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), 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. Seit 1991 arbeitet er als *chief security officer* bei Sun Microsystems. Seine dortige home page hat den URL `http://research.sun.com/people/diffie/` .

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.

Im akademischen Bereich gab es ein Jahr nach Erscheinen der Arbeit von DIFFIE und HELLMAN das erste Kryptosystem mit öffentlichen Schlüsseln: Drei Professoren am Massachussetts Institute of Technology fanden nach rund vierzig erfolglosen Ansätzen 1977 schließlich jenes System, das heute nach ihren Anfangsbuchstaben mit RSA bezeichnet wird: RON RIVEST, ADI SHAMIR und LEN ADLEMAN.

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.

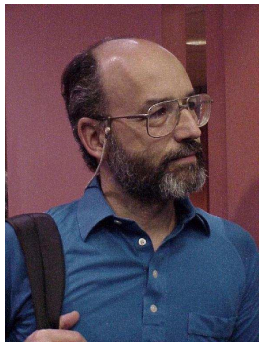
RSA ist übrigens identisch mit dem von COCKS vorgeschlagenen System, so daß einige Historiker auch Zweifel an den Behauptungen der GCHQ haben. Die Beschreibung durch RIVEST, SHAMIR und ADLEMAN erschien 1978 unter dem Titel *A method for obtaining digital signatures and public-key cryptosystems* in Comm. ACM **21**, 120–126.

§2: Die Grundidee des RSA-Verfahrens

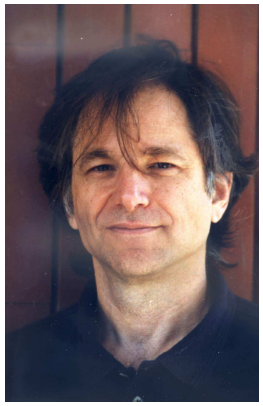
a) Allgemeine Vorüberlegungen

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 $\varphi\colon \mathbb{Z}/N \to \mathbb{Z}/N$.

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 ist 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`

Von dieser Funktion erwarten wir drei Dinge:

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

Forderung 3 ist, mathematisch gesehen, natürlich unerfüllbar: φ 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 φ einfach dadurch verschaffen, daß man zur Berechnung von $\varphi^{-1}(y)$ für jedes $x \in \mathbb{Z}/N$ ausprobiert, ob $\varphi(x) = y$ 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 φ für die gängigen Verfahren eine einfache mathematische Struktur hat, so daß es bessere Ansätze zur Berechnung von φ^{-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 $n \bmod m$, 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, jedenfall 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.

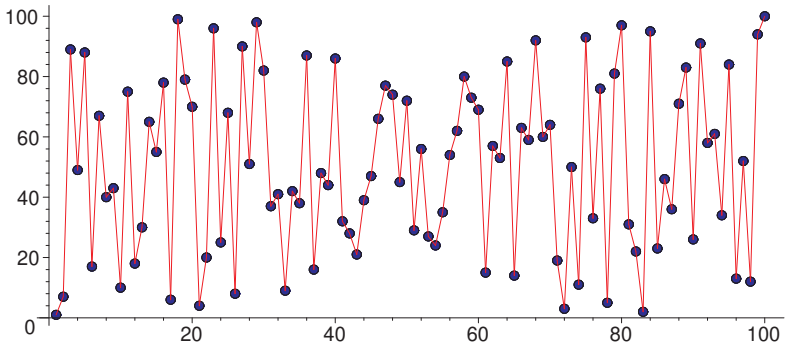
Falls n_1 mod m und n_2 mod m gleich sind, sagen wir auch, n_1 und n_2 seien kongruent modulo m; in Zeichen $n_1 \equiv n_2 \bmod m$. Dies ist offensichtlich genau dann der Fall, wenn $n_1 - n_2$ durch m teilbar ist. Um einen Kandidaten für eine Verschlüsselungsfunktion zu bekommen, kombinieren wir dies zum Beispiel mit einer Potenzfunktion, d.h. wir betrachten die Abbildung

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

Natürlich gibt es keinen Grund, daß φ bijektiv sein sollte; in der Tat ist die Abbildung etwa für $e = 2$ und ungerades $N > 1$ mit Sicherheit nicht injektiv, da dann stets $\varphi(x) = \varphi(N - x)$ ist. Wir müssen also einschränkende Bedingungen an die Zahlen N und e stellen.

b) 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



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

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 $a \geq 0$. 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 $n \geq 3$ keine ganzzahlige Lösung mit $xyz \neq 0$ hat. Dieser „große" Satzes von FERMAT, von dem FERMAT lediglich in einer Randnotitz 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 den meisten 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

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

zueinander invers sind. Insbesondere ist φ dann also bijektiv und kann als Verschlüsselungsfunktion verwendet werden mit Entschlüsselungsfunktion ψ. Die ersten beiden Forderungen, die wir in Abschnitt *a)* an eine asymmetrische Verschlüsselungsfunktion gestellt haben, sind damit erfüllt. Die dritte Forderung allerdings ist ganz eklatant verletzt: Wer φ kennt, kennt insbesondere die Zahlen p und e, und daraus läßt sich das für die Entschlüsselung benötigte d nach obigem Satz effizient berechnet. Somit läßt sich φ 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 φ 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.

c) 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 $0 \leq r < y$ 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 berechnens, 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 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 $\mathrm{ggT}(x, y) = r_{i-1}$; 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

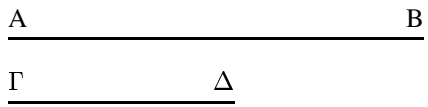
$$\mathrm{ggT}(x, y) = \mathrm{ggT}(y, x \bmod y)$$

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

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

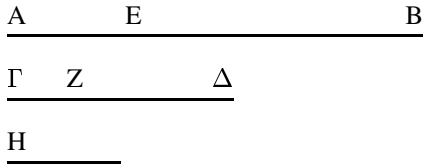
Die zwei gegebenen Zahlen, die nicht prim, gegeneinander sind, seien AB, ΓΔ. Man soll das größte gemeinsame Maß von AB, ΓΔ finden.



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 übrigbleiben, 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

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

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.

d) Die RSA-Verschlüsselungsfunktion

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

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

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 *b)*

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

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 *b)* 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 *b)* 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 *b):* 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 φ, 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 φ 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 berechen, 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 $\varphi(x) = x^e \bmod N$ 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 Standardoperatiønen, die von jeder Programmiersprache als einfache Befehle bereitsgestellt 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 unter `www.bundesnetzagentur.de` zu finden, indem man der Linkkette *Telekommunikation* → *Elektronische Signaturen* → *Veröffentlichungen* → *Geeignete Algorithmen* folgt.)

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 reichen, 2007 aber mindestens 1536 notwendig werden. Die Mindestlänge von 2048 Bit wurde wieder zur „Empfehlung“ zurückgestuft.

Am 2. Januar 2003 erschienen endlich die offiziellen Richtlinien des Jahres 2002; veröffentlich wurden sie am 11. März 2003 im Bundesanzeiger Nr. 48, S. 4202–4203. Danach reichen 1024 Bit auch noch bis Ende 2007, erst 2008 werden 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 22. Februar 2007 (Bundesanzeiger Nr. 69 vom 12. April 2007, S. 3759). Sie empfehlen grundsätzlich schon heute 2048 Bit, aber wirklich verbindlich sind

bis Ende	2007	2008	2009	2010	2012	
Minimallänge	1024	1280	1536	1728	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.)

Der (noch nicht offiziell veröffentlichte) Entwurf der Empfehlungen für 2008 sieht vor, daß die bis 2012 gültigen Werte sogar bis 2014 gelten sollen.

Die beiden Primfaktoren p, q sollen zufällig und unabhängig voneinander 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

$$114381625757888867669235779976146612010218296721242362562561842 93\backslash$$
$$5706935245733897830597123563958705058989075147599290026879543541$$

(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

$$4905295108476509491478496199038981334177646384933878439908205 77$$
$$\times 3276913299326670954996198819083446141317764296799294253979828853 3$$

gefunden, und wie sich zeigte, hatten RIVEST, SHAMIR und ADELMAN 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 hintereinandergeschrieben 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 verschiednen 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 $\varphi\colon \mathbb{Z}/N \to \mathbb{Z}/N$, die durch RSA realisiert werden, haben die Eigenschaft, daß für beliebige Nachrichten $m_1, m_2 \in \mathbb{Z}/n$ gilt $\varphi(m_1 m_2) \equiv \varphi(m_1)\varphi(m_2) \bmod N$.

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 $c = \varphi(m)$ entschlüsseln: Wir berechnen für $k = 1, \ldots, S$ die Werte

$$\varphi(k) = k^e \bmod N \quad \text{und} \quad c/\varphi(k) \bmod N\,.$$

Die Werte $\varphi(k)$ sortieren wird dann der Größe nach und schauen für jedes $c/\varphi(k)$ nach, ob es in dieser Liste vorkommt. Falls wir ein Paar (m_1, m_2) gefunden haben mit $c/\varphi(m_1) = \varphi(m_2)$, ist $c = \varphi(m_1 m_2)$; 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 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) Implementierung in Maple, Scheme, Java, C++, ...

Zumindest kurz sei erwähnt, wie diese Algorithmen in den Programmiersprachen 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; in den Übungen wurde dazu Maple kurz vorgestellt.

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 $ax + by = \mathrm{ggT}(x, y)$ ist.

2.) Scheme: Auch Scheme kann (wie wohl alle Dialekte von LISP) mit ganzen Zahl praktisch beliebiger Länge rechnen; 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, wie bereits erwähnt, 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))
```

3.) 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.

4.) 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.

Praktisch alle Pakete zur Langzahlarithmethik 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 mehr 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 kann niemand sonst ein derartiges y erzeugen.

Ähnliche Verfahren werden in Zugangskontrollsystemen tatsächlich angewandt; zumindest wenn man seine Identität gegenüber *jedermann* so etablieren möchte, dürfen dazu auber auf keinen Fall dieselben RSA-Parameter benutzt werden, die man zur Ver- und Entschlüsselung geheimer Nachrichten einsetzt:

Die offensichtlichste Sicherheitslücke hier ist, 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 praktisch 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 $x = greeny^e \bmod N$ 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 $0 \leq a < N$ 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 obengenannten 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 authentizieren, 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 (KERKHOFFS 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 Authentizierungsprozedur 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:

$$
\begin{aligned}
& 2135987035920910082395022704999628797051095341 82\backslash \\
& 6417406442524165008583957746445088405009430865999 \\
= \; & 11139543251488279879254901754770248440709228448 43 \\
\times \; & 19174817025245044393757862682308621806969341892 93
\end{aligned}
$$

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; es gibt zwar im Augenblick noch keine veröffentlichten Faktorisierungen von 768-Bit RSA-Moduln, jedoch erwartet jedermann, daß diese kurz bevorstehen. Die Geheimdienste reicher und technologisch entwickelter Staaten sind wahrscheinlich bereits heute dazu in der Lage.

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 Beachrtung der Symmetrie) zufällig erzeugt. Für jede solche Seriennummer m erzeugt der Kunde eine Zufallszahl r, schickt $mr^e \bmod N$ 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 $v = m^d \bmod N$ für die Seriennummer m. Mit dieser Zahl v kann er bezahlen.

Der Zahlungsempfänger berechnet $v^e \bmod N$; 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 realisert 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 einfach noch deutlich kleiner war als heute, läßt sich von außerhalb nur schwer beurteilen.

CHAUMES' 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. Trotzdem kennt bis heute niemand auch nur eine Primzahl mit mindestens zehn Millionen Stellen (wofür die *Electronic Frontier Foundation,* die uns bereits beim DES-Cracker begegnet ist, einen Preis von 100 000 $ ausgesetzt hat).

Die größte derzeit bekannte Primzahl ist $2^{32\,582\,657} - 1$; sie hat hat 9 808 358 Dezimalstellen und wurde am 4. September 2006 im Rahmen der *Great Internet Mersenne Prime Search* (GIMPS) gefunden; siehe `www.mersenne.org`.

Wie bei allen Rekord-Primzahlen der letzten Jahre auch ist sie eine sogenannte MERSENNEsche Primzahl, d.h. eine Primzahl der Form $2^n - 1$. Das letzte Mal war zwischen 1989 und 1992 eine andere Zahl

$(391\,581 \cdot 2^{216\,193} - 1)$ größte bekannte Primzahl; ansonsten war es 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 $2 \equiv 1 \bmod 2^u - 1$, 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 anderen. 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 44 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 private 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 eine dreihundertstellige Zufallszahl erzeugen, nur knapp 10^{12} Möglichkeiten. Die kann ein entschlossener Gegner mit vertretbarem Aufwand durchprobieren.

Der hier eingesetzte Zufallsgenerator muß daher für seinen Startwert eine Anzahl von Möglichkeiten haben, die jenseits der Möglichkeiten eines zu erwartenden Gegners liegt. Wenn wir, wie bei symmetrischen Blockchiffren, davon ausgehens, daß niemand 2^{128} Möglichkeiten durchprobieren kann, 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, ist im Skriptum der Zahlentheorievorlesung des vergangenen Semesters zu finden.)

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 Millenium 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 uns auch, 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 (275–194 v.Chr.) 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.

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.

Da die Feinverteilung der Primzahlen sehr inhomogen ist, 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}
$\varepsilon \leq 7{,}16 \cdot 10^{-2}$	$2{,}87 \cdot 10^{-3}$	$8{,}46 \cdot 10^{-5}$	$1{,}70 \cdot 10^{-6}$	$2{,}77 \cdot 10^{-8}$
$p \approx 10^{120}$	10^{140}	10^{160}	10^{180}	10^{200}
$\varepsilon \leq 5{,}28 \cdot 10^{-12}$	$1{,}08 \cdot 10^{-15}$	$1{,}81 \cdot 10^{-19}$	$2{,}76 \cdot 10^{-23}$	$3{,}85 \cdot 10^{-27}$
$p \approx 10^{300}$	10^{400}	10^{500}	10^{600}	10^{700}
$\varepsilon \leq 5{,}8 \cdot 10^{-29}$	$5{,}7 \cdot 10^{-42}$	$2{,}3 \cdot 10^{-55}$	$1{,}7 \cdot 10^{-68}$	$1{,}8 \cdot 10^{-82}$
$p \approx 10^{800}$	10^{900}	10^{1000}	10^{2000}	10^{3000}
$\varepsilon \leq 5{,}4 \cdot 10^{-96}$	$1{,}0 \cdot 10^{-109}$	$1{,}2 \cdot 10^{-123}$	$8{,}6 \cdot 10^{-262}$	$3{,}8 \cdot 10^{-397}$

(Sie geben natürlich auch eine allgemeine Formel an, jedoch ist diese zu grausam zum Abtippen.)

Selbst wenn wir noch mit 512-Bit-Moduln arbeiten und somit knapp achtzigstellige Primzahlen brauchen, liegt also die Fehlerwahrscheinlichkeit bei nur etwa 10^{-5}; falls man sie erniedrigen möchte, testet man einfach mit mehreren zufällig gewählten Basen und hat dann etwa bei drei verschiedenen Basen eine Irrtumswahrscheinlichkeit von höchsten 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 aber selbst letzterer Wert schon bei einem einzigen FERMAT-Test unterschritten.

Einige Leute reden bei Zahlen, die einen FERMAT-Test bestanden haben, von „wahrscheinlichen Primzahlen“. 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 $\mathbb{Z}/p$.)

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 schnellen 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 mod 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: Fall 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 daher 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 kann man die Karte durchaus so beeinflussen, daß Fehler wahrscheinlich, aber nicht zu wahrscheinlich werden, so daß man eine gute Chance hat, daß genau eines der beiden Zwischenergebnisse falsch berechnet wird.

Sicherer ist es also auf jeden Fall, wenn man trotz des rund doppelt so hohen Aufwands direkt modulo N rechnet; die oben aufgestellte Regel, wonach man die Primzahlen so schnell wie möglich vergessen sollte, hat durchaus ihren Sinn.

Die Methode, eine Zahl modulo N aus ihren Restklassen modulo von Teilern von N zu bestimmen, hat aber noch viele andere Anwendungen, so daß wir den Satz hier allgemein formulieren wollen. Er wurde angeblich früher von chinesischen Generälen benutzt, um Truppenstärken zu berechnen, indem sie die Soldaten in Reihen verschiedener Breite antreten ließen und dann 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 linear in der Ziffernzahl des Exponenten steigt, wird gelegentlich vorgeschlagen, einen kleinen Verschlüsselungsexponenten e zu verwenden; populär sind etwa $e = 3$ 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, der ungefähr die gleiche Größenordnung hat wie der Moduk $N = pq$; 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.

(Für den *Beweis* sei auf das Zahlentheorie-Skriptum des vorigen Semesters verwiesen.)

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:

1. Schritt: Setze $x_0 = x$ und $a_0 = [x]$.

n-ter Schritt, $n \geq 1$*:* Falls $x_{n-1} = a_{n-1}$ 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 letzten 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 Antragsstellers 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 entfernetern Freunden, und wenn es zu Freunden von Freunden geht, nimmt das Vertrauen naturgemäs ab.

Bei PGP läst 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 vorkomen, 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 will und nich an jemand, 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 Einzel-

genehmigung 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 Lizens 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 assymetrischen 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 $0 \leq c \leq N - 1$ 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 $0 \leq c \leq N - 1$ 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

$$a_i = c_i^d \bmod N = as_i \bmod N \in [B_1, B_2] i\,,$$

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

$$\frac{B_1 + rN}{s_i} \leq v \quad \text{und} \quad \frac{B_2 + rN}{s_i} \geq u\,,$$

also

$$\frac{s_i u - B_2}{N} \leq r \leq \frac{s_i v - B_1}{N}$$

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 $s \leq s_0$ höchstens gleich N aber natürlich größer als B_2 ist, so daß as dann unmöglich akzeptiert werden kann.

Im i-ten Schritt für $i \geq 1$ 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

$$\left[\frac{B_1 + rN}{s_i}, \frac{B_2 + rN}{s_i}\right] \cap [u, v]$$

$$\text{mit} \quad [u, v] \in L \quad \text{und} \quad \frac{s_i u - B_2}{N} \leq r \leq \frac{s_i v - B_1}{N}\,.$$

Dieses Verfahren wird so lange forgesetzt, 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, Serveranrfragen 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 blcokweise 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 Zufallsbit 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 Verfikation 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 Unterschreiber 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 plaziert.

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 $n - r + 1$ Einsern 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 Einsern 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; zumindest 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 abdividert 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-Modulus 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 nun für $y = 0, 1, 2, \ldots$ die Zahlen $N + y^2$; falls er auf ein Quadrat x^2 stößt, berechnet er

$$\mathrm{ggT}(N, x+y) \quad \text{und} \quad \mathrm{ggT}(N, x-y)\,.$$

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 Faktoriserung 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 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

$$\mathrm{ggT}(x+y, N) \quad \text{und} \quad \mathrm{ggT}(x-y, N)$$

nichttriviale Teiler von N sind.

Beim quadratischen Sieb betrachten wir dazu das Polynom

$$f(x) = \left(x + \left[\sqrt{N}\right]\right)^2 - N\,.$$

Offensichtlich ist für jedes x

$$f(x) \equiv \left(x + \left[\sqrt{N}\right]\right)^2 \bmod N\,,$$

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

$$\prod_{i=1}^{r} f(x_i) \equiv \prod_{i=1}^{r} \left(x + \left[\sqrt{N}\right]\right)^2 \bmod N$$

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.

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 ε_i mod 2 und den e_{ip} mod 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

$$\prod_{i=1}^{r} f(x_i)^{\varepsilon_i} \equiv \prod_{i=1}^{r} \left(x + \left[\sqrt{N}\right]\right)^{2\varepsilon_i} \bmod N$$

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.

Da ε_i nur die Werte 0 und 1 annimmt, stehen in obigem Produkt natürlich keine echten Potenzen: Man multipliziert einfach nur die Faktoren miteinander, für die $\varepsilon_i = 1$ ist. Außerdem interessieren nicht die links- und rechtsstehenden Quadrate, sondern deren Quadratwurzeln; tatsächlich also berechnet man

$$x = \prod_{p\in\mathcal{B}} p^{\frac{1}{2}\sum_{i=1}^{r} \varepsilon_i e_{ip}} \quad \text{und} \quad y = \prod_{i=1}^{r} \left(x + \left[\sqrt{N}\right]\right)^{\varepsilon_i}\,.$$

wobei beide Produkte nur modulo N berechnet werden müssen.

Zum besseren Verständnis des Verfahrens wollen wir versuchen, damit die Zahl 15 zu faktorisieren. Dies ist zwar eine sehr untypische Anwendung, da das quadratische Sieb üblicherweise erst für mindestens etwa vierzigstellige Zahlen angewandt wird, aber zumindestens das Prinzip sollte auch damit klarwerden.

Als Faktorbasis verwenden wir die Menge

$$\mathcal{B} = \{2, 3, 7, 11\}\,;$$

die Primzahl fünf fehlt, da $3 \cdot 5 = 15$ ist und daher bei einer Faktorbasis, die sowohl drei als auch fünf enthält, die Gefahr zu groß ist, daß die linke wie auch die rechte Seite der Kongruenz durch fünfzehn teilbar ist. Bei realistischen Anwendungen muß man auf solche Überlegungen keine Rücksicht nehmen, denn dann sind die Elemente der Faktorbasis höchstens siebenstellig und somit erheblich kleiner als die gesuchten Faktoren.

Wir berechnen $f(x)$ für $x = 1, 2, \ldots,$ bis wir einige Funktionswerte haben, die über der Faktorbasis faktorisiert werden können. Die faktorisierbaren Werte sind in folgender Tabelle zusammengestellt:

x	$x + \left\lceil \sqrt{N} \right\rceil$	$f(x)$	*Faktorisierung*
1	4	1	
3	6	21	$3 \cdot 7$
5	8	49	7^2
6	9	66	$2 \cdot 3 \cdot 11$
10	13	154	$2 \cdot 7 \cdot 11$
54	57	3234	$2 \cdot 3 \cdot 7^2 \cdot 11$

Die erste und die dritte Zeile sind selbst schon Relationen der gesuchten Art, nämlich $4^2 \equiv 1 \bmod 15$ und $8^2 \equiv 7^2 \bmod 15$.

Die zweite Relation ist nutzlos, denn $8 - 7 = 1$ und $8 + 7 = 15$. Die erste dagegen führt zur Faktorisierung, denn

$$\mathrm{ggT}(4 + 1, 15) = 5 \quad \text{und} \quad \mathrm{ggT}(4 - 1, 15) = 3\,.$$

Da dies aber ein Zufall ist, der bei großen Werten von N so gut wie nie vorkommt, wollen wir das ignorieren und mit den Relationen zu $x = 3, 6, 10$ und 51 arbeiten:

$$\begin{aligned}
6^2 &\equiv 3 \cdot 7 && \bmod 15\\
9^2 &\equiv 2 \cdot 3 \cdot 11 && \bmod 15\\
13^2 &\equiv 2 \cdot 7 \cdot 11 && \bmod 15\\
54^2 &\equiv 2 \cdot 3 \cdot 7^2 \cdot 11 && \bmod 15
\end{aligned}$$

Multipliziert man die ersten drei dieser Relationen miteinander, folgt

$$(6 \cdot 9 \cdot 13)^2 \equiv (2 \cdot 3 \cdot 7 \cdot 11)^2 \bmod 15$$

oder $702^2 \equiv 462^2 \bmod 15$. Da

$$\mathrm{ggT}(702 - 462, 15) = \mathrm{ggT}(240, 15) = 15$$

ist, bringt das leider nichts.

Wir erhalten auch dann rechts ein Quadrat, wenn wir das Produkt der ersten, dritten und vierten Relation bilden; dies führt auf

$$(6 \cdot 13 \cdot 57)^2 \equiv (2 \cdot 3 \cdot 7^2 \cdot 11)^2 \bmod 15$$

oder $4446^2 \equiv 3234^2 \bmod 15$. Hier ist

$$\mathrm{ggT}(4446 - 3234, 15) = \mathrm{ggT}(1212, 15) = 3\,,$$

womit wir die Zahl 15 faktorisiert haben – wenn auch nicht unbedingt auf die einfachstmögliche Weise.

Bei realistischen Beispielen sind die Funktionswerte $f(x)$ deutlich größer als die Primzahlen aus der Faktorbasis; außerdem liegen die vollständig faktorisierbaren Zahlen viel dünner als hier: Bei der Faktorisierung einer hundertstelligen Zahl etwa muß man davon ausgehen, daß nur etwa jeder 10^9-te Funktionswert über der Faktorbasis zerfällt.

Daher ist es wichtig, ein Verfahren zu finden, mit dem diese wenigen Funktionswerte schnell und einfach bestimmt werden können. Das ist zum Glück möglich:

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 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

$$f(x) = \left(x - \left[\sqrt{N}\right]\right)^2 - N = 0 \Longleftrightarrow \left(x - \left[\sqrt{N}\right]\right)^2 = N\,,$$

und diese Gleichung ist genau dann lösbar, wenn es ein Element $w \in \mathbb{F}_p$ gibt mit Quadrat N, wenn also in $\mathbb{Z}$

$$w^2 \equiv N \bmod p$$

ist. Für $p > 2$ hat $f(x) = 0$ in $\mathbb{F}_p$ dann die beiden Nullstellen

$$x = \left[\sqrt{N}\right] \pm w\,;$$

andernfalls gibt es keine 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, die Faktorbasis sollte also nur Primzahlen enthalten, für die N modulo p ein Quadrat ist.

Für solche p kann man dann 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.

Das eigentliche Sieben zum Auffinden der komplett über der Faktorbasis zerlegbaren Funktionswerte $f(x)$ geht nun folgendermaßen vor sich: Man legt ein Siebintervall $x = 0, 1, \ldots, M$ fest und speichert in einem

Feld der Länge $M+1$ für jedes x eine ganzzahlige Approximation von $\log_2 f(x)$.

Für jede Primzahl p aus der Faktorbasis berechnet man dann die beiden Nullstellen $x_{1/2}$ von f modulo p im Intervall von 0 bis $p-1$ und subtrahiert von jedem Feldelement mit Index der Form $x_1 + kp$ oder $x_2 + kp$ eine ganzzahlige Approximation von $\log_2 p$.

Falls $f(x)$ über der Faktorbasis komplett faktorisierbar ist, sollte dann am Ende der entsprechende Feldeintrag bis auf Rundungsfehler gleich null sein; um keine Fehler zu machen, untersucht man daher für alle Feldelemente, die unterhalb einer gewissen Grenze liegen, durch Abdividieren, ob sie wirklich komplett faktorisieren, und man bestimmt auf diese Weise auch *wie* sie faktorisieren. Damit läßt sich dann das oben erwähnte Gleichungssystem über $\mathbb{F}_2$ aufstellen und, falls genügend viele Relationen gefunden sind, nichttrivial so lösen, daß eine der daraus resultierenden Gleichungen $x^2 \equiv y^2 \bmod p$ zu einer nichttrivialen Faktorisierung von N führt.

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.

Dabei muß natürlich berücksichtigt werden, daß nun auch negative Werte $f(x)$ auftreten können; dies geschieht einfach dadurch, daß man die Zahl -1 wie eine Primzahl behandelt, die genau dann in der Faktorisierung von $f(x)$ auftritt, wenn $f(x)$ negativ ist.

Als Polynome betrachtet man quadratische Polynome der Form

$$f(x) = ax^2 + 2bx + c \quad \text{mit} \quad 0 \le b < a \quad \text{und} \quad b^2 - ac = N\,.$$

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. Bei geschickter Vorgehensweise fällt diese Zeit jedoch kaum ins Gewicht:

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.

Mit Hilfe der Theorie quadratischer Reste lassen sich die Nullstellen modulo p aller möglicher Polynome zu einem gegebenen Wert von a simultan berechnen, wobei der Aufwand pro Polynom ziemlich gering ist. Man verwendet daher bei der Multipolynomialversion des quadratischen Siebs sehr viele Polynome und relativ kleine Siebintervalle $[-M,\ M]$.

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 bei optimaler Wahl aller Größen der Rechenaufwand zum Faktorisieren einer Zahl N mit der Multipolynomialvariante des quadratischen Siebs proportional ist zu $e^{\sqrt{\ln N \ln\ln N}}$.

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

$$F(x,y) = a_dx^d + a_{d-1}x^{d-1}y + \cdots + a_1xy^{d-1} + a_0y^d\,,$$

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

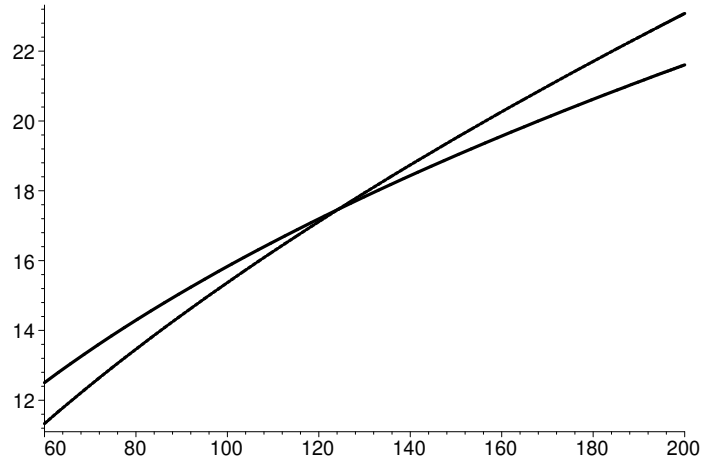
$$e^{c(\ln N)^{\frac{1}{3}}(\ln\ln N)^{\frac{2}{3}}}\ c = \sqrt[3]{\frac{64}{9}} \approx 1{,}923$$

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 (durgezogene Linie) ab etwa 125-stelligen Zahlen dem quadratischen Sieb (gestrichelte 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



Zeitaufwand für quadratisches Sieb und Zahlkörpersieb

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* regelmäßig zusammenstellte, denn hier geht es um Zahlen, die sorgfältig unter kryptographischen Gesichtspunkten ausgewählt wurden. Die größte erfolgreich faktorisierten unter diesen Zahlen war RSA-200, eine 200-stellige Dezimalzahl (entsprechend 663 Bit), deren Faktorisierung am 8. Mail 2005 beendet war. Begonnen wurde sie Ende 2003 von einem Team von Mathematikern aus Bonn und den Niederlanden (BAHR, BOEHM, FRANKE, KLEINJUNG). 2007 beendete *RSA Computer Security Incorperated* den Wettbewerb.

Der durch sein populärwissenschaftliches Buch zum Beweis der FERMAT-Vermutung bekannte Autor SIMON SINGH veröffentlichte 1999 unter dem Titel *The Code Book* ein Buch über Kryptographie und verband dies mit einem zehnstufigen Preisrätsel, in dem Kryptogramme verschiedener Schwierigkeitsstufe geknackt werden mußten. Die zehnte Stufe enthielt unter anderem eine RSA-Verschlüsselung mit 512 Bit.

Die Gruppe schwedischer Studenten, die das Gesamträtsel im Oktober 2000 als erste löste, bestand nicht, wie die Wissenschaftlergruppe, die RSA-200 (und eine Reihe anderer RSA-*challenges*) knackte, aus Experten für algorithmische Zahlentheorie; sie enthielt kein einziges Mitglied, das die Mathematik des Zahlkörpersiebs verstanden hätte. Da der Programmcode der RSA-155-Faktorisierung allerdings frei erhältlich ist, konnten sie damit arbeiten und ihn an ihre Bedürfnisse anpassen.

Wegen der fehlenden Erfahrung der Anwender war ihre Polynomauswahl deutlich schlechter als bei der nicht lange zuvor durchgefürten RSA-155-Faktorisierung durch Experten, so daß sie eine fast doppelt so langen Siebzeit brauchten. Da es an Universitäten, Instituten und Firmen viele Personen mit Zugriff auf große Rechnernetze gibt, war das Sieben aber kein großes Problem.

Das nächste Problem war die Lösung des linearen Gleichungssystems, für die bis dahin stets Supercomputer verwendet wurden, was natürlich jenseits der Möglichkeiten von Privatpersonen ist. Die Gruppe entwickelte daher ein neues Programm, das auch auf kleineren Rechnern läuft, und löste das Gleichungssystem in dreizehn Tagen auf einem Compaq quad processor ES40 System mit vier Alpha-Prozessoren mit 667 MHz und 8 GB Hauptspeicher, der von der Herstellerfirma zu diesem Zweck zur Verfügung gestellt wurde. Damit war gezeigt, daß auch relative Laien mit relativ beschränkten Mitteln eine 512-Bit-Faktorisierung durchführen können; einschließlich aller theoretischer Vorbereitungen mußten sie dazu ungefähr elf Monate aufwenden.

Es steht zu erwarten, daß entsprechende Faktorisierungen in wenigen Jahren routinemäßig durchgeführt werden können; die Tage der 512-Bit-Moduln sollten also hoffentlich bald gezählt sein. Im April 2004 erschien ein *preprint* von vier japanischen Mathematikern, die sich die Zahlen RSA-100 bis RSA-150 der alten Liste noch einmal vornahmen und alle mit derselben Hardware faktorisierten:

KAZAMARO AOKI, YUJI KIDA, TAKESHI SHIMOYAMA, HIROKI UEADA: GNFS Factoring Statistics of RSA-100, 110, . . . , 150

Sie benutzten Pentium 4 Computer mit 2,53 GHz und 1024 MB RDRAM unter FreeBSD und berechneten die Siebzeit *hochgerechnet auf einen solchen PC.* Zur Lösung des linearen Gleichungssystems verwendeten sie ein Cluster aus 16 über 100 Leitungen vernetzten PCs. Die Zeiten waren:

	Siebzeit dd:hh:mm:ss	*LGS-Zeit hh:mm:ss*
RSA-100	0 : 15 : 04 : 31	00:07:19
RSA-110	2 : 05 : 47 : 56	00:28:49
RSA-120	9 : 03 : 28 : 58	02:24:15
RSA-130	26 : 19 : 09 : 07	08:47:28
RSA-140	77 : 20 : 24 : 18	15:07:39
RSA-150	238 : 09 : 27 : 40	101:31:17

Die für RSA-150 angegebenen gut 238 Tage Siebzeit würden bei parallelem Rechnen auf 16 PCs zu knapp 15 Tagen, dazu kommen gut vier Tage für das LGS. Mit knapp drei Wochen Rechenzeit kann eine solche Zahl also bereits heute von jedem faktorisiert werden, der Zugang zu einem PC-Pool hat.

e) Faktorisierung durch Spezialhardware

Faktorisierungen mit dem quadratischen oder Zahlkörpersieb benötigen zwar zumindest für einige Schritte wie die Lösung des linearen Gleichungssystems leistungsfähige Rechner mit viel Speicher, aber im wesentlichen handelt es sich dabei doch um Standardcomputer. Da die Grundoperation des Siebens so einfach ist, dürfte es sich wohl auch kaum lohnen, dafür spezielle Chips zu entwerfen und einzusetzen.

Es gibt allerdings zumindest einen Vorschlag, wie man das Sieben mit einem optoelektronischen Gerät etwa um den Faktor Tausend beschlëunigen kann: ADI SHAMIR, einer der Erfinder des RSA-Verfahrens, entwarf 1999 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 Geschwindikeit von etwa 300 000 km/sec legt es pro Takt etwa drei Zentimeter zurück. Die Laufwegunterschiede der Lichtstahlen 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 fertigwerden.

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 Hundertausende 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 verbessertes Gerät 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 mann allerding bedenkt, daß bei der NSA Ingenieuere sitzen, die sehr viel mehr Erfahrung mit dem Bau elektronischer Schaltungen haben als Wissenschaftler an einer Fakultät für Informatik und angewandte Mathematik, spricht schon einiges dafür, daß es dort irgendwelche Hardware gibt, die 1024 Bit Zahlen faktorisieren kann – zehn Millionen Dollar sind beim Budget der NSA kein Problem, wenn es um wirklich wichtige Daten geht.

f) Folgerungen für die Wahl der Schlüssellänge

Wie wir bereits gesehen haben, veröffentlichen das Bundesamt für Sicherheit in der Informationstechnik und die Bundesnetzagentur jedes Jahr Empfehlungen für die Länge von RSA-Schlüsseln; falls man rechtsgültige elektronische Unterschriften braucht, muß man auf jeden Fall darauf achten, daß während der Vertragslaufzeit diese Mindestanforderungen erfüllt sind.

Für die Übermittlung verschlüsselter privater oder auch geschäftlicher Informationen darf allerdings jeder seine eigenen Standards festlegen; wir wollen sehen, was sich auf Grund der vorherigen Abschnitte für die Sicherheit in Abhängigkeit von der Schlüssellänge folgern läßt.

Schlüssel mit 512 Bit sind auch heute noch recht verbreitet, da amerikanische Firmen bis vor wenigen Jahren weder Hardware noch Software zur RSA-Verschlüsselung exportieren durften, die mit längeren Schlüsseln arbeitet. Wie wir oben gesehen haben, bieten diese Schlüssel heute noch einen einigermaßen akzeptablen Schutz gegen Gelegenheitshacker, aber definitiv nicht mehr. In wenigen Jahren dürften sie wohl völlig unsicher sein.

Eine weitere magische Grenze sind 1024 Bit, da viele Hersteller von kryptographischer Hardware keine längeren Schlüssel unterstützen oder gerade erst anfangen, sie zu unterstützen. Dies war schließlich auch der Hauptgrund dafür, daß die Regulierungsbehörde für Telekommunikation

und Post so lange brauchte, Empfehlungen für Schlüssel jenseits dieser Grenze durchzusetzen.

Es ist unwahrscheinlich, daß heute außerhalb vom Geheimdiensten großer Staaten (oder vielleicht sogar mit Ausnahme nur der NSA) irgendjemand in der Lage ist, ohne Zusatzinformationen einen 1024 Bit-Modul zu faktorisieren, allerdings kann niemand vorhersagen, welche neuen Faktorisierungsverfahren in den nächsten Jahren möglicherweise entdeckt werden oder welche neuen Rechnerarchitekturen möglichweise für einen Durchbruch auf dem Gebiet der Faktorisierung sorgen werden. Daher muß man für die Verschlüsselung von Information, die auch in einigen Jahren noch geheim sein muß, mit einer recht großzügig bemessenen Sicherheitsreserve arbeiten, und vor diesem Hintergrund erscheinen 1024 Bit-Moduln auch dann bedenklich, wenn man sich nur gegen Wirtschaftsspionage der Konkurrenz oder gegen Hacker schützen möchte.

Selbst bei 2048 Bit-Moduln können wir nicht wirklich wissen, ob sie noch in mehreren Jahren sicher sind: Falls beispielsweise die „Quantencomputer“, mit denen wir uns in Kapitel 9 beschäftigen werden, je Realität werden mit Quantenregistern, deren Länge in der Größenordnung der Bitzahl von RSA-Moduln liegt, wird die Faktorisierung dieser Moduln nicht mehr aufwendiger sein als die Verschlüsselung eines kurzen Texts. Dann wird also RSA *als Verfahren* unsicher.

Bislang gibt es allerdings gerade erst einem Quantencomputer, der leistungsfähig genug war, um die Zahl 15 in ihre Primfaktoren zu zerlegen; er benutzte einen Ansatz, bei dem es unwahrscheinlich erscheint, daß man damit wesentlich mehr als die sieben dort realisierten Quantenbits erreichen kann. Bei anderen Technologien, die erheblich besser skalierbar sein sollten, kam man bislang noch nicht über drei Quantenbits hinaus. Wir wollen Quantencomputer daher im Augenblick ignorieren und einfach ausrechnen, wie stark der Aufwand für eine Faktorisierung mit dem Zahlkörpersieb als dem schnellsten derzeit bekannten Algorithmus ansteigt, wenn wir die Bitzahl auf mehr als 512 erhöhen.

Die folgende Tabelle zeigt dies anhand der oben angegebenen Formel für den Zeitbedarf des Zahlkörpersiebs:

Dezimalziffern	*Bits*	*Zeitfaktor*
155	512	1,00
160	529	1,89
165	545	3,53
170	562	6,52
175	579	11,93
180	595	21,56
185	612	38,63
190	628	68,51
195	645	120,29
200	662	209,75
210	695	621,91
220	728	1788,09
230	761	4996,63
240	794	13596,95
250	828	36094,23
260	861	93615,09
270	894	237562,59
280	927	590593,30
290	961	1440060,02
300	994	3447569,30
310	1027	8111574,17

Falls man abschätzen möchte, welchen Aufwand ein Gegner *heute* treiben muß, um einen RSA-Schlüssel einer gegebenen Länge zu faktorisieren, gibt diese Tabelle wohl einen recht guten Überblick; falls man allerdings abschätzen möchte, wie lange eine Schlüssellänge noch sicher ist, dürfte sie auf deutlich zu optimistische Schlußfolgerungen führen.

Wie in der Vergangenheit werden wohl auch in der Zukunft immer neue Verfahren gefunden und bei den alten die Konstanten verbessert: So war beispielsweise die Faktorisierung von RSA-155 etwa viermal so schnell, wie anhand der (ebenfalls mit dem Zahlkörpersieb durchgeführten) Faktorisierung von RSA-150 zu erwarten gewesen wäre. Hinzu kommt, daß die Hardware immer schneller wird und vielleicht auch neue Spezialhardware hinzukommt.

Die Autoren der RSA-155-Faktorisierung haben Rekordfaktorisierungen der letzten dreißig Jahre untersucht (erster Rekord war 1970 die 39-stellige FERMAT-Zahl $2^{2^7} + 1$, die Maple heute auf einem Pentium II PC mit 266 MHz und 128 kB RAM in gut einer Viertelstunde faktorisiert) und kamen zu der heuristischen Formel, daß eine d-stellige allgemeine Dezimalzahl wohl erstmals im Jahr

$$13{,}24\sqrt[3]{d} + 1928{,}6$$

faktorisiert werden dürfte. Für gängige Bitlängen von RSA-Moduln sind also folgende Jahre zu erwarten, in denen entsprechende Moduln erstmalig faktorisiert werden:

Bit:	768	1024	1280	1536	2048	2560	3072	4096
Jahr:	2010	2018	2025	2031	2041	2050	2057	2070

Im Jahr 2100 würden demnach Zahlen mit etwa 7200 Bit faktorisiert; für das Jahr 3000, bis zu dem die Formel ganz sicher nicht gilt, ergäben sich etwa 1,7 Millionen Bit.

Dies bietet zumindest einen gewissen Anhaltspunkt für Verschlüsselungen, die längerfristig sicher sein müssen – auch wenn niemand garantieren kann, daß die nächsten Jahren keinen so dramatischen Durchbruch bei den Faktorisierungsverfahren bringen, daß dadurch alle diese Extrapolationen über den Haufen geworfen werden.

§9: Literatur

RSA ist immer noch das gebräuchlichste asymmetrische Kryptographieverfahren; es gibt daher kaum ein nach etwa 1980 erschienenes Lehrbuch der Kryptologie, das nichts darüber enthält. Lehrbücher mit einem Schwergewicht auf asymmetrischer Kryptographie sind

Dazu kommen eine ganze Reihe von Lehrbüchern der algorithmischen Zahlentheorie, die RSA behandeln, dabei aber ihr Hauptaugenmerk auf Primzahlen und auch Faktorisierung legen.

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 dieser Ausarbeitung 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` .