Wolfgang K. Seiler:
Kryptographie mit öffentlichen Schlüsseln

Sommerschule Datensicherheit, Mannheim 2003

§ 1: New directions in cryptography

Bei klassischen Kryptoverfahren 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.*

Ihr Nachteil 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 in Stanford, 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 es jedem Teilnehmer zu erlauben, mit jeden anderen so 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.

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 Gegner mit unbegrenzten Ressourcen 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.

Ein realer Gegner, der 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 es gab unter den Experten einige Skepsis bezüglich der Möglichkeit, solche Funktionen zu finden.

Tatsächlich gab es 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 solchen 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 *Government Communications Headquarters* (GCHQ), zu denen CESG gehört, einige Arbeiten aus der damaligen Zeit im Internet (`http://www.cesg.gov.uk/`), wo sie inzwischen allerdings anscheinend nicht mehr zugänglich sind.

Im akademischen Bereich gab es ein Jahr nach Erscheinen der Arbeit von Diffie und Hellman das erste Kryptosystem mit öffentlichen Schlüsseln: Drei junge Wissenschaftler 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 entwickelte beispielsweise auch einen Kandidaten für AES, der es immerhin bis in die Endrunde schaffte.

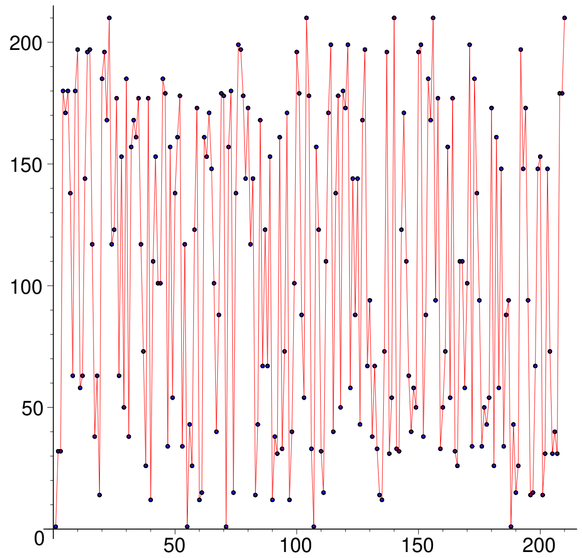
RSA ist übrigens identisch mit dem von Cocks vorgeschlagenen System, so daß einige Historiker auch Zweifel an den Behauptungen der GCHQ haben. Die Beschreibung durch Rivest, Shamir und Adleman erschien 1978 unter dem Titel *A method for obtaining digital signatures and public-key cryptosystems* in Comm.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

§2: Das RSA-Verfahren

Auch asymmetrische Kryptoverfahren beruhen auf den Shannonschen Prinzipien von *Konfusion* und *Diffusion.* Speziell beim RSA-Verfahren benutzt man dazu die Funktion

$$\{0, \ldots, N-1\} \to \{0, \ldots, N-1\}; \quad x \mapsto x^e \bmod N$$

für eine „geeignete“ Zahl N. Ein Blick auf den Graphen einer solchen Funktion legt die Vermutung nahe, daß hier wohl ein ziemliches Maß an Konfusion erreicht werden kann, und für hinreichend großes N sollte es auch einen bedeutenden Diffusionseffekt geben.



Die Funktion $x \mapsto x^5 \bmod 211$

In der Tat ist kein effizientes Verfahren bekannt, wie man für beliebiges N eine Umkehrfunktion berechnen kann, weder als Lauscher noch als legitimer Empfänger.

Für spezielle N allerdings gibt es solche Verfahren: Ist etwa $N = p$ eine Primzahl, so sagt uns der kleine Satz von FERMAT aus dem vorigen Vortrag, daß im Körper mit p Elementen jedes von Null verschiedene Elemente $(p-1)$-te Potenz eins hat, d.h.

$$a^{p-1} \equiv 1 \bmod p \quad \text{für alle } a \in \mathbb{Z} \text{ mit } a \not\equiv 0 \bmod p\,,$$

was man mittels der binomischen Formel auch leicht direkt beweisen kann: Da der Binomialkoeffizient $\binom{p}{i}$ für $i = 1, \dots, p-1$ durch p teilbar ist, folgt

$$(x+y)^p \equiv x^p + y^p \bmod p\,,$$

was mit $y = 1$ induktiv zeigt, daß

$$a^p \equiv a \bmod p \quad \text{für alle } a \in \mathbb{Z}\,.$$

Falls a kein Vielfaches von p ist, kam man in $\mathbb{F}_p$ durch a kürzen und erhält die Behauptung.

Falls nun e teilerfremd ist zu $p-1$, können wir nach EUKLID ganze Zahlen d, s finden, so daß

$$de + s(p-1) = \operatorname{ggT}(e, p-1) = 1$$

ist oder

$$de = 1 - s(p-1) \equiv 1 \bmod (p-1)\,.$$

Für jedes $a = 1, \dots, p-1$ ist dann

$$\left(a^e\right)^d \equiv a^{de} \equiv a^{1-s(p-1)} \equiv a \cdot \left(a^{(p-1)}\right)^s \equiv a \bmod p\,,$$

d.h. Potenzieren mit d ist invers zum Potenzieren mit e.

Falls d und e beide geheimgehalten werden, läßt sich daraus ein symmetrisches Kryptoverfahren machen, das sogenannte POHLIG-HELLMAN-Verfahren, mit dem man beispielsweise Poker per Telephon spielen kann. Für ein Verfahren mit öffentlichen Schlüsseln ist es allerdings offensichtlich ungeeignet, denn jeder, der e und p kennt, kann leicht d berechnen.

Für RSA wählt man daher N nicht als Primzahl, sondern als Produkt $N = pq$ zweier verschiedener Primzahlen. Dann ist für jede weder durch p noch durch q teilbare ganze Zahl a

$$a^{(p-1)(q-1)} \equiv \left(a^{p-1}\right)^{q-1} \equiv 1^{q-1} \equiv 1 \bmod p$$

und

$$a^{(p-1)(q-1)} \equiv \left(a^{q-1}\right)^{p-1} \equiv 1^{p-1} \equiv 1 \bmod q\,,$$

also

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

Man beachte, daß sich die Zahl $(p-1)(q-1) = N + 1 - (p+q)$ nicht ohne Kenntnis von p und q berechnen läßt; falls man sie kennt, erhält man p und q leicht als Lösungen einer quadratischen Gleichung.

Zur Erzeugung eines öffentlichen RSA-Schlüssels (N, e) wählt man daher zwei große Primzahlen p und q, die strikt geheim gehalten werden müssen, und berechnet $N = pq$. Außerdem wählt man einen Exponenten e, der zu $(p-1)(q-1)$ teilerfremd ist. Das Paar (N, e) wird veröffentlicht.

Den geheimen Schlüssel d berechnet man nach EUKLID: Da e und $(p-1)(q-1)$ größten gemeinsamen Teiler eins haben, liefert dessen Algorithmus Zahlen d, s, so daß

$$de + s(p-1)(q-1) = 1$$

ist und damit

$$\left(a^e\right)^d \equiv a \bmod N \quad \text{für alle zu } pq \text{ teilerfremden Zahlen } a\,.$$

Sind p und q hinreichend groß, ist das Auftreten einer Nachricht a, für die $\operatorname{ggT}(pq, a) \neq 1$ ist, so unwahrscheinlich, daß man diese Möglichkeit (die zur Faktorisierung von N führt!) getrost vernachlässigen kann.

a) Verschlüsselung

Jeder, der den öffentlichen Schlüssel (N, e) kennt, kann Nachrichten verschlüsseln: Er bricht die Nachricht auf in Blöcke, die durch ganze Zahlen zwischen Null und $N-1$ dargestellt werden können, berechnet für jeden so dargestellten Block den Chiffretext $b \equiv a^e \bmod N$, der als Zahl zwischen Null und $N-1$ interpretiert und an den Inhaber des geheimen Schlüssels geschickt wird. (Wir schreiben in solchen Fällen in Zukunft kurz $b = a^e \bmod N$, wobei „mod" in diesem Zusammenhang als die Berechnung des Divisionsrests bei Division durch N zu interpretieren ist.)

Der Empfänger berechnet $b^d \bmod N$; da

$$b^d \equiv a^{ed} \equiv a \bmod N$$

ist, entschlüsselt dies die Nachricht.

b) Identitätsnachweis

Im Gegensatz zu symmetrischen Kryptoverfahren endet die Nützlichkeit des RSA-Verfahrens nicht mit der bloßen Möglichkeit einer Verschlüsselung; das Verfahren kann beispielsweise auch benutzt werden, um in Zugangskontrollsystemen, vor Geldautomaten oder bei einer Bestellung im Internet die Identität einer Person Zu beweisen: Nur der Inhaber des geheimen Schlüssels d kann zu einem gegebenen a eine Zahl b berechnen, für die $b^e \equiv a \bmod N$ ist.

Falls also der jeweilige Gegenüber eine Zufallszahl a erzeugt und als Antwort das zugehörige b verlangt, kann er anhand eines öffentlichen Schlüsselverzeichnisses die Richtigkeit von b überprüfen und sich so von der Identität seines Partners überzeugen. Im Gegensatz zu Kreditkarteninformation oder Paßwörtern ist dieses Verfahren auch immun gegen Abhören: Falls jedesmal ein neues zufälliges a erzeugt wird, nützt ein einmal abgehörtes b nichts.

Trotzdem ist das Verfahren in dieser Form nicht als Ersatz zur Übertragung von Kreditkarteninformation oder ähnlichem geeignet, da der Gegenüber anhand des öffentlichen Schlüssels jederzeit zu einer willkürlich gewählten Zahl b die Zahl $a = b^e \bmod N$ erzeugen kann um dann zu behaupten, er habe b als Antwort darauf empfangen. Man müßte also beispielsweise noch zusätzlich verlangen, daß die Zahl a eine spezielle Form hat, etwa daß die vordere Hälte der Ziffernfolge identisch mit der hinteren Hälfte ist.

c) Elektronische Unterschriften

Praktische Bedeutung hat vor allem eine weitere Variante: die elektronische Unterschrift. Her 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, sofern gewisse formale Voraussetzungen erfüllt sind, rechtsverbindlich.)

Um einen Nachrichtenblock a mit $0 \leq a < N$ zu unterschreiben, berechnet der Inhaber des öffentlichen Schlüsseln (N, e) mit seinem geheimen Schlüssel d die Zahl

$$b = a^d \bmod N$$

und sendet das Paar (a, b) an den Empfänger. Dieser überprüft, ob

$$b^e \equiv a \bmod N\,;$$

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, b) zu erzeugen, kann er auch gegenüber einem Dritten beweisen, daß der Absender 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 $b^e \bmod N$ für ein falsch berechnetes b mit an Sicherheit grenzender Wahrscheinlichkeit keine sinnvolle Nachricht ergibt.

Falls die übermittelte Nachricht geheimgehalten werden soll, müssen a und b 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 unterschreibt man meist nicht die Nachricht selbst, sondern einen daraus extrahierten 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 beschäftigt sich ein eigenes Teilgebiet der Kryptologie.

d) Blinde Unterschriften und elektronisches Bargeld

Einer der erfolgversprechendsten Ansätze zum Aushebeln eines Kryptosystems besteht darin, sich auf die Dummheit seiner Mitmenschen zu verlassen.

So sollte es durch gutes Zureden nicht schwer sein, jemanden zu Demonstrationszwecken zum Unterschreiben einer sinnlosen Nachricht zu bewegen: Eine Folge von Nullen und Einsen ohne sinnvolle Interpretation hat schließlich keine rechtliche Wirkung.

Nun muß eine sinnlose Nachricht aber nicht unbedingt eine Zufallszahl sein: Sie kann sorgfältig präpariert sein. Sei dazu etwa m eine Nachricht, die ein Zahlungsversprechen enthält, (N, e) der öffentliche Schlüssel des Opfers und r eine Zufallszahl zwischen 2 und $N - 2$. Dann wird

$$x = m \cdot r^e \bmod N$$

wie eine Zufallsfolge aussehen, für die man eine Unterschrift

$$u = x^d \bmod N = (mr^e)^d \bmod N = m^d r \bmod N$$

bekommt. Multiplikation mit r^{-1} macht daraus eine Unterschrift unter die Zahlungsverpflichtung m.

Das angegebene Verfahren kann nicht nur von Trickbetrügern benutzt werden; blinde Unterschriften sind auch die Grundlage von *digitalem 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.

Es wir ausgegeben von einer Bank, die für jede angebotene Stückelung einen öffentlichen Schlüssel (N, e) bekannt gibt. Eine 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 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 oben berechnet er daraus durch Multiplikation mit r^{-1} die Unterschrift $v = m^d \bmod N$ für die Seriennummer N, und mit diesem Block kann er bezahlen.

Der Zahlungsempfänger berechnet $v^e \bmod N$; falls dies die Form einer gültigen Seriennummer hat, kann er 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 sie darin noch nicht vorkommt, wird sie eingetragen und der Händler bekommt sein Geld; andernfalls verweigert sie die Zahlung.

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.

Das System kann nur funktionieren, wenn in diesem Fall 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.

Da digitales Bargeld überdies 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 auch nicht sehr groß.

§3: Wie findet man große Primzahlen?

a) Wie groß sollten die Primzahlen sein?

Das RSA-Verfahren ist nur sicher, wenn die Primzahlen p und q hinreichend groß gewählt werden. Als erstes müssen wir uns also die Frage stellen, was „hinreichend groß“ bedeuten soll.

Ein treu sorgender Staat bleibt seinen Bürgern auf eine so wichtige Frage natürlich keine Antwort schuldig: Zwar gibt es noch keine oberste Bundesbehörde für Primzahlen, aber das Bundesamt für Sicherheit in der Informationstechnik (BSI) und die Regulierungsbehörde für Telekommunikation und Post erarbeiten jedes Jahr ein gemeinsames Dokument mit dem Titel *Geeignete Kryptoalgorithmen gemäß §17 (2) SigV.*

SigV steht für die aufgrund des Signaturgesetzes SigG erlassene Signaturverordnung; beide gemeinsam 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 mit der verwendeten Schlüssellänge ein „geeigneter Kryptoalgorihmus“ im Sinne der jeweils gültigen Veröffentlichung der Regulierungsbehörde ist.

Da Rechner immer schneller und leistungsfähiger werden und auch auf der theoretisch-algorithmischen Seite fast jedes Jahr kleinere oder größere Fortschritte zu verzeichnen sind, gelten die jeweiligen Empfehlungen

nur für etwa sechs Jahre. Offiziell geht es dabei jeweils nur um die Empfehlung geeigneter Algorithmen für elektronische Unterschriften und 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 interessierte Öffentlichkeit, von der die Kommentare zu den Entwürfen kommen, besteht naturgemäß in erster Linie aus Anbietern von Kryptographie-Software, 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 die Verfahren, die keines der Konkurrenzunternehmen anbietet.)

Derzeit erhältliche Hardware-Implementierungen von RSA unterstützen typischerweise Schlüssellängen von bis zu 1024 Bit; größere Schlüssel sind vor allem 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, aber die Richtlinien für 2000 legten fest, daß danach bis Mitte 2005 eine Mindestgröße von 1024 Bit erforderlich sei, 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 2002-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; veröffentlicht 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 bleiben dringend empfohlen, und in Kürze wird wohl der erste Entwurf der Richtlinien für 2003 erscheinen . . .

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

$$\varepsilon_1 = 0{,}5 \quad \text{und} \quad \varepsilon_2 = 30$$

vorgeschlagen; ist p die kleinere der beiden Primzahlen, soll also

$$\sqrt{2}p < q < 2^{30}p \approx 10^9 p$$

sein.

Wir sollten also, wenn wir wirklich sicher sein wollen, in der Lage sein, Primzahlen mit etwa 1024 Bit oder 309 Dezimalstellen zu finden.

Im Land der unbegrenzten Möglichkeiten ist das kein großes Problem: Dort kann man Primzahlen, wie alles andere auch, einfach kaufen. Unter Sicherheitsgesichtspunkten ist das allerdings 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 die öffentlichen Schlüssel seiner Konkurrenz durch die gerade gekauften Primzahlen teilen kann und damit teilweise Erfolg hat.

Sowohl aus Sicherheitsgründen als auch weil wir Mathematiker sind, sollten wir also unsere Primzahlen selbst erzeugen. Die größte derzeit bekannte Primzahl, gefunden am 14.November 2001, ist $2^{13\,466\,917} - 1$ mit 4 053 946 Dezimalstellen; da etwa 300 Dezimalstellen (entsprechend 1024 Bit) verglichen damit recht wenig ist, sollte die Suche nach solchen Primzahlen eigentlich nicht sonderlich schwierig sein.

Zu bedenken ist allerdings, daß Rekordprimzahlen allesamt eine sehr spezielle Struktur haben; 13 466 917 etwa ist eine Primzahl, so daß $2^{13\,466\,917} - 1$ die Form $2^p - 1$ hat; solche Zahlen bezeichnet man als MERSENNEsche Primzahlen. Bislang sind 39 solcher Zahlen bekannt, und schon daraus folgt, daß diese Zahlen für die Kryptographie völlig nutzlos sind: 39 Möglichkeiten kann schließlich jeder durchprobieren.

Unsere Primzahlen müssen also zufällig und allgemein sein, da sonst ein Gegner einfach alle Primzahlen einer speziellen Form durchprobieren kann. Idealerweise sollte jede Primzahl der richtigen Größenordnung die gleiche Chance haben, daß sie gewählt wird. Das läßt sich am sichersten dadurch erreichen, daß wir Zufallszahlen erzeugen und diese Testen, ob sie Primzahlen sind; wir brauchen daher einen Test, der uns sagt, ob eine gegebene Zahl prim ist.

b) Der Fermat-Test

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

$$3^{F_{20}-1} \equiv 1 \bmod F_{20}\,.$$

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 meinen, das sei die teuerste bislang produzierte 1-Bit-Information.)

Die Umkehrung der obigen Aussage gilt leider nicht: Ist beispielsweise

$$p = (6t+1)(12t+1)(18t+1)$$

Produkt dreier *Primzahlen* dieser speziellen Form, so ist nach FERMAT

$$a^{6t} = 1 \bmod (6t+1), \qquad a^{12t} = 1 \bmod (12t+1)$$

und

$$a^{18t} = 1 \bmod (18t+1)\,,$$

also

$$a^{36t} = 1 \bmod p\,.$$

Wegen

$$p-1 = 36t(36t^2+11t+1)$$

ist dann auch

$$a^{p-1} = 1 \bmod p$$

für alle zu p teilerfremden ganzen Zahlen a.

Zahlen dieser Art gibt es tatsächlich, z.B.

$$1729 = 7\times 13\times 19, \quad 294409 = 37\times 73\times 109 \quad usw.\,;$$

es gibt auch entsprechende Zahlen, die nicht von dieser speziellen Form sind: Wir benötigen offenbar nur, daß für jeden Primteiler q von p auch $q-1$ ein Teiler von $p-1$ ist. Es ist nicht bekannt, ob es unendlich viele Zahlen mit dieser Eigenschaft gibt, es gilt aber als wahrscheinlich.

Trotzdem wird es für große Zahlen zunehmend unwahrscheinlich, daß eine Zahl p für auch nur ein a den obigen Test besteht, ohne Primzahl zu sein. Rechnungen von

SU HEE KIM, CARL POMERANCE: The probability that a Random Probable Prime is Composite, *Math. Comp.* **53** (1989), 721–741

geben folgende obere Schranke für die Fehlerwahrscheinlichkeit ε:

$p \approx 10^{60}$	10^{70}	10^{80}	10^{90}	10^{100}
$\varepsilon \le 7{,}16\cdot 10^{-2}$	$2{,}87\cdot 10^{-3}$	$8{,}46\cdot 10^{-5}$	$1{,}70\cdot 10^{-6}$	$2{,}77\cdot 10^{-8}$
$p \approx 10^{120}$	10^{140}	10^{160}	10^{180}	10^{200}
$\varepsilon \le 5{,}28\cdot 10^{-12}$	$1{,}08\cdot 10^{-15}$	$1{,}81\cdot 10^{-19}$	$2{,}76\cdot 10^{-23}$	$3{,}85\cdot 10^{-27}$

(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.

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 Regulierungsbehörde empfiehlt allerdings, bei probabilistischen Primzahltests eine Irrtumswahrscheinlichkeit von höchsten $2^{-80} \approx 8{,}27\cdot 10^{-25}$ zuzulassen; diese Schranke wird erst bei knapp zweihundertstelligen Primzahlen mit einem einzigen FERMAT-Test erreicht. Immerhin, wenn sich die 1280 Bit Mindestlänge von RSA-Moduln durchsetzen sollte, braucht man etwas mehr als 190-stellige Primzahlen, und damit dürfte man in diesem Bereich sein.

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 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.

Natürlich kennt die Mathematik auch Verfahren, um exakt festzustellen, ob eine Zahl prim ist oder nicht; das einfachste besteht darin, alle potentiellen Primteiler einfach auszuprobieren. Es gibt aber natürlich deutlich bessere Verfahren, die auch für tausendstellige Zahlen in wenigen Minuten entscheiden können, ob diese prim sind. Das für Zahlen im kryptographisch interessanten Bereich derzeit beste Verfahren beruht auf elliptischen Kurven und soll hier nicht weiter behandelt werden.

c) Wie dicht liegen die Primzahlen?

Bevor man daran geht, Zufallszahlen zu testen, sollte man zunächst wissen, wie wahrscheinlich es ist, daß eine zufällig gewählte Zahl prim ist. Dies ist ein sehr altes Problem, das immer noch nicht ganz vollständig gelöst ist; die bekannte Antwort ist aber für praktische Zwecke gut genug. Die Mathematik, die im Beweis der folgenden Aussagen steckt, ist allerdings 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 $x/\ln x$. 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 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 $c > 0$.

Für uns wichtig ist die Folgerung: Unter den Zahlen der Größenordnung N haben der Primzahlen die Dichte $1/\ln N$, d.h. der Abstand zwischen zwei Primzahlen liegt im Durchschnitt 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.

d) Das Sieb des Eratosthenes

Da ein FERMAT-Test für Zahlen mit mehreren hundert Stellen einiges an Zeit kostet, verzichtet man meist auf die Erzeugung wirklich zufälliger Primzahlen sondern nimmt stattdessen zu einer Zufallszahl n die kleinste Primzahl $p \geq n$. Damit sind nicht mehr alle Primzahlen gleich wahrscheinlich, sondern eine Primzahl wird mit umso größerer Wahrscheinlichkeit gewählt, je weiter sie von der nächstkleineren Primzahl entfernt ist. Bislang wurden keine Ansätze bekannt, wie ein Gegner diese Ungleichbehandlung der Primzahlen ausnutzen kann.

Diese Zahl p kann man wie folgt suchen: Man definiert ein Suchintervall $[n, n+a]$, in dem Primzahlen zu erwarten sind; beispielsweise könnte man $a \approx 5\ln n$ oder $a \approx 10\ln n$ setzen, so daß etwa fünf oder zehn Primzahlen zu erwarten sind.

In diesem Intervall sind alle geraden Zahlen uninteressant; diese können also gleich gestrichen werden. Sodann sind auch alle Dreierzahlen uninteressant; wir bestimmen also n mod 3; sodann kennen wir die erste auf n folgende Dreierzahl und streichen von dort ausgehend jede dritte Zahl. Genauso geht es weiter mit fünft, sieben und so weiter, bis zu einer Primzahl, die etwa die Gröse von a hat. (Bei größeren Primzahlen gibt es nur noch selten etwas zu streichen.) Die übriggebliebenen Zahlen haben dann zumindest keine kleinen Primteiler, und wir beschränken uns darauf, den FERMAT-Test für diese Zahlen durchzuführen.

§4: 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 (z.B. $x_{i+1} = x_i^2 + c \bmod N$) Zufallszahlen und berechnet deren ggT mit der zu faktorisierenden Zahl. Da der EUKLIDische Algorithmus im Vergleich zur Erzeugung der Zufallszahlen relativ teuer ist, empfiehlt es sich, die erzeugten Zufallszahlen zunächst modulo N miteinander zu multiplizieren und dann erst in etwa jedem hundertsten Schritt den ggT von N mit diesem Produkt zu berechnen. (Dies setzt voraus, daß alle sehr kleinen Faktoren bereits abdividiert sind; sonst ist die Gefahr zu groß, daß im Produkt von hundert Zufallszahlen mehr als ein Primfaktor steckt.) Bei Faktoren mit mehr als acht Dezimalstellen wird die Methode schnell langsamer, so daß man dann zu alternativen Verfahren übergehen sollte.

Die nächste Klasse von Verfahren beruht auf gruppentheoretischen Überlegungen, im wesentlichen dem kleinen Satz von FERMAT im Falle zyklischer Gruppen und Verallgemeinerungen auf weitere Gruppen wie die multiplikative Gruppe eines Körpers $\mathbb{F}_{p^2}$ oder einer elliptischen Kurve. Diese Verfahren sind sehr effizient, wenn die Gruppenordnung nur relativ kleine Primteiler hat. Aus diesem Grund wurde früher häufig empfohlen, daß für die Primteiler p eines RSA-Modulus N sowohl $p-1$ als auch $p+1$ jeweils mindestens einen „großen“ Primfaktor haben sollen. Da diese Verfahren jedoch ihre Stärke bei Faktoren mit einer Länge von bis etwa 30 oder 35 Dezimalstellen haben und ein N mit 70 Dezimalstellen, wie wir bald sehen werden, heute auch aus anderen Gründen völlig unsicher ist, gilt diese Empfehlung heute nicht mehr, und wir müssen uns nicht genauer mit diesen Verfahren befassen.

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

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

Wenn er Pech hat, sind dies die beiden Zahlen eins und N, wenn er Glück hat, sind es p und q. Für zufällig gewählte Paare (x, y) kommt beides jeweils mit fünfzigprozentiger Wahrscheinlichkeit vor.

Falls p und q nahe beieinander liegen, führt schon ein kleines y zur korrekten Faktorisierung; bei der Wahl der Primzahlen muß also darauf geachtet werden, daß sie zwar die gleiche *Größenordnung* haben, aber nicht zu weit beieinander liegen. Liegt etwa p in der Größenordnung von $2q$, hat die Differenz die gleiche Größenordnung wie p, und FERMATS Verfahren bräuchte etwa p Rechenschritte, wird also vom Aufwand her vergleichbar mit der Faktorisierung durch Abdividieren. Somit kann man sich auch gegen diese Attacke recht gut schützen.

Wirklich gefährlich sind eine Klasse von Siebverfahren, die auf FERMATS Methode aufbauen. Diese Verfahren sind die schnellsten derzeit bekannten zur Faktorisierung von RSA-Moduln; die Wahl einer sicheren Ziffernlänge hängt also davon ab, welche Zahlen diese Verfahren faktorisieren können.

b) Das quadratische Sieb

Das quadratische Sieb ist der Grundalgorithmus der ganzen Klasse; es ist logisch einfacher, allerdings vor allem für große Zahlen auch deutlich langsamer als die Variationen, mit denen wir uns im nächsten Abschnitt 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

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

wobei links und rechts verschiedene Zahlen stehen. Insbesondere steht links im allgemeinen keine Quadratzahl.

Falls wir allerdings Werte $x_1, x_2, \dots, x_r$ 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. Ist

$$f(x_i) = \prod_{p\in\mathcal{B}} p^{e_{ip}}\,,$$

so ist

$$\prod_{i=1}^{r} f(x_i)^{\varepsilon_i} = \prod_{p\in\mathcal{B}} p^{\sum_{i=1}^{r} \varepsilon_i e_{ip}}$$

genau dann ein Quadrat, wenn

$$\sum_{i=1}^{r} \varepsilon_i e_{ip}$$

für alle $p \in \mathcal{B}$ gerade ist. Dies hängt natürlich nur ab von den $\varepsilon_i \bmod 2$ und den $e_{ip} \bmod 2$; wir können ε_i und e_{ip} daher als Elemente des Körpers mit zwei Elementen auffassen und bekommen dann über $\mathbb{F}_2$

das Gleichungssystem

$$\sum_{i=1}^{r} \varepsilon_i e_{ip} = 0 \quad \text{für alle } p \in \mathcal{B}\,.$$

Betrachten wir die ε_i als Variablen, ist dies ein homogenes lineares Gleichungssystem in r Variablen mit soviel Gleichungen, wie es Primzahlen in der Faktorbasis gibt. Dieses Gleichungssystem hat nichttriviale Lösungen, falls die Anzahl der Variablen die der Gleichungen übersteigt, falls es also mehr Zahlen x_i gibt, für die $f(x_i)$ über der Faktorbasis faktorisiert werden kann, als Primzahlen in der Faktorbasis.

Für jede nichttriviale Lösung ist

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

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

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

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	$x + \left[\sqrt{N}\right]$	$f(x)$	*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

$$4^2 \equiv 1 \bmod 15 \quad \text{und} \quad 8^2 \equiv 7^2 \bmod 15\,.$$

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:

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

$$54^2 \equiv 2 \cdot 3 \cdot 7^2 \cdot 11 \mod 15$$

Multipliziert man die ersten drei dieser Relationen miteinander, folgt

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

oder $702^2 \equiv 462^2 \mod 15$. 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

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

oder $4446^2 \equiv 3234^2 \mod 15$. 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

$$f(x) \equiv 0 \mod p$$

ist. Da für $x, y, a, b \in \mathbb{Z}$ und mit $x \equiv y \mod p$ und $a \equiv b \mod p$ gilt

$$a + x \equiv b + y \mod p \quad \text{und} \quad a \cdot x \equiv b \cdot y \mod p$$

und für $n \in \mathbb{N}$ auch

$$x^n \equiv y^n \mod p\,,$$

ist für jedes Polynom f mit ganzzahligen Koeffizienten

$$f(x) \equiv f(y) \mod p\,.$$

Ist also insbesondere $f(x) \equiv 0 \mod p$, so ist auch

$$f(x + kp) \equiv 0 \mod p \quad \text{für alle } k \in \mathbb{Z}\,.$$

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 $\mathbb{Z}$

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

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 $x \leq 10^{10}$ 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

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

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

$$F(x, y) = a_d x^d + a_{d-1} x^{d-1} y + \cdots + a_1 x y^{d-1} + a_0 y^d ,$$

wobei die a_i mit $i < d$ höchstens Betrag $m/2$ haben.

Hinzu kommt das homogene lineare Polynom

$$G(x, y) = x - my ;$$

das Sieb sucht nach Zahlenpaaren (x, y), für die sowohl $F(x, y)$ als auch $G(x, y)$ über der Faktorbasis zerfallen. Da die Polynome von zwei Variablen abhängen, muß man entweder jeweils eine Variable festhalten und über die andere sieben, oder aber ein zweidimensionales Siebverfahren anwenden; üblich ist eine Kombination beider Methoden.

Gesucht sind Paare (x, y) teilerfremder ganzer Zahlen, für die

$$F(x, y) \quad \text{und} \quad G(x, y)$$

beide über der gewählten Faktorbasis zerfallen, und das Ziel ist, wie beim quadratischen Sieb, eine Relation der Form

$$\prod_{(x,y)\in\mathcal{M}} F(x, y) \equiv \prod_{(x,y)\in\mathcal{M}} G(x, y) \mod N ,$$

in der rechts und links Quadrate stehen.

Die besten derzeit bekannten Strategien zur Polynomauswahl *usw.* führen auf eine Laufzeitabschätzung proportional

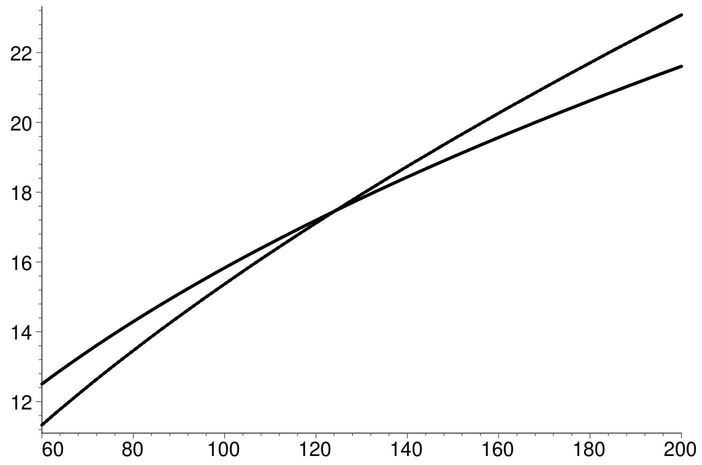
$$e^{c(\ln N)^{\frac{1}{3}}(\ln\ln N)^{\frac{2}{3}}} c = \sqrt[3]{\frac{64}{9}} \approx 1{,}923$$

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 (durchgezogene 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 zusammenstellt, denn hier geht es um Zahlen, die sorgfältig unter kryptographischen Gesichtspunkten ausgewählt wurden.

Die größte bislang faktorisierte Zahl aus dieser Liste ist RSA-155, eine, wie der Name sagt, 155-stellige natürliche Zahl. Diese Ziffernlänge ist vor allem deshalb interessant, weil sie 512 Binärziffern entspricht, also jener Schlüssellänge, die Web-Browser typischerweise für die sogenannten „sicheren“ Verbindungen wählen und die auch im SSL-Protokoll *(Secure Socket Layer)* verwendet wird.

RSA-155 wurde 1999 mit dem Zahlkörpersieb faktorisiert; beteiligt waren 16 Wissenschaftler von 14 Institutionen aus drei Kontinenten, 160 *workstations,* acht R10000-Prozessoren, 120 Pentium II PCs, vier Digital/Compaq boxes, die in 3,7 Monaten mit einer Rechenleistung von 35,7 CPU-Jahren das Sieben durchführten. Sodann brauchte ein SGI

Origin 2000 Computer einen Monat, um eine Matrix mit 6 699 191 Zeilen und 6 711 336 Spalten aufzustellen, aus der schließlich eine CRAY C916 in 214 CPU-Stunden mit zwei Gigabyte RAM 64 Relationen fand. Für vier dieser Relationen berechnete die SGI Origin 2000 die Quadratwurzeln links und rechts mit einem Rechenaufwand von zwischen etwa vierzig und etwa sechzig CPU-Stunden pro Relation. Die anschließenden Faktorisierungsversuche durch Anwendung des EUKLIDischen Algorithmus benötigten natürlich kaum eine Sekunde und lieferten die Zerlegung in zwei 78-stellige Primzahlen.

Wie man sieht, ist diese Faktorisierung noch deutlich jenseits der Möglichkeiten eines Gelegenheitshackers; Zugriff auf ein Netz von etwa dreihundert *workstations* und PCs ist zwar kein großes Problem, aber 214 CPU-Stunden auf einer CRAY sind außer für Regierungsstellen und Forschungszentren höchstens noch für Großunternehmen realistisch, wobei selbst bei denen einiges an kreativer Buchführung notwendig wäre, falls der Faktorisierungsversuch geheim bleiben soll.

Dieser aus Sicht eines Anwenders von 512-Bit RSA-Verschlüsselung einzig positive Aspekt der Faktorisierung von RSA-155 galt aber leider nur wenige Monate; dann wurde eine weitere Faktorisierung einer 512-Bit-Zahl bekannt, die unter völlig anderen Rahmenbedingungen zustande kam.

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 RSA-155-Gruppe, aus führenden 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 die Polynomauswahl hier deutlich schlechter als bei RSA-155, was zu einer fast doppelt so langen Siebzeit führte.

Das nächste Problem war die Lösung des linearen Gleichungssystems, für die bis dahin nur CRAY-Code zur Verfügung stand. 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.

Die aktuellen *Factoring Challenges* von RSA Security sind unter `www.rsasecurity.com/rsalabs/challenges/factoring/numbers.html` zu finden; sie beginnen mit einer 576-Bit-Zahl (174 Dezimalstellen), die immer noch darauf wartet, daß jemand die 10 000 $ einfordert, die für ihre Faktorisierung ausgesetzt sind. Für die größte Zahl der Liste, RSA-2048 mit 617 Dezimalstellen, gibt es 200 000$.

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 beschleünigen kann: ADI SHAMIR, einer der Erfinder des RSA-Verfahrens, entwarf 1999 TWINKLE, *The Weitzman INstitut Key Locating Engine.*

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 zehn Mal pro Sekunde. Mit diesen Datenraten können auch einfache Computer leicht fertig werden.

Das Hauptproblem ist der Bau das Chips mit den Dioden und deren Steuerelektronik; SHAMIR meint, daß dies mit der heute existierende

GaAs-Technologie gerade möglich sein sollte, und möglicherweise stehen inzwischen schon solche oder ähnliche Maschinen in Labors von NSA und ähnlichen Organisationen. In der offenen Literatur ist nicht über die Existenz solcher Maschinen bekannt und auch nichts über Pläne welche zu bauen. Der Entwicklungsaufwand dürfte sicherlich in die 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.

f) Folgerungen für die Wahl der Schlüssellänge

Wie wir bereits gesehen haben, veröffentlichen das Bundesamt für Sicherheit in der Informationstechnik und die Regulierungsbehörde für Telekommunikation und Post 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.

Die Faktorisierung von RSA-155 zeigt, daß ein öffentlicher Schlüssel N mit 512 Bit faktorisiert werden kann, wenn auch derzeit noch mit beträchtlichem Aufwand sowohl an Rechenzeit als auch an Kalenderzeit. Ein solcher Schlüssel bietet also keine Sicherheit, gegen einen Opponenten, der ein größeres Netzwerk an Rechnern einsetzen kann und für den

die Information auch noch in etwa sechs Monaten diesen Einsatz wert ist. (In den nächsten Jahren wird natürlich sowohl die Gesamtrechenzeit als auch die verflossene absolute Zeit für eine solche Faktorisierung recht schnell sinken.) Zur Übertragung einer Kreditkartennummer mit einem Kreditlimit von wenigen Tausend Euro oder eines Paßworts, das man (hoffentlich) ohnehin in wenigen Monaten ändert, sollten 512 Bit allerdings zumindest gegen Gelegenheitshacker im Augenblick noch genügend Schutz bieten.

Jedes weitere Bit Schlüssellänge erhöht die Sicherheit, macht aber wegen der magischen Grenze von 512 Bit bei vieler Software einen Upgrade erforderlich, der in vielen Fällen nicht erhältlich sein dürfte: Für amerikanische Hersteller etwa gelten 512 Bit immer noch als Grenze, jenseits derer für RSA zumindest keine Exportgenehmigungen im vereinfachten Verfahren mehr gegeben werden. Trotzdem gibt es natürlich sowohl Hardware-Unterstützung als auch reine Software-Implementierungen für 2048-Bit-RSA, allerdings kostet Hardware und meist auch Software Geld, und 2048-Bit-RSA ist um den Faktor $4^3 = 64$ aufwendiger als 512-Bit-RSA, so daß viele Anwender weiter mit nur 512 Bit arbeiten.

Niemand kann vorhersagen, welche neuen Faktorisierungsverfahren in den nächsten Jahren möglicherweise entdeckt werden oder welche völlig neuen Rechnerarchitekturen möglichweise für einen Durchbruch bei der Faktorisierung sorgen werden: Falls beispielsweise die „Quantencomputer“, mit denen wir uns in verschiedenen Vorträgen dieser Sommerschule noch beschäftigen werden, je Realität werden mit Quantenregistern, deren Länge in der Größenordnung der Bitzahl der RSA-Moduln liegt, wird die Faktorisierung der Moduln nicht mehr schwieriger sein als die Verschlüsselung eines kurzen Texts.

Bislang ist es allerdings noch ein ungelöstes Problem, die Zahl 15 mit einem Quantencomputer zu faktorisieren; daher wollen wir diese 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 all diese Extrapolationen über den Haufen geworfen werden.

§ 5: Verfahren mit diskreten Logarithmen

Wie wir gesehen haben, ist die Schlüsselverteilung bei RSA zwar deutlich einfacher als bei symmetrischen Verfahren, aber es gibt immer noch Probleme, dem Adressaten einer Nachricht sicher den korrekten Schlüssel zuzuordnen, was oft nur über eine Zentrale möglich ist.

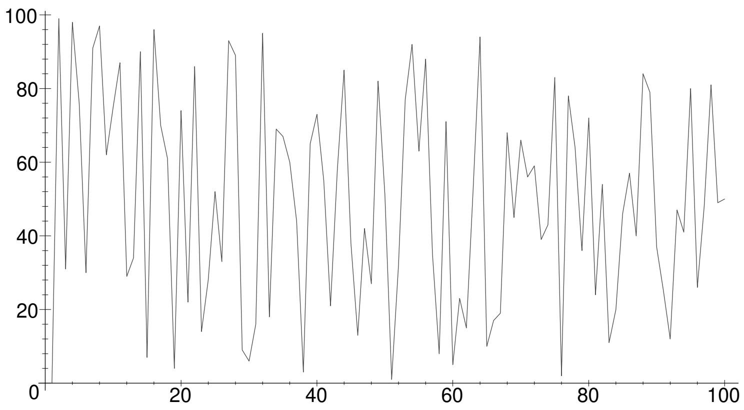
Für die bloße Übertragung einer Leihkartennummer an eine Bibliothek oder kleinere Einkäufe im Internet lohnt es sich kaum, sich bei einer Zentrale registrieren zu lassen (und dafür zu bezahlen); dafür braucht man alternative Verfahren, die ohne den Austausch geheimer Schlüsselinformation auskommen. Typischerweise arbeiten diese mit diskreten Logarithmen.

a) Diskrete Logarithmen

In $\mathbb{R}$ ist der Logarithmus zur Basis a die Umkehrfunktion der Funktion $x \mapsto a^x$; genauso definieren wir ihn auch für endliche Körper:

$$y = a^x \Longrightarrow x = \log_a y\,.$$

Trotz dieser formalen Übereinstimmung gibt es es allerdings große Unterschiede zwischen reellen Logarithmen und ihren Analoga in endlichen Körpern: Während reelle Logarithmen sanft ansteigende stetige Funktionen sind, die man leicht mit beliebig guter Genauigkeit annähern kann, sieht der diskrete Logarithmus typischerweise so aus wie in der Abbildung zu sehen ist. Auch ist im Reellen der Logarithmus zur Basis $a > 1$ für jede positive Zahl definiert; in endlichen Körpern ist es viel schwerer zu entscheiden, ob ein bestimmter Logarithmus existiert: Modulo sieben etwa sind $2, 4$ und 1 die einzigen Zweierpotenzen, so daß $3, 5$ und 6 keine Zweierlogarithmen haben. Wie wir im letzten Vortrag gesehen haben, gibt es aber in jedem endlichen Körper primitive Wurzeln, d.h. Elemente a, für die a^x jeden Wert außer der Null annimmt. In $\mathbb{F}_7$ sind dies die beiden Zahlen drei und fünf.



Die Funktion $\log_{51} x$ in $\mathbb{F}_{101}$

Die Berechnung der Potenzfunktion durch sukzessives Quadrieren und

Multiplizieren ist auch in endlichen Körpern einfach, für ihre Umkehrfunktion, den diskreten Logarithmus gibt es aber derzeit nur deutlich schlechtere Verfahren. Die derzeit besten Verfahren zur Berechnung von diskreten Logarithmen in Körpern mit N Elementen erfordern etwa denselben Aufwand wie die Faktorisierung eines RSA-Moduls der Größenordnung N; diese Diskrepanz zwischen Potenzfunktion und Logarithmen kann kryptologisch ausgenutzt werden.

Als Körper verwendet man entweder Körper von Zweipotenzordnung, da man in diesen gut rechnen kann, oder Körper von Primzahlordnung. Da es für viele interessante Körper von Zweipotenzordnung bereits Chips gibt, die dort diskrete Logarithmen berechnen, und Angreifer eher bereit sind, das Geld dafür auszugeben, als einfache Benutzer, dürften Körper von Primzahlordnung bei ungefähr gleicher Elementanzahl wohl etwas sicherer sein: Es gibt einfach viel mehr Primzahlen als Zweierpotenzen, und jeder Fall erfordert einen neue Hardwareentwurf. Falls man die Primzahlen hinreichend häufig wechselt, dürfte sich dieser Aufwand für kaum einen Gegner lohnen.

Da Körper von Primzahlordnung auch einfacher sind als solche von Primzahlpotenzordnung, wollen wir uns hier zur ersten Vorstellung von Kryptoverfahren auf der Basis diskreter Logarithmen auf Körper $\mathbb{F}_p$ beschränken; im weiteren Verlauf der Sommerschule werden dann nicht nur beliebige Körper $\mathbb{F}_q$ betrachtet werden, sondern auch allgemeinere Gruppen als nur die multiplikative Gruppe eines solchen Körpers. Am Prinzip der Kryptoverfahren ändert sich dadurch nichts.

b) Diffie-Hellman und ElGamal-Verfahren

Beim Diffie-Hellman-Verfahren, dem ältesten auf der Grundlage diskreter Logarithmen, geht es darum, wie zwei Teilnehmer, die weder über gemeinsame Schlüsselinformation noch über eine sichere Leitung verfügen, einen Schlüssel vereinbaren können.

Nach DIFFIE-HELLMAN einigen sie sich zunächst (über die unsichere Leitung) auf eine Primzahl p und eine natürliche Zahl a derart, daß die Potenzfunktion $x \mapsto a^x$ möglichst viele Werte annimmt.

Als nächstes wählt Teilnehmer A eine Zufallszahl $x < p$ und B entsprechend $y < p$; A schickt $u = a^x \bmod p$ an B und erhält dafür $v = a^y \bmod p$.

Sodann berechnet A die Zahl

$$v^x \bmod p = \left(a^y\right)^x \bmod p = a^{xy} \bmod p$$

und B entsprechend

$$u^y \bmod p = \left(a^x\right)^y \bmod p = a^{xy} \bmod p\,;$$

beide haben also auf verschiedene Weise dieselbe Zahl berechnet, die sie nun als Schlüssel in einem klassischen Kryptosystem verwenden können: Beispielsweise könnten die letzten 128 Bit der Zahl als AES-Schlüssel dienen.

Ein Gegner, der den Datenaustausch abgehört hat, kennt die Zahlen p, a, u und v; um $a^{xy} \bmod p$ zu finden, muß er den diskreten Logarithmus von u oder v berechnen.

Das Verfahren von ELGAMAL ist eine einfache Variante, die daraus ein Verschlüsselungsverfahren macht: Nachrichten werden durch ganze Zahlen zwischen 0 und $p-1$ dargestellt und vor der Übertragung mit a^{xy} modulo p multipliziert; der Chiffretext zum Klartext m ist also

$$a^{xy}m \bmod p\,.$$

Da Inversenbildung im Körper mit p Elementen mittels des EUKLIDischen Algorithmus leicht möglich ist, kann der Empfänger durch Multiplikation mit dem inversen Element zu $a^{xy} \bmod p$ den Klartext rekonstruieren.

In dieser Form ist das Verfahren nur für die Übertragung eines Blocks geeignet, denn da die Verschlüsselungsfunktion einfach eine Multiplikation ist, gilt für die Chiffretexte c_1, c_2 zu zwei Klartexten m_1, m_2

$$c_1^{-1}c_2 = m_1^{-1}m_2\,;$$

ein Lauscher, der einen Klartextblock kennt oder errät (Briefkopf, Schlußformel, ...), kann mithin auch alle anderen Blöcke entschlüsseln.

Falls also A an B sendet, muß a für jeden Block m_i eine neue Zufallszahl x_i erzeugen und $u_i = a^{x_i} \bmod p$ berechnen; tatsächlich übertragen werden also die beiden Blöcke

$$u_i \quad \text{und} \quad a^{x_i y}m_i\,.$$

Der Chiffretext ist damit doppelt so lang wie der Klartext, so daß das Verfahren für lange Texte nicht attraktiv ist.

c) Das Verfahren von Massey-Omura

Auch bei diesem Verfahren geht es um Nachrichtenaustausch zwischen zwei Partnern A und B, die über keinerlei Schlüsselinformation verfügen. Die beiden einigen sich auf eine Primzahl p, und jeder erzeugt sich einen (geheimzuhaltenden) Exponenten e_A *bzw.* e_B, der prim ist zu $p-1$. Dazu berechnet er nach dem erweiterten EUKLIDischen Algorithmus ein (ebenfalls geheimzuhaltendes) Inverses modulo $p-1$; diese Inversen seien d_A und d_B. Nach dem kleinen Satz von FERMAT ist somit für jedes $m \in \mathbb{Z}$

$$m^{e_A d_A} \equiv m^{e_B d_B} \equiv m \bmod p\,.$$

Will nun A eine Nachricht m verschlüsselt an B schicken, so schickt er $m^{e_A} \bmod p$. Damit kann natürlich weder B noch ein etwaiger Lauscher etwas anfangen: Da niemand außer A den Verschlüsselungsexponenten e_A und den Entschlüsselungsexponenten d_A kennt, ist das einfach *irgendeine* Potenz zu *irgendeiner* Basis. Selbst ein BAYESscher Gegner, der alle Kombinationen (M, e) mit $M^e \equiv m^{e_A} \bmod p$ durchprobieren kann, wird dort für große p eine Fülle von potentiellen Klartexten finden, die alle ungefähr gleich wahrscheinlich sind.

Der Empfänger schickt die Nachricht daher gleich wieder zurück, potenziert sie aber vorher mit seinem Verschlüsselungsexponenten e_B. Was A erhält, ist also $m^{e_A e_B}$, eine Nachricht die niemand entschlüsseln kann.

A potenziert diese Nachricht mit seinem Entschlüsselungsexponenten d_A und erhält

$$\left(m^{e_A e_B}\right)^{d_A} = m^{e_A e_B d_A} = m^{e_A d_A e_B} = \left(m^{e_A d_A}\right)^{e_B} \equiv m^{e_B} \bmod p\,.$$

Diese Nachricht schickt er an B, der nun mit seinem Entschlüsselungsexponenten d_B leicht den Klartext ermitteln kann.

Auch die Sicherheit dieses Verfahrens hängt an diskreten Logarithmen: Ein etwaiger Lauscher kennt die Zahlen

$$m^{e_A} \bmod p, \quad m^{e_A e_B} = \left(m^{e_A}\right)^{e_B} \quad \text{und} \quad m^{e_B} = \left(m^{e_A e_B}\right)^{d_A} ;$$

falls er in der Lage ist, diskrete Logarithmen modulo p zu berechnen, kann er also e_B bestimmen als den diskreten Logarithmus von $m^{e_A e_B}$ zur Basis m^{e_A} und d_A als diskreten Logarithmus von $\left(m^{e_A e_B}\right)^{d_A}$ zur Basis $m^{e_A e_B}$. Damit kann auch er m berechnen, indem er beispielsweise m^{e_A} modulo p mit d_A potenziert. Die Primzahl p muß also auch bei diesem Verfahren so groß sein, daß die Berechnung diskreter Logarithmen modulo p zumindest praktisch undurchführbar ist.

d) Strategien zur Berechnung diskreter Logarithmen

Genau wie es zahlreiche Ansätze gibt, ganze Zahlen auf mehr oder weniger effiziente Weise zu faktorisieren, gibt es auch die verschiedensten Methoden, den diskreten Logarithmus x einer Zahl u zur Basis a modulo p zu berechnen.

Am einfachsten und langwierigsten ist Probieren: Man berechnet einfach systematisch alle Potenzen von a modulo p, bis man das Ergebnis u erhält; im schlimmsten Fall sind dies p Multiplikationen, im Durchschnitt $p/2$.

Besser ist das *baby step – giant step* Verfahren, mit dem sich der Aufwand immerhin auf etwa $\sqrt{p}$ reduzieren läßt. Dazu wählt man eine natürliche Zahl m die ungefähr gleich $\sqrt{p} \cdot \log_2 p$ ist, und berechnet zunächst die Potenzen $a^i \bmod p$ für $i \le m$; das sind m sogenannte *baby steps.*

Bei den dann folgenden *giant steps* berechnet man, um den diskreten Logarithmus von x zu erhalten, die Elemente $xa^{-mj} \bmod p$ für $j = 1, 2, \ldots$ und vergleicht sie mit den Potenzen aus dem ersten Teil. Ein solcher Vergleich kann etwa über eine binäre Suche oder eine *hash*-Tabelle implementiert werden und hat einen Aufwand von $\log_2 p$.

Sobald man einen Wert xa^{-mj} gefunden hat, der mit einer der in den *baby steps* berechneten Potenzen a^i übereinstimmt, hat man die Gleichung

$$xa^{-mj} = a^i \bmod p \quad \text{oder} \quad x = a^{mj+i} \bmod p\,,$$

der diskrete Logarithmus von x zur Basis a ist also $mj + i$. Die notwendige Anzahl von *giant steps* liegt im schlimmsten Fall bei $p/m \approx \sqrt{p}$; im Durchschnitt ist sie halb so groß.

Diese Reduktion des Rechenaufwands auf die Quadratwurzel des naiven Werts erinnert an die allererste Reduktion bei den Faktorisierungsalgorithmen für eine natürliche Zahl N: Auch hier kommt man auf einen Aufwand von „nur" $\sqrt{N}$, wenn man sich auf das Durchprobieren potentieller Teiler bis $\sqrt{N}$ beschränkt.

Allgemein zeigt die bisherige Erfahrung, daß es zu jedem Faktorisierungsalgorithmus einen Algorithmus zur Berechnung diskreter Logarithmen gibt, der die gleiche Komplexität hat – auch wenn bislang niemand beweisen konnte, daß es einen solchen Zusammenhang gibt und auch kein Grund erkennbar ist, warum es ihn geben sollte.

Die derzeit besten Faktorisierungsalgorithmen beruhen auf dem quadratischen und dem Zahlkörpersieb; für beide wurden bald nach ihrer Einführung ähnliche Siebalgorithmen gefunden, die zur Berechnung diskreter Logarithmen führen. Wir beschränken uns hier, wie auch schon bei der Faktorisierung, auf das quadratische Sieb, dessen Variante für diskrete Logarithmen als *Indexkalül* bezeichnet wird, und auch hier beschränken wir uns auf eine einfache Variante speziell für Primkörper $\mathbb{F}_p$.

Wie beim quadratischen Sieb wird eine Schranke B festgelegt und damit eine Faktorbasis $\mathcal{B}$ definiert; diese besteht hier aus *allen* Primzahlen $q \le B$. Der Algorithmus besteht aus zwei Teilen:

Im ersten Teil berechnet man die diskreten Logarithmen aller Primzahlen q aus der Faktorbasis zur gegebenen Basis a modulo p. Dies mag auf den ersten Blick unsinnig erscheinen, denn schließlich suchen wir den diskreten Logarithmus *einer* Zahl und beginnen dazu mit der Berechnung der diskreten Logarithmen vieler Zahlen.

Die Logarithmen der Primzahlen lassen sich aber simultan wie folgt berechnen: Man berechne viele Potenzen $a^y \bmod p$ und suche diejenigen, die eine Primfaktorzerlegung mit lauter Faktoren aus $\mathcal{B}$ haben. Ist

$$a^y \bmod p = q_1^{e_1} \cdots q_r^{e_r} ,$$

so ist

$$y \equiv e_1 \log_a q_1 + \cdots + e_r \log_a q_r \bmod m ,$$

wobei m die kleinste natürliche Zahl ist, für die $a^m \equiv 1 \bmod p$. Für eine primitive Wurzel a modulo p ist $m = p-1$, ansonsten kann m auch ein echter Teiler davon sein.

Mit genügend vielen Gleichungen dieser Form hat man ein lineares Gleichungssystem für die Logarithmen der $q \in \mathcal{B}$, allerdings leider nicht über einem Körper, sondern modulo der im allgemeinen zusammengesetzten Zahl m. Falls m Produkt von Primzahlen ist, löst man das Gleichungssystem modulo jeder dieser Primzahlen und setzt die Lösungen nach dem chinesischen Restesatz zusammen; wenn auch echte Primzahlpotenzen P^s in m stecken, schreibt man die e_i und die linken Seiten y im Zahlensystem zur Basis P und erhält dann für jede Ziffer ein lineares Gleichungssystem über dem Körper mit P Elementen, aus denen man die Lösung modulo P^s zusammensetzen kann.

Dieser erste Schritt ist offensichtlich völlig unabhängig vom Element x, dessen Logarithmus wir suchen; er kann für eine gegebene Basis a und Primzahl p ein für allemal im voraus durchgeführt werden.

Im zweiten Schritt betrachtet man für zufällig gewählte Exponenten y die Elemente $a^y x \bmod p$, bis man eines findet, das nur durch Primzahlen aus der Faktorbasis teilbar ist. Falls etwa

$$a^y x \bmod p = q_1^{f_1} \cdots q_s^{f_s}$$

ist, muß

$$\log_a x = f_1 \log_a q_1 + \cdots + f_s \log_a q_s - y \bmod m$$

sein.

e) Allgemeinere DL-Systeme

Wie wir in den vorangegangen Paragraphen gesehen haben, kommen Kryptoverfahren, die auf diskreten Logarithmen beruhen, ohne vorher bekannte Schlüssel aus, so daß sie gerade für sichere Kommunikation im Internet sehr attraktiv sind. Problematisch sind allerdings die großen Zahlen, mit denen man hier arbeiten muß: Im Augenblick 1024 Bit, bald sogar 2048. Für einen PC ist das zwar nicht sonderlich problematisch, für den WAP-Browser eines Mobiltelephons aber ist es ein fast unüberwindliches Hindernis. Auch beim elektronischen Bargeld, egal ob auf der Basis von RSA oder in der hier nicht behandelten Variante mit diskreten Logarithmen, sind die langen Schlüssel ein Problem, da die Bank alle Seriennummern bereits eingelöster „Banknoten" speichern muß.

Zum Glück lassen sich diskrete Logarithmen nicht nur für die Multiplikation ganzer Zahlen modulo einer Primzahl p definieren, sondern in jeder Situation mit einer Rechenoperation hat, die sich iterieren läßt.

Beispielsweise könnten wir eine $n \times n$-Matrix A über einem endlichen Körper hernehmen und die Umkehrfunktion von $x \mapsto A^x$ betrachten; das diskrete Logarithmenproblem bestünde also darin, zu einer Matrix B eine ganze Zahl $x \geq 0$ zu finden, so daß $B = A^x$ ist. Das wäre allerdings nicht sonderlich sinnvoll, denn der Aufwand bei der Berechnung von A^x steigt mit wachsendem n stark an, wohingegen das diskrete Logarithmenproblem über die JORDAN-Zerlegung von A und B leicht auf das im Grundkörper zurückgeführt werden kann – wenn nicht gar der nilpotente Anteil noch zusätzliche Hinweise liefert.

Für kryptographische Anwendungen ideal wäre eine Operation, die nicht sehr viel aufwendiger ist als die Multiplikation modulo p, für die aber die Berechnung des darauf beruhenden diskreten Logarithmus deutlich komplexer ist. Solche Operationen liefert die algebraische Geometrie; der einfachste Fall (und im Augenblick der einzige, der in der Praxis eine nennenswerte Rolle spielt) ist der der *elliptischen Kurven,* allerdings gibt es auch bereits Systeme auf der Grundlage der etwas komplizierteren sogenannten *hyperelliptischen* Kurven. Ihr großer Vorteil liegt darin, daß man dort die Anwendung des Indexkalküls zur Berechnung diskreter Logarithmen so erschweren kann, daß sie langsamer wird als etwa die

Berechnung via *baby step – giant step.* Das wiederum bedeutet, daß man mit deutlich kleineren Körpern auskommt und nur noch Zahlen mit etwa 160–180 Bit betrachten muß.

Mehr dazu wird in den fünf Vorträgen von Herr FREY zu hören sein.

§6: Literatur

RSA und Verfahren auf der Grundlage diskreter Logarithmen sind die gebräuchlichsten Algorithmen der *public key* Kryptographie; es gibt daher kaum ein nach etwa 1980 erschienenes allgemeines Lehrbuch der Kryptologie, das nichts darüber enthält. Beispielhaft zitiert seien

JOHANNES BUCHMANN: *Einführung in die Kryptographie,* Springer, 1999

und

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

Ein Lehrbuch, das sich vor allem mit asymmetrischer Kryptographie beschäftigt, ist

NEAL KOBLITZ: *A Course in Number Theory and Cryptography,* Graduate Texts in Mathematics **114**, Springer ^{2}1994

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

HANS RIESEL: *Prime Numbers and Computer Methods for Factorization,* Birkhäuser, 1985

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

Einen Überblick über andere Angriffe auf RSA als die Faktorisierung des Moduls bietet

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` .

Einen Eindruck über die Diskussionen, die zu den jährlichen Empfehlungen über „Geeignete Algorithmen“ führen gibt der Vortrag

SCHABHÜSER: *How to find the "‘socially accepted"’ minimal Keylength for Digital Signature Algorithms,* Vortrag ECC 2002, Essen, `www.exp-math.uni-essen.de/w̃eng/schabhueser_ECC_2002.`$\left\{\begin{matrix}\texttt{PDF}\\\texttt{ppt}\end{matrix}\right.$

WOLFGANG K. SEILER
Lehrstuhl VI für Mathematik
Universität Mannheim
Seminargebäude A5, C201
68131 Mannheim

Tel. 181-2515
seiler@math.uni-mannheim.de
http://hilbert.math.uni-mannheim.de/~seiler/