum Schlüsselaustausch verwendet, möchte man auch beim Unterschreiben das asymmetrische Verfahren nur ein einziges Mal anwenden müssen. Dies wird dadurch möglich, daß man nicht die Nachricht selbst unterschreibt, sondern nur einen daraus berechneten geeigneten Hashwert.

Allgemein ist ein Hashwert eine Art Prüfsumme, die von der gesamten Nachricht abhängt; wohlbekannt sind etwa die Prüfziffern am Ende eines Artikelcodes oder einer ISBN. Bei der Europäischen Artikelnummer EAN, die zumindest als Strichcode auf praktisch allen Waren zu finden ist, wird die letzte Ziffer so gewählt, daß eine durch zehn teilbare Zahl entsteht, wenn man die Summe bildet aus einmal der ersten, dreimal der zweiten, einmal der dritten, dreimal der vierten Ziffer *usw.* Bei der Internationalen Standardbuchnummer ISBN wird die erste der zehn Ziffern mit zehn multipliziert, die zweite mit neun *usw.* und die Summe muß durch elf teilbar sein. Falls man dazu eine zehnte „Ziffer“ zehn braucht, wird diese als „X“ geschrieben. Ähnlich aufgebaut sind auch die Hashfunktionen, die Informatiker in Suchalgorithmen verwenden.

Solche Prüfsummen sind für die Fehlererkennung bei Scannerkassen oder bei Buchbestellungen gut geeignet, da sie die typischen Flüchtigkeitsfehler mit hoher Wahrscheinlichkeit erkennen. Für kryptographische Anwendungen sind sind sie allerdings völlig unbrauchbar, denn hier haben wir es mit intelligenten Gegnern zu tun, die bereit sind, einen großen Aufwand zu treiben um zu einem unterschriebenen Hashwert eine zweite Nachricht mit dem gleichen Hashwert zu konstruieren um dann zu behaupten, die Unterschrift gehöre zu dieser zweiten Nachricht.

Von einer kryptographisch brauchbaren Hashfunktion müssen wir daher fordern, daß es rechnerisch nicht mit vertretbarem Aufwand möglich sein darf, zu einem gegebenen Hashwert einen Text zu konstruieren. Die Hashfunktion muß also, genau wie ein symmetrisches Kryptoverfahren, mit Konfusion und Diffusion arbeiten, so daß möglichst jedes Bit des Texts jedes Bit des Hashwerts in einer schwer durchschaubaren Weise beeinflußt.

Dies legt es nahe, den Hashwert über ein symmetrisches Kryptoverfahren zu berechnen: Man nimmt etwa den ersten Block als Schlüssel, verschlüsselt damit den zweiten, nimmt das Ergebnis als Schlüssel zur Verschlüsselung des dritten und so weiter; die Verschlüsselung des letzten Blocks ist dann der Hashwert.

Ein solches Verfahren ist allerdings gleichzeitig zu aufwendig und zu unsicher: Da jeder Block der Nachricht zum Endergebnis beiträgt, erhalten wir automatisch eine deutlich Reduktion der Redundanz; wir brauchen daher nicht so viele oder nicht so aufwendige Runden pro Block wie bei der Verschlüsselung eines einzelnen Blocks.

Gleichzeitig wäre bei einer solchen Vorgehensweise die Sicherheit drastisch reduziert, wenn wie der nächste Paragraph zeigen wird, brauchen wir für Hashverfahren bei gleicher Sicherheit eine doppelt so große Blocklänge wie bei Verschlüsselungsverfahren.

§2: Das Geburtstagsparadoxon

Der Grund dafür ist das sogenannte „Geburtstagsparadoxon“: Angenommen, in einem Raum befinden sich n Personen. Wie groß ist die

Wahrscheinlichkeit dafür, daß zwei davon am gleichen Tag Geburtstag haben?

Um diese Frage wirklich beantworten zu können, müßte man die (recht inhomogene) Verteilung der Geburtstage über das Jahr kennen; wir beschränken uns stattdessen auf ein grob vereinfachtes Modell ohne Schaltjahre mit 365 gleich wahrscheinlichen Geburtstagen. Dann ist die Wahrscheinlichkeit dafür, daß von n Personen keine zwei am gleichen Tag Geburtstage haben,

$$\prod_{k=0}^{n-1} \left(1 - \frac{k}{365}\right) ,$$

denn für eine Person ist das überhaupt keine Bedingung, und jede weitere Person muß die Geburtstage der schon betrachteten Personen vermeiden. (Da der Faktor mit $k = 365$ verschwindet, wird die Wahrscheinlichkeit für $n > 365$ zu null, wie es nach dem DIRICHLETschen Schubfachprinzip auch sein muß.)

Nachrechnen ergibt für $n = 23$ ungefähr den Wert 0,4927; bei 23 Personen liegt also die Wahrscheinlichkeit für zwei gleiche Geburtstage bei 50,7%. Tatsächlich dürfte sie noch deutlich höher liegen, denn bei Geburtstagen ist die Annahme einer Gleichverteilung sicherlich falsch.

Bei einer guten Hashfunktion allerdings sollten die Hashwerte in sehr guter Näherung gleichverteilt sein; falls es N mögliche Hashwerte gibt, liegt die Wahrscheinlichkeit dafür, daß unter n Nachrichten zwei zum selben führen daher bei

$$p_n = \prod_{k=0}^{n-1} \left(1 - \frac{k}{N}\right) .$$

Da wir uns für große Werte von N interessieren, können wir davon ausgehen, daß

$$\left(1 - \frac{1}{N}\right)^N \approx e \quad \text{und} \quad \left(1 - \frac{1}{N}\right) \approx e^{-1/N}$$

ist; für nicht zu große Werte von k ist dann auch

$$\left(1 - \frac{k}{N}\right) \approx e^{-k/N}$$

und für nicht zu große Werte von n gilt

$$p_n = \prod_{k=0}^{n-1} \left(1 - \frac{k}{N}\right) \approx \prod_{k=0}^{n-1} e^{-k/N} = e^{-\frac{1}{N}\sum_{k=0}^{n-1} k} = e^{-\frac{n(n-1)}{2N}} .$$

Für $N = 365$ etwa ergibt dies den Näherungswert $p_{23} \approx 0{,}499998$ für den korrekten Wert 0,4927.

Wenn wir im Exponenten noch den Term $n(n-1)$ durch n^2 approximieren, können wir abschätzen, für welches n die Wahrscheinlichkeit p_n einen vorgegebenen Wert erreicht:

$$e^{-\frac{n^2}{2N}} = p \iff \frac{n^2}{2N} = -\ln p \iff n = \sqrt{-2N \ln p} .$$

Damit liegt p_n bei etwa 50%, falls $n \approx \sqrt{2N \ln 2} \approx 1{,}177\sqrt{N}$ ist; für $N = 365$ ergibt dies die immer noch recht gute Näherung 22,494.

Für $p = 1/1000$ ergibt sich $n \approx 3{,}717\sqrt{N}$, für $p = 999/1000$ entsprechend $n \approx 0{,}0447\sqrt{N}$. Die Wahrscheinlichkeit dafür, daß es unter n Nachrichten zwei mit demselben Hashwert gibt, wechselt also bei der Größenordnung $n \approx \sqrt{N}$ ziemlich schnell von sehr unwahrscheinlich zu sehr wahrscheinlich.

Damit ist klar, daß bei einem kryptographisch brauchbares Hashverfahren die Zahl N der möglichen Hashwerte so groß sein muß, daß die Erzeugung von $\sqrt{N}$ verschiedenen Nachrichten rechnerisch unmöglich ist. Ansonsten könnte nämlich ein Gegner zwei verschiedene Texte a und b erzeugen, von denen a so ist, daß ihn das Opfer unterschreibt, b aber für dieses äußerst nachteilig wäre. Dazu könnte er jeweils etwa $\sqrt{N}$ sinngleiche Modifikationen a_i und b_j erzeugen, indem er beispielsweise unabhängig voneinander an jedem Zeilenenden entweder ein Leerzeichen einfügt oder auch nicht, was bei z Zeilen bereits 2^z Varianten ergibt; genauso könnte er Kombinationen aus Leerzeichen und Rücktaste einfügen oder auch nicht, Tabulatoren durch Leerzeichen ersetzen oder auch nicht, und so weiter. Wie wir gerade gesehen haben, hätte er dann eine gute Chance, daß ein a_i denselben Hashwert hätte wie ein b_j; jede Unterschrift unter a_i wäre gleichzeitig eine unter b_j.

Die Schlüssellänge K eines symmetrischen Kryptoverfahrens wird so gewählt, daß die 2^K Schlüssel nicht mit realistischem Aufwand durchprobiert werden können. Bei einem kryptographisch sicheren Hashverfahren muß dementsprechend die Länge L des Hashwerts so gewählt werden, daß niemand mit realistischem Aufwand $\sqrt{2^L} = 2^{L/2}$ Nachrichten erzeugen kann, d.h. bei gleichen Sicherheitsanforderungen muß ein Hashwert etwa doppelt so lang sein wie ein Schlüssel eines symmetrischen Kryptosystems. Da AES mit Schlüssellängen 128, 192 und 256 arbeitet, sollte man also entsprechend mit Hashwerten der Länge 256, 384 und 512 arbeiten; Hashwerte mit nur 128 Bit sind genau wie Kryptoverfahren mit 64 Bit-Schlüsseln heute nicht mehr sicher.

§ 3: Die Familie der SHA-Algorithmen

Da es mit kryptographisch sicheren Hashfunktionen deutlich weniger Erfahrung gibt als mit Verschlüsselung, liest sich die bisherige Geschichte solcher Funktionen eher enttäuschend: Zu den meisten Verfahren wurden über kurz oder lang Angriffe gefunden.

In der alltäglichen Praxis wurden bis vor kurzer Zeit hauptsächlich zwei Hashverfahren verwendet: RIPEMD-160 und SHA-1, die beide mit 160 Bit Hashwerten arbeiten. In der Tat gibt es eine ganze Reihe chipkartenbasierter Systeme für elektronische Unterschriften, die nur mit Hashwerten dieser Länge arbeiten konnten. Die Sicherheit solcher Unterschriften entsprach nur der eines Kryptoverfahrens mit Schlüssellänge 80 war somit nach heutigen Anforderungen recht gering.

Von daher sollte es niemanden verwundern, daß in den letzten Jahren zunehmend Ansätze für Kollisionsattacken gegen die beiden Verfahren publiziert wurden und daß die Bundesnetzagentur nun längere Hashwerte vorschreibt. Genau wie vor einigen Jahren bei der „Schallgrenze“ 1024 Bit für RSA stieß sie auch hier wieder auf erbitterten Widerstand einiger Anwender; nach den Empfehlungen für 2010 sind nun aber für neue Unterschriften nur noch Verfahren erlaubt, die Hashwerte mit Mindestlänge 224 liefern; das etwa der Sicherheit von Triple-DES. Die einzigen dafür zugelassenen Verfahren sind derzeit die entsprechenden

Algorithmen aus der SHA-Familie, mit denen wir uns daher in diesem Paragraphen beschäftigen wollen.

Der *Secure Hash Algorithm* SHA wurde im Januar 1992 veröffentlicht und am 11. Mai 1993 als amerikanischer Standard FIPS 180 verkündet. Wegen einer (nie publizierten) „technischen Schwäche“ musste auch dieser Algorithmus alsbald nachgebessert werden; am 11. Juli 1994 wurde die Modifikation SHA-1 als Nachfolgestandard FIPS 180-1 eingesetzt. Dessen Nachfolger FIPS 180-2 vom 1. August 2002 ließ SHA-1 unangetastet, fügte aber drei neue, ähnlich aufgebaute Algorithmen SHA-256, SHA-384 und SHA-512 mit längeren Hashwerten dazu. In einem Zusatz vom 15. Februar 2004 wurde schließlich auch noch eine Variante SHA-224 normiert. Am 15. März 2006 wurden die amerikanischen Bundesbehörden angewiesen, künftig nur noch die neueren Versionen mit mindestens 224 Bit zu benutzen. Die derzeitig gültige Version FIPS 180-3 vom Oktober 2008 stellt im wesentlich nur bekanntes Material in anderer Weise zusammen.

Die fünf Algorithmen sind sehr ähnlich aufgebaut; SHA-1, SHA-224 und SHA-256 arbeiten mit 32-Bit-Wörtern und Blöcken der Länge 512, bei SHA-384 und SHA-512 verdoppeln sich diese Längen. Die Nachrichten, die SHA-1 bis SHA-256 verarbeiten können, müssen kürzer sein als 2^{64} Bit, also etwa zwei Millionen Terabyte; für SHA-384 und SHA-512 liegt die Grenze bei 2^{128} Bit. Da die gesamte in unserem Universum enthaltene Information nach Schätzungen der Physiker bei etwa 10^{120} Bit liegt, dürfte dies für alle praktischen Zwecke ausreichen.

Die Maximallänge spielt eine Rolle bei der Vorverarbeitung der Nachricht: Ist M die zu verarbeitende Nachricht, bestehend aus ℓ Bit, so wird am Ende ein Bit „1“ eingefügt, dann k Nullen und schließlich noch die Zahl ℓ als Block von 64 *bzw.* 128 Bit. Die Zahl k wird als kleinste nichtnegative ganze Zahl gewählt, für die $\ell + 1 + k + 64$ durch 512 teilbar ist für SHA-1, SHA-224 und SHA-256 *bzw.* für die $\ell + 1 + k + 128$ durch 1024 teilbar ist für SHA-384 und SHA-512, so daß in jedem Fall die Länge der Nachricht ein ganzzahliges Vielfaches der Blocklänge ist.

Damit läßt sich die Nachricht nun aufteilen in Blöcke $M^{(1)}, M^{(2)}, \ldots$;

jeder dieser Blöcke wiederum wird aufgeteilt in Wörter $M_0^{(i)}, \ldots, M_{15}^{(i)}$. (Man beachte, daß bei jedem der fünf Algorithmen die Blocklänge gleich der 16-fachen Wortlänge ist.)

Jeder der Algorithmen startet mit seinem eigenen Anfangshashwert. Bei SHA-1 besteht dieser aus den willkürlich *(?)* festgelegten fünf (hexadezimal angegebenen) Wörtern

$$H_0^{(0)} = 67452301, \quad H_1^{(0)} = \texttt{EFCDAB89}, \quad H_2^{(0)} = 98\texttt{BADCFE},$$
$$H_3^{(0)} = 10325476, \quad H_4^{(0)} = \texttt{C3D2E1F0}\,,$$

ähnlich auch bei SHA-224 mit

$$H_0^{(0)} = \texttt{C1059ED8}, \quad H_1^{(0)} = 367\texttt{CD507}, \quad H_2^{(0)} = 3070\texttt{DD17},$$
$$H_3^{(0)} = \texttt{F70E5939}, \quad H_4^{(0)} = \texttt{FFC00B31}, \quad H_5^{(0)} = 68581511,$$
$$H_6^{(0)} = 64\texttt{F98FA7}, \quad H_7^{(0)} = \texttt{BEFA4FA4}\,.$$

Bei den drei Standards mit längeren Hashwerten zeigt sich der Trend zu nachvollziehbaren mathematischen Definitionen: Hier gibt es jeweils acht Wörter $H_0^{(0)}, \ldots, H_7^{(0)}$, die über die hexadezimal geschriebenen gebrochenen Anteile der Quadratwurzeln von Primzahlen p definiert sind; es geht also um die hexadezimal geschriebenen ganzen Zahlen $\left[2^r\{\sqrt{p}\}\right]$, wobei $r = 32$ *bzw.* 64 die Wortlänge des jeweiligen Algorithmus ist. Für SHA-256 und SHA-512 nimmt man die erste bis achte Primzahl, für SHA-384 die neunte bis sechzehnte. In der nachstehenden Tabelle sind diese Werte zu finden, wobei für SHA-256 natürlich nur die jeweils linke Hälfte relevant ist.

Daneben gibt es noch Konstanten, die auch bei SHA-1 wieder willkürlich *(?)* festgelegt sind als

$$K_t = \begin{cases} 5A827999 & \text{für } \ 0 \le t \le 19 \\ \texttt{6ED9EBAL} & \text{für } 20 \le t \le 39 \\ \texttt{8FLBBCDC} & \text{für } 40 \le t \le 59 \\ \texttt{CA62C1D6} & \text{für } 60 \le t \le 99 \end{cases},$$

und für die drei anderen Algorithmen gegeben sind durch die ersten Bits der gebrochenen Anteile der Kubikwurzeln der ersten Primzahlen. Für SHA-224 und SHA-256 nimmt man die ersten 32 Bit und die

i	p_i	$H_i^{(0)} = \left[2^r\{\sqrt{p_i}\}\right]$
1	2	6A09E667 F3BCC908
2	3	BB67AE85 84CAA73B
3	5	3C6EF372 FE94F82B
4	7	A54FF53A 5F1D36F1
5	11	510E527F ADE682D1
6	13	9B05688C 2B3E6C1F
7	17	1F83D9AB FB41BD6B
8	19	5BE0CD19 137E2179
9	23	CBBB9D5D C1059ED8
10	29	629A292A 367CD507
11	31	9159015A 3070DD17
12	37	152FECD8 F70E5939
13	41	67332667 FFC00B31
14	43	8EB44A87 68581511
15	47	DB0C2E0D 64F98FA7
16	53	47B5481D BEFA4FA4

ersten 64 Primzahlen, für SHA-384 und SHA-512 entsprechend die ersten 64 Bit und die ersten achtzig Primzahlen. In jedem Fall hat man also eine Folge von Wörtern $K_0, K_1, \ldots$ bis K_{63} *bzw.* K_{79}. Die Hexadezimalentwicklungen der gebrochenen Anteile der Kubikwurzeln der ersten achtzig Primzahlen sind unten in der Tabelle zu finden, wobei wieder darauf zu achten ist, daß K_t für SHA-224 und SHA-256 nur aus den ersten 32 Bit (oder acht Hexadezimalziffern) der angegebenen Werte besteht.

Außer diesen Konstanten sind für die fünf Algorithmen noch Funktionen definiert, die später im Konfusionsschritt eingesetzt werden; sie verknüpfen jeweils drei Wörter zu einem vierten, indem die angegebenen logischen Operationen bitweise angewendet werden.

In allen fünf Algorithmen wird die Funktion

$$Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z)$$

eingesetzt, in der $\oplus$ für die Addition in $\mathbb{F}_2$ oder (äquivalent) das exklusive Oder steht, hat für wahres x den Wahrheitswert $y \oplus$ *falsch*, also

i	p_i	$K_t = \lfloor 2^r \{\sqrt[3]{p_i}\} \rfloor$	i	p_i	$K_t = \lfloor 2^r \{\sqrt[3]{p_i}\} \rfloor$
1	2	428A2F98 D728AE22	41	179	A2BFE8A1 4CF10364
2	3	71374491 23EF65CD	42	181	A81A664B BC423001
3	5	B5C0FBCF EC4D3B2F	43	191	C24B8B70 D0F89791
4	7	E9B5DBA5 8189DBBC	44	193	C76C51A3 0654BE30
5	11	3956C25B F348B538	45	197	D192E819 D6EF5218
6	13	59F111F1 B605D019	46	199	D6990624 5565A910
7	17	923F82A4 AF194F9B	47	211	F40E3585 5771202A
8	19	AB1C5ED5 DA6D8118	48	223	106AA070 32BBD1B8
9	23	D807AA98 A3030242	49	227	19A4C116 B8D2D0C8
10	29	12835B01 45706FBE	50	229	1E376C08 5141AB53
11	31	243185BE 4EE4B28C	51	233	2748774C DF8EEB99
12	37	550C7DC3 D5FFB4E2	52	239	34B0BCB5 E19B48A8
13	41	72BE5D74 F27B896F	53	241	391C0CB3 C5C95A63
14	43	80DEB1FE 3B1696B1	54	251	4ED8AA4A E3418ACB
15	47	9BDC06A7 25C71235	55	257	5B9CCA4F 7763E373
16	53	C19BF174 CF692694	56	263	682E6FF3 D6B2B8A3
17	59	E49B69C1 9EF14AD2	57	269	748F82EE 5DEFB2FC
18	61	EFBE4786 384F25E3	58	271	78A5636F 43172F60
19	67	FC19DC68 2B8CD5B5	59	277	84C87814 A1F0AB72
20	71	240CA1CC 77AC9C65	60	281	8CC70208 1A6439EC
21	73	2DE92C6F 592B0275	61	283	90BEFFFA 23631E28
22	79	4A7484AA 6EA6E483	62	293	A4506CEB DE82BDE9
23	83	5CB0A9DC BD41FBD4	63	307	BEF9A3F7 B2C67915
24	89	76F988DA 831153B5	64	311	C67178F2 E372532B
25	97	983E5152 EE66DFAB	65	313	CA273ECE EA26619C
26	101	A831C66D 2DB43210	66	317	D186B8C7 21C0C207
27	103	B00327C8 98FB213F	67	331	EADA7DD6 CDE0EB1E
28	107	BF597FC7 BEEF0EE4	68	337	F57D4F7F EE6ED178
29	109	C6E00BF3 3DA88FC2	69	347	06F067AA 72176FBA
30	113	D5A79147 930AA725	70	349	0A637DC5 A2C898A6
31	127	06CA6351 E003826F	71	353	113F9804B EF90DAE
32	131	14292967 0A0E6E70	72	359	1B710B35 131C471B
33	137	27B70A85 46D22FFC	73	367	28DB77F5 23047D84
34	139	2E1B2138 5C26C926	74	373	32CAAB7B 40C72493
35	149	4D2C6DFC 5AC42AED	75	379	3C9EBE0A 15C9BEBC
36	151	53380D13 9D95B3DF	76	383	431D67C4 9C100D4C
37	157	650A7354 8BAF63DE	77	389	4CC5D4BE CB3E42B6
38	163	766A0ABB 3C77B2A8	78	397	597F299C FC657E2A
39	167	81C2C92E 47EDAEE6	79	401	5FCB6FAB 3AD6FAEC
40	173	92722C85 1482353B	80	409	6C44198C 4A475817

denselben Wert wie y. Ist x dagegen falsch, erhalten wir *falsch* $\oplus\, z$, also den Wahrheitswert von z. In SHA-1, SHA-224 und SHA-256 wird sie bitweise auf Wörter der Länge 32 angewandt, bei SHA-384 und SHA-512 auf solche der Länge 64.

Ebenfalls allen Algorithmen gemeinsam ist die Mehrheitsfunktion $Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$, die in derselben Weise bitweise angewandt wird. Falls genau zwei ihrer Eingabebits gesetzt sind, ist genau eine der drei Klammern eins, also auch das Ergebnis. Sind alle drei Eingabebits gesetzt, erhalten wir eine Summe von drei Einsen, also ebenfalls eins. In allen anderen Fällen sind alle drei Summanden null, also auch das Ergebnis.

Nur SHA-1 arbeitet mit der Funktion $Maj(x, y, z) = x \oplus y \oplus z$, deren Ergebnisbit genau dann gleich eins ist, wenn eine ungerade Anzahl der drei Eingabebits gleich eins ist.

Zur Diffusion verwenden alle fünf Algorithmen Verschiebungen. Die zyklischen Verschiebungen nach rechts und links bezeichnen wir mit ROTR und ROTL, die entsprechenden nichtzyklischen Verschiebungen mit SHL und SHR. Diese Operatoren verschieben um jeweils ein Bit; für Verschiebungen um größere Distanzen sorgen ihre Potenzen.

SHA-1 verwendet diese Operatoren direkt, bei den neueren Algorithmen gibt es stattdessen vier Funktionen auf Wörtern der jeweiligen Länge, die mehrere dieser Verschiebungen additiv kombinieren. Bei SHA-224 und SHA-256 sind dies

$$\begin{aligned}
\Sigma_0^{(256)}(x) &= \mathrm{ROTR}^2(x) \oplus \mathrm{ROTR}^{13}(x) \oplus \mathrm{ROTR}^{22}(x),\\
\Sigma_1^{(256)}(x) &= \mathrm{ROTR}^6(x) \oplus \mathrm{ROTR}^{11}(x) \oplus \mathrm{ROTR}^{25}(x),\\
\sigma_0^{(256)}(x) &= \mathrm{ROTR}^7(x) \oplus \mathrm{ROTR}^{18}(x) \oplus \mathrm{SHR}^3(x),\\
\sigma_1^{(256)}(x) &= \mathrm{ROTR}^{17}(x) \oplus \mathrm{ROTR}^{19}(x) \oplus \mathrm{SHR}^{10}(x),
\end{aligned}$$

bei SHA-384 und SHA-512

$$\begin{aligned}
\Sigma_0^{(512)}(x) &= \mathrm{ROTR}^{28}(x) \oplus \mathrm{ROTR}^{34}(x) \oplus \mathrm{ROTR}^{39}(x),\\
\Sigma_1^{(512)}(x) &= \mathrm{ROTR}^{14}(x) \oplus \mathrm{ROTR}^{18}(x) \oplus \mathrm{ROTR}^{41}(x),\\
\sigma_0^{(512)}(x) &= \mathrm{ROTR}^1(x) \oplus \mathrm{ROTR}^8(x) \oplus \mathrm{SHR}^7(x),\\
\sigma_1^{(512)}(x) &= \mathrm{ROTR}^{19}(x) \oplus \mathrm{ROTR}^{61}(x) \oplus \mathrm{SHR}^6(x).
\end{aligned}$$

Die weitere Vorgehensweise ist bei den neueren Algorithmen etwas anders als bei SHA-1; wir betrachten diese daher getrennt. In allen Fällen gehen wir davon aus, daß die Nachricht als Folge von Blöcken

$M^{(1)}, M^{(2)}, \ldots$ vorliegt gemäß der eingangs erläuterten Vorverarbeitung.

Der Algorithmus SHA-1 unterwirft diese Blöcke den folgenden Verarbeitungsschritten:

1. Setze

$$W_t = \begin{cases} M_t^{(i)} & \text{für } 0 \le t \le 15 \\ \text{ROTL}(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) & \text{für } 16 \le t \le 79 \end{cases}.$$

Dies ist offensichtlich ein Diffusionsschritt.

2. Initialisiere die fünf internen Variablen mit den fünf Hashwerten der vorigen Runde (für die erste Runde wurden die Hashwerte $H_j^{(0)}$ oben definiert):

$$a \leftarrow H_0^{(i-1)}, \quad b \leftarrow H_1^{(i-1)}, \quad c \leftarrow H_2^{(i-1)},$$
$$d \leftarrow H_3^{(i-1)}, \quad e \leftarrow H_4^{(i-1)}$$

3. Der Konfusionsschritt: Führe für $t = 0$ bis $t = 79$ die folgenden Anweisungen aus:

$$T \leftarrow \text{ROTL}^5(a) + f_t(b,c,d) + K_t + W_t, \quad e \leftarrow d, \quad d \leftarrow c,$$
$$c \leftarrow \text{ROTL}^{30}(b), \quad b \leftarrow a, \quad a \leftarrow T,$$

wobei $f_t = \begin{cases} Ch & \text{für } \le t \le 19 \\ Maj & \text{für } 40 \le t \le 59 \text{ ist.} \\ Maj & \text{sonst} \end{cases}$

(Die Konstanten K_t wurden oben definiert.)

4. Berechne den Hashwert der Runde:

$$H_0^{(i)} \leftarrow a + H_0^{(i-1)}, \quad H_1^{(i)} \leftarrow b + H_1^{(i-1)}, \quad H_2^{(i)} \leftarrow c + H_2^{(i-1)},$$
$$H_3^{(i)} \leftarrow d + H_3^{(i-1)}, \quad H_4^{(i)} \leftarrow e + H_4^{(i-1)}$$

Ergebnis des Algorithmus sind die aneinandergesetzten Hashwerte der letzten Runde.

Die Algorithmen SHA-224, SHA-256, SHA-382 und SHA-512 unterscheiden sich abgesehen von Längenparametern und den bereits definierten Anfangskonstanten kaum voneinander. Für jeden Block $M^{(i)}$,

aufgeteilt in die Wörter $M_0^{(i)}$ bis $M_{15}^{(i)}$, werden die folgenden Operationen durchgeführt:

1. Setze

$$W_t = \begin{cases} M_t^{(i)} & \text{für } 0 \le t \le 15 \\ \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16} & \text{für } 16 \le t \le r \end{cases},$$

wobei $r = 63$ für SHA-224 und SHA-256 und $r = 80$ sonst.

Das Pluszeichen steht hier folgendermaßen zu interpretieren: Die einzelnen Summanden werden als ganze Zahlen zwischen Null und 2^{32} *bzw.* 2^{64} interpretiert und dann als ganze Zahlen addiert. Das Ergebnis wird modulo 2^{32} *bzw.* 2^{64} betrachtet, d.h. ein etwaiger Überlauf wird ignoriert.

Bei σ_0 und σ_1 ist darauf zu achten, daß es sich hier bei SHA-224 und SHA-256 um $\sigma_0^{(256)}$ und $\sigma_1^{(256)}$ handelt, sonst aber um $\sigma_0^{(512)}$ und $\sigma_1^{(512)}$.

2. Initialisiere die acht internen Variablen mit den acht Hashwerten der vorigen Runde (für die erste Runde wurden die Hashwerte $H_j^{(0)}$ oben definiert):

$$a \leftarrow H_0^{(i-1)}, \quad b \leftarrow H_1^{(i-1)}, \quad c \leftarrow H_2^{(i-1)}, \quad d \leftarrow H_3^{(i-1)},$$
$$e \leftarrow H_4^{(i-1)}, \quad f \leftarrow H_5^{(i-1)}, \quad g \leftarrow H_6^{(i-1)}, \quad h \leftarrow H_7^{(i-1)}$$

3. Der Konfusionsschritt: Führe für $t = 0$ bis $t = r$ (also je nach Algorithmus 63 oder 79) die folgenden Anweisungen aus:

$$T_1 \leftarrow h + \Sigma_1(e) + Ch(e,f,g) + K_t + W_t$$
$$T_2 \leftarrow \Sigma_0(a) + Maj(a,b,c), \quad h \leftarrow g, \quad g \leftarrow f, \quad f \leftarrow e,$$
$$e \leftarrow d + T_1, \quad d \leftarrow c,$$
$$c \leftarrow b, \quad b \leftarrow a, \quad a \leftarrow T_1 + T_2.$$

4. Berechne den Hashwert der Runde:

$$H_0^{(i)} \leftarrow a + H_0^{(i-1)}, \quad H_1^{(i)} \leftarrow b + H_1^{(i-1)}, \quad H_2^{(i)} \leftarrow c + H_2^{(i-1)},$$
$$H_3^{(i)} \leftarrow d + H_3^{(i-1)}, \quad H_4^{(i)} \leftarrow e + H_4^{(i-1)}, \quad H_5^{(i)} \leftarrow f + H_5^{(i-1)},$$
$$H_6^{(i)} \leftarrow g + H_6^{(i-1)}, \quad H_7^{(i)} \leftarrow h + H_7^{(i-1)},$$

Ergebnis des Algorithmus sind die aneinandergesetzten Hashwerte der letzten Runde.

§4: Weitere Anwendungen sicherer Hashfunktionen

Die Nützlichkeit kryptographisch sicherer Hashfunktionen beschränkt sich nicht auf elektronische Unterschriften; sie sind auch Teil einer Reihe von weiteren Protokollen zum Schutz der Integrität von Daten, für sogenannte „Zeitstempel“ und einiges mehr. Auch die amerikanische Norm für den DSA sieht für die Schlüsselerzeugung den Einsatz von SHA vor. Hier soll vor allem dieser letztere Fall behandelt werden sowie als erstes der

a) Schutz der Integrität von Daten

Wie bereits zu Beginn der Vorlesung erwähnt, hat ein Gegner viele Angriffsmöglichkeiten; angesichts der geringen Sicherheitsstandards der am häufigsten verwendeten Betriebssysteme wird oft der direkte Angriff auf einen Computer die einfachste Möglichkeit sein. Gegen das Ausnutzen von Sicherheitslücken ist die Kryptographie machtlos; sie kann aber doch helfen, zumindest gewisse Manipulationen zu entdecken.

Schädliche Programme lassen sich am leichtesten dann in einen Computer einschleusen, wenn sie der Besitzer selbst installiert. Das wird er natürlich kaum freiwillig tun, aber wenn wenn man ihm eine manipulierte Variante eines Programms unterschiebt, das er ohnehin installieren möchte, wird es vielleicht versehentlich tun.

Teilweise können schon kleinste Manipulationen ein Einfallstor für künftige Angriffe sein: Wie wir bei der Diskussion von SSL/TLS gesehen haben, sind beispielsweise die elektronischen Unterschriften der wichtigsten Zertifizierungsagenturen im Programmcode der gängigen Browser enthalten. Fügt man dort nur einen einzigen Datensatz mit der elektronischen Unterschrift einer Bogus-Agentur hinzu, wird der Browser künftig ohne Nachfrage beliebige Seiten als sicher bezeichnen, wenn sie von dieser Agentur zertifiziert sind. Falls der manipulierte Browsercode auch noch einige Anweisungen mit „Ersatzadressen“ für populäre Ziele enthält, sind praktisch alle Arten von Angriffen möglich.

Ein Anwender muß daher sicher sein, daß seine Programme von einer sicheren Quelle kommen und nicht manipuliert sind. Falls er seine Programme von dubiosen Adressen herunterlädt (die selbstverständlich allesamt seriöse Namen haben) oder wenn er das in der Vergangenheit getan hat, kann ihm auch die Kryptographie nicht mehr helfen.

Falls er aber seinen Computer samt Software-Erstausstattung von seriösen und sorgfältig arbeitenden Produzenten und Händlern erworben hat (so es das im Konsumentenbereich geben sollte), dann kann er sichere Internetverbindungen zu seriösen Anbietern aufbauen und auch sicher sein, daß er mit dem gewünschten Partner verbunden ist. Jetzt muß er nur noch wissen, daß die Software, die er von dort (oder einer näher gelegenen und/oder billigeren Quelle) herunterlädt identisch ist mir der des Anbieters.

Zu diesem Zweck veröffentlichen viele Anbieter Hashwerte zu ihren Programmen. Falls diese mit einem kryptographisch sicheren Hashverfahren berechnet werden, kann man damit nicht nur überprüfen, ob die Software fehlerfrei übertragen wurde, sondern man kann auch überprüfen, daß sie, selbst wenn sie von einem unbekannten *mirror* kommt, identisch ist mit dem Original.

Auf dieselbe Weise lassen sich auch kritische Dateien auf dem eigenen Rechner schützen: Falls man zu jeder einen Hashwert berechnet und diesen idealerweise noch auch einem externen Speichermedium festhält, kann man jederzeit überprüfen, ob eine Datei verändert wurde.

b) Wie zufällig müssen unsere Schlüssel sein?

Idealerweise sollten unsere Schlüssel stets zufällig gewählt sein; jeder in Frage kommende Schlüssel sollte also genau dieselbe Wahrscheinlichkeit haben. Alles andere gibt einem Gegner zumindest prinzipiell Ansätze zu einer Kryptanalyse, die schneller ist als das Durchprobieren aller Schlüssel.

Nun sind aber in der asymmetrischen Kryptographie ohnehin alle vernünftigen kryptanalytischen Ansätze deutlich schneller als bloßes Probieren – deshalb brauchen wir ja auch so lange Schlüssel. Unter Sicherheitsgesichtspunkten ist völlig ausreichend, daß ein Gegner das

Verfahren nicht mit weniger als etwa 2^{128} Rechenschritten knacken kann; das Bundesamt für Sicherheit in der Informationstechnik ist sogar bereits mit einer Unterschranke von 2^{100} zufrieden.

Deshalb müssen wir bei RSA und DSA, selbst wenn wir Primzahlen der Länge 1024 oder gar 2048 Bit suchen, nicht unbedingt jeder solchen Primzahl dieselbe Chance geben: Es reicht, daß wir einen (echten) Zufallsanteil von mindestens 128 Bit haben; der Rest kann dann durchaus deterministisch daraus abgeleitet sein. Dabei muß freilich sichergestellt sein, daß das deterministische Verfahren keine erkennbare (und damit vielleicht auch ausnutzbare) Struktur in die Schlüssel bringt, und dazu ist eine kryptographisch sichere Hashfunktion ein geeignetes Werkzeug.

c) Erzeugung großer Zufallszahlen aus kleinen

Angenommen, wir haben eine Hashfunktion h, die Werte zwischen Null und $2^m - 1$ liefert, und wir haben als Startwert eine Zufallszahl $0 < x_0 < 2^m - 1$. (In der englischsprachigen Literatur wird ein solcher Startwert als *seed* = Keim bezeichnet.) Was wir wirklich wollen, ist aber eine Zahl y zwischen 2^{L-1} und 2^L für ein L, das deutlich größer ist als m.

Dazu gehen wir ähnlich vor wie beim Counter-Mode eines symmetrischen Kryptoverfahrens (*s.* Kap. 3, §5*e*): Wir schreiben $L - 1 = nm + b$ mit $0 \le b < m$ und berechnen $m + 1$ „Ziffern“ zur Basis 2^m als $v_i = h(x_0 + i)$. Dann setzen wir

$$z = \sum_{i=0}^{n-1} v_i \cdot 2^{mi} + (v_n \bmod 2^b) \cdot 2^{(n+1)m} .$$

Das ist noch kein geeigneter Kandidat, denn da $v_n \bmod 2^b$ kleiner ist als 2^b, ist auch $z < 2^{L-1}$. Genau deshalb aber ist $y = 2^{L-1} + z$ eine Zahl zwischen 2^{L-1} und 2^L, die auf einer Zufallszahl mit n zufälligen Bit beruht.

Auf der Suche nach RSA-Moduln können wir dann beispielsweise von dort aus das Sieb des ERATOSTHENES laufen lassen und nach der nächsten Primzahl suchen.

Falls wir extrem vorsichtig sind und dem Gegner keinen Ansatz geben möchten, die sehr inhomogene Verteilung der Differenzen aufeinanderfolgender Primzahlen auszunutzen, werden wir allerdings nur y auf Primalität testen und bei negativem Ausgang des Tests von vorne anfangen.

Ein ERATOSTHENES-Schritt ist jedoch selbst unter diesen Bedingungen völlig problemlos: Da es in dem Größenbereich, in dem wir suchen, keine geraden Primzahlen gibt, sollten wir im Fall eines geraden y den Wert von $y + 1$ testen. Auf diese Weise halbieren wir die Anzahl der zu erwartenden Durchläufe und haben selbst theoretisch keinen Sicherheitsverlust.

d) Primzahlen für DSA

In manchen Fällen brauchen wir nicht einfach *irgendwelche* Primzahlen, sondern zwei Primzahlen, die in einer bestimmten Relation zueinander stehen. Als Beispiel dazu betrachten wir den aus Kapitel 5 bekannten digitalen Unterschriftsalgorithmus DSA, in den USA standardisiert als *Digital Signature Standard* DSS

In seiner derzeit gültigen Form wurde er am 27. Januar 2000 als FIPS 186-2 veröffentlicht. Er benötigt bekanntlich eine kleine Primzahl q und eine große Primzahl $p \equiv 2q \bmod 1$, wobei FIPS 186-2 die Größenordnungen $2^{159} < q < 2^{160}$ und $2^{L-1} < p < 2^L$ mit $512 \le L \le 1024$ vorsieht; ein Nachtrag vom 5. Oktober 2001 läßt nur noch den Wert $L = 1024$ zu, und hier in Deutschland sehen die Empfehlungen der Bundesnetzagentur ohnehin höhere Werte auch für q vor. Im Jahr 2000 gab es allerdings nur den Algorithmus SHA-1 und der war auf die Unterschriftenlänge 160 des DSS abgestimmt. Heute wird man wohl eher mit den neueren SHA-Algorithmen arbeiten und auch längere Unterschriften bevorzugen. Da der Leser keine Schwierigkeiten haben sollte, im folgenden Algorithmus die Zahl 160 überall durch einen anderen Wert zu ersetzen, ist hier aber trotzdem der Originalalgorithmus aus FIPS 186-2 angegeben. Wenn man mit einer SHA-Variante arbeitet, die längere Werte als die gewünschte Unterschrift liefert, muß man sich gegebenenfalls bei der Anwendung von SHA auf die ersten Bits beschränken.

In *Appendix 2* des Standards wird folgendes Verfahren zur Produktion dieser Primzahlen vorgeschlagen: Sei

$$L - 1 = 160n + b \quad \text{mit} \quad 0 \leq b < 160\,.$$

1. Wähle eine zufällige 160-Bit-Zahl x_0.

2. Berechne u = SHA-1(x_0) $\oplus$ SHA-1($x_0 + 1 \bmod 2^{160}$).

u ist eine Folge von 160 Bit; falls allerdings das führende Bit gleich Null ist, beschreibt sie eine Zahl, die kleiner ist als 2^{159}, und wenn das letzte Bit gleich Null ist, haben wir eine gerade Zahl die ebenfalls nicht als Wert für q in Frage kommt. Deshalb:

3. Ersetze in u das führende wie auch das letzte Bit durch Einsen.

4. Teste, ob q eine Primzahl ist.

5. Falls nicht: Zurück zum ersten Schritt.

6. Setze *Zähler* = 0 und *offset* = 2.

7. Berechne v_k = SHA-1(x_0 + *offset* + $k \bmod 2^{160}$) für $k = 0, \ldots, n$.

8. Setze $w = v_0 + v_1 \cdot 2^{160} + \ldots + v_{n-1} \cdot 2^{160(n-1)} + (v_n \bmod 2^b) \cdot 2^{160n}$. Diese Zahl ist kleiner als 2^{L-1}; daher liegt $x0w + 2^{L-1}$ zwischen 2^{L-1} und 2^L.

x soll natürlich ein Kandidat für p sein; allerdings wird x im allgemeinen nicht kongruent eins modulo $2q$ sein. Dies korrigieren wir im nächsten Schritt:

9. Sei $c = x \bmod 2q$ und $p = x - (c - 1)$

10. Falls $p < 2^{L-1}$: Weiter bei Schritt 13.

11. Teste, ob p prim ist.

12. Falls ja, weiter bei Schritt 15

13. Setze *Zähler* = *Zähler* + 1 und *offset* = *offset* + n + 1.

14. Falls *Zähler* $\geq 2^{12} = 4096$, fängt alles wieder bei Schritt eins von vorne an, andernfalls geht es mit den neuen Werten zurück zu Schritt sieben.

15. Wenn der Algorithmus hier ankommt, haben wir geeignete Werte für q und p gefunden. Um beweisen zu können, daß alles mit rechten Dingen zuging, verlangt der Algorithmus von uns, daß wir uns den aktuellen Wert von x_0 merken.

e) Wie bekommt man echte Zufallszahlen?

Durch Hashverfahren können wir zwar die Anzahl benötigter Zufallsbits zumindest bei asymmetrischen Kryptoverfahren deutlich reduzieren, wir können aber definitiv nicht ohne sie auskommen. Alles beruht darauf, daß wir mit mindestens 128 „wirklich“ zufälligen Bits starten. Wie bekommen wir diese? Und was bedeutet das Wort „zufällig“?

Im Alltagsgebrauch bezeichnen wir ein Ereignis als zufällig, wenn es keine Erklärung gibt, warum ausgerechnet dieses Ereignis an Stelle von mehreren ebenfalls möglichen Alternativen eingetreten ist: Fällt beim Würfel die Zahl „drei“, so ist das Zufall.

Untersuchen wir allerdings die Bewegung des Würfels genauer, so können wir ohne größere Probleme aus den Anfangsbedingungen (Ort, Geschwindigkeit, Drehimpuls des Würfels beim Abwurf) zumindest im Prinzip dessen Bahn berechnen und das Ergebnis vorhersagen; es ist also nicht mehr zufällig. In der Tat gibt es Spieler, die in der Lage sind, mit recht guter Trefferwahrscheinlichkeit jede gewünschte Zahl zu würfeln. Bei genauerer Betrachtung des Vorgangs wird die Zufälligkeit des Ergebnisses hier also doch eher problematisch.

In der Tat ist es algorithmisch unmöglich, zu entscheiden, ob eine gegebene Zahlenfolge zufällig ist; wir können nie ausschließen, daß wir einfach das korrekte Bildungsgesetz noch nicht gefunden haben.

Auch bei physikalischen Phänomenen, die uns zufällig erscheinen, müssen wir immer die Möglichkeit im Auge behalten, daß wir vielleicht einfach noch nicht die korrekte Theorie zu ihrer Erklärung gefunden haben. Zumindest in der Quantentheorie gibt es allerdings durchaus Sätze, wonach das Verhalten gewisser Systeme *nicht* mit bislang noch unbekannten „verborgenen Parametern“ deterministisch erklärt werden kann, und Phänomene wie der radioaktive Zerfall oder thermisches Rauschen liefern Werte, die als zufällig interpretiert werden können. Sie sind zwar

im allgemeinen nicht gleichverteilt, aber dieses Problem läßt sich durch geeignete statistische Aufbereitung beheben.

Dem Normalanwender stehen physikalische Quellen üblicherweise nicht zur Verfügung. Trotzdem gibt es auch auf seinem Computer eine ganze Reihe von nicht vorhersehbaren Ereignissen, die keineswegs alle auf Softwarefehlern beruhen: Mißt man etwa den Abstand zwischen zwei Tastaturinterrupts mit einer Genauigkeit von einer Tausendstel Sekunde, so verhält sich die letzte Ziffer sicherlich zufällig: Niemand kann seine Bewegungen mit einer Genauigkeit in diesem Bereich steuern. Auch aus Mausbewegungen, Festplattenzugriffen und aus Netzwerkaktivitäten lassen sich entsprechende Zufallszahlen gewinnen.

Linux (und verschiedenen andere UNIX-Systeme) sammeln die so gewonnene Entropie und stellen sie in /dev/random als Folge gleichverteilter Zufallszahlen zur Verfügung; mit

`dd if=/dev/random of=Dateiname bs=1 count=`n

lassen sich n Zufallsbytes gewinnen – sofern hinreichend viel Entropie zur Verfügung steht. Andernfalls blockiert /dev/random und liefert erst dann wieder neue Bytes, wenn neue zufällige Ereignisse eingetreten sind.

Eine zweite Spezialdatei, /dev/urandom, greift ebenfalls auf den Entropiepool des Betriebssystems zu, liefert aber bei nicht ausreichender Entropie algorithmisch berechnete Pseudozufallsbytes anstatt zu blockieren.

Unter cygwin stehen /dev/random und /dev/urandom im Prinzip auch für Windows-Anwender zur Verfügung, allerdings liefern dann beide nur Pseudozufallszahlen in üblicher Windowsqualität, die auf keinen Fall für kryptographische Zwecke verwendet werden dürfen.

Wer keinen Zugang zu physikalischen Zufallsquellen oder Computern mit echten Entropiequellen hat, muß leider seine Zufallsbit auf konventionelle Weise erzeugen, d.h. er muß Münzen oder Würfel werfen und versuchen, dies möglichst „zufällig“ zu tun. Mit einiger Übung ist es nicht sehr schwer, Münzen oder Würfel so zu werfen, daß das Ergebnis

ziemlich vorhersehbar ist; man sollte für „echten“ Zufall auf jeden Fall möglichst hoch werfen und auch da die Kraft variieren.

§5: Literatur

SHA-1 ist bei BUCHMANN und in weiteren neueren Lehrbüchern der Kryptologie behandelt; die übrigen SHA-Algorithmen sind dort allerdings noch nicht zu finden. Alle sind aber ausführlich beschrieben in den Originaldokumenten, deren neueste Version man via

`http://csrc.nist.gov/CryptoToolkit/tkhash.html`

und den Link zu *Secure hashing* finden sollte. Entsprechend führt der Link *Digital Signatures* zum DSS.

Grundsätzliches über kryptographische Hashfunktionen und Angriffe dagegen findet man im fünften Kapitel des Buchs

NIELS FERGUSON, BRUCE SCHNEIER, TADAYOSHI KOHNO: *Cryptography Engineering – Design Principles and Practical Applications*, Wiley,2010

Kapitel 8
Kryptographische Protokolle

Bislang hatten wir Kryptologie nur im Zusammenhang mit Verschlüsselung und mit elektronischen Unterschriften betrachtet; in diesem Kapitel sollen einige darüber hinausgehende Aspekte betrachtet werden.

Eine wichtige solche Anwendung ist beispielsweise die Identitätsfeststellung: Der Gebrauch einer Geldkarte (oder auch eines Mobiltelephons) hängt wesentlich davon ab, daß die Auszahlung *bzw.* die Gesprächsgebühren dem richtigen Konto belastet werden können.

Bei Geldkarten wird dies heute so realisiert, daß der Benutzer eine Geheimzahl eingeben muß, die in verschlüsselter Form im Magnetstreifen der Karte kodiert ist. Die Verschlüsselung hängt nicht nur ab von der Geheimzahl, sondern auch von den Kontodaten des Inhabers, so daß keine Bijektion zwischen den nur knapp zehn Tausend verschiedenen Geheimzahlen und Feldern auf dem Magnetstreifen besteht. Zur Verschlüsselung wird ein Triple DES benutzt, dessen Schlüssel im gesamten System konstant ist und der daher sehr sorgfältig geheimgehalten werden muß; er ist nur den Computern der Clearingstellen bekannt.

Geldautomaten oder *point of sale* Terminals müssen daher sowohl die eingetippte Geheimzahl als auch die Information auf dem Magnetstreifen an so eine Clearingstelle übermitteln; dort wird beides verglichen und die Zahlung entweder autorisiert oder auch nicht.

Die dabei verwendeten Terminals funktionieren so, daß ein Händler die übermittelte Kundendaten nicht zu Gesicht bekommt; allerdings ist natürlich denkbar, daß ein betrügerischer Händler Geräte so manipuliert, daß sie sowohl eine Kopie des Magnetstreifens als auch die eingetippte Geheimzahl in einer ihm zugänglichen Weise speichern. Mit diesen

Informationen kann er sich dann gegenüber Dritten als Karteninhaber ausgeben und beliebig über dessen Konto verfügen. Sicherer wäre ein Verfahren, das dem Händler zwar garantiert, daß er den legitimen Karteninhaber vor sich hat, bei dem er aber keine Chance hat, in betrügerischer Absicht an dessen Daten zu gelangen.

Andere nichtklassische Anwendungen der Kryptologie sind etwa das Werfen von Münzen (für eine zufällige Entscheidung „Kopf oder Zahl“) via Telephon oder auch ein Pokerspiel per Internet. Die dafür eingesetzten Protokolle können durchaus auch ernste Anwendungen haben, beispielsweise beim verteilten Rechnen, wenn die Zuweisung von Aufgaben an die einzelnen Rechner nach einem Zufallsverfahren erfolgt. Falls die Kosten für die Inanspruchnahme der verschiedenen Rechner von verschiedenen Personen getragen werden, sollten diese definitiv daran interessiert sein, daß niemand dem Zufall auf ihre Kosten nachhilft.

§ 1: Werfen einer Münze per Telephon

Beim Werfen einer Münze geht es darum, daß zwei Partner A und B eine Entscheidung herbeiführen, die für beide als zufällig erkennbar ist. Bei einer Telephonverbindung ohne Videokanal sind echte Münzen natürlich nutzlos.

Die Zahlentheorie liefert eine praktikable Alternative: A wählt zwei große Primzahlen p und q und schickt deren Produkt $N = pq$ an B. Dieser wählt eine Zufallszahl x zwischen $\sqrt{N} + 1$ und $N - 1 - \sqrt{N}$ und schickt $y = x^2$ an A.

Da y modulo N ein Quadrat ist, ist es erst recht ein Quadrat modulo p und modulo q. Der wesentliche Punkt ist nun, daß sich Quadratwurzeln modulo einer Primzahl leicht berechnen lassen: Im einfachsten Fall, wenn $p \equiv 3 \bmod 4$ ist gilt für ein Quadrat $y \equiv x^2 \bmod p$ nach dem kleinen Satz von FERMAT die Gleichung

$$\left(y^{\frac{p+1}{4}}\right)^2 \equiv y^{\frac{p+1}{2}} \equiv x^{p+1} = x^p \cdot x \equiv x \cdot x \equiv y \bmod p\,,$$

die beiden Quadratwurzeln von y modulo p sind also $\pm y^{\frac{p+1}{4}} \bmod p$.

Für Primzahlen $p \equiv 1 \bmod 4$ ist die Berechnung der Quadratwurzel etwas aufwendiger, aber wie bereits in Zusammenhang mit dem quadratischen Sieb in Kap. 4, §8*b)* diskutiert, gibt es entsprechende Algorithmen. Falls A damit nicht vertraut ist, kann er sich natürlich auch einfach auf Primzahlen $p, q \equiv 3 \bmod 4$ beschränken.

A berechnet nun die Quadratwurzeln $\pm w_1, \pm w_2$ von y modulo p und modulo q; zwei davon setzt er nach dem chinesischen Restesatz zusammen zu einer Quadratwurzel w modulo N. Dabei hat er *vier* Möglichkeiten: Je nach Wahl der Vorzeichen bekommt er entweder den (ihm unbekannten) Wert x oder dessen Negatives, oder aber einen neuen Wert $\pm u$, der modulo der einen Primzahl kongruent x, modulo der anderen aber kongruent $-x$ ist. Er muß sich für eine dieser vier Möglichkeiten entscheiden und schickt die betreffende Zahl an B. Diese Entscheidung von A simuliert den Münzwurf.

Falls die geschickte Zahl gleich $\pm x$ ist, erhält B keine neuen Informationen und hat verloren; dies geschieht offenbar in 50% aller Fälle.

In den übrigen 50% der Fälle schickt A eine Zahl u, die modulo genau einer der beiden Primzahlen kongruent x ist. Somit ist ggT$(x - u, N)$ gleich dieser Primzahl, und auf diese Weise kann B die Zahl N faktorisieren. In diesem Fall hat B gewonnen und schickt zum Beweis einen der beiden Primfaktoren an A.

Falls p und q hinreichend groß und verschieden sind, hat B keine realistische Möglichkeit, N auf andere Weise zu faktorisieren, insbesondere nicht in den wenigen Sekunden, mit denen er bei korrekter Durchführung des Protokolls auskommen muß. Auch sonst kann er den Ausgang nicht zu seinen Gunsten zu beeinflussen: Er könnte zwar versuchen, zwei Zahlen x und $u \neq \pm x$ mit gleichem Quadrat zu finden und eine davon an A schicken, aber wie wir bei der Diskussion des quadratischen Siebs gesehen haben, ist genau das die derzeit effizienteste Methode zur Faktorisierung von N und damit nicht leichter als diese Faktorisierung.

Auch A hat keine Möglichkeit, das Ergebnis zu seinen Gunsten zu beeinflussen, denn er kann zwar alle vier Quadratwurzeln von y modulo N berechnen, weiß aber nicht, welche davon die Zahl x ist, deren Quadrat ihm B übermittelte.

§2: Poker per Telephon

Poker ist ein Kartenspiel, das traditionellerweise in verrauchten Hinterzimmern von Restaurants gespielt wurde, wobei meist auch viel Alkohol im Spiel war. Der moderne Internetuser allerdings möchte sich nicht einen ganzen Pokerabend lang von seinem Computer trennen und muß daher einen anderen Weg finden. Das kryptographische Problem besteht darin, daß bei einem traditionellen Pokerspiel die Karten jeweils gemischt, abgehoben und verteilt werden müssen und am Ende niemand die Karten seiner Mitspieler kennen darf.

Der Ansatz ist ähnlich wie bei den blinden Unterschriften, die für elektronisches Bargeld benutzt werden, allerdings wird statt RSA ein einfacheres Verfahren verwendet, das Verfahren von POHLIG und HELLMANN. Es funktioniert so ähnlich wie RSA, jedoch wird anstelle des Produkts zweier Primzahlen nur eine Primzahl p als Modul verwendet. Zur Verschlüsselung dient ein zu $p - 1$ teilerfremder Exponent e, mit dem eine Nachricht m verschlüsselt wird als m^e mod p. Zur Entschlüsselung dient ein zweiter Exponent d mit der Eigenschaft, daß $de \equiv 1 \bmod p-1$ ist, denn nach dem kleinen Satz von FERMAT ist dann $m^{de} \equiv m \bmod p$. Ähnlich wie bei RSA kann d mit dem erweiterten EUKLIDischen Algorithmus (angewandt auf e und $p - 1$) bestimmt werden.

Die Berechnung von d kann jeder ausführen, der die zur Anwendung des Algorithmus notwendigen Zahlen p und e kennt; der Algorithmus von POHLIG und HELLMANN ist also kein asymmetrisches Verfahren, sondern ein symmetrisches Kryptoverfahren, dessen Schlüssel geheim bleiben muß. Man kann wahlweise das Paar (p, e) als Schlüssel betrachten oder aber die Primzahl p innerhalb eines Netzwerks ein für alle man fest wählen und öffentlich bekanntgeben und dann nur den Exponenten e als geheimen Schlüssel betrachten.

Auf den ersten Blick vereint das Verfahren von POHLIG und HELLMANN alle Nachteile der symmetrischen und der asymmetrischen Kryptographie: Wie bei allen symmetrischen Verfahren hat man das Problem des Schlüsselaustauschs, und das bei einem Rechenaufwand, der dem des RSA-Verfahrens entspricht!

Tatsächlich muß die Sicherheit des Verfahrens von POHLIG/HELLMANN nach völlig anderen Kriterien beurteilt werden als die des RSA-Verfahrens: Die empfohlene Schlüssellänge von 2048 Bit bei RSA erklärt sich aus dem Stand und dem zu erwartenden Fortschritt bei Faktorisierungsalgorithmen; diese aber spielen für die Sicherheit von POHLIG/HELLMANN keinerlei Rolle. Hier muß ein Angreifer versuchen, den bei RSA öffentlich bekannten Exponenten e zu ermitteln; falls er die möglichen Exponenten einfach durchprobiert, ist er ungefähr in derselben Situation wie bei einem Angriff auf DES oder AES, so daß man vielleicht argumentieren könnte, daß ungefähr dieselben Sicherheitsparameter wie für Algorithmen dieser Art gewählt werden sollten, d.h. nach heutigem Stand mindestens 128 Bit für Primzahl und Exponent.

Dies ist aber zu optimistisch: Wie wir schon bei der Diskussion der Sicherheit von DES gesehen haben, müssen sich bei einer guten Blockchiffre die Transformationen wie eine Zufallsauswahl aus der vollen Permutationsgruppe über der Menge aller möglicher Blöcke verhalten; insbesondere dürfen sie keine zu kleine Untergruppe dieser symmetrischen Gruppe erzeugen.

Diese Bedingung ist hier klar verletzt: Die Transformationen von POHLIG und HELLMANN bilden sogar bereits eine (zyklische) Untergruppe der vollen Permutationsgruppe. Dies gibt einem Angreifer eine ganze Reihe zusätzlicher Möglichkeiten; insbesondere muß er bei einer Attacke mit bekanntem Klartext nur ein diskretes Logarithmenproblem lösen, wozu es, wie wir aus §4 von Kapitel fünf wissen, deutlich schnellere Algorithmen gibt als das vollständige Durchsuchen des Schlüsselraums.

Wenn wir trotzdem oft mit deutlich kürzeren Schlüssellängen arbeiten als bei Verfahren mit diskreten Logarithmen, rechtfertigt sich das vor allem aus der Art der Anwendungen: Das Verfahren von POHLIG und HELLMANN wird in erster Linie eingesetzt für Protokolle, die in Echtzeit ablaufen; falls man dazu dann auch noch *ad hoc*-Schlüssel einsetzt, nützt eine Kryptanalyse dem Gegner nur dann, wenn er sie innerhalb weniger Sekunden oder höchstens Minuten durchführen kann. In solchen Situationen sind die Sicherheitsanforderungen natürlich erheblich geringer als etwa bei elektronischen Unterschriften, die oft jahrelang sicher sein müssen.

Speziell beim Kartenspiel per Telephon (oder Internet) wird das Verfahren folgendermaßen eingesetzt:

Die n Teilnehmer einigen sich auf eine Primzahl p, die hinreichend groß sein muß, daß niemand in der zur Verfügung stehenden Zeit einen nennenswerten Teil aller möglicher Exponenten durchprobieren kann. Eine Größenordnung von 100 Bit oder dreißig Dezimalstellen sollte dazu mehr als ausreichend sein. Außerdem wird jeder der n Spielkarten eine natürliche Zahl $1 < x_i < p-1$ zugeordnet. Dabei kann es um eine fortlaufende Nummerierung handeln oder aber auch um eine strukturierte, bei der beispielsweise Farbe und Wert der Karte in den Ziffern von x_i kodiert sind.

Sodann wählt jeder der r Spieler zwei Exponenten d_k, e_k derart, daß $d_k e_k \equiv 1 \bmod p-1$ ist; nach dem kleinen Satz von FERMAT ist also $\left(x^{e_k}\right)^{d_k} \equiv x \bmod p$ für alle natürlichen Zahlen x.

Vor dem Start des eigentlichen Spiels müssen die Karten gemischt werden. Üblicherweise ist dies Aufgabe eines der Spieler; die anderen sehen nur zu, daß alles seine Richtigkeit hat, und vielleicht hebt einer von ihnen auch noch ab.

Beim Spiel per Telephon kann niemand beim Mischen zusehen; deshalb werden *alle* Spieler daran beteiligt. Der Kartenstapel wird simuliert durch die Folge $(i, x_i)_{i=1,\dots,n}$ der Karten, wobei die erste Komponente eines jeden Paars die Position der Karte im Stapel angibt.

Jeder Spieler wählt eine Permutation π_k der Menge der Zahlen von eins bis n. Mit dieser mischt er den Stapel, wobei er gleichzeitig die Karten mit seinem Exponenten e_k verschlüsselt.

Der erste Spieler ersetzt also jedes Paar (i, x_i) auf dem Stapel durch $(\pi_1(i), x_i^{e_1} \bmod p)$, sortiert die entstehende Liste wieder nach ihren ersten Komponenten und gibt die so entstandene Folge $(i, y_i)_{i=1,\dots,n}$ weiter an den zweiten. Der ersetzt jedes Paar (i, y_i) durch $(\pi_2(i), y_i^{e_2} \bmod p)$, sortiert wieder nach der ersten Komponente *usw.*, bis jeder Spieler seine Permutation angewandt hat. Der „gemischte“ Stapel besteht also aus den Paaren $\left(\pi_r \circ \cdots \circ \pi_1(i),\ x_i^{e_1 \cdots e_r} \bmod p\right)$, sortiert nach ihren ersten Komponenten.

Zum Spielen müssen die Karten wieder entschlüsselt werden, allerdings so, daß nur der Empfänger den Klartext x_i erkennen kann. Wenn daher einer der Spieler eine Karte vom Stapel erhalten soll, nimmt sein rechter Nachbar die oberste Karte vom Stapel und entschlüsselt sie mit seinem Exponenten d_k. Dann reicht er sie weiter an *seinen* rechten Nachbarn, der seinen Exponenten d_{k+1} (oder d_1, falls $k = r$) anwendet *usw.*, bis die Karte ihren Empfänger erreicht hat. Nachdem auch dieser noch mit seinem d-Exponenten potenziert hat, wurde die Karte x_i zu

$$x_i^{e_1 \cdots e_r d_1 \cdots d_r} = x_i^{(e_1 d_1)(e_2 d_2) \cdots (e_r d_r)} \equiv x_i \bmod p\,,$$

ist also wieder erkennbar.

Ausgespielt werden die Karten nun unverschlüsselt nach den üblichen Regeln des jeweiligen Kartenspiels; wer also die Karte x_i ausspielen möchte, schickt die Zahl x_i per E-Mail oder sonstige Software an alle Spielteilnehmer.

Im Gegensatz zur Situation bei einem realen Kartenspiel können diese nun allerdings nicht sicher sein, daß jeder nur Karten ausspielt, die ihm auch wirklich ausgeteilt wurden: Beim Skat etwa könnte der Alleinspieler während des Spiels unbemerkt eine „gedrückte“ Karte ausspielen. Um dies zu verhindern, werden nach Spielende alle Exponenten d_k, e_k bekanntgegeben, so daß sich jeder vergewissern kann, daß regelgerecht gespielt wurde.

Speziell beim Poker, wo Bluffen ein wesentlicher Teil des Spiels ist, sollte allerdings auch nach Spielende nicht bekannt werden, ob jemand wirklich Karten auf der Hand hatte, die soviel wert sind, wie er suggerierte. Mit etwas komplizierteren Verfahren kann auch hier die Einhaltung der Spielregeln kontrolliert werden.

§3: Zero Knowledge Protokolle

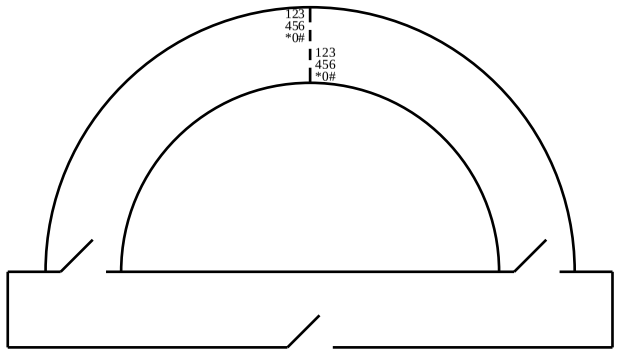
Wer heute mit einer Kreditkarte oder Bankkarte bezahlt, gibt dem Zahlungsempfänger recht viel Information in die Hand: Außer der Karten- oder Kontonummer sind auch sein Name, seine Bankverbindung und ähnliches auf der Karte kodiert. Darüber hinaus muß er beim Bezahlen entweder eine Unterschrift hinterlassen (die oft nicht allzu schwer

nachzumachen ist) oder eine PIN eintippen, die von einem manipulierten Gerät aufgezeichnet werden kann. Der Karteninhaber muß also ziemliches Vertrauen in den Zahlungsempfänger haben, kann dessen Vertrauenswürdigkeit aber oft nicht überprüfen: Schließlich hat nicht jedes von der Mafia geführte Restaurant ein Bronzeschild an der Tür mit der Aufschrift A PROUD MEMBER OF MOB ENTERPRISES CENTRAL EUROPE LTD.

Eine Alternative zur Kartenzahlung wäre das im Zusammenhang mit RSA behandelte elektronische Bargeld, jedoch scheint sich dieses zumindest derzeit kommerziell nicht durchzusetzen.

Eine andere Möglichkeit bestünde darin, daß der Karteninhaber keine PIN eintippt, sondern nur beweist, daß er die PIN kennt. Verfahren, bei denen jemand beweist, daß er über eine Information verfügt *ohne* irgendeinen Teil dieser Information preiszugeben, bezeichnet man in der Kryptologie als Zero Knowledge Protokolle oder, falls nur sehr wenig Information preisgegeben wird, als Minimal Disclosure Protokolle.

Zur Illustration der grundsätzlichen Idee betrachten wir zunächst eine durch ein Nummernschloß gesicherte Tür. Diese sei in einem runden Gang, dessen beide Enden durch Türen mit einem Vorraum verbunden sind:



Wenn B sieht, daß A durch die eine Tür in den halbkreisförmigen Gang eintritt und durch die andere herauskommt, kann er sicher sein, daß A

durch die mit Nummernschloß gesicherte Tür gegangen ist, erfährt aber nicht die dort einzugebende Geheimzahl.

Kann umgekehrt auch A sicher sein, daß B *überhaupt nichts* erfahren hat außer der Tatsache, daß A die Nummer kennt? Offensichtlich nicht, denn auf jeden Fall weiß B, daß A durch die Tür gegangen ist. Das ist nicht weiter schlimm, aber könnte B noch mehr erfahren haben?

Die Antwort auf diese Frage ist vor allem deshalb sehr schwierig, weil „etwas“ nur schwer zu definieren ist. Mit „nichts“ haben wir weniger Schwierigkeiten:

Angenommen, wir filmen den gesamten Ablauf der Verifikation. Falls dabei ein Film entsteht, den B auch ohne Teilnahme von A zusammen mit einem Statisten C auch drehen könnte, hat er offensichtlich nichts erfahren, was er nicht schon vorher wußte. Im vorliegenden Fall ist dieses Kriterium nicht erfüllt, denn C kann ohne Kenntnis der Geheimzahl den halbkreisförmigen Gang nicht durchqueren.

Mit einer kleinen Änderung des Protokolls geht das: Nun geht A zunächst allein in den Vorraum und verschwindet dort nach seiner Wahl hinter einer der beiden Türen. Danach erst kommt B mit seiner Kamera in den Vorraum und kann bestimmen, durch welche der beiden Türen A zu ihm kommen soll.

Damit kann B allerdings nicht wissen, ob A wirklich durch die gesicherte Tür gegangen ist: Falls er zu Beginn durch die Türe verschwunden ist, aus der ihn B kommen sehen möchte, kann er einfach herauskommen ohne seine Geheimzahl anwendenden zu müssen. B wird daher erst dann glauben, daß A diese wirklich kennt, wenn das Protokoll mehrfach wiederholt wird, und selbst dann gibt es immer noch ein Restrisiko von 2^{-n} bei n Versuchen.

A kann nun aber wirklich sicher sein, daß B keine neuen Informationen bekommen hat: Nun könnte B auch mit einem Statisten C ein Video drehen, in dem alles so abläuft wie beim echten Protokoll mit A: B muß nur C vorher informieren, durch welche Tür er kommen soll, oder aber er schneidet nachträglich alle Szenen heraus, in denen C durch die falsche Tür kommt.

In einer digitalen Version könnte man die Geheimzahl beispielsweise ersetzen durch die beiden Primteiler eines Produkts $N = pq$ zweier großer Primzahlen. A könnte die Kenntnis dieser Primteiler beweisen, indem er zu einer von B vorgegebenen Quadratzahl y modulo N eine Quadratwurzel produziert. Hierbei erfährt allerdings B, wie wir beim Münzwurf per Telephon gesehen haben, mit einer Wahrscheinlichkeit von 50% die beiden Primzahlen, was natürlich inakzeptabel ist.

Praktikabel ist dagegen die folgende Version, die SHAMIR (den wir vom RSA-Verfahren her kennen) und sein damaiger Doktorand AMOS FIAT 1986 vorgeschlagen haben: A wählt als Geheimzahl irgendein x mit $\sqrt{N} < x < N - \sqrt{N}$ und veröffentlicht $y = x^2 \bmod N$. Soll er nun gegenüber A nachweisen, daß er x kennt, erzeugt er zunächst eine nur für diesen einen Austausch gültige Zahl $\sqrt{N} < u < N - \sqrt{N}$ und schickt $v = u^2 \bmod N$ an B. Dieser kann nun entscheiden, ob er eine Quadratwurzel (modulo N) aus v oder aus yv sehen möchte. Falls er sich für v entscheidet, schickt ihm A entweder u oder $-u$, ansonsten $\pm xu$. A kann also beide Anfragen beantworten, braucht aber nur bei der zweiten Alternative seine Geheimzahl x.

Trotzdem sollte B stets zufällig zwischen seinen beiden Möglichkeiten wählen, denn wenn er sich stets oder überwiegend für die zweite entscheidet, könnte sich ein Betrüger C für A ausgeben, indem er y über den EUKLIDischen Algorithmus invertiert, eine Zufallszahl u erzeugt, und $v = y^{-1}u^2 \bmod N$ an B schickt. Verlangt B nun eine Wurzel aus yv, so kann er einfach u schicken. Würde B jedoch eine Wurzel aus v verlangen, müßte C passen, denn so etwas kann er nur dann berechnen, wenn er eine Wurzel aus y kennt. C kann sich also immer so vorbereiten, daß er *eine* der beiden möglichen Fragen von B korrekt beantworten kann; beide Fragen kann aber nur beantworten, wer eine Wurzel aus y modulo N kennt.

Die Faktorisierung von N spielt hier offensichtlich keinerlei Rolle, denn A muß nie Wurzeln modulo p oder modulo q ziehen. Daher muß N auch nicht unbedingt ein Produkt zweier Primzahlen sein, allerdings muß die Primzerlegung von N so schwierig sein, daß sie niemand finden kann, denn wer immer ein z finden kann mit $z^2 \equiv y \bmod N$ kann sich bei diesem Protokoll als A ausgeben.

Die Simulation mit einem Statisten ist auch hier wieder problemlos: Entweder der Statist erfährt vorher, welche Frage B stellen wird und kann sich darauf vorbereiten, oder aber alle Szenen, in denen er sich für die falsche Alternative entschieden hat, werden anschließend herausgeschnitten.

§4: Schlußbemerkung

Münzwurf oder Kartenspielen per Telephon gehören sicherlich nicht zu den praktisch relevantesten Anwendungen der Kryptologie, sie sind jedoch relativ elementare Beispiele aus einem Problemkreis, der durchaus auch ernstzunehmende Themen behandelt wie etwa das Rechnen mit verdeckten Daten:

Hinter Daten aus geologischen Explorationen oder Sensordaten von Satelliten steckt meist ein gewaltiger (auch finanzieller) Aufwand, so daß diese Daten einen großen Wert haben und nicht ohne Bezahlung an andere weitergegeben werden. Sie müssen allerdings oft auch mit sehr spezialisierten Verfahren ausgewertet und aufbereitet werden, und auch das kann nicht jeder.

Falls der Besitzer der Daten eine Auswertung wünscht, die er selbst nicht durchführen kann, braucht er also die Hilfe eines Spezialisten. Er möchte diesem jedoch nicht seine Daten anvertrauen, denn der Spezialist weiß schließlich, was man damit machen kann und wird sich möglicherweise über Strohmänner dann Schürfrechte, Optionen und ähnliches verschaffen. Umgekehrt möchte aber auch der Spezialist seine Programme nicht weitergeben, denn wer darüber verfügt, braucht ihn künftig nicht mehr.

Auch beim sogenannten *cloud computing* wäre ein Rechnen mit verschlüsselten Daten nützlich: Hier werden die einzelnen Schritte eines komplexen Algorithmus mehr oder weniger zufällig auf Rechner mit freier Kapazität verteilt in einem großen Cluster, das durchaus nicht nur vertrauenswürdige Computer enthalten muß.

Benötigt werden hierzu vollständig homomorphe Verschlüsselungsverfahren, d.h. Verschlüsselungsfunktionen φ, die nicht nur (wie etwa RSA und POHLIG/HELLMAN) mit der Multiplikation kompatibel sind, sondern

mit *allen* Rechenoperationen. Erste solche Verfahren gibt es, allerdings sind sie bislang so aufwendig, daß es kaum eine Rechnung geben dürfte, bei der sich ihr Einsatz lohnt. Die entsprechende Forschung steht aber noch ganz am Anfang; vielleicht wird es schon in naher Zukunft auch praktikable Verfahren geben.

Um zu sehen, daß Rechnen mit verdeckten Daten zumindest grundsätzlich möglich ist, betrachten wir ein extrem einfaches Beispiel: Angenommen, n Personen, die sich gegenseitig vertrauen, wollen ihr Durchschnittsgehalt berechnen, allerdings möchte (oder darf) keiner den anderen sein Gehalt nennen.

In diesem Fall reicht zur „Verschlüsselung“ der Daten die Addition einer zufälligen Zahl: Der erste wählt eine Zufallszahl z und addiert dazu sein Gehalt g_1; das Ergebnis $z_1 = z + g_1$ gibt er weiter an den zweiten. Dieser addiert sein Gehalt g_2 und gibt $z_2 = z_1 + g_2$ weiter an den dritten *usw.* Der letzte schließlich gibt $z_n = z_{n-1} + g_n$ weiter an den ersten. Da

$$z_n = z_{n-1} + g_n = z_{n-2} + g_{n-1} + g_n = \dots = z + g_1 + \dots + g_n$$

ist, kann dieser die Summe der Gehälter berechnen als $z_n - z$, und damit ist auch der Durchschnitt bekannt.

§5: Literatur

Kryptographische Protokolle werden zunehmend auch in Standardlehrbüchern behandelt; ein einfach lesbares relativ dünnes Buch, das sich ausschließlich darauf spezialisiert, ist

ALBRECHT BEUTELSPACHER, JÖRG SCHWENK, KLAUS-DIETER WOLFENSTETTER: *Moderne Verfahren der Kryptographie – Von RSA zu Zero-Knowledge*, Vieweg, ^{7}2010

Dort findet man auch einige Literaturhinweise zur ausführlicheren Beschäftigung mit weitergehenden Verfahren.

Kapitel 9
Kryptologie und Quantenphysik

Computer sind physikalische Systeme, und die Rechnungen, die sie ausführen, sind physikalische Prozesse. Diese Sichtweise ist zwar in der deutschen Hochschulinformatik nicht sehr verbreitet, bestimmte aber den Fortschritt der Informationstechnik in den letzten Jahrzehnten.

Nach einer berühmten, 1965 in der Zeitschrift *Electronics* veröffentlichten Beobachtung des Intel-Mitbegründers GORDON E. MOORE (*1929) verdoppelt sich die Anzahl der Transistoren eines integrierten Schaltkreises etwa alle zwei Jahre. Zumindest bislang beschrieb dies die Entwicklung recht gut, und auch für die für andere Parameter wie die Anzahl der Rechenoperationen pro Sekunden gelten ähnliche Aussagen, die heute allesamt als MOOREsches Gesetz bezeichnet werden.

Möglich war dieser dramatische Anstieg bislang nur durch immer weitere Verkleinerung elektronischer Bauteile; hier bewahrheiteten sich immer wieder Titel und Inhalt eines Vortrags, den der spätere Physik-Nobelpreisträger RICHARD FEYNMAN (1918–1988) bereits 1959 am California Institute of Technology (Caltech) gehalten hatte: *There's Plenty of Room at the Bottom*.

Falls dieser Trend auch künftig anhalten sollte, sind bald Dimensionen erreicht, bei denen nicht mehr die Gesetze der klassischen Physik gelten, sondern die der Quantentheorie.

Zu diesen kommt man allerdings auch auf Grund ganz anderer Überlegungen: In seinem Vortrag *Simulating Physics with Computers* wies FEYNMAN schon 1982 darauf hin, daß eine Simulation beliebiger physikalischer Vorgänge nur möglich sei mit einem quantenmechanischen

System, das *zum Teil* aus sogenannten Quantencomputern besteht (mit denen wir uns in §3 beschäftigen werden). Solche Quantencomputer gab es zwar 1982 noch nicht, und auch heute gibt es noch nicht viel mehr als erste Ansätze zu ihrer Realisierung. Trotzdem ist ziemlich klar, daß die Quantenphysik die Welt der Computer und auch die Kryptologie in den nächsten Jahrzehnten wesentlich verändern wird.

Bevor wir über entsprechende Entwicklungen spekulieren, müssen wir uns allerdings zunächst mit zumindest einigen Grundlagen der Quantenphysik vertraut machen.

§1: Grundzüge der Quantenmechanik

Als zu Beginn des zwanzigsten Jahrhunderts Experimente in immer kleineren Größenordnungen möglich wurden, mußten die Physiker feststellen, daß sich viele Ergebnisse nicht mit den klassischen Gesetze der Physik erklären ließen: In der Welt der Atome und Elementarteilchen ist es beispielsweise nicht mehr möglich, das künftige Verhalten eines Systems aus dem bisherigen Zustand und den Naturgesetzen vorherzusagen, außerdem lassen sich verschiedene physikalische Kenngrößen eines Teilchens wie beispielsweise Ort und Impuls nicht gleichzeitig genau bestimmen. Wie sich bald herausstellte, lag dies nicht nur an technischen Beschränkungen oder unvollständigen Theorien, in denen wesentliche Parameter fehlten: Man konnte zeigen, daß *keine* Theorie mit egal welchen zusätzlichen Parametern den Ausgang der Experimente erklären konnte.

Erste Versuche, das klassische Weltbild durch *ad hoc* Postulate (meist in Form irgendwelcher Verbote) zu retten, führten zu keinen befriedigenden Ergebnissen; dazu kam es erst, als ab etwa 1925 aus den Arbeiten verschiedener Physiker ein radikal neuer Erklärungsansatz entstand, die Quantenmechanik. Sie ist sehr viel weniger anschaulich als die klassische Physik und widerspricht auch teilweise unseren Vorstellungen, hat sich aber seither in unzähligen Experimenten bestätigt und führte auch bereits zu praktischen Anwendungen, wie beispielsweise der im nächsten Paragraphen behandelten Quantenkryptographie. Die klassi-

sche Physik kann auf der Grundlage der Quantenphysik erklärt werden als deren Limes für immer größer werdende Teilchenzahl.

Grundlage der Beschreibung eines quantenmechanischen Systems sind *nicht* wie in der klassischen Physik die Werte verschiedener Meßgrößen, die den Bestandteilen des Systems zugeordnet sind; stattdessen wird der Zustand des gesamten Systems beschrieben durch einen einzigen Vektor, der im allgemeinen *nicht* bestimmt werden kann.

Der *Zustandsraum*, in dem diese Vektoren liegen, ist ein (im allgemeinen unendlichdimensionaler) Vektorraum über dem Körper $\mathbb{C}$ der komplexen Zahlen mit einem HERMITEschen Skalarprodukt; bevor wir uns mit Einzelheiten befassen, müssen wir also zunächst dieses Produkt verstehen.

HERMITEsche Skalarprodukte sind das Analogon der bekannten EUKLIDischen Skalarprodukte für Vektoren mit komplexen Einträgen. Mit dem Standardskalarprodukt auf $\mathbb{R}^2$ ist

$$\begin{pmatrix} a \\ b \end{pmatrix} \begin{pmatrix} c \\ d \end{pmatrix} = ac + bd\,.$$

Für komplexe Zahlen a, b, c, d würde dies beispielsweise bedeuten, daß $\binom{1}{i}\binom{1}{i} = 1^2 + i^2 = 0$ wäre, ein vom Nullvektor verschiedener Vektor hätte also die Länge Null. Dies widerspricht allen unseren Vorstellungen von einem Skalarprodukt und muß daher vermieden werden. Die einfachste Lösung besteht darin, daß wir die Einträge des zweiten Vektors vor der Multiplikation komplex konjugieren: Mit

$$\begin{pmatrix} a \\ b \end{pmatrix} \odot \begin{pmatrix} c \\ d \end{pmatrix} \underset{\text{def}}{=} a\overline{c} + b\overline{d}$$

ist $\binom{1}{i} \odot \binom{1}{i} = 1 \cdot 1 + i \cdot (-i) = 2$ eine positive Zahl, wie es sein soll.

Was die Bezeichnung angeht, haben Physiker eine andere Konvention als Mathematiker; sie geht zurück auf den englischen Physik-Nobelpreisträger PAUL DIRAC (1902–1984). Danach schreibt man das Produkt in der Form $\langle u|v\rangle$ mit einer spitzen Klammer, englisch *bracket*. Die Vektoren aus dem Zustandsraum werden in der Form $|v\rangle$ geschrieben und als *kets* bezeichnet.

Diese Schreibweise mag einem Mathematiker auf den ersten Blick verwirrend erscheinen; er sollte sich aber klarmachen, daß es hier nur um ein Symbol geht und sich an der Mathematik natürlich nichts ändert, egal ob man einen Vektor als $\mathfrak{v}$, $\vec{v}$ oder eben $|v\rangle$ schreibt.

Die formale Definition eines HERMITEschen Skalarprodukts ist

Definition: Ein HERMITEsches Skalarprodukt auf einem komplexen Vektorraum V ist eine Abbildung

$$\langle\cdot\,|\,\cdot\rangle\colon V \times V \to \mathbb{C}; \quad \big(|v\rangle, |w\rangle\big) \mapsto \langle v|w\rangle$$

mit folgenden Eigenschaften:

1.) $\langle \lambda v_1 + \mu v_2|w\rangle = \lambda\,\langle v_1|w\rangle + \mu\,\langle v_2|w\rangle$

2.) $\langle v|w\rangle = \overline{\langle w|v\rangle}$

3.) $\langle v|v\rangle \geq 0$ und $\langle v|v\rangle = 0$ genau dann, wenn $|v\rangle$ der Nullvektor ist.

Typisches Beispiel eines endlichdimensionalen Vektorraums mit einem HERMITEschen Skalarprodukt ist der $\mathbb{C}^n$ mit dem Produkt

$$\left\langle \begin{pmatrix} z_1 \\ \vdots \\ z_n \end{pmatrix} \middle| \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix} \right\rangle = z_1\overline{w}_1 + \ldots + z_n\overline{w}_n\,.$$

Die ersten beiden Forderungen sind trivialerweise erfüllt, und für die dritte müssen wir nur beachten, daß

$$\left\langle \begin{pmatrix} z_1 \\ \vdots \\ z_n \end{pmatrix} \middle| \begin{pmatrix} z_1 \\ \vdots \\ z_n \end{pmatrix} \right\rangle = z_1\overline{z}_1 + \ldots + z_n\overline{z}_n = |z_1|^2 + \ldots + |z_n|^2$$

als Summe von Betragsquadraten eine nichtnegative reelle Zahl ist, die genau dann verschwindet. wenn alle z_i verschwinden.

Man beachte, daß ein HERMITEsches Skalarprodukt nur in seinem ersten Argument linear ist; für das zweite Argument haben wir nach den Forderungen 1.) und 2.) die Beziehung

$$\begin{aligned}\langle v, \lambda w_1 + \mu w_2\rangle &= \overline{\langle \lambda w_1 + \mu w_2, v\rangle} = \overline{\lambda\,\langle w_1, v\rangle + \mu\,\langle w_2, v\rangle} \\ &= \overline{\lambda}\,\overline{\langle w_1, v\rangle} + \overline{\mu}\,\overline{\langle w_2, v\rangle} = \overline{\lambda}\,\langle v, w_1\rangle + \overline{\mu}\,\langle v, w_2\rangle\,.\end{aligned}$$

Ein HERMITEsches Skalarprodukt ist somit nicht bilinear, hat aber doch auch im zweiten Argument eine linearitätsähnliche Eigenschaft; man spricht hier von einer *Sesquilinearform*, d.h. von „anderthalbfacher“ Linearität.



CHARLES HERMITE (1822–1901) war einer der bedeutendsten Mathematiker des neunzehnten Jahrhunderts. Zu seinen Resultaten zählen eine Vereinfachung des ABELschen Beweises, daß Gleichungen fünften Grades im allgemeinen nicht durch Wurzelausdrücke gelöst werden können, die explizite Lösung solcher Gleichungen durch elliptische Funktionen, der Nachweis, daß e eine transzendente Zahl ist, also keiner algebraischen Gleichung über $\mathbb{Q}$ genügt, eine Interpolationsformel und vieles mehr. HERMITE galt als ein sehr guter akademischer Lehrer; er unterrichtete an der École Polytechnique, dem Collège de France, der École Normale Supérieure und der Sorbonne.

In der Quantenmechanik werden die Zustände eines Systems beschrieben durch die Vektoren $|v\rangle \neq |0\rangle$ aus einem komplexen Vektorraum V; dabei sollen allerdings zueinander proportionale Vektoren denselben Zustand beschreiben. Tatsächlich entsprechen die möglichen Zustände somit den eindimensionalen Untervektorräumen von V; die Menge aller dieser Unterräume bezeichnet man auch als den projektiven Raum $\mathbb{P}(V)$. Für praktische Rechnungen betrachten Physiker allerdings immer konkrete Vektoren; da es auf deren Längen nicht ankommt, werden diese oft auf eins normiert.

Meßbare physikalische Größen werden durch sogenannte HERMITEsche Operatoren beschrieben, das sind lineare Abbildungen $A\colon V \to V$, für die gilt $\langle Av|Aw\rangle = \langle v|w\rangle$ für alle $|v\rangle, |w\rangle \in V$. Schreibt man den Operator A im endlichdimensionalen Fall bezüglich einer Orthonormalbasis als Matrix, so ist das äquivalent zur Gleichung $A^T = \overline{A}$. Falls alle Einträge von A reell sind, bedeutet das einfach, daß A eine symmetrische Matrix ist; im Komplexen reden wir von einer HERMITEschen Matrix.

Bekanntlich sind alle Eigenwerte einer symmetrischen reellen Matrix reell; dies gilt auch für HERMITEsche Matrizen: Ist nämlich $\lambda \in \mathbb{C}$ ein Eigenwert von A mit Eigenvektor $|v\rangle$, so ist

$$\langle v|Av\rangle = \langle v|\lambda v\rangle = \overline{\lambda}\,\langle v|v\rangle \ .$$

Andererseits können wir den Vektor $|v\rangle$ auch als eine einspaltigen Matrix v auffassen; dann ist

$$\langle v|Av\rangle = v^T\overline{A}\overline{v} = v^TA^T\overline{v} = (Av)^T\overline{v} = (\lambda v)^T\overline{v} = \langle \lambda v|v\rangle = \lambda\,\langle v|v\rangle \ .$$

Somit ist $\overline{\lambda}\,\langle v|v\rangle = \lambda\,\langle v|v\rangle$, was wegen $\langle v|v\rangle \neq 0$ nur dann möglich ist, wenn $\lambda = \overline{\lambda}$ reell ist.

Genauso wie im Reellen folgt auch, daß eine HERMITEsche Matrix stets diagonalisierbar ist und daß Eigenvektoren zu verschiedenen Eigenwerten orthogonal bezüglich des HERMITEschen Skalarprodukts sind.

Wie bereits erwähnt, ist das Ergebnis einer Messungen in der Quantenphysik nicht allein durch den Zustand des Systems festgelegt. Dies wird mathematisch wie folgt modelliert:

Angenommen, das System befindet sich vor der Messung im Zustand, der durch den Einheitsvektor $|v\rangle$ beschrieben wird, und wir messen die physikalische Größe, die durch die Matrix A beschrieben wird. Dann ist das Ergebnis der Messung *einer* der Eigenwerte λ von A, und das System befindet sich anschließend in einem Zustand, der durch einen Eigenvektor $|w\rangle$ zum Eigenwert λ beschrieben wird. Dies entspricht der experimentell gut belegten Tatsache, daß man in einem quantenmechanischen System nichts messen kann, ohne dabei den Zustand des Systems zu verändern.

Auch wenn wir das Ergebnis einer Messung nicht mit Sicherheit voraussagen können, können wir doch immerhin die Wahrscheinlichkeiten der möglichen Meßergebnisse angeben: Die Wahrscheinlichkeit dafür, daß sich das System nach der Messung in dem Zustand befindet, der durch einen Eigenvektor $|w\rangle$ der Länge eins beschrieben wird, ist das Quadrat des Skalarprodukts $\langle v|w\rangle$. Falls $|w\rangle$ zu einem Eigenwert der Vielfachheit eins gehört, ist das auch die Wahrscheinlichkeit dafür, daß wir diesen Eigenwert als Meßergebnis erhalten; bei höherer Vielfachheit müssen die entsprechenden Wahrscheinlichkeiten für die Vektoren aus einer Basis des Eigenraums aufaddiert werden.

Das einzige quantenmechanische System, mit dem wir uns in diesem Kapitel näher beschäftigen werden, ist ein polarisiertes Photon. Polarisiertes Licht ist zumindest einigen Lesern vielleicht aus der Photographie

vertraut: Dort vermeidet man störende Reflexe (z.B. auf Fensterscheiben oder Metallflächen), indem man ein Polarisationsfilter auf das Objektiv schraubt und es in eine geeignete Richtung dreht.

Licht besteht aus ebenen Wellen, wobei im Raumlicht üblicherweise alle Schwingungsrichtungen vertreten sind, während reflektiertes Licht in einer festen Ebenen schwingt. Ein Polarisationsfilter läßt nur Licht einer Schwingungsebenen ungehindert passieren; Licht mit dazu senkrechter Schwingungsebene wird vollständig, sonstiges Licht teilweise absorbiert. Zur Vermeidung von Reflexen muß man also nur das Filter in eine Richtung drehen, in der es das reflektierte Licht nicht durchläßt oder zumindest hinreichend stark abdämpft.

Nach der Quantentheorie ist Licht zusammengesetzt aus sogenannten Photonen. Diese kleinsten Bestandteile haben ebenfalls eine Schwingungsebene; da es sich um kleinste Teile handelt, kann sie ein Polarisationsfilter aber nicht *teilweise* durchlassen, sondern entweder ganz oder gar nicht. Dieses Phänomen wollen wir hier quantenmechanisch beschreiben:

Der Zustandsraum für ein polarisiertes Photon ist der Vektorraum $\mathbb{C}^2$ mit seinem üblichen HERMITEschen Skalarprodukt. Der Zustand eines Photons mit Schwingungsrichtung φ wird beschrieben durch den Vektor $\begin{pmatrix}\cos\varphi\\ \sin\varphi\end{pmatrix}$ (und seine Vielfachen). Eine Messung die bestimmt, ob das Photon durch ein Polarisationsfilter mit Durchlassrichtung ψ kommt, wird beschrieben durch den Operator

$$A_\psi = \begin{pmatrix}\cos 2\psi & \sin 2\psi\\ \sin 2\psi & -\cos 2\psi\end{pmatrix} = \begin{pmatrix}\cos^2\psi-\sin^2\psi & 2\sin\psi\cos\psi\\ 2\sin\psi\cos\psi & \sin^2\psi-\cos^2\psi\end{pmatrix}.$$

Dessen charakteristisches Polynom ist

$$\begin{aligned}\det(A_\psi-\lambda E) &= (\cos 2\psi-\lambda)(-\cos 2\psi-\lambda)-\sin^2 2\psi\\ &= \lambda^2-\cos^2 2\psi-\sin^2 2\psi-1 = \lambda^2-1\,;\end{aligned}$$

wir haben also zwei mögliche Meßergebnisse +1 und −1 entsprechend der Tatsache, daß das Photon entweder durchgelassen wird oder nicht.

Die Eigenvektoren dazu lassen sich ebenfalls leicht bestimmen: Nach der zweiten Darstellung von A_ψ ist

$$\begin{aligned}A_\psi - E &= \begin{pmatrix}\cos^2\psi-\sin^2\psi-1 & 2\sin\psi\cos\psi\\ 2\sin\psi\cos\psi & \sin^2\psi-\cos^2\psi-1\end{pmatrix}\\ &= \begin{pmatrix}-2\sin^2\psi & 2\sin\psi\cos\psi\\ 2\sin\psi\cos\psi & -2\cos^2\psi\end{pmatrix},\end{aligned}$$

und der Lösungsraum des homogene lineare Gleichungssystem dazu wird offensichtlich erzeugt vom Vektor $\begin{pmatrix}\cos\psi\\ \sin\psi\end{pmatrix}$; genauso ist

$$\begin{aligned}A_\psi + E &= \begin{pmatrix}\cos^2\psi-\sin^2\psi+1 & 2\sin\psi\cos\psi\\ 2\sin\psi\cos\psi & \sin^2\psi-\cos^2\psi+1\end{pmatrix}\\ &= \begin{pmatrix}2\cos^2\psi & 2\sin\psi\cos\psi\\ 2\sin\psi\cos\psi & 2\sin^2\psi\end{pmatrix};\end{aligned}$$

hier wird der Lösungsraum erzeugt von

$$\begin{pmatrix}-\sin\psi\\ \cos\psi\end{pmatrix} = \begin{pmatrix}\cos\left(\psi+\frac{\pi}{2}\right)\\ \sin\left(\psi+\frac{\pi}{2}\right)\end{pmatrix};$$

wie zu erwarten, schwingt also ein durchgelassenes Photon anschließend in Durchlassrichtung des Polarisationsfilters und ein absorbiertes in der dazu senkrechten Sperrichtung.

Mit diesen Vorbereitungen sind wir zwar weit entfernt von einem auch nur rudimentären Verständnis der Quantentheorie, sie sollten aber ausreichen, um den Rest dieses Kapitels zumindest im Prinzip zu verstehen.

§ 2: Quantenkryptographie

Die Quantenkryptographie ist die am wenigsten spekulative Anwendung der Quantenphysik auf die Kryptographie; sie wurde bereits in vielen Versuchen erfolgreich getestet, und die dazu benötigten Geräte sind kommerziell erhältlich. Die Reichweiten sind zwar bislang auf unter etwa 150 Kilometer begrenzt, so daß sich Anwendungen auf Bereiche wie Washington, D.C. oder die Londoner City beschränken; bereits ein realistisches Nahziel sind aber Experimente mit niedrig fliegenden

Satelliten, mit deren Hilfe ein globales Netzwerk aufgebaut werden könnte.

Von allen Kryptosystemen, die wir bislang kennengelernt haben, hat nur der *one time pad* beweisbare absolute Sicherheit. Das liegt nicht nur daran, daß wir nur wenige Kryptosysteme kennen: Nach dem in Kapitel 3, §1 erwähnten Satz von SHANNON muß ein Kryptosystem, das absolute Sicherheit bietet, mit Schlüsseln arbeiten, die mindestens so lang sind wie die Summe aller damit übertragenen Nachrichten. Solche Schlüssel lassen sich mit klassischen Methoden nur schwer und aufwendig vereinbaren. Die Quantenkryptographie stellt ein Protokoll zur Verfügung, mit dem zwei räumlich getrennte Partner solche Schlüssel über eine unsichere Leitung so vereinbaren können, daß ein Lauscher mit einer beliebig nahe bei eins liegenden vorgegebenen Wahrscheinlichkeit kein einziges Bit an Information erhalten kann.

a) Informationsübertragung mit einzelnen Photonen

Die Grundidee des Verfahrens besteht darin, die Bits durch einzelne polarisierte Photonen zu kodieren und beispielsweise vereinbaren, daß eine horizontale Schwingungsebene für eine Eins und eine vertikale für eine Null stehen soll. Der Empfänger stellt dann sein Polarisationsfilter horizontal; falls er dahinter ein Photon mißt, wurde eine Null gesendet, ansonsten eine Eins.

Praktisch werden die Photonen meist approximiert durch sehr kurze Lichtblitze, die mit hoher Wahrscheinlichkeit nicht mehr als ein Photon enthalten, weil sie beispielsweise so kurz sind, daß sie nur mir Wahrscheinlichkeit $1/10$ überhaupt ein Photon enthalten. Die Wahrscheinlichkeit für zwei oder mehr Photonen in einem nichtleeren Lichtblitz liegt dann bei etwa 6%, was tolerierbar ist.

Da drehbare Polarisationsfilter nur langsam auf eine neue Richtung eingestellt werden können, verwendet man in der Praxis andere Methoden: Beispielsweise läßt sich der POCKELS-Effekt aus der nichtlinearen Optik ausnutzen, wonach gewisse anisotrope Kristalle beim Anlegen einer Spannung ihren Brechungsindex ändern, oder aber man verwendet (da die Spannung beim POCKELS-Effekt typischerweise in der Größenordnung einiger hundert Volt liegen muß, was sich nicht sonderlich schnell

schalten läßt) für jede gewünschte Polarisationsrichtung eine eigene Laserdiode oder sonstige geeignete Lichtquelle mit dahinterstehendem Polarisationsfilter und vereinigt anschließend die Strahlengänge. Auf diese Weise lassen sich Lichtblitze mit einer Frequenz von bis zu 1 GHz erzeugen.

Da nicht jeder Puls ein Photon enthält und auch während der Übertragung Photonen verloren gehen, muß der Empfänger zunächst wissen, *ob* ein Photon angekommen ist; außerdem muß er dann dessen Polarisationsrichtung bestimmen. Dazu läßt er das Photon durch einen doppelbrechenden Kristall (z.B. einen Kalkspat) gehen; je nachdem ob es parallel oder senkrecht zu dessen optischer Achse polarisiert ist, verläßt es den Kristall an einer von zwei wohldefinierten Stellen, hinter denen Photomultiplikatoren und Detektoren stehen, so daß für jede der beiden Polarisationsrichtungen ein Detektor anspricht. Natürlich müssen Sender und Empfänger synchronisiert werden; dies geschieht beispielsweise dadurch, daß eine festgelegte Zeitspanne vor jedem Puls ein heller Lichtblitz gesendet wird, der garantiert ankommt.

Entsprechende Apparaturen wurden erstmals im Oktober 1989 experimentell getestet, damals über eine Entfernung von nur 32 cm. Inzwischen ist man bei Glasfaserkabeln einer Länge von über 100 km angelangt, sowohl bei aufgerollter Glasfaser im Labor als auch bei „echten“ Glasfaserverbindungen wie etwa der der Swiss Telekom von Genf nach Nyon (22,8 km) unter dem Genfer See. Für eine Vernetzung innerhalb einer Stadt ist Quantenkryptographie also bereits heute praktikabel. Entsprechende Strecken über Tausende von Kilometern erscheinen nach derzeitigem Stand der Technik unwahrscheinlich wegen der Absorption in Glasfaserkabeln: Bei den für Telekommunikation typischen Wellenlängen um 1300 und 1550 nm hat man eine Dämpfung von 0,35 beziehungsweise 0,2 dB/km, so daß nach knapp 30 beziehungsweise 50 km nur noch ein Zehntel der abgeschickten Photonen übriggeblieben ist. Für Photonen mit nur 800 nm Wellenlänge, die sich einfacher detektieren lassen, beträgt die Dämpfung sogar 2 dB/km, man hat also schon nach fünf Kilometern neun von zehn Photonen verloren.

Verglichen mit anderen Kryptosystemen, die auch für die drahtlose Übermittlung von Nachrichten benutzt werden können, ist die Notwendigkeit

einer Glasfaserverbindung überhaupt ein Nachteil: Besser wäre es, wenn die Photonen kabellos übertragen werden könnten.

Das Problem hierbei ist natürlich, daß es in der Luft, vor allem bei Tageslicht, bereits ziemlich viele Photonen gibt. Das ist allerdings nicht ganz so schlimm, wie es auf den ersten Blick aussieht, denn bei hinreichender Sorgfalt kann man den Ort, die Zeit und die Wellenlänge der übermittelten Photonen sehr genau festlegen. Bei einer Wellenlänge von etwa 770 nm (im infraroten Bereich also) ist die Atmosphäre auch so durchlässig, daß sich die Verlustrate in Grenzen hält.

Wissenschaftler der National Laboratories in Los Alomos testeten drahtlose Quantenkryptographie erstmals am 13. August 1999 von 9^{30} bis 11^{30} Uhr unter dem wolkenlosen blauen Himmel von New Mexico über eine Entfernung von 1,6 km; 2002 testeten Physiker aus München und Innsbruck kabellose Quantenkryptographie erfolgreich zwischen dem Gipfeln der Zugspitze und der Westlichen Karwendelspitze, d.h. über eine Entfernung von 23,4 km Luftlinie, wobei sie auf eine Netto-Bitrate von 1000-1500 gemeinsamen Bits pro Sekunde kamen, und 2006 schließlich testete eine Gruppe von Physikern aus München, Wien Singapur, Bristol und von der ESA Freiluft-Quantenkryptographie erfolgreich über eine Strecke von 144 km zwischen La Palma und Teneriffa, allerdings nur mit einer Ausbeute von 12,8 Bit pro Sekunde. Fernziel ist die magische Grenze von 30 km, die es erlauben würde, Photonen an Satelliten in erdnahen Umlaufbahnen zu schicken. Ein solcher Satellit ist zwar von einem festen Punkt der Erde aus nur etwa acht Minuten pro Tag in direkter Sichtverbindung; diese Zeit würde aber, bei den angestrebten Datenraten, ausreichen, um einen Schlüssel zu vereinbaren, mit dem sich deutlich mehr als die typischerweise in einem Unternehmen innerhalb von 24 Stunden anfallenden Nachrichten vom und zum Satelliten verschlüsseln lassen.

Die erste bekanntgewordene „praktische“ Anwendung der Quantenkryptographie war bei der Schweizer Nationalratswahl am 21. Oktober 2007: Ein Wahllokal bei Genf übertrug seine Auszählungsergebnisse via Quantenkryptographie an die vier Kilometer entfernte Zentrale des Kantons. Da die Stimmzettel öffentlich ausgezählt werden und die Ergebnisse am nächsten Tag in der Zeitung stehen, dürfte allerdings auch hier der

kryptographische Aspekt im Hintergrund gestanden haben; in erster Linie ging es wohl um eine Werbemaßnahme der Firma *id Quantique* (einem Spin-off Unternehmen der Universität Genf), die den Verkauf ihrer Technologie ankurbeln wollte. Im Oktober 2008 wurde in Wien im Rahmen des Forschungsprojekts SECOQC ein quantenkryptographisches Netzwerk aus sechs Knoten und acht Verbindungen auf dem Glasfasernetz von Siemens Wien demonstriert.

b) Protokolle zur Quantenkryptographie

Natürlich wurde bei keinem der vorgestellten Experimente genau das oben skizzierte Verfahren getestet, denn so wie beschrieben bietet es keinerlei kryptographische Sicherheit: Ein Gegner könnte einfach alle Photonen abfangen und neue auf die Reise schicken; falls er diese genauso polarisiert, wie die abgefangenen, wird weder der Sender noch der Empfänger etwas bemerken.

Um dies zu verhindert, arbeitet der Sender nicht nur mit vertikal oder horizontal polarisierten Photonen sondern auch mit solchen in Schwingungsebenen von 45° und 135°. Wie wir in §1 gesehen haben, wird ein Photon, dessen Schwingungsebene um 45° gegenüber der Durchlassrichtung des Polarisationsfilters gedreht ist, jeweils mit Wahrscheinlichkeit ein halb durchgelassen oder nicht.

Der erste Ansatz, dies für kryptographische Zwecke auszunutzen, stammt von CHARLES BENNET und GILLES BRASSARD aus dem Jahr 1984; er wird heute kurz als das BB84-Protokoll bezeichnet. Hier entscheidet sich der Sender vor der Übertragung eines jeden Photons zufällig für ein Referenzsystem, bestehend entweder aus den Richtungen 0° und 90°, oder aus den Richtungen 45° und 135°. Danach entscheidet er sich, wiederum zufällig, für einen der beiden Bitwerte 0 oder 1 und kodiert beispielsweise die Eins durch Polarisationsrichtung 0° oder 45°, die Null durch 90° oder 135°.

Praktisch können die Zufallsentscheidungen beispielsweise durch Digitalisieren von weißem Rauschen erfolgen oder aber indem man polarisierte Photonen durch ein um 45° gedrehtes Filter schießt und mißt, welche transmittiert werden.

Auch der Empfänger wählt für jedes Bit zufällig eines der beiden Referenzsysteme; nimmt er dasselbe System wie der Absender, was etwa in der Hälfte der Fälle vorkommt, kann er das abgeschickte Bit messen, andernfalls erhält er ein Zufallsergebnis.

Die folgende Tabelle faßt die verschiedenen Möglichkeiten für Sender und Empfänger noch einmal zusammen:

gesendet wird:	0°	0°	45°	45°	90°	90°	135°	135°
empfangen mit:	+	×	+	×	+	×	+	×
Meßergebnis:	0°	**?**	**?**	45°	90°	**?**	**?**	135°

Hier bedeutet „+“, daß der Empfänger seinen Doppelspat so orientiert, daß er 0° und 90°-Polarisierung messen kann, während „×“ entsprechend für das um 45° gedrehte Bezugssystem steht. Ein „**?**“ in der letzten Zeile soll besagen, daß hier das Meßergebnis nur vom Zufall abhängt und somit keine sinnvolle Information enthält.

Um festzustellen, welche Hälfte der gemessenen Bits sinnvolle Information enthält, informiert der Empfänger den Sender anschließend über einen gewöhnlichen, nicht abhörsicheren Kanal, welche Photonen angekommen sind und mit welchem Referenzsystem er sie gemessen hat. Der Sender teilt ihm dazu jeweils mit, ob dies die richtige Wahl war oder nicht. Dieser Austausch soll im folgenden kurz als die „öffentliche“ Diskussion bezeichnet werden.

Für die Photonen, bei denen beide mit demselben Referenzsystem gearbeitet haben, notiert sich der Sender den gesendeten und der Empfänger den gemessenen Bitwert; bis auf allfällige Übertragungsfehler sollten diese somit beide dieselbe Bitfolge notieren. Der Lauscher, der nachträglich zwar das korrekte Referenzsystem erfahren hat, nicht aber den übertragenen Bitwert, kann zumindest aus der „öffentlichen“ Diskussion nichts darüber erfahren. Seine sonstigen Möglichkeiten sollen weiter unten erörtert werden.

Zunächst aber soll hier noch das 1992 von BENNET vorgeschlagene logisch und technisch etwas einfachere B92-Protokoll vorgestellt werden. Hier verwendet der Sender nur die beiden Polarisationsrichtungen 0° für Null und 45° für Eins; der Empfänger nur 135° für Null und 90° für Eins.

Insbesondere braucht der Empfänger also keinen doppelbrechenden Kristall mehr mit zwei Detektoren, sondern nur noch eine POCKELS-Zelle mit *einem* nachgeschalteten Detektor. Die möglichen Meßergebnisse sind in folgender Tabelle zusammengefaßt:

gesendet wird	0°	0°	45°	45°
gemessen wird mit	90°	135°	90°	135°
Photon wird detektiert?	nein	**?**	**?**	nein

Das „**?**“ in der letzten Zeile soll dabei bedeuten, daß der Empfänger mit gleicher Wahrscheinlichkeit ein Photon mißt oder auch nicht.

Es gibt also nur zwei Kombinationen, bei denen der Empfänger überhaupt ein Photon finden kann: Falls der Empfänger mit 0° gesendet und der Empfänger mit 135° gemessen hat, oder wenn der Empfänger mit 45° gesendet und der Empfänger mit 90° gemessen hat. Falls ein Photon gemessen wird, weiß der Empfänger also, mit welcher Polarisationsrichtung gesendet wurde und kann je nachdem eine Null oder Eins notieren.

Im anschließenden „öffentlichen“ Dialog muß er dem Sender dann nur mitteilen, in welchen Positionen er Photonen gemessen hat, so daß auch der sich die Bitwerte dafür notieren kann. Man beachte, daß bei diesem Protokoll nur jedes vierte übermittelte Photon zu einem Bitwert führt.

Mittlerweile wurde auch ein völlig verschiedener Ansatz zur Quantenkryptographie getestet, die Verwendung sogenannter EPR-Paare. EPR steht für EINSTEIN-PODOLSKY-ROSEN und bezieht sich auf ein von diesen drei Physikern entdecktes Paradoxon, das EINSTEIN zunächst an der Richtigkeit der Quantentheorie zweifeln ließ:

Angenommen, bei einem Experiment werden simultan zwei Photonen gleicher Polarisierung erzeugt; die Polarisierung sei jeweils eine Überlagerung der beiden Zustände die wir als $|0\rangle$ und $|1\rangle$ bezeichnen. Der Zustand des System ist dann $\frac{\sqrt{2}}{2}|00\rangle + \frac{\sqrt{2}}{2}|11\rangle$. Nun fliegen die beiden Photonen in entgegengesetzte Richtungen und das eine davon wird gemessen. Danach ist es entweder im Zustand $|0\rangle$ oder im Zustand $|1\rangle$, und da beide Photonen ein quantenmechanisches System aus identisch polarisierten Photonen bilden, ist der Zustand des Gesamtsystems entweder

$|00\rangle$ oder $|11\rangle$. Wird also kurz darauf auch das andere Photon gemessen, erhält man notwendigerweise denselben Wert wie beim ersten Photon. Dies gilt auch dann, wenn erst durch die Messung des ersten Photons festgelegt wird, was wir als $|0\rangle$ und was als $|1\rangle$ interpretieren wollen.

Sobald das erste Photon gemessen wird (und dadurch seinen Zustand ändert) muß also auch das zweite Photon, egal wie weit es inzwischen entfernt ist, seinen Zustand ändern. Dies widerspricht der Relativitätstheorie, wonach keine Information mit einer größeren Signalgeschwindigkeit als der Vakuumlichtgeschwindigkeit übertragen werden kann. Es ist allerdings inzwischen vielfach experimentell verifiziert und ist auch nur einer der Punkte, bei denen die Physiker noch große Probleme haben, die für ihre Hauptanwendungsgebiete hervorragend bestätigten Gesetze der Quantentheorie und der Relativitätstheorie unter einen Hut zu bringen: Die Suche nach der *großen vereinheitlichten Feldtheorie* hat zwar bereits eine ganze Reihe vielversprechender Ansätze hervorgebracht, von einem Durchbruch ist die Physik aber noch weit entfernt.

Für die Quantenkryptographie bedeutet das EPR-Paradoxon, daß man auch von der Mitte der Leitung aus EPR-Paare auf den Weg schicken kann. Diese tragen unterwegs noch keinerlei Information (es sei denn, ein Lauscher erzwingt das), sondern bekommen diese erst, wenn an einem Ende der Leitung gemessen wird. Um Angriffe des Lauschers entdecken zu können, muß man auch hier mit zwei Referenzsystemen arbeiten, also z.B. mit einem der beiden oben beschriebenen Protokolle.

Wiener Physiker testeten diese Art der Quantenkryptographie erfolgreich für die Übermittlung einer (wahrscheinlich nicht realen) Banküberweisung durch die Wiener Kanalisation unter der Donau hindurch. (Daß der Versuch in der Wiener Kanalisation stattfand, hat wohl weniger mit dem Kinoklassiker „Der dritte Mann“ zu tun als damit, daß die Wiener Stadtverwaltung entdeckt hat, daß sie mit ihrem weitverzweigten Kanalnetz Geld verdienen kann, wenn sie es für die Verlegung von Glasfaserkabeln zu Verfügung stellt.)

c) Angriffsmöglichkeiten

Sowohl beim BB84- als auch beim B92-Protokoll kennen Sender und Empfänger am Ende eine Bitfolge, über die ein Gegner zumindest durch

Abhören der „öffentlichen“ Diskussion nichts in Erfahrung bringen kann. Er hat aber natürlich auch noch die Möglichkeit, sich in den anfänglichen Photonenaustausch einzuschalten.

Wie wir aus §1 wissen, hat er aber keine Möglichkeit, ein Photon zu messen, ohne dessen Zustand zu verändern. Falls er beispielsweise ein mit 0°-Polarisierung gesendetes Photon mit Durchlassrichtung 45° mißt, hat er anschließend mit gleicher Wahrscheinlichkeit ein mit 45° oder 135° polarisiertes Photon; er weiß aber nicht, ob das Photon wirklich mit dieser Polarisierung ankam, oder ob es eine der beiden Polarisierungsrichtungen 0° und 90° hatte.

Hier ist extrem wichtig, daß mit einzelnen Photonen gearbeitet wird: Falls zwei Photonen im selben Zustand übertragen werden, kann man sie durch einen Strahlteiler trennen und jedes in einem anderen Referenzsystem messen; beim BB84-Protokoll wird dann in der „öffentlichen“ Diskussion klar, welche der Messungen zum richtigen Bitwert führte, beim B92-Protokoll ist er in drei Viertel aller Fälle sofort klar, da eine Polarisationsrichtung von 90° im +-System nur gemessen werden kann, wenn das Photon im $\times$-System übertragen wurde, d.h. mit Polarisationsrichtung 45°. Entsprechend kann 135° im $\times$-System nur gemessen werden kann, wenn das Photon mit 0° polarisiert war. Lediglich bei der Kombination $(0^\circ, 45^\circ)$ ist nicht klar, welches Bit der Sender übermitteln wollte. Da Photonen durch kurze Lichtblitze realisiert werden, lassen sich jedoch Blitze mit mehr als einem Photon nicht vollständig vermeiden, man muß also damit rechnen, daß ein Gegner unbemerkt eine gewisse Anzahl von Photonen messen kann. Eine Modifikation des BB84-Protokolls, das sogenannte SARG04-Protokoll, erschwert dies, da ein Gegner dort zum unbemerkten Abhören *zwei* Photonen aus einem Blitz entnehmen muß, so daß ihm nur Blitze mit mindestens drei Photonen etwas nützen, aber auch die kommen natürlich vor.

Bei Blitzen, die nur ein Photon enthalten, muß der Gegner dieses messen und anschließend wieder ein Photon auf die Reise schicken. Am sichersten wäre es, ein Photon zu senden, das genau dieselbe Polarisationsrichtung hat wie das abgefangene; in diesem Fall bliebe sein Lauschen unbemerkt. Diese Möglichkeit wird aber durch die Quantentheorie ausgeschlossen, da er den Zustand des Photons nicht vollständig

bestimmen kann.

Am wenigsten verfälscht er beim BB84-Protokoll, wenn er ein Photon losschickt, das die von ihm gemessene Polarisationsrichtung hat: Falls er im richtigen Referenzsystem maß, bleibt sein Eingriff bei diesem Photon unbemerkt, andernfalls gibt es immerhin noch eine 50%-ige Wahrscheinlichkeit, daß der Empfänger, falls er im richtigen System mißt, den korrekten gesendeten Wert mißt. (Falls auch der Empfänger im falschen System mißt, wird das Bit nicht verwendet, so daß es auf seinen Wert nicht ankommt.) In etwa einem Viertel aller Fälle sorgt der Lauscher durch seinen Eingriff dafür, daß der Empfänger trotz gleichen Referenzsystems einen anderen Bitwert mißt als den ursprünglich gesendeten.

Beim B92-Protokoll kann ein Lauscher, der wie der Empfänger beim B84-Protokoll mit Doppelbrechung arbeitet, jedes zweite Bit messen: Falls er im +-System 90° mißt, muß das Photon mit 45° polarisiert gewesen sein, wenn er im ×-System 135° mißt, mit 0°. In diesem Fall kann er ein entsprechendes Ersatzphoton schicken. In der anderen Hälfte der Fälle muß er aber raten und verfälscht somit auch hier wieder insgesamt rund ein Viertel aller übertragener Bitwerte.

d) Fehlerkorrektur

Unabhängig von der Aktivität eines Lauschers entstehen auch aus rein physikalischen Gründen Fehler; eine Fehlerkorrektur ist also auf jeden Fall notwendig. Dazu muß als erstes die Fehlerrate abgeschätzt werden. Sender und Empfänger einigen sich daher in der „öffentlichen“ Diskussion auf eine Zufallsstichprobe der notierten Bits und vergleichen diese; durch Vergleich der so geschätzte Fehlerrate mit der aus physikalischen Gründen zu erwartende können sie auch abschätzen, wieviel Information in die Hände von Gegnern gefallen sein kann.

Vor der Fehlerkorrektur wird natürlich die Stichprobe aus der Bitfolge eliminiert: Diese Bits wurden schließlich in der „öffentlichen“ Diskussion verglichen und sind somit dem Gegner bekannt.

Die Grundidee zur Fehlerkorrektur ist dieselbe wir in der klassischen Kodierungstheorie: Bei jeder Informationsübertragung gibt es Störungen, die zu Fehlern führen. Um diese zu eliminieren, überträgt man zusätzlich zur eigentlichen Information noch zusätzliche Prüfbits, mit deren Hilfe man eine (vom verwendeten Code abhängige) Anzahl von Bitfehlern pro Codewort korrigieren kann. Mathematisch gesehen ist ein fehlerkorrigierender Code eine (meist lineare) Abbildung, die jedem Vektor aus einem gewissen Vektorraum $\mathbb{F}_2^n$ einen Vektor aus einem höherdimensionalen Vektorraum $\mathbb{F}_2^m$ zuordnet. Bei einem Code, der d Fehler korrigieren kann, sind die Bilder der Vektoren aus $\mathbb{F}_2^m$ so verteilt, daß sich zwei verschiedene Vektoren aus dem Bild in mindestens $2d + 1$ Bit unterscheiden; der Veränderung von höchstens d Bit eines Bildvektors läßt sich dieser also eindeutig rekonstruieren.

Nehmen wir an, die Fehlerrate sei so, daß ein Code, der in einer Folge von m Bits d Fehler korrigieren kann, ausreicht. Die „gemeinsame“ Bitfolge sei aus Sicht des Senders der Vektor $\mathfrak{v}$, aus Sicht des Empfängers aber $\mathfrak{v}' = \mathfrak{v} + \mathfrak{u}$, wobei $\mathfrak{u}$ der Fehlervektor ist. Wir nehmen an, daß die Länge aller dieser Vektoren ein Vielfaches von m ist.

Falls sich $\mathfrak{v}$ als Folge von Worten aus dem fehlerkorrigierenden Code auffassen ließe, könnte der Empfänger einfach die Dekodierungsabbildung dieses Codes auf $\mathfrak{v}'$ anwenden, um $\mathfrak{v}$ zu erhalten. Da aber nur ein Bruchteil der Blitze zu Komponenten von $\mathfrak{v}$ führt, hat der Sender keine Möglichkeit, $\mathfrak{v}$ entsprechend zu präparieren. Er behilft sich daher mit einem Trick: Er wählt einen zufälligen Bitvektor und kodiert diesen zu einem Codevektor $\mathfrak{w}$ derselben Länge wie $\mathfrak{v}$. Sodann berechnet er $\mathfrak{v} + \mathfrak{w}$ und überträgt diesen Vektor über die ungesicherte Leitung – je nach deren Qualität eventuell wiederum gesichert durch einen fehlerkorrigierenden Code. Auch die Art der verwendeten Codes wird dem Empfänger über die ungesicherte Leitung mitgeteilt.

Der Empfänger kennt dann sowohl $\mathfrak{v} + \mathfrak{w}$ als auch $\mathfrak{v}' = \mathfrak{v} + \mathfrak{u}$, er kann also deren Differenz $\mathfrak{v} + \mathfrak{w} - \mathfrak{v}' = \mathfrak{w} - \mathfrak{u} = \mathfrak{w} + \mathfrak{u}$ berechnen. (Da wir mit Bitvektoren rechnen, gibt es keinen Unterschied zwischen plus und minus.) Das ist aber das mit dem Fehler $\mathfrak{u}$ gestörte Codewort $\mathfrak{w}$, die Dekodierungsfunktion des fehlerkorrigierenden Codes erlaubt also die Rekonstruktion von $\mathfrak{w}$. Damit ist dann aber auch $\mathfrak{u}$ berechenbar und somit auch $\mathfrak{v}$.

e) Elimination der gegnerischen Information

Der Lauscher weiß weniger über $\mathfrak{v}$ als der Empfänger; er kann zwar eventuell auch etwas von der Fehlerkorrektur im zweiten Schritt profitieren, hat aber immer noch keine vollständige Information über $\mathfrak{v}$. Dieses unterschiedliche Wissen über $\mathfrak{v}$ wird nun im dritten Schritt, der sogenannten *privacy amplification*, ausgenutzt, um $\mathfrak{v}$ auf einen Vektor $\mathfrak{z}$ zu verkürzen, über den der Lauscher mit hoher Wahrscheinlichkeit so gut wie nichts weiß. Die Längendifferenz zwischen $\mathfrak{v}$ und $\mathfrak{z}$ entspricht dabei der Information, die der Lauscher über $\mathfrak{v}$ gewonnen hat; diese kann leicht aus der Fehlerrate berechnet werden.

Zur Berechnung von $\mathfrak{z}$ einigen sich Sender und Empfänger wieder in „öffentlicher“ Diskussion, über die (zufällige) Auswahl einer Hashfunktion φ aus einer großen Anzahl vorher vereinbarter solcher Funktionen. Dies darf erst *nach* Übertragung der Photonen geschehen, denn wenn der Lauscher diese Funktion kennt, kann er seine Abhörstrategie darauf abstimmen und sie praktisch wirkungslos machen; insbesondere gibt es also immer ein kleines Restrisiko, daß der Lauscher durch vorheriges Erraten der richtigen Funktion durch deren Anwendung keine Information verliert.

Bei einem hinreichend großen Raum von Hashfunktionen ist dieses Restrisiko jedoch verschwindend klein, und eine genauere informationstheoretische Analyse zeigt, daß die Information, die der Lauscher über $\varphi(\mathfrak{v})$ hat, mit sehr hoher Wahrscheinlichkeit sehr nahe bei Null liegt. Die Behandlung der Einzelheiten dieser Analyse und des dahinterstehenden Begriffsapparats würde hier zu weit führen; interessierte Leser seien auf die am Ende des Kapitels zitierte Originalarbeit **[BBR]** verwiesen, vor allen den dortigen Paragraphen 4.2.

Zum Schluß sei nach angemerkt, daß auch die Sätze aus **[BBR]** noch nicht beweisen, daß Quantenkryptographie wirklich sicher ist – auch nicht bis auf das erwähnte Restrisiko. Wir sind nämlich immer nur davon ausgegangen, daß der Lauscher sich darauf beschränkt, einzelne Photonen zu messen und in geeigneter Weise zu ersetzen. Tatsächlich könnte er stattdessen auch irgendwelche Interferenzerscheinungen oder andere Daten über aus vielen Photonen zusammengesetzte Zustände messen und daraus Schlußfolgerungen ziehen.

Realistischerweise muß man zur Abschätzung der Sicherheit des Verfahrens dem Gegner erlauben, alle Messungen durchzuführen, die die Quantentheorie nicht ausdrücklich verbietet, darunter auch solche, an die bislang noch niemand gedacht hat.

Unter diesen Umständen erfordert die Analyse seiner Möglichkeiten erheblich tiefere Methoden aus der Quantentheorie, als die einfache Darstellung in dieser Vorlesung bieten kann. Verfügt man über solche Methoden, kann man dann allerdings beweisen, daß Quantenkryptographie auch gegenüber einem Gegner mit unbeschränkten Ressourcen im erwähnten Sinne sicher ist; siehe dazu etwa **[SP]**.

§ 3: Quantencomputer

Die Quantentheorie ist nicht nur in der Lage, Kryptographie sicherer zu machen, sie stellt potentiell auch eine ernste Bedrohung existierender Kryptoverfahren dar, die mit einem sogenannten Quantencomputer möglicherweise leicht gebrochen werden können. Besonders gefährdet sind die asymmetrischen oder *public key*-Verfahren, mit denen wir uns in den Kapiteln vier und fünf beschäftigt haben.

a) Quantenregister und QBits

In einem klassischen Computer werden Bits dargestellt durch die beiden Zustände eines bistabilen Schaltelements wie etwa eines Flipflops oder durch eine Magnetisierungsrichtung oder etwas ähnliches. Unabhängig von der physikalisch-technischen Realisierung hat man immer ein Element, das sich stets in einem von zwei wohldefinierten Zuständen befindet; diese werden traditionell als 0 und 1 bezeichnet.

Angenommen, wir verwenden stattdessen ein Photon. Das ist beim heutigen Stand der Technologie zwar völlig unrealistisch, aber es ist wohl die anschaulichste Art, das Prinzip eines Quantencomputers zu verstehen; weiter unten werden wir auch realitätsnähere Ansätze diskutieren.

Zur Kodierung eines Bits können wir etwa vereinbaren, daß die horizontale Polarisation einer Null und die vertikale einer Eins entsprechen soll;

mit einem Polarisationsfilter lassen sich Photonen in jedem der beiden Zustände unschwer produzieren; wir setzen zur Abkürzung

$$|0\rangle = \begin{pmatrix}\cos 0\\ \sin 0\end{pmatrix} = \begin{pmatrix}1\\0\end{pmatrix} \quad \text{und} \quad |1\rangle = \begin{pmatrix}\cos\frac{\pi}{2}\\ \sin\frac{\pi}{2}\end{pmatrix}.$$

Genauso einfach lassen sich aber auch Photonen jeder anderen Polarisationsrichtung produzieren; wir können unser Bit daher auch mit einem Photon der Polarisationsrichtung 45° besetzen; sein Zustand ist dann

$$\begin{pmatrix}\cos\frac{\pi}{4}\\ \sin\frac{\pi}{4}\end{pmatrix} = \begin{pmatrix}\sqrt{2}/2\\ \sqrt{2}/2\end{pmatrix} = \frac{\sqrt{2}}{2}|0\rangle + \frac{\sqrt{2}}{2}|1\rangle .$$

Diesen Zustand bezeichnen wir als eine *Überlagerung* der beiden Zustände $|0\rangle$ und $|1\rangle$. denn wenn wir ihn messen mir einem Polarisationsfilter waagrechter oder senkrechter Durchlassrichtung erhalten wir eines der Ergebnisse $|0\rangle$ oder $|1\rangle$, können aber nicht im Voraus sagen, welches der beiden.

Indem wir den Winkel 45° durch einen anderen ersetzen, können wir offenbar auch jeden anderen Zustand der Form $\alpha|0\rangle + \beta|1\rangle$ erzeugen; ein solches quantenmechanisches System mit einem zweidimensionalen Zustandsraum bezeichnen wir als ein *QBit*. Im Gegensatz zu einem gewöhnlichen Bit, das nur die Werte 0 und 1 annehmen kann, sind für ein QBit also beliebige Werte der Form $\alpha|0\rangle + \beta|1\rangle$ mit $\alpha, \beta \in \mathbb{C}$ möglich. ($\alpha^2 + \beta^2 = 1$ bei Normierung auf Länge eins)

Setzt man mehrere QBits zusammen, entsteht ein Quantenregister. Wichtig ist dabei, daß dieses Quantenregister ein einziges quantenmechanisches System ist, es ist also beispielsweise durchaus möglich, daß der Zustand „alle QBits sind $|0\rangle$“ mit dem Zustand „alle QBits sind $|1\rangle$“ überlagert ist. In diesem Fall wüßten wir zwar nicht im voraus, welches Ergebnis eine Messung eines QBits aus dem Register hätte, wir könnten uns aber sicher sein, daß nach der Messung des ersten QBits jede Messung eines weiteren QBits zum selben Ergebnis führte.

In DIRAC-Notation schreibt man $|\alpha_1\alpha_2\ldots\alpha_n\rangle$ für den Zustand, in dem das i-te QBit den Wert $\alpha_i \in \{0,1\}$ hat; die 2^n Vektoren, die man auf diese Weise erhält, bilden offenbar eine Basis des Zustandsraums für

das Quantenregister. (Mathematisch betrachtet ist dieser das Tensorprodukt $V_1 \otimes V_2 \otimes \cdots \otimes V_n$ der Zustandsräume V_i der einzelnen QBits.) Der Zustand eines Quantenregisters wird also beschrieben durch eine Linearkombination von Vektoren der obigen Form wie etwa

$$\frac{\sqrt{2}}{2}|00\ldots0\rangle + \frac{\sqrt{2}}{2}|11\ldots1\rangle$$

für das Beispiel aus dem vorigen Absatz.

b) Quantencomputer

Genau wie ein klassischer Computer den Inhalt klassischer Register manipuliert, manipuliert ein Quantencomputer den Inhalt von Quantenregistern.

Das fundamentale Grundgesetz der Quantenmechanik, die SCHRÖDINGER-Gleichung, sagt aus, wie sich deren Inhalt verändern kann: Ist $|\psi(t)\rangle$ der Zustandsvektor zur Zeit t und wird das System durch keine äußeren Einflüsse gestört, so ist

$$i\hbar\frac{\partial|\psi(t)\rangle}{\partial t} = H|\psi(t)\rangle ,$$

wobei H den HAMILTONschen Operator des Systems bezeichnet, jenen HERMITEschen Operator also, der die Gesamtenergie des Systems beschreibt, und $\hbar = h/2\pi \approx 1{,}054589 \times 10^{-34}$ Js das durch 2π dividierte PLANCKsche Wirkungsquantum.

Die SCHRÖDINGER-Gleichung ist ein System linearer Differentialgleichungen mit konstanten Koeffizienten; seine Lösung läßt sich mit Hilfe der Matrixexponentialfunktion sofort hinschreiben:

$$|\psi(t)\rangle = e^{-iHt/\hbar}|\psi(0)\rangle = U(t)|\psi(0)\rangle \quad \text{mit} \quad U(t) = e^{-iHt/\hbar}.$$

Dabei ist

$$U(t)\cdot\overline{U(t)}^T = e^{-iHt/\hbar}\cdot e^{i\overline{H}^T t/\hbar} = E$$

die Einheitsmatrix, denn $\overline{H}^T = H$ nach Definition eines HERMITEschen Operators. Somit ist der Operator $U(t)$, der die zeitliche Entwicklung des Systems beschreibt, unitär und insbesondere auch invertierbar.

Diese Invertierbarkeit ist ein großer Unterschied zu klassischen Computern: Die Addition $3+5=8$ etwa läßt sich nicht invertieren, denn das Ergebnis „8“ enthält keine Information mehr darüber, wie es zustandegekommen ist. Langfristig werden aber möglicherweise auch klassische Computer bei immer weitergehender Miniaturisierung der Bauelemente mit reversibler Logik rechnen müssen, denn die einzige Stelle beim Rechnen, bei der Energieverbrauch aus physikalischen Gründen nicht vermieden werden kann, ist die Vernichtung von Information.

Von den klassischen Logikoperationen, mit denen heutige Computer arbeiten, ist nur die Negation reversibel und natürlich auch unitär: Bezüglich der Basis $\{|0\rangle, |1\rangle\}$ des Zustandsraums eines Photons wird sie durch die Matrix

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

beschrieben. Konjunktion und Disjunktion sind aus demselben Grund wie die Addition nicht reversibel: Für eine reversible Logikoperation muß die Anzahl der Ausgangsbits gleich der der Eingangsbits sein. Ein Beispiel einer reversiblen Operation mit zwei Eingangsbits ist die kontrollierte Negation

$$(x,y) \mapsto \begin{cases} (x,\neg y) & \text{falls } x=1 \\ (x,\ \ y) & \text{falls } x=0 \end{cases} = (x, x\oplus y)\,,$$

wobei $\oplus$ die Addition modulo 2 bezeichnet.

Bezüglich der Basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$ wird die kontrollierte Negation durch die unitäre Matrix

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

beschrieben, und sie ist auch quantenmechanisch realisierbar.

Wichtiger ist das TOFFOLI-Gate, das auf drei QBits operiert:

$$(x,y,z) \mapsto T(x,y,z) \underset{\text{def}}{=} \big((x,y,z\oplus(x\wedge y)\big)\,,$$

denn mit seiner Hilfe lassen sich auf Kosten eines zusätzlichen Bits „und“ und „oder“ realisieren:

$$T(x,y,|0\rangle) = (x,y,x\wedge y) \quad\text{und}\quad T(\neg x,\neg y,|1\rangle) = (\neg x,\neg y, x\vee y)\,.$$

In der Standardbasis des Zustandsraums für drei QBits vertauscht das TOFFOLI-Gate einfach die beiden Vektoren $|110\rangle$ und $|111\rangle$, es wird also durch eine unitäre Matrix beschrieben. Seine quantenmechanische Realisierung *ohne* Phasenverschiebung ist etwas trickreich, aber möglich. Insbesondere kann er als eine Folge von Operationen mit jeweils höchstens zwei Eingangsvariablen geschrieben werden, was für die technische Realisierung von großer Bedeutung ist: Interaktionen zwischen drei Quantenbits gleichzeitig wären zu weit jenseits des derzeit Machbaren.

Ein Quantencomputer kann also mit der Negation und dem TOFFOLI-Gate alle logischen Berechnungen durchführen. Arithmetische Operationen können nach den klassischen Regeln der Schaltalgebra auf logische zurückgeführt werden; auch sie sind somit in einem Quantencomputer realisierbar.

c) Der Algorithmus von Shor

Als Beispiel für die kryptographische Relevanz eines Quantencomputers wollen wir die Faktorisierung einer ganzen Zahl N betrachten.

Der dümmste Ansatz zur Faktorisierung von Hand beseht darin, daß wir x und y unabhängig voneinander die Zahlen von 2 bis N durchlaufen lassen und jeweils das Produkt xy berechnen; falls dieses gleich N ist, haben wir eine Zerlegung von N gefunden.

Für eine etwa hundertstellige Zahl N (mit deren Faktorisierung ein heutiger Computer keine größeren Schwierigkeiten hat) wären hierzu rund 10^{200} Multiplikationen notwendig, mit klassischen Computern ein Ding der Unmöglichkeit: Selbst wenn alle heutigen Computer seit Beginn des Universums daran gerechnet hätten, wäre erst ein verschwindend kleiner Bruchteil der Produkte berechnet.

Für einen Quantencomputer dagegen sind diese 10^{200} Multiplikationen überhaupt kein Problem: Da $10^{100} \approx 2^{332}$ ist, nehmen wir zwei Quantenregister x, y aus etwa 350 QBits und bringen beide in den Zustand, in

dem alle Basisvektoren denselben Koeffizienten haben. Sodann berechnen wir das Produkt der beiden Registerinhalte auf reversible Weise, d.h. wir berechnen das Tripel (x, y, xy). Dieses ist in einem Zustand, in dem alle Kombinationen (x, y, xy) mit $0 \leq x, y < 2^{350}$ überlagert sind, insbesondere also auch die, für die $xy = N$ ist. Der Quantencomputer kann also all diese Multiplikationen gleichzeitig durchführen.

Damit ist allerdings leider das Faktorisierungsproblem noch nicht gelöst, denn wir müssen das Tripel ja auch noch messen. Dabei kollabiert der überlagerte Zustand und wir erhalten als Ergebnis ein Tripel (x, y, xy) aus natürlichen Zahlen, über das wir keinerlei Kontrolle haben. Insbesondere ist die Wahrscheinlichkeit, daß an dritter Stelle die Zahl N steht, verschwindend gering, so daß dieser sehr einfache Ansatz definitiv nicht zum Ziel führt.

Ein Quantencomputer kann also zwar sehr viele Operationen gleichzeitig durchführen, aber dies läßt sich nur ausnutzen, wenn man das Ziel der Rechnung anschließend auch wirklich messen kann.

Deshalb geht der Algorithmus von SHOR das Faktorisierungsproblem völlig anders an: Wie von FERMAT bis hin zu den modernsten Siebalgorithmen immer wieder ausgenutzt wird, hat man dann eine gute Chancen, eine Zahl zu faktorisieren, wenn man sie selbst oder ein Vielfaches als Differenz zweier Quadrate darstellen kann: Ist

$$kN = x^2 - y^2 = (x - y)(x + y)\,,$$

so kann man hoffen, daß ggT$(x - y, N)$ und ggT$(x + y, N)$ echte Teiler von N sind.

Der Algorithmus von SHOR nutzt dies aus, indem er für eine zufällig gewählte Zahl x zwischen 2 und $N - 2$ ihre Ordnung modulo N berechnet, d.h. die kleinste natürliche Zahl r, für die

$$x^r \equiv 1 \bmod N$$

ist. Eine solche Zahl r muß es nicht geben; für $N = 4$ und $x = 2$ beispielsweise gibt es keine. Elementare zahlentheoretische Betrachtungen zeigen, daß die zu N teilerfremde Zahlen bezüglich der Multiplikation modulo N eine zyklische Gruppe bilden; genau für diese Zahlen gibt

es also ein solches r, und für alle anderen ist der ggT von x und N ein echter Teiler von N.

Falls r existiert und ungerade ist, hat man Pech gehabt und beginnt noch einmal mit einem neuen x; andernfalls ist

$$(x^{r/2} + 1)(x^{r/2} - 1) = x^r - 1 \equiv 0 \bmod N\,,$$

wir sind also genau in der obigen Situation und können ggTs berechnen. Falls dies keine echten Teiler sind, haben wir wieder Pech gehabt und müssen mit einem neuen x von vorne anfangen. Da r die kleinste Zahl ist, für die $x^r - 1$ durch N teilbar ist, passiert das genau dann, wenn $x^{r/2} + 1$ durch N teilbar ist, also im Fall, daß $x^{r/2} \equiv -1 \bmod N$. Der Algorithmus ist somit genau dann nützlich, wenn dieser Fall nicht allzu oft (oder gar immer) eintritt.

Ist etwa p eine ungerade Primzahl und $N = p^n$, so kann bei geraden r die Primzahl p keinesfalls Teiler *beider* Faktoren

$$x^{r/2} + 1 \quad \text{und} \quad x^{r/2} - 1$$

sein, da sich die beiden nur um zwei unterscheiden. Also ist einer der beiden durch p^n teilbar, was natürlich nur der erste sein kann. Somit versagt der Algorithmus von SHOR in diesem Fall für *jedes* x.

Im kryptographisch besonders interessanten Fall, daß $N = pq$ Produkt zweier ungerader Primzahlen ist, sieht es aber – je nach Standpunkt leider oder zum Glück – ganz anders aus: Da modulo einer Primzahl $y \equiv \pm 1$ die beiden einzigen Lösungen der Gleichung $y^2 \equiv 1$ sind, zeigt eine einfache, wenn auch etwas langwierige Argumentation über den chinesischen Restesatz, daß für mindestens die Hälfte aller zu N primen Zahlen die Ordnung r gerade und $x^{r/2}$ nicht kongruent -1 modulo N ist. Hier liegt also die Erfolgswahrscheinlichkeit bei über 50%; wiederholt man die die Rechnung mit einem anderen x, kommt man bereits auf 75%, nach sieben zufällig gewählten Werten von x auf über 99%, nach zehn auf über 99,9%. Falls es also gelingt, r effizient zu berechnen, ist dieses Verfahren extrem gefährlich für das RSA-System.

Die Berechnung von r stellt uns vor ein ähnliches Problem wie die Berechnung eines diskreten Logarithmus, und in der Tat führt der Algorithmus von SHOR zur Bestimmung von r auch auf einen Algorithmus

zur Berechnung diskreter Logarithmen; auch die Verallgemeinerung auf entsprechende Probleme für elliptische Kurven stellt keine prinzipiellen Probleme.

SHOR geht folgendermaßen vor: Für $N < 2^L$ braucht er einen Quantencomputer mit zwei Quantenregistern der Längen $2L$ beziehungsweise L. Im ersten speichert er die Überlagerung aller Zahlen a von 0 bis $2^{2L}-1$, für das zweite berechnet er den Zustand x hoch erstes Register modulo N. Der Aufwand hierfür liegt für die klassische Berechnungsmethode durch fortgesetztes Quadrieren in der Größenordnung von L^3 Operationen.

Der Computer befindet sich dann in einem Zustand, in dem alle Paare $(a, x^a \bmod N)$ mit $0 \le a < 2^{2L}$ überlagert sind. Wird nun der Inhalt des zweiten Registers gemessen, ist die Chance für das Messen der Zahl eins natürlich verschwindend gering; das Meßergebnis wird *irgendeine* Zahl b zwischen 0 und $N-1$ sein.

Durch die Messung des zweiten Registers hat sich aber der Inhalt des ersten verändert: Der *gesamte* Computer ist *ein* quantenmechanisches System, das in einen Zustand gebracht wurde, in dem der Inhalt des zweiten Registers gleich x hoch dem Inhalt des ersten Registers ist. Die Messung des zweiten Registers kann an dieser Tatsache nichts ändern, und da das zweite Register nach der Messung die Zahl b enthält, kann eine Messung des ersten Registers nun nur noch ein Ergebnis a liefern, für das $x^a \bmod N = b$ ist. Ein solches Ergebnis nützt uns aber nichts, und deshalb dürfen wir das erste Register keinesfalls messen.

Da das erste Register doppelt so lang ist wie das zweite, befindet es sich immer noch in einem überlagerten Zustand, nämlich in der Überlagerung aller Zahlen $0 \le a < 2^{2L}$, für die $x^a \equiv b \bmod N$ ist. Ist a_0 die kleinste solche Zahl, sind dies genau diejenigen Zahlen der Form $a_0 + kr$ mit $k \in \mathbb{N}_0$, die kleiner sind als 2^{2L}; hierbei ist r die gesuchte Ordnung.

Bei den Zahlen im überlagerten Zustand im Register handelt es sich also um eine periodische Folge von $m = \left[\frac{2^{2L}-a_0}{r}\right]$ Zahlen; was uns interessiert, ist ihre Periode r.

Die Periode eines Gitters läßt sich optisch aus dem Beugungsspektrum des Gitters bestimmen; genauso wollen wir auch hier vorgehen und nicht den *Inhalt* des ersten Registers messen, sondern so etwas wie sein *Beugungsspektrum*. Ein wesentlicher Aspekt der Quantentheorie ist schließlich, daß auch einander ausschließende Zustände eines Teilchens miteinander interferieren: Falls etwa ein Photon zwei mögliche Wege zu einem Punkt zurücklegen kann, tritt auch im Falle eines einzigen Photons Interferenz auf.

Das rechnerische Analogon zum Beugungsspektrum ist die *Quanten-*FOURIER*-Transformation*. Ersteres leitet man bekanntlich aus dem HUYGENSschen Prinzip ab, wonach jeder Punkt, in dem das Gitter durchlässig ist, Ausgangspunkt einer neuen Welle ist; diese wird beschrieben durch eine komplexe Exponentialfunktion, und das Beugungsspektrum ist gegeben durch die Summe aller dieser Exponentialfunktionen.

Ganz entsprechend ordnet die Quanten-FOURIER-Transformation eines Registers der Länge $2L$ dem Inhalt $|x\rangle$ dieses Registers (aufgefaßt als Zahl zwischen 0 und $2^{2L}-1$ oder als Überlagerung mehrerer solcher Zahlen) den Zustand

$$2^{-L} \sum_{\nu=0}^{2^{2L}-1} e^{2\pi i |x\rangle \nu / 2^{2L}} \, |\nu\rangle$$

zu. Man überzeugt sich leicht, daß dies eine unitäre Abbildung definiert; das liegt einfach an der wohlbekannten Formel

$$\sum_{\nu=1}^{m} e^{2k\pi i \cdot \nu/m} = \begin{cases} m & \text{falls } m|k \\ 0 & \text{sonst} \end{cases},$$

hinter der die Symmetrie der n-ten Einheitswurzeln zum Nullpunkt der komplexen Zahlenebene steckt – falls $n = m/ggT(m,k)$ größer als eins ist. Rechnerisch kann man die Formel auch so beweisen, daß man die Summe mit $e^{2k\pi i/m}$ multipliziert:

$$e^{2k\pi i/m} \cdot \sum_{\nu=1}^{m} e^{2k\pi i \cdot \nu/m} = \sum_{\nu=2}^{m+1} e^{2k\pi i \nu/m} = \sum_{\nu=1}^{m} e^{2k\pi i \cdot \nu/m},$$

denn $e^{2k\pi i(m+1)/m} = e^{2k\pi i} \cdot e^{2k\pi i/m} = e^{2k\pi i/m}$. Somit ist

$$(1 - e^{2k\pi i/m}) \cdot \sum_{\nu=1}^{m} e^{2k\pi i \cdot \nu/m} = 0\,.$$

Falls m kein Teiler von k ist, verschwindet der erste Faktor nicht, also muß die Summe verschwinden. Ist dagegen m ein Teiler von k, so sind alle Summanden gleich eins, die Summe also gleich m.

Schwieriger ist es, diese Abbildung quantenmechanisch zu realisieren; dazu braucht man die sogenannte schnelle FOURIER-Transformation, die durch geschickte Gruppierung der Summanden nach dem Prinzip *Teile und herrsche* eine deutlich effizientere Berechnung der Abbildung erlaubt. Wie SHOR gezeigt hat, läßt sie sich auch so aus einzelnen Rechenschritten zusammensetzen, daß das Ergebnis jedes einzelnen Schritts sich durch einen Quantenprozesse ermitteln läßt, dessen Ergebnis nur von zwei QBits abhängt.

Diese Quanten-FOURIER-Transformation wird nun also auf das erste Register angewandt; dadurch kommt dieses in den Zustand

$$\begin{aligned} &2^{-L} \sum_{\nu=0}^{2^{2L}-1} \frac{1}{\sqrt{m}} \sum_{\mu=1}^{m} e^{2\pi i(a_0+\mu r)\nu/2^{2L}} \,|\nu\rangle \\ &= \frac{1}{2^L\sqrt{m}} \sum_{\nu=0}^{2^{2L}-1} e^{2\pi i a_0} \left(\sum_{\mu=1}^{m} e^{2\pi i r\mu\nu/2^{2L}} \right) |\nu\rangle \end{aligned}\,.$$

Da $m = \left[\frac{2^{2L}-a_0}{r}\right] \approx 2^{2L}/r$ ist, läßt sich die Klammer approximieren durch

$$\sum_{\mu=1}^{m} e^{2\pi i r\mu\nu/2^{2L}} \approx \sum_{\mu=1}^{m} e^{2\pi i\mu\nu/m} = \begin{cases} m & \text{falls } m|\nu \\ 0 & \text{sonst} \end{cases}\,.$$

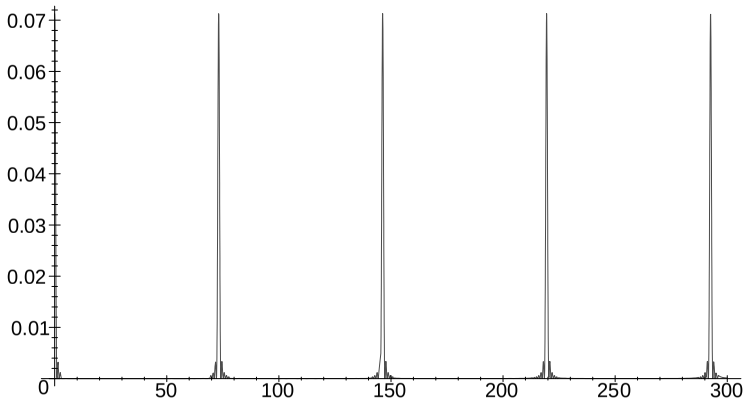
Der Zustand des ersten Registers ist also ungefähr gleich

$$\frac{1}{2^L\sqrt{m}} \sum_{\nu=0}^{m-1} e^{2\pi i a_0} m\,|r\nu\rangle = \frac{e^{2\pi i a_0}\sqrt{m}}{2^L} \sum_{\nu=0}^{m-1} |r\nu\rangle\,.$$

Durch Aufsummieren der geometrischen Reihe kann man natürlich auch leicht den genauen Wert der geometrischen Reihe berechnen.

Falls wir ein Register messen, dessen Inhalt durch die approximative Zustandsfunktion beschrieben wird, erhalten wir mit *Sicherheit* ein Vielfaches von r als Ergebnis, wobei die verschiedenen Vielfachen allerdings gleiche Wahrscheinlichkeit haben.

Die exakte Zustandsfunktion führt zu leichten Abweichungen davon: Die Betragsmaxima der Koeffizienten liegen zwar bei den Vielfachen der wirklichen Periode $2^{2L}/r$, aber weiter davon entfernt liegende Werte sind nicht mehr unmöglich, sondern nur noch unwahrscheinlich. Die Abbildung zeigt die berechneten Wahrscheinlichkeiten für den Fall $L = 5$ und $r = 14$ mit $2^{2L}/r \approx 73{,}14$



Wahrscheinlichkeitsamplitiden für die exakte Zustandsfunktion

Wir wiederholen deshalb das Experiment mit demselben x mehrfach (eine genauere Analyse zeigt, daß etwa L Wiederholungen mit hoher Wahrscheinlichkeit genügen) und haben dann verschiedene Meßwerte für ν. Für die meisten, wenn nicht gar alle dieser Werte gibt es eine Zahl λ, so daß

$$\nu \approx \lambda\frac{2^{2L}}{r} \quad \text{oder} \quad \frac{\nu}{2^{2L}} \approx \frac{\lambda}{r}$$

ist.

In der hinteren Gleichung ist die linke Seite bekannt; von der rechten kennen wir weder Zähler noch Nenner. Wir wissen aber, daß der Nenner r als Ordnung von x modulo N kleiner als N ist und damit sehr viel kleiner als der Nenner 2^{2L} auf der linken Seite, der ja größer als N^2 ist. Zur Bestimmung von r müssen wir also für die verschiedenen Brüche $\lambda/2^{2L}$ Approximationen finden, deren Nenner kleiner ist als N.

Die besten rationalen Approximationen einer reellen Zahl mit Nennern einer vorgegebenen Größenordnung liefert, wie wir in Kap. 4, §6*d)*, gesehen haben, deren Kettenbruchentwicklung; also berechnen wir für jeden Meßwert μ alle Konvergenten des Kettenbruchs zu $\frac{\mu}{2^{2L}}$ mit Nenner kleiner N und suchen dann nach einem Nenner r, der bei möglichst vielen Meßwerten auftritt. Dieser ist ein Kandidat für die gesuchte Ordnung von x, und wir verfahren damit wie im SHORschen Algorithmus angegeben.

d) Was können Quantencomputer?

Wie wir gerade gesehen haben, gibt es für Quantencomputer einen Faktorisierungsalgorithmus, der sehr viel effizienter ist als alles, was wir für konventionelle Computer kennen. Damit stellt sich natürlich die Frage, was ein Quantencomputer sonst noch alles so kann.

In der ersten Hälfte des zwanzigsten Jahrhunderts gab es ausführliche Diskussionen nicht nur über die Grundlagen der Mathematik, sondern auch über den Begriff der „Berechenbarkeit“. Es gab viele Versuche, diesen intuitiven Begriff mathematisch exakt zu definieren, unter anderem durch verschiedene Klassen rekursiv definierter Funktionen, den λ-Kalkül oder durch TURING-Maschinen. Als es in der zweiten Hälfte des zwanzigsten Jahrhunderts Computer gab, kamen dann auch Definitionen dazu, die vereinfachte Modelle eines Computers formalisieren, vor allem die sogenannten RAM-Maschinen. (RAM = **R**andom **A**ccess **M**ashine.) Wie sich bei jeder Definition ziemlich schnell herausstellte, war sie äquivalent zu den vorherigen.

Bereits 1936 stellten ALONZO CHURCH (1903–1995) und ALAN TURING (1912–1954) die These auf, daß ihre jeweiligen Definitionen den intuitiven Berechenbarkeitsbegriff formalisieren. Bis zum Beginn unseres Jahrhunderts hat sich diese These bewährt. Es gab zwar gelegentlich Ansätze, beispielsweise mit Analogcomputern Berechnungen auszuführen, die über die Möglichkeiten einer RAM-Maschine hinausgehen, aber genauere Untersuchungen zeigten stets, daß dies nicht funktionierte.

2003 veröffentlichte TIEN D. KIEU einen probabilistischen Quantenalgorithmus, der das sogenannte zehnte HILBERTsche Problem löst, d.h. er kann entscheiden, ob ein Polynom mit ganzzahligen Koeffizienten ganzzahlige Nullstellen hat. Dieses Problem ist nach den klassischen Berechenbarkeitsbegriffen unlösbar: Beispielsweise läßt sich das Halteproblem für TURING-Maschinen auf die Lösbarkeit einer diophantischen Gleichung zurückführen. Das von KIEU vorgeschlagene Verfahren benutzt allerdings keinen Quantencomputer, sondern ein allgemeineres System; wie DAVID DEUTSCH schon 1985 zeigte, kann ein Quantencomputer nur Probleme lösen, die zumindest im Prinzip auch ein klassischer Computer lösen könnte. Für Quantencomputer gilt also die These von CHURCH und TURING. Eine ähnliche Erkenntnis steckt wohl auch hinter FEYNMANs Vortrag von 1982, wo er zur Simulation physikalischer Vorgänge ein quantenmechanisches System vorschlug, das nur *teilweise* auf Quantencomputern beruht.

Das Ergebnis von DEUTSCH schließt allerdings nicht aus, das ein Quantencomputer ein Problem möglicherweise sehr viel schneller und damit überhaupt erst praktikabel lösen kann. Es gibt Beispiele von Problemen, die ein Quantencomputer in einer Zeit proportional t lösen kann, während ein klassischer Computer eine Zeit proportional zu 2^t benötigen würde; allerdings handelt es sich dabei um eher uninteressante speziell zu diesem Zweck konstruierte Probleme.

Nachdem wir gesehen haben, daß Quantencomputer so viel schneller faktorisieren können als klassische Computer, daß es für RSA keine sicheren Parameter mehr gibt, und auch erwähnt wurde, daß dasselbe für diskrete Logarithmen gilt, ist klar, daß Quantencomputer mit großen Registerlängen alle in dieser Vorlesung behandelten asymmetrischen Kryptoverfahren obsolet machen werden oder würden. Man kann allerdings auch zeigen, daß es asymmetrische Kryptoverfahren gibt, die auch gegen Angreifer mit Quantencomputern sicher sind – nur ist das

bislang ein reiner Existenzbeweis und es gibt meines Wissens noch keine Ansätze für ein konkretes Verfahren dieser Art.

Bleibt also die Frage, welche Auswirkungen Quantencomputer auf die hier behandelten symmetrischen Kryptoverfahren haben.

Wie wir hoffen, gibt es gegen die heute gebräuchlichen dieser Verfahren keine Angriffsmöglichkeit, die schneller ist als das Durchprobieren aller Schlüssel. Selbst wenn dies zutrifft, schließt es natürlich nicht aus, daß es Quantenalgorithmen geben könnten, die schneller sind als die besten Quantenalgorithmen zum Durchprobieren aller Schlüssel, allerdings sind bislang in der offenen Literatur noch keine entsprechenden Ansätze aufgetaucht.

Wie LOV GROVER 1996 gezeigt hat, können Quantencomputer allerdings schneller alle Schlüssel durchprobieren als klassische: Während einer klassischer Computer im Extremfall N Versuche braucht um N Schlüssel durchzuprobieren und im Mittel $N/2$, braucht ein Quantencomputer im Mittel nur etwa $\sqrt{N}$ Versuche. Um auch noch gegenüber einem Gegner mit Quantencomputer gegen diesen Angriff sicher zu sein brauchen wir also bei symmetrischen Kryptoverfahren für gleiche Sicherheit ungefähr die doppelte Schlüssellänge.

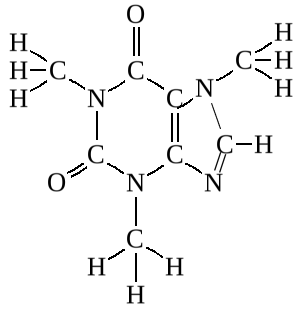
Nun könnte es natürlich sein, daß ein Gegner einen besseren Algorithmus als den von GROVER kennt. Im Gegensatz zur klassischen Situation haben wir allerdings hier die Sicherheit, daß ein solcher Algorithmus nicht wesentlich besser sein kann: Während es in der klassischen Komplexitätstheorie so gut wie keine Beweise für die in der Kryptologie relevanten unteren Schranken gibt, haben BENNETT, BERNSTEIN, BRASSARD und VAZIRANI 1997 bewiesen, daß es keinen schnelleren Quantenalgorithmus als den von GROVER zur Suche in einer Liste ungeordneter Daten der Länge N geben kann. Grundsätzlich reicht also bei symmetrischen Kryptoverfahren die Verdoppelung der Schlüssellänge zur Verteidigung gegen Quantencomputer – sofern man spezifische Angriffe gegen ein bestimmtes Verfahren ausschließen kann, was natürlich praktisch nie möglich ist. Aber dieses Problem kennen wir schon zu Genüge, denn auch bei konventioneller Rechnung können wir uns nur gegen aktuell bekannte Angriffe schützen.

e) Experimentelle Realisierung

Das große Problem beim Bau von Quantencomputern ist die *Dekohärenz:* Ein überlagerter Zustand bleibt nur erhalten, wenn keinerlei Wechselwirkung mit äußeren Systemen eintritt; dies läßt sich auch mit großem Aufwand nur für sehr kurze Zeitspannen sicherstellen. SHOR entwickelte *Quantencodes*, die nach dem Vorbild klassischer fehlerkorrigierender Codes ein gewisses Maß an Dekohärenz verkraften können, aber auch damit läßt sich diese Zeitspanne nur begrenzt ausdehnen. Ein Quantencomputer auf der Basis von Photonen mit ihren extrem kurzen Dekohärenzzeiten erscheint deshalb im Augenblick jenseits jeder technischen Realisierbarkeit.

Am vielversprechendsten ist im Augenblick ein Ansatz, der mit NMR-Spektroskopie arbeitet. Bei dieser handelt es sich um eine schon seit über dreißig Jahren gebräuchliche chemische Analysemethode, die inzwischen auch für bildgebende Verfahren in der Medizintechnik eingesetzt wird. Sie beruht auf der magnetischen Resonanz gewisser Atomkerne. Zwar zeigen nur wenige Isotopen Kernresonanz, aber es handelt sich dabei um Isotope häufiger und wichtiger Atome: ^{1}H, ^{2}H, ^{10}B, ^{11}B, ^{13}C, ^{14}N, ^{15}N, ^{17}O, ^{19}F, ^{29}Si und ^{31}Si.

Als Beispiel eines für einen ersten Quantencomputer geeigneten Moleküls wurde (nicht unbedingt ganz ernsthaft) das *Koffein* vorgeschlagen, das in der Tat genug solche Atome enthält, um einen kleinen „Supercomputer in der Kaffeetasse“ zu realisieren:



Protonen und Neutronen in einem Atom haben jeweils Spin $\pm 1/2$, wobei

sich benachbarte Protonen sowie benachbarte Neutronen jeweils antiparallel ausrichten. Bleibt dabei ein Gesamtspin übrig, richtet sich das Atom in einem umgebenden Magnetfeld der Stärke von etwa 9 bis 15 Tesla in einer von wenigen wohldefinierten Richtungen aus, zwischen denen es durch Aufnahme beziehungsweise Abgabe von Strahlungsquanten wechseln kann. Mit geeigneten Radiowellen läßt sich also zwischen verschiedenen Zuständen hin- und herschalten; durch Aufnahme eines Resonanzspektrums lassen sich die (über viele Atome gemittelten) Zustände ablesen. Dadurch, daß man hier nicht mit Quantenzuständen einzelner Partikel arbeitet, sondern über viele Partikel mittelt, wird auch das Problem der Dekohärenz entschärft.

2001 realisierten Wissenschaftler bei IBM auf dieser Basis einen Quantencomputer mit sieben QBits und schafften es, damit die Zahl 15 nach SHORs Algorithmus zu faktorisieren. Sie arbeiteten allerdings nicht mit Koffein, sonder mit einem Dicarbonylcyclopentadienyl (Perfluorobutadien-2-yl) Eisen ($C_{11}H_5F_5O_2Fe$), wobei die sieben QBits den fünf Fluor-19 Atomen sowie zwei Kohlenstoff-13 Atomen zugeordnet waren.

QBits in einzelnen Atomen benutzt die um 1995 in Innsbruck konzipierte *Ionenfalle*. Hier werden Ionen durch elektromagnetische Felder in einer linearen Anordnung gehalten; die QBits werden kodiert durch Anregungszustände der Ionen (von denen jedes durch einen eigenen Laser kontrolliert wird) und der Gitterschwingungen (Phononen) zwischen den einzelnen Ionen. Überlagerte Zustände, die mehrere QBits umfassen, werden über die Vibrationsenergie ihres Schwerpunkts kontrolliert.

Am National Institut of Standards in Boulder, Colorado, wurde so die kontrollierte Negation implementiert; Faktorisierung kleiner Zahlen erscheint durch Weiterentwicklung und Vergrößerung der Apparatur möglich, allerdings müssen dazu noch einige technische Probleme gelöst werden.

§ 4: Andere nichtkonventionelle Rechnerarchitekturen

Nicht nur Quantencomputer können für zumindest einige Kryptoverfahren gefährlich werden, sondern jede nichtklassische Rechnerarchitektur ist zumindest potentiell eine Bedrohung. Entsprechende Ansätze

für nichtkonventionelle Architekturen gibt es viele, beispielsweise zelluläre Automaten und neuronale Netze in all ihren verschiedenen Ausprägungen; abgesehen von Quantencomputern gibt es aber nur noch einen Ansatz, der in Hinblick auf die Kryptographie ernsthaft diskutiert wurde: Die sogenannten DNS-Computer. Sie stellen nach heutigem Kenntnisstand keine Bedrohung gängiger Kryptoverfahren dar, könnten allerdings interessant werden für die Sicherung von Markenwaren gegen Fälschungen. Für Interessenten sei daher kurz eines der Prinzipien skizziert, nach denen DNS-Computer arbeiten können.

Alles Leben beruht auf der Informationsverarbeitung in den Zellen eines Organismus. Diese arbeitet mit komplexen chemischen Reaktionszykeln, die bei weitem noch nicht alle verstanden sind. Das wichtigste Speichermedium ist die Desoxyribonukleinsäure, kurz DNS, der Zelle.

Die Informationsdichte dort ist ungeheuer hoch: In trockenem Zustand benötigt ein Bit gerade einmal ein Volumen von $1\,\mathrm{nm}^3$, in der Zelle etwa $100\,\mathrm{nm}^3$. Für einen Kubikzentimeter trockener DNS kommt man somit auf eine Speicherkapazität von 10^{21} Bit, das ist eine achtel Billion Gigabyte; für DNS in der Zelle kommt man mit einem Kubikzentimeter immerhin noch auch 125 Milliarden Gigabyte –verglichen mit Speicherchips auf Siliziumbasis oder CDs und DVDs eine ungeheure Menge: Beispielsweise bräuchte man größenordnungsmäßig fast eine Billion CDs, um die in einem Kubikzentimeter DNS enthaltene Information zu speichern.

Hinzu kommt, daß die Information in der DNS massiv parallel verarbeitet werden kann mit einem Energieaufwand pro Operation, der ungefähr zehn Größenordnungen unter dem heutiger Supercomputer liegt.

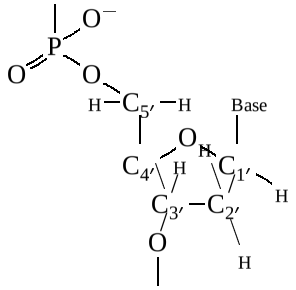
Wenn man dieses Potential für klassische Probleme der wissenschaftlichen oder gewerblichen Informationsverarbeitung nutzbar machen könnte, wäre dies eine Revolution, die auch die Kryptographie massiv verändern würde.

Seit 1994 gibt es einen ersten Hinweis darauf, daß dies vielleicht nicht völlig unrealistisch ist: Damals löste der Mathematiker und Informatiker LEONARD M. ADLEMAN (der hier bereits im Zusammenhang mit dem RSA-Verfahren erwähnt wurde) in einem molekularbiologischen Labor

ein (ziemlich einfaches) graphentheoretisches Problem mit Hilfe von DNS. Um seine Vorgehensweise und das Potential der Methode zu verstehen, müssen wir zunächst kurz den Aufbau der DNS betrachten.

a) Die Desoxyribonukleinsäure

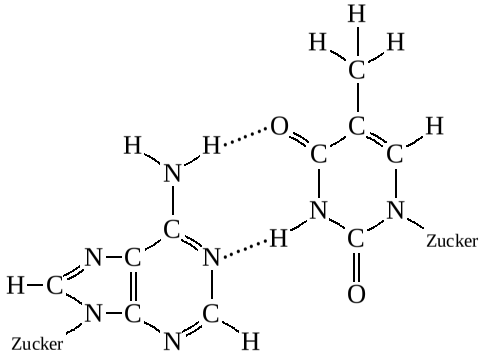
Die Desoxyribonukleinsäure hat ihren Namen vom hier abgebildeten Zuckermolekül Desoxyribose, aus dem ihr Gerüst aufgebaut ist.



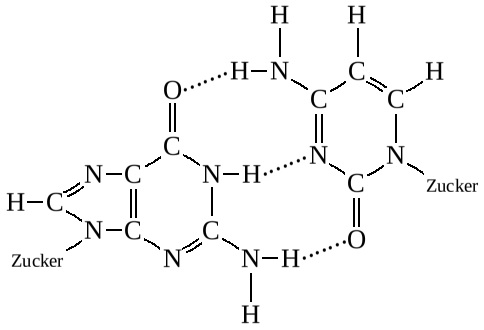
Die Desoxyribose

Die fünf Kohlenstoffatome werden mit 1′ bis 5′ bezeichnet; in der Abbildung ist dies als Index eingetragen. An der Phosphatgruppe von 5′ und der Hydroxygruppe von 3′ fehlt jeweils ein Wasserstoffatom: Hier werden die Zucker zu einer Kette zusammengesetzt, wobei stets auf die 5′-Phosphatgruppe eine 3′-Hydroxygruppe folgt. Am Kohlenstoffatom 1′ steht *Base;* hier wird eine jener vier Basen eingebaut, die die eigentlichen Informationsträger der DNS sind: Adenin, Guanin, Thymin und Cytosin.

Wie man an den Abbildungen sieht, können sich zwischen Adenin und Thymin sowie zwischen Guanin und Cytosin die gestrichelt eingezeichneten Wasserstoffbrücken ausbilden; diese sorgen dafür, daß DNS nur selten in einzelnen Strängen vorkommt: Meist kombinieren sich zwei Stränge zu einem Doppelstrang, wobei jedem Adenin ein Thymin und jedem Guanin ein Cytosin gegenübersteht. Man bezeichnet diese Basen deshalb als zueinander WATSON-CRICK-komplementär.



Adenin Thymin



Guanin Cytosin

Aus geometrischen Gründen hat ein DNS-Doppelstrang die berühmte Form der Doppelhelix, die wiederum selbst weitere geometrische Strukturen bildet. Für die Informationsverarbeitung in der Zelle sind alle diese geometrischen Strukturen sehr wichtig; die entsprechenden Mechanismen sind aber noch nicht so gut verstanden, daß man sie auch in Laborexperimenten ausnutzen könnte, so daß diese Strukturen für uns irrelevant sind.

b) Die Polymerase-Kettenreaktion

Ein wesentlicher Bestandteil des molekularen Rechnens ist der Aufbau von DNS-Sequenzen sowie die Vervielfältigung von Sequenzen mit erwünschten Eigenschaften. Ein wichtiges Hilfsmittel dafür ist die 1987 von KARY B. MULLIS entwickelte Polymerase-Kettenreaktion. (Es gibt inzwischen auch theoretische Ansätze, die ohne diese relativ langsame Operation auskommen wollen.)

Polymerasen sind Enzyme, die in der Zelle die Duplizierung der Information aus DNS-Strängen steuern; sie sind je nach Lebewesen verschieden, und auch innerhalb derselben Zelle gibt es oft mehrere Polymerasen mit verschiedenen Aufgaben. Das besonders gut erforschte Bakterium *Escherichia coli* etwa enthält drei DNS-Polymerasen; die am stärksten vertretene DNS-Polymerase I ist eine Kette mit 928 Aminosäuren. Es handelt sich hier also um sehr komplexe Enzyme, die deshalb auch heute noch nicht synthetisiert werden, sondern auch für die Laborarbeit von Bakterien produziert werden.

Polymerasen werden zusammen mit zwei sogenannten *Primern* eingesetzt, d.h. mit zwei Folgen von DNS-Basen; diese sorgen dafür, daß selektiv ein DNS-Strang produziert wird, der WATSON-CRICK-komplementär ist zum Stück zwischen den beiden Primern auf einem vorhandenen DNS-Strang. Insbesondere muß DNS also einsträngig vorliegen, bevor eine Polymerase aktiv werden kann. In der lebenden Zelle sorgen Enzyme dafür, daß benötigte Teilstränge zur Verfügung stehen; im Labor geht man brutaler vor und teilt den ganzen Doppelstrang durch Erhitzen auf eine Temperatur von $90^\circ - 95^\circ$ C.

Da diese Temperatur praktisch alle Enzyme zerstört, arbeitet man heute nicht mehr mit der Polymerase von *Escherichia coli*, sondern meist mit der sogenannten Taq-Polymerase des Bakteriums *Thermus aquaticus*, das in heißem Wasser lebt und daher auch eine hitzebeständige Polymerase hat, die ihre optimale katalytische Wirkung bei etwa 75° C entfaltet.

Zur Vervielfältigung von DNS geht man folgendermaßen vor:

Man gibt die DNS zusammen mit den beiden Primern, der Taq-Polymerase und den vier Nucleotiden, aus denen die DNS aufgebaut

ist, in ein Gefäß und denaturiert zunächst die DNS durch Erhitzen, d.h. man zerlegt sie in ihre Einzelstränge. Sodann kühlt man ab auf etwa 55° C; dies führt dazu, daß sich die Primer an die passenden Stellen der DNS-Stränge angliedern. Eine Erhöhung der Temperatur auf etwa 75° C aktiviert die Polymerase, die nun dafür sorgt, daß (ausgehend von den $3'$-Enden der Primer) WATSON-CRICK-komplementäre Nucleotide an die DNS-Stränge angelegt werden.

Damit hat man die gewünschten DNS-Sequenzen verdoppelt, was im allgemeinen nicht ausreicht; deshalb wird das Verfahren etwa 25–35 Mal wiederholt, was zu einer etwa millionenfachen Vermehrung führt. Das Verfahren kann in sogenannten Thermocyclern automatisch durchgeführt werden, da es nur auf die zyklische Temperatursteuerung ankommt.

c) Adlemans Experiment

Gegeben seien eine gewisse Anzahl von Städten sowie Straßen, die Städte gewissen anderen Städten verbinden. Man finde eine Straßenverbindung, die eine vorgegebene Stadt mit einem vorgegebenen Ziel so verbindet, daß jede Stadt genau einmal besucht wird.

Dies ist eine von vielen Varianten eines graphentheoretischen Problems, das für große Anzahlen von Städten und Verbindungsstraßen rechnerisch nur mit großem Aufwand gelöst werden kann. (Es ist NP-vollständig.) ADLEMAN beschränkte sich allerdings auf nur sieben Städte und fand heraus, daß der durchschnittliche menschliche Betrachter etwa 54 Sekunden braucht um sein spezielles Problem zu lösen. Interessanter ist aber die Art und Weise, wie er diese Lösung statt durch Hinschauen durch sieben Tage Arbeit im Labor produzierte.

Die Grundidee ist folgende: Jede Stadt wird durch eine Folge von zwanzig DNS-Basen kodiert; als Lösung soll jene Folge von 140 DNS-Basen produziert werden, die den (in ADLEMANs Beispiel einzigen) Lösungsweg angibt.

Die Herstellung kurzer DNS-Sequenzen mit vorgegebener Basenfolge gehört heutzutage zu den Standardtechniken der Molekularbiologie; große Labors haben Automaten, die solche Sequenzen (über

die Polymerase-Kettenreaktion) erzeugen, kleinere kaufen sie von den großen: Für 25$ kann man innerhalb von wenigen Tagen ein Reagenzglas bekommen, das etwa 10^{18} DNS-Stränge mit einer vorgegebenen Folge von zwanzig Basen enthält.

Für jede der sieben Städte wurde also ein solches Reagenzglas benötigt, dazu aber auch noch für jede Straße zwischen zwei Städten. Natürlich müssen die Basenfolgen für Städte und für Straßen aufeinander abgestimmt sein: ADLEMAN faßte die Basenfolge für die i-te Stadt auf als ein Paar (x_i, y_i) aus zwei Basenfolgen der Länge zehn, und wenn es eine Straßenverbindung zwischen i-ter und j-ter Stadt gibt, kodierte er diese durch die Basenfolge $(\overline{y_i}, \overline{x_j})$, wobei die Überstreichung hier für das WATSON-CRICK-Komplement stehen soll.

Der erste Schritt der eigentlichen Berechnung bestand darin, alle diese Sequenzen (jeweils etwa 10^{14} Moleküle) in Wasser aufzulösen und Ligase dazuzugeben. (Ligasen sind Enzyme, die kurze DNS-Sequenzen zu langen zusammenfügen.) Dazu kommt noch etwas Puffer und ähnliches, und ungefähr eine Sekunde später sind im Reagenzglas viele doppelsträngige DNS-Sequenzen zu finden, die möglichen Wegen durch den Graph entsprechen.

Unter diesen Sequenzen sind mit praktisch 100%-iger Sicherheit auch solche, die der Lösung des Problems entsprechen; das Problem besteht genau wie bei den überlagerten Zuständen in Quantencomputern darin, diese herauszufiltern. Ein Vorteil gegenüber Quantencomputern ist allerdings, daß man beim molekularen Rechnen durch Messungen üblicherweise nichts zerstört.

Im zweiten Schritt ging es darum, Sequenzen zu finden, die zu Wegen mit dem Richtigen Anfangs- und Zielort gehören. Finden bedeutete in diesem Fall, daß diese Sequenzen stark vermehrt werden sollten, und das ist ein klarer Fall für die Polymerase-Kettenreaktion: Man nehme die WATSON-CRICK-Komplemente des Start- und des Zielorts als Primer und lasse die Reaktion laufen. Danach sind alle Sequenzen, bei denen Anfangs- und Zielort stimmen, etwa millionenfach verstärkt; die, bei denen nur einer der beiden Orte stimmt, etwa dreißigfach, und der Rest überhaupt nicht. Entnimmt man also eine kleine Probe zur Weiterverarbeitung, so enthält diese kaum noch Sequenzen, bei denen einer der beiden Orte falsch ist.

Im dritten Schritt ging es darum, alle Sequenzen auszusondern, die eine falsche Länge haben. Ein Weg, der jede Stadt genau einmal berührt, entspricht einer Folge von sieben Städten und damit einem DNS-Strang der Länge 140. Zur Isolation dieser Stränge führt eine Art chromatographisches Verfahren, die *Gel-Elektrophorese.*

Hierbei wird ausgenutzt, daß die betrachteten Moleküle negativ geladen sind (man achte auf das O^- in der Desoxyribose). Bringt man die Lösung also auf einen Gel auf, meist Agarose oder Polyacrylamid, und legt ein elektrisches Feld an, so hängt die Wanderungsgeschwindigkeit ab sowohl von der elektrischen Ladung als auch der Molekülgröse und (in geringerem Ausmaß) einigen anderen Gegebenheiten. Die Methode ist aber jedenfalls trennscharf genug, um Sequenzen, deren Länge sich um zwanzig Nucleotide unterscheidet, zuverlässig zu trennen; nachdem die Moleküle einige Zeit gewandert sind, entnimmt man die aus der 140er-Region und verwirft den Rest.

Unter den noch verbleibenden Sequenzen befinden sich immer noch zahlreiche Nichtlösungen, denn eine Folge von sieben Städten erhält man auch, wenn man einige Städte ausläßt und andere dafür mehrfach besucht. Es muß also entweder sichergestellt werden, daß keine Stadt mehrfach besucht wurde, was mit molekularbiologischen Methoden sehr schwer sein dürfte, oder aber, daß jede Stadt mindestens einmal besucht wurde.

Letzteres ist für den Anfangs- und den Zielort bereits bekannt; bleiben also noch fünf Städte. Für diese verwendete ADLEMAN eine Kombination aus molekularbiologischer und physikalischer Vorgehensweise: Um diejenigen Moleküle auszusondern, die die Sequenz für eine gegebene Stadt nicht enthalten, heftete er WATSON-CRICK-Komplemente dieser Stadt an kleine Eisenkügelchen mit einem Durchmesser von etwa 1 μm, gab dies zu der denaturierten (d.h. wieder durch Erhitzen in Einzelstränge zerlegten) noch verbliebenen Lösung und lies die Stränge sich wieder kombinieren. Dabei paarten sich die Sequenzen an Eisenkugeln mit Teilsequenzen jener Moleküle, die einen Weg durch die betrachtete

Stadt beschreiben. Die so entstandenen DNS-Sequenzen waren also mit Eisenkugeln verbunden und konnten so mit einem Magneten am Rand des Reagenzglases festgehalten werden, während der Rest des Glases ausgeschüttet wurde.

Danach wurde frisches Wasser zugegeben und das ganze wieder erhitzt zur Denaturierung; jetzt aber wurde der Magnet an das erhitzte Reagenzglas gehalten, so daß der die Sequenzen mit Eisenkügelchen festhielt. Der Rest wurde umgegossen in ein anderes Reagenzglas, wo dann das gleiche Spiel für die nächste Stadt wiederholt werden konnte, bis alle fünf Städte abgearbeitet waren.

Falls danach noch Moleküle übrig waren, konnte es sich nur um Lösungen handeln. Da es aber wahrscheinlich nur noch relativ wenige waren, vermehrte sie ADLEMAN zunächst durch eine Polymerase-Kettenreaktion und überzeugte sich durch Gel-Elektrophorese davon, daß seine Lösung wirklich Sequenzen der Länge 140 enthielt. Zur vollständigen Lösung des Problems mußten diese dann nur noch analysiert werden.

Auch hierzu dient wieder die Polymerase-Kettenreaktion in Verbindung mit Gel-Elektrophorese: Für jeden Zwischenort führte ADLEMAN eine Polymerase-Kettenreaktion durch, wobei er als Primer die WATSON-CRICK-Komplemente von Anfangsstadt und Zwischenort nahm. Dadurch wurde die Teilsequenz bis zu diesem Zwischenort stark vermehrt, und durch Gel-Elektrophorese läßt sich feststellen, wie lange sie ist. Diese Länge, geteilt durch zwanzig, ist die Position der Stadt im Lösungsweg.

d) Wie geht es weiter?

ADLEMAN hat mit seinem Experiment zwar kein neues Problem gelöst, aber er hat gezeigt, daß molekularbiologische Verfahren zumindest grundsätzlich zur Lösung rechnerischer Probleme benutzt werden können – bis zu ihrem effizienten Einsatz ist es sicherlich noch ein langer Weg.

Das durchgeführte Experiment überprüft mit brutaler Gewalt (fast) *alle* möglichen Wege durch den Graphen, was schon bei 200 Städten eine

DNS-Menge verlangen würde, die mehr wiegt als die Erde. Mit klassischen Computern und den besten derzeit bekannten Algorithmen lassen sich dagegen auch Probleme mit einigen Tausend Städten behandeln.

Auch der letzte Schritt des Experiments funktionierte nur deshalb, weil das Problem eine eindeutige Lösung hatte. Dies ist allerdings kein großes Problem, denn alternative Methoden zur Bestimmung der Basenfolge in einem DNS-Strang gibt es natürlich: Schließlich wurde inzwischen sogar das gesamte menschliche Genom entschlüsselt.

Um ein Gefühl für die mögliche Relevanz des molekularen Rechnens in der Zukunft zu bekommen, insbesondere auch in Bezug auf die Kryptologie, ist es vielleicht ganz nützlich, zum Vergleich ein völlig anderes Thema zu betrachten, die bisherige Geschichte der Faktorisierung ganzer Zahlen.

Das einfachste Verfahren zur Faktorisierung einer ganzen Zahl N ist das systematische Durchprobieren aller Zahlen $d \leq \sqrt{N}$; dieser Algorithmus ist vergleichbar mit ADLEMANs oben beschriebener Methode.

Wie wir bei der Diskussion von SHORs Algorithmus gesehen haben, liefert der Ansatz von FERMAT ein alternatives Verfahren; berechnet man, wie es FERMAT tat, für $a = 1, 2, 3, \ldots$ systematisch die Zahlen $N + a^2$ in der Hoffnung, darunter ein Quadrat zu finden, so lassen sich zumindest Zahlen mit zwei nahe beieinanderliegenden Faktoren deutlich schneller zerlegen.

Das neunzehnte Jahrhundert war von der Mechanisierung geprägt; in der ersten Hälfte des zwanzigsten Jahrhunderts erreichte diese auch die Zahlentheorie, als D.H. LEHMER eine Siebversion der FERMAT-Methode mit Zahnrädern und Fahrradketten zur Faktorisierung verwendete.

Bei den elektrischen und elektronischen Rechner griff die neue Technik schneller auf die Zahlentheorie über: D.H. LEHMER hatte bereits auf den ersten Computern seiner Universität Hintergrundprogramme laufen, die sich mit der Faktorisierung von Zahlen beschäftigten, wenn es sonst nichts zu tun gab.

In den Siebzigerjahren, als Computer leistungsfähig genug waren, um in Zahlbereiche vorzudringen, die vorher jenseits praktisch durchführbarer

Rechnungen lagen, war wieder die Mathematik gefragt, die in den letzten dreißig Jahren immer neue Faktorisierungsalgorithmen entwickelte. Diese mußten auf den jeweils aktuellen Computern implementiert werden, wobei in den Achtzigerjahren vor allem die CRAY-Computer eine sehr große Rolle spielten und aufgrund ihrer Pipeline-Architektur zur Entwicklung und Optimierung neuer Programmiertechniken zwangen.

In den Neunzigerjahren brachte das Internet die Möglichkeit zum verteilten Rechnen; 1994 wurde dadurch jene berühmte 129-stellige Zahl faktorisiert, die 1978 bei der Vorstellung des RSA-Systems als Beispiel diente und von der man damals überzeugt war, daß sie auch in hundert Jahren noch sicher sei. (Heute hält das Bundesamt für Sicherheit in der Informationstechnik Zahlen mit mindestens 1024 Bit für sicher; ab 2005 verlangt es die doppelte Länge.)

Für Fortschritte bei der Faktorisierung waren also jeweils drei Aspekte maßgeblich:

- Bessere mathematische Algorithmen
- Bessere Maschinen
- Besseres Verständnis des Umgangs mit diesen Maschinen.

Auch die weitere Entwicklung des molekularen Rechnens wird wohl von drei ähnlichen Aspekten abhängen.

Sechzehn Jahre nach ADLEMANs Experiment kann man versuchen, zumindest ein bißchen über jeden dieser drei Aspekte zu spekulieren.

Die weitaus größte Zahl von Publikationen im Umkreis der DNS-Computer beschäftigt sich mit theoretischen Algorithmen sowie mit abstrakten Maschinen und formalen Sprachen zur Modellierung des molekularen Rechnens. Viele dieser Arbeiten kommen von theoretischen Informatikern ohne Chemieausbildung und dürften wohl selbst innerhalb der Informatik schon in wenigen Jahren vergessen sein; bis zu einer Erprobung im Labor dürfte es kurzfristig auch vom verbleibenden Rest kaum eine bringen. Langfristig allerdings könnten einige wenige dieser Ansätze durchaus zu brauchbaren Verfahren weiterentwickelt werden.

Von besseren *Maschinen* dürfte der Fortschritt des molekularen Rechnens angesichts der hochentwickelten verfügbaren chemischen Labortechnik in den nächsten Jahren wohl kaum abhängen; vielmehr geht

es darum, zunächst zu erforschen, welche primitiven Grundoperationen wie durchgeführt werden können, um effizient molekular zu rechnen. Mit dieser Frage beschäftigen sich eine ganze Reihe molekularbiologischer Laboratorien, die beispielsweise mit an Oberflächen angehefteten Molekülen und ähnlichem arbeiten und teilweise auch schon erste Erfolge melden können: So gelang inzwischen die molekulare Addition zweier (kleiner) Binärzahlen. Im Augenblick ist noch nicht abzusehen, ob sich mehrere Grundparadigmen durchsetzen werden, oder ob die Entwicklung auf *den* DNS-Computer zusteuert.

Der dritte Aspekt, der bessere Umgang mit der Maschinerie, kann wohl erst dann zum Tragen kommen, wenn die Entwicklungen beim Algorithmenentwurf und bei den molekularbiologischen Ansätzen anfangen, gegeneinander zu konvergieren, wenn also die entsprechenden Experten anfangen, Notiz voneinander zu nehmen. Dies setzt einerseits voraus, daß die Labors stabile und zuverlässige Verfahren entwickeln. Vor allem aber wird ein wirklicher Fortschritt erst dann möglich sein, wenn es eine nennenswerte Anzahl von Wissenschaftlern gibt, die sowohl in der Molekularbiologie als auch in der Algorithmik zumindest eine gewisse Mindestqualifikation haben.

Der Zeitrahmen dafür hängt wesentlich davon ab, ob es gelingt, ausreichend Studenten für eine solche duale Qualifikation zu begeistern und sie ihnen auch zu ermöglichen, oder ob die Studenten angesichts des Booms sowohl in der Informationstechnik als auch der Molekularbiologie vollauf mit der Qualifikation in einem dieser Gebiete zufrieden sind.

An diesem Punkt geraten die Spekulationen ins Gebiet der Politik, wo Spekulationen nur selten zu etwas vernünftigem führen; daher soll diese Vorlesung besser hier enden.

e) Literaturhinweise

Für einen ersten Einstieg in die Welt der Quantentheorie speziell im Bezug auf Quantenkryptographie und Quantencomputer ist das (derzeit leider vergriffene) Taschenbuch

DAGMAR BRUSS: Quanteninformation, *Fischer,* 2003

sehr gut geeignet; eine kurze Übersicht über die Prinzipien der Quantenkryptographie bietet auch der Artikel

CHARLES H. BENNET, GILLES BRASSARD, ARTHUR K. EKERT: Quantenkryptographie, *Spektrum der Wissenschaft,* Dezember 1992, S. 96–104

Die beiden in §2 zitierten Originalarbeiten sind

[BBR] CHARLSES BENNETT, GILLES BRASSARD, JEAN MARC ROBERT: Privacy amplification by public discussion, *SIAM J. Comp.* **17** (1988), 210–229

und

[SP] PETER W. SHOR, JOHN PRESKILL: Simple proof of security of the BB84 quantum key distribution protocol, *Phys. Rev. Letters* **85**,2 *(10. Juli 2000),*441–444

Es gibt inzwischen eine ganze Reihe von Büchern, die sich speziell mit Quantencomputern befassen. Für einen allerersten Einstieg eignet sich vor allem

JULIAN BROWN: Minds, Mashines, and the Multiverse: the Quest for the Quantum Computer, *Simon & Schuster,* 2002

sowie das deutlich technischer geschriebene Buch

STIG STENHOLM, KALLE-ANTTI SUOMINEN: Quantum Approach to Informatics, *Wiley,* 2005

Hauptsächlich mit Quantenalgorithmen, insbesondere denen von SHOR und GROVER beschäftigen sich

MIKA HIRVENSALO: Quantum Computing, *Springer,* ^{2}2003

PHILLIP KAYE, RAYMOND LAFLAMME, MICHELE MOSCA: An Introduction to Quantum Computing, *Oxford,* 2007

A.YU. KITAEV, A.H. SHEN, M.N. VYALYI: Classical and Quantum Computation, *Graduate Studies in Mathematics* **47,** American Mathematical Society, 2002

SHORs Originalarbeit wurde 1999 in überarbeiteter Form nachgedruckt:

PETER W. SHOR: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, *SIAM Rev.* **41** (1999), S. 303–332

Einen weiteren Übersichtsartikel, in dem auch auf das hier nicht behandelte Problem der Fehlerkorrektur in Quantencomputern eingegangen wird, ist SHORs Vortrag auf dem internationalen Mathematikerkongreß 1998 in Berlin:

PETER W. SHOR: Quantum Computing, *Proceedings of the international congress of mathematicians Berlin 1998, Documenta Mathematica, Extra volume ICM* 1998 · *I*, DEUTSCHE MATHEMATIKER-VEREINIGUNG, 1998, S. 467–486 *oder*
www.mathematik.uni-bielefeld.de/documenta/xvol-icm/00/00.html

Die Originalarbeit von ADLEMAN zum molekularen Rechnen ist

LEONARD M. ADLEMAN: Molecular computation of solutions to combinatorial problems, *Science* **266** (1994), S. 1021–1024

Eine erste Diskussion, bestehend aus mehreren Leserbriefen und einer Antwort ADLEMANs ist in *Science* **268** (1995), S. 481–484 zu finden. Eine gut lesbare Darstellung der wesentlichen Punkte des Experiments ist

LEONARD M. ADLEMAN: Rechnen mit DNA, *Spektrum der Wissenschaft,* November 1998, S. 70–77

Eine Übersicht über die molekularbiologischen Methoden, die dem molekularen Rechnen zugrunde liegen, findet man beispielsweise in den Büchern

BERNARD L. GLICK, JACK J. PASTERNAK: *Molekulare Biotechnologie,* Spektrum Akademischer Verlag, 1995 *und*

ROLF KNIPPERS: *Molekulare Genetik,* Thieme, ^{7}1997,

von denen das erste mit etwas anschaulicheren Darstellungen arbeitet und das zweite stärker auf den biochemischen Hintergrund eingeht. Detaillierte Laboranweisungen zu den Grundtechniken findet man etwa in

HANS GÜNTHER GASSEN, GANGOLF SCHRIMPF [Hrsg.]: *Gentechnische Methoden,* Spektrum Akademischer Verlag, ^{2}1999