Public Key Kryptosysteme in Theorie und Programmierung

Johann Wiesenbauer, TU Wien

Zusammenfassung

Obwohl es Chiffrierverfahren mit öffentlichem Schlüssel, wie z.B. das RSA-Verfahren, erst seit wenig mehr als zwei Jahrzehnten gibt, sind sie heute aus dem Alltagsleben nicht mehr wegzudenken wie ihre weite Verbreitung u.a. im Bankenverkehr und bei der Verschlüsselung von Nachrichten im Internet (z.B. mittels PGP) zeigt. Diese Tatsache sollte sich auch im Schulunterricht niederschlagen. Dies umso mehr, als ihre Behandlung einen in vieler Hinsicht idealen genetischen Zugang zur heute so wichtig gewordenen sog. algorithmischen Zahlentheorie bietet. Es wird versucht, dafür einige Anregungen zu geben, wobei insbesondere auf die konkrete Realisierung der vorgestellten Algorithmen mit Hilfe von DERIVE - in vielen Fällen mit realistisch großen Zahlen - großer Wert gelegt wird.

1. Einleitung

> "...both Gauss and lesser mathematicians may be justified in rejoicing that there is one science [number theory] at any rate, and that their own, whose very remoteness from ordinary human activities should keep it gentle and clean."
>
> - G.H.Hardy, A Mathematician's Apology, 1940

Wie obiges Zitat des berühmten Mathematikers Hardy belegt, galt die Zahlentheorie bis vor gar nicht langer Zeit als eines der "reinsten", d.h. den Anwendungen am fernstehendsten Teilgebiete der Mathematik. Dieser Auffassung mochten auch noch jene Verantwortlichen gewesen sein, welche anfangs der 80-er Jahre, also am Ende der sog. "new math"-Ära an unseren Schulen, die als "Mengenlehre" diskredidierten und ohnehin spärlichen Ansätze zur Algebra und Zahlentheorie wieder aus den Lehrplänen entfernten. Es muß daher als eine besondere Ironie des Schicksals angesehen werden, daß etwa um diese Zeit die wenig Jahre zuvor erfundenen Public Key Kryptosysteme einen wahren Siegeszug um die Welt antraten, welche zu einem gutem Teil auf einfachen und für die Schule durchaus zugänglichen Sätzen der Algebra und Zahlentheorie beruhten und somit auch jene "Anwendungen" des abstrakten Stoffes geboten hätten, welche viele bis dahin vermißten. Insgesamt kann gesagt werden, daß parallel zu der in letzten Jahrzehnten ungeheuer angestiegenen Computerisierung die gesamte Diskrete Mathematik und mit ihr vor allem auch die Zahlentheorie stark an Bedeutung für die Anwendungen gewonnen hat, sodaß heute von deren "remoteness from ordinary human activities" wie vielleicht noch zu Hardy's Zeiten keine Rede mehr sein kann.

Was Public Key Kryptosysteme für den Schulunterricht so interessant macht - wenn leider auch derzeit nur im Rahmen von Projektstudien bzw. im Vertiefenden Wahlfach Mathematik - ist die Fülle von wichtigen Algorithmen der Mathematik, die dabei zum Einsatz kommen. Als besonders bedeutsam hat sich dabei die Untersuchung gewisser Teilbarkeitseigenschaften von natürlichen Zahlen herausgestellt, welche bereits seit altersher betrieben und insbesondere auch von Gauß in seinen "Disquisitiones arithmeticae" (1801) angesprochen wird:

"Daß die Aufgabe, die Primzahlen von den zusammengesetzten zu unterscheiden und letzere in ihre Primfaktoren zu zerlegen, zu den wichtigsten und nützlichsten der gesamten Arithmetik gehört und die Bemühungen und den Scharfsinn sowie der alten wie auch der neuen Geometer in Anspruch genommen hat, ist so bekannt, daß es überflüssig wäre, hierüber viele Worte zu verlieren. Trotzdem muß man gestehen, daß alle bisher angewendeten Methoden entweder auf spezielle Fälle beschränkt oder so mühsam und weitläufig sind, daß sie auf größere Zahlen meistenteils kaum angewendet werden können. Außerdem aber dürfte es die Würde der Wissenschaft erheischen, alle Hilfsmittel zur Lösung jenes berühmten Problems fleißig zu vervollkommnen."

Gauß wäre sicher nicht schlecht erstaunt, könnte er sehen, wie intensiv heute auf diesem Gebiet geforscht wird und wieviele Hilfsmittel zur Lösung dieser beiden fundamentalen Probleme, d.h. der Feststellung der Primalität einer vorgelegten großen natürlichen Zahl bzw. der Bestimmung von nichttrivialen Faktoren im Fall ihrer Zusammengesetztheit, inzwischen zusammengetragen wurden. Obwohl die meisten Mathematiker nicht daran glauben, ist es doch nicht von vornherein ausgeschlossen, daß die Forschungen eines Tages zu einem sog. Polynomialzeitalgorithmus für das Faktorisierungsproblem führen, was dann einen Kollaps eines wesentlichen Teils der modernen Kryptographie zur Folge hätte. Im letzten Kapitel wird versucht, anhand von einigen leichter zugänglichen Beispielen einen kleinen Einblick in die Methoden dieses hochaktuellen Forschungsgebiets zu gewähren.

Ein Wort noch zu DERIVE. Es wurde nicht nur deshalb gewählt, weil es an Österreichs Mittelschulen aufgrund einer Generallizenz allgemein zugänglich ist, sondern auch um zu beweisen, daß man es trotz seiner im Vergleich zu manchen anderen Computeralgebrasystemen eingeschränkten Programmiermöglichkeiten - wobei sich übrigens in diesem Bereich mit der im Herbst 1999 herauskommenden Version 5 von DERIVE for Windows (DfW) starke Verbesserungen abzeichnen - durchaus auch zur Lösung von relativ komplexen Aufgabenstellungen algebraisch-zahlentheoretischer Natur heranziehen kann. Was im übrigen die Ausführungsgeschwindigkeit der Programme angeht, so braucht hier DERIVE einen Vergleich in der Regel nicht zu scheuen. Alle Programme wurden dabei nur mit der zur Zeit aktuellen Version DfW 4.11 getestet, für ältere Versionen 4.xx besteht allerdings die Möglichkeit eines "free update" über Internet (www.derive.com/dfwfixsm.htm). Soweit Rechenzeiten angegeben wurden, beziehen sie sich stets auf einen Pentium 166 PC mit 64 MB.

2. Die Grundidee von Public Key Kryptosystemen

Public Key Kryptosysteme - auch Chiffrierverfahren mit öffentlichem Schlüssel oder asymmetrische Chiffrierverfahren genannt - wurden von W.Diffie und M.Hellman 1976 eingeführt (s. [2]). Ihre Grundidee kann folgendermaßen beschrieben werden:

Jedem Benutzer X sind zwei Funktionen $E_X: M_X \to C_X$ und $D_X: C_X \to M_X$ zugeordnet. E_X wird dabei zum Chiffrieren von Nachrichten $m \in M_X$ an X verwendet und ist in einer öffentlich zugänglichen Schlüsselkartei abgelegt, wohingegen die Funktion D_X, welche X seinerseits zum Dechiffrieren eines Chiffrats $c \in C_X$ verwendet, natürlich von diesem geheimgehalten wird.

E_X und D_X haben dabei folgende Eigenschaften:

1) $D_X(E_X(m))=m \; \forall m \in M_X$. Insbesondere ist also E_X injektiv.

2) Die Funktionswerte von E_X und D_X sind leicht berechenbar.

3) E_X ist eine sog. Falltürfunktion (engl. trapdoor function), d.h. es ist "praktisch unmöglich" D_X aus E_X (ohne Zusatzinformationen, über die aber nur X verfügt) zu gewinnen.

Manchmal gilt zusätzlich

4) $E_X(D_X(c))=c \; \forall c \in C_X$, d.h. E_X und D_X sind dann sogar zueinander invers und damit auch bijektiv.

Gegenüber den bis dahin verwendeten Verschlüsselungssystemen (welche auch "symmetrisch" genannt werden, da die genaue Angabe der Verschlüsselung auch gleichzeitig der Schlüssel zur Entschlüsselung ist) haben diese asymmetrischen Chiffrierverfahren folgende Vor- und Nachteile.

Vorteile:

- Kein Schlüsselaustausch mehr notwendig, da die erforderlichen Schlüssel für den Datenaustausch nicht mehr geheimgehalten werden müssen und öffentlich zugänglich sind.
- Man braucht sehr wenige Schlüssel (bei n Teilnehmern waren es früher n(n-1)/2, jetzt nur mehr mehr 2n)
- Public Key Kryptosysteme, für die auch 4) gilt, bieten i.allg. darüber hinaus die Möglichkeit einer sog. elektronischen Unterschrift (digital signature). Z.B. könnte A sich gegenüber B dadurch ausweisen, indem er ihm seine Signatur s_A in der Form $E_B(D_A(s_A))$ mitübermittelt, welche dieser umgekehrt durch Anwendung von $E_A \circ D_B$ entschlüsseln kann. Allerdings muß dazu sichergestellt sein, daß $M_A \subseteq M_B$ gilt, da sonst $D_A(s_A)$ i.allg. nicht im Definitionsbereich von E_B liegt.

Nachteile

- Weitaus geringere Geschwindigkeit (softwaremäßig um einen Faktor 100-1000, bei Verwendung spezieller Hardware sogar bis zu einem Faktor von ca. 10000) gegenüber gängigen symmetrischen Chiffrierverfahren. Aus diesem Grund wird daher damit oft nur der Schlüssel für ein schnelleres symmetrischen Chiffrierverfahren übermittelt. Soche hybride Verschlüsselungssysteme sind in der Praxis sehr beliebt, da sie Sicherheit und Schnelligkeit in sich vereinigen (z.B. geschieht die eigentliche Verschlüsselung bei PGP mit Hilfe von CAST,IDEA oder TripleDES während RSA nur für das Schlüsselmanagement verwendet wird).
- Alle der bisher bekannten Public Key Kryptosysteme basieren auf der angenommenen, aber in keinem Fall streng bewiesenen, Schwierigkeit der Lösung von gewissen mathematischen Problemen.
- Gegenüber symmetrischen Chiffrierverfahren sind die Längen der privaten Schlüssel (in bits) in der Regel um ein Vielfaches größer. Dasselbe gilt für elektronische Unterschriften.

3. Alt aber trotzdem nicht gut: Knapsack-Verfahren

Knapsack-Verfahren waren historisch gesehen die erste Realisierungen von Public-Key Kryptosystemen. Die nachfolgende Originalversion von R.C.Merkle und M.E.Hellman (1978) sowie fast alle Nachfolgerversionen gelten aber heute nicht mehr als sicher, sodaß wir heir nur kurz darauf eingehen wollen.

Ausgangspunkt ist ein spezielles "Knapsackproblem", welches zur Klasse der NP-vollständigen Probleme gehört, die abgesehen von gewissen Spezialfällen nur sehr schwer zu "knacken" sind.

Wir nehmen dazu an, daß b ein festgewählter Vektor von n natürlichen Zahlen ist, sodaß die Abbildung, welche jeder Nachricht $x=(x_1,x_2,\ldots,x_n)\in\{0,1\}^n$ in binär codierter Form das innere Produkt s:=xb zuordnet, injektiv ist. Ist dann n genügend groß, so ist es dann i.allg. "praktisch unmöglich" aus s die Nachricht x rückzurechnen. Falls allerdings b die Eigenschaft

$$b_i > b_1 + b_2 + \ldots + b_{i-1} \quad (i=2,3,\ldots,n)$$

hat (man nennt b dann "superincreasing"), ist dies ganz einfach:

x_n ist dann nämlich ganz einfach der Quotient bei der Division $s:b_n$, x_{n-1} der Quotient bei der Division $(s-x_nb_n):b_{n-1}$, x_{n-2} der Quotient bei der Division $(s-x_nb_n-x_{n-1}b_{n-1}):b_{n-2}$, usw.

Jeder Benutzer X des Systems wählt nun einen genügend langen Vektor b, der superincreasing ist. Den anderen Benutzern wird aber natürlich nicht dieser Vektor zur Verfügung gestellt - sonst wäre für sie das Entschlüsseln gleichermaßen einfach - , sondern eine "maskierte" Version a davon, welche in dieser Variante aus b auf folgende Weise erhalten wird:

Man wählt dazu natürliche Zahlen m und w mit

$$m > b_1 + b_2 + \ldots + b_n$$

und (w,m)=1, sowie eine Permutation π der Menge $\{1,2,\ldots,n\}$ und setzt

$$a_i := wb_{\pi(i)} \bmod m,\ i=1,2,\ldots,n.$$

Der Vektor $a=(a_1,a_2,\ldots,a_n)$ ist dann der öffentliche Schlüssel von X, während m,w und π natürlich von diesem geheimgehalten werden.

Ist nun $x=(x_1,x_2,\ldots,x_n)\in\{0,1\}^n$ die Nachricht, welche zu s:=xa verschlüsselt wird, so kann X mit Hilfe seines privaten Schlüssels, ein Inverses w' zu w mod m bestimmen und mit $s':=w's \bmod m$ und $y=(y_1,y_2,\ldots,y_n)$ das "leichte" Knapsackproblem

$$yb=s'$$

lösen, woraus sich dann wegen

$$x_i = y_{\pi(i)},\ i=1,2,\ldots,n,$$

sofort auch x selbst ergibt.

Nachfolgend einige DERIVE-Routinen, mit deren Hilfe alle diese Rechnungen vollautomatisch durchgeführt werden. Als erstes führen wie eine Routine ein, die eine

Folge von n natürlichen Zahlen erzeugt, welche superincreasing ist. (Achtung: "Doppelte" Zuweisungen mit Hilfe von := = müssen immer sofort simplifiziert werden, damit sie wirksam werden. Im Gegensatz zur "einfachen" Zuweisung mittels := werden dabei die rechtsstehenden Ausdrücke "ein für allemal" berechnet und erst danach wird das numerische Ergebnis der linksstehenden Variablen zugewiesen.)

SUPERINCREASING(n) := ITERATE(APPEND(s_, [SUM(s_) + RANDOM(10) + 1]), s_, [], n)

b := = SUPERINCREASING(100)=
[8, 11, 29, 50, 108, 208, 417, 832, 1667, 3335, 6671, 13337, 26680, 53357, ...,
2064145631437549916970333744801, 4128291262875099833940667489603]

Damit könnten nun jeder Binärvektor x der gleichen Länge wie b durch Bildung des inneren Produkts xb verschlüsselt werden, z.B.

x := = VECTOR(RANDOM(2), k_, 1, 100)= [1, 1, 1, 1, 1, 0, 0, ..., 0, 1, 0, 1, 0, 0, 1]

(s :== b·x) = 479753886900606330963548355 1456

Wie oben beschrieben, könnte aber dieses x sofort von jedem, der b kennt, leicht entschlüsselt werden:

DECRYPT0(s, b) := REVERSE_VECTOR((ITERATE([APPEND(a_, [FLOOR(s_, bSUBk_)]), MOD(s_, bSUBk_), k_ - 1], [a_, s_, k_], [[], s, DIMENSION(b)], DIMENSION(b)))SUB1)

DECRYPT0(s, b) = x = true (0.1s !)

Die Zahlen m, w und die Permutation r, sowie der "maskierte" Vektor a werden dann aus b etwa wie im folgenden Beispiel berechnet:

APPROX(SUM(b)) = 8.25658·10^30

(m :== RANDOM(10^32)) = 92519416676036416422540341059254

m > SUM(b) = true

(w :== ITERATE(IF(GCD(w_, m) = 1, w_, w_ + 1), w_, RANDOM(m))) =
21864428968323363187924298358313

RANDOM_PERM(n) := APPEND(ITERATE(ITERATE([APPEND(p_, [q_]), SELECT(s_ /= q_, s_, r_)], q_, r_SUB(RANDOM(DIMENSION(r_)) + 1), 1), [p_, r_], [[], [1, ..., n]], n - 1))

(r := = RANDOM_PERM(100)) = [27, 42, 53, 65, 32, 33, ...,11, 94, 4, 40, 55, 95, 8]

a :== VECTOR(MOD(w·b_, m), b_, VECTOR(bSUBk_, k_, r))=
[4643900204281023920232836672776 6, 419240186127379859386469023447 32,...,
39583767400934363223433022251 49, 573992331419005535351093865026 32]

Dieser Vektor a wird nun anstelle von b öffentlich bekanntgegeben und die Verschlüsselung von x damit ergibt das tatsächliche Chiffrat, nämlich

(s := = a·x) = 188359803673554525133952054311 5933

Für die Entschlüsselung muß nur die modulare Multiplikation mit w mod m wieder rückgängig gemacht (durch Multiplikation von s mit dem Inversen von w mod m) und dann für b und das so erhaltene s das dann "leichte" Knapsackproblem gelöst werden.

Auf die so erhaltene Lösung muß schließlich noch die Permutation r angewandt werden. (Man beachte, daß DECRYPT die Routine INVERSE_MOD aus dem Utility-File NUMBER.MTH verwendet, welches daher vorher geladen sein muß!)

DECRYPT(s, b, w, m, r) := ITERATE(VECTOR(d_SUBk_, k_, r), d_, DECRYPT0(MOD(INVERSE_MOD(w, m)·s, m), b), 1)

DECRYPT(s, b, w, m, r) = x = true (0.1s)

Leider zeigte es sich, daß selbst nach der Maskierung mit Hilfe der modularen Multiplikation das entsprechende Knapsack-Problem noch gewisse spezielle Eigenschaften aufweist, welche eine Angriffsfläche bieten, sodaß heute, wie bereits eingangs erwähnt, alle Knapsack-Verfahren (mit Ausnahme einer Variante von Chor-Rivest, welche eine grundsätzlich andere Art der Maskierung verwendet) als gebrochen gelten. Bezüglich Einzelheiten sei auf die einschlägige Literatur (wie z.B. [3]) verwiesen.

4. RSA: Die Nummer 1 unter den Public Key Kryptosystemen

Ebenfalls eines der ersten Public Key Kryptosysteme und das heute mit Abstand verbreitetste ist RSA, welche nach seinen Erfindern R.L.Rivest, A.Shamir und L.Adleman benannt ist, die es 1978 publizierten (siehe [4]).

Jeder RSA-Teilnehmer A muß dazu folgende Schritte durchführen:

- Generierung von zwei großen Primzahlen p und q, $p \neq q$, von etwa derselben Größe und Berechnung von n:=pq.
- Wahl eines e mit $1<e<kgV(p-1,q-1)$ und $ggT(e,(p-1)(q-1))=1$.
- Berechnung eines d mit $1<d<kgV(p-1,q-1)$, sodaß gilt

$$de \equiv 1 \bmod kgV(p-1,q-1)$$

- A's öffentlicher Schlüssel ist das Paar (n,e), sein privater Schlüssel d.

Will B an A eine mit dem RSA-Verfahren verschlüsselte Nachricht schicken, so muß er dazu folgendes machen:

- A's öffentlichen Schlüssel (n,e) nachschlagen.
- Die Nachricht als Dezimalzahl m mit $0<m<n$ darstellen (bei langen Nachrichten könnte dazu eventuell eine Unterteilung in mehrere solche Dezimalblöcke notwendig sein).
- $c = m^e \bmod n$ berechnen und an A senden.

A kann dann seinerseits m mittels $m = c^d \bmod n$ zurückgewinnen.

Mathematisch gesehen basiert RSA auf der Gültigkeit von

$$a^{1+k\,kgV(p-1,q-1)} \equiv a \bmod pq \quad \forall a \in \mathbf{Z} \quad \forall k \in \mathbf{N} \qquad (*)$$

denn wegen $de \equiv 1 \bmod kgV(p-1,q-1)$ ist de in der Form $de = 1 + k\,kgV(p-1,q-1)$ für ein $k \in \mathbf{N}$ darstellbar, woraus dann nach (*) tatsächlich $(a^e)^d = (a^d)^e = a^{de} \equiv a \bmod pq$ folgt.

Die Kongruenz in (*) gilt aber offensichtlich genau dann, wenn sie mod p und mod q gilt, womit sie sich dann weiter zurückführen läßt auf die Gültigkeit von

$$a^{1+r(p-1)} \equiv a \bmod p \quad \forall a \in \mathbf{Z}\, \forall r \in \mathbf{N} \quad \text{bzw.} \quad a^{1+s(q-1)} \equiv a \bmod q \quad \forall a \in \mathbf{Z}\, s \in \mathbf{N} \quad (**)$$

indem man hierin

$$r = \frac{k(q-1)}{ggT(p-1,q-1)} \quad \text{bzw.} \quad s = \frac{k(p-1)}{ggT(p-1,q-1)}$$

setzt und dann die bekannte Formel

$$kgV(p-1,q-1) = \frac{(p-1)(q-1)}{ggT(p-1,q-1)}$$

verwendet.

Aus Symmetriegründen genügt es, etwa die erste der beiden Formeln in (**) zu beweisen. Diese ist aber offenbar richtig für r=0 und der Schritt von r auf r+1 wird geleistet durch

$$a^{1+(r+1)(p-1)} = a^p a^{r(p-1)} \equiv a a^{r(p-1)} = a^{1+r(p-1)} \equiv a \bmod p \quad \forall a \in \mathbf{Z}$$

wobei wir nur den sog. "Kleinen Fermatschen Satz" für die Primzahl p, nämlich

$$a^p \equiv a \bmod p \quad \forall a \in \mathbf{Z}$$

verwendet haben. Diese letzte Formel, auf die sich also alles zurückführen läßt, könnte man aber am einfachsten durch Induktion nach a beweisen - so wie dies übrigens auch Euler im allerersten Beweis dieses Satzes gemacht hat. Dabei ist der Induktionsanfang a=0 trivial und für den Schluß von a auf a+1 wird nur benötigt, daß alle Koeffizienten in der Entwicklung von $(a+1)^p$ nach dem binomischen Lehrsatz mit Ausnahme des ersten und letzten durch p teilbar sind:

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

Übrigens findet man in vielen Lehrbüchern - und übrigens auch in der Originalarbeit [4] seiner Erfinder - eine Darstellung von RSA, wo anstelle des kgV(p-1,q-1) überall das Produkt (p-1)(q-1) genommen wird. Natürlich funktioniert die Entschlüsselung auch mit dem so erhaltenen d - aus $de \equiv 1 \bmod (p-1)(q-1)$ folgt ja erst recht $de \equiv 1 \bmod kgV(p-1,q-1)$ -, doch ist das so erhaltene d im allgemeinen etwas größer als es sein müßte. (In der Regel zwar nur einige bits, was in der Praxis kaum Probleme macht, aber immerhin! Dies ist in jedem Falle ein schönes Beispiel dafür, daß auch die Verfasser von Mathematiklehrbüchern oft nur gedankenlos voneinander abschreiben!)

Speziell auf Fragen der Sicherheit von RSA, insbesondere auch was die Auswahl der Primzahlen p und q betrifft, werden wir im letzten Kapitel noch etwas näher darauf eingehen. Hier wollen wir nur noch anhand eines Beispiels mit Hilfe von DERIVE die einzelnen Schritte der Verschlüsselung und Entschlüsselung im Detail nachvollziehen.

Wie wählen dazu jene berühmte Preisaufgabe von den Erfindern des RSA selbst, damals am MIT tätig, welche in der bekannten Spalte "Mathematical Games" von Martin Gardner in der August Nummer der Zeitschrift "Scientific American" aus dem Jahr 1977 den Lesern dieser Zeitschrift gestellt wurde. Als Preis waren $ 100

ausgesetzt. Die Kolumne hatte den Untertitel "A new kind of cipher that would take millions of years to break". Als diese Aufgabe im April 1994 durch die Mithilfe von etwa 600 Freiwilligen über Internet schließlich gelöst wurde, hatte die tatsächliche Projektdauer nur etwa 8 Monate betragen.

Wie wir heute wissen, war die MIT-Gruppe bei der Zusammenstellung ihrer Preisaufgabe folgendermaßen vorgangen. Sie hatte zunächst die 64- bzw. 65-stelligen Primzahlen p und q wie folgt gewählt

p := 3490529510847650949147849619903898133417764638493387843990820577

q := 32769132993266709549961988190834461413177642967992942539798288533

und danach deren Produkt gebildet:

(n := p·q) =
114381625757888867669235779976146612010218296721242362562561842935706
9352457338978305971235639587050589890751475992900268795435 41

Anschließend wurde die Nachricht

THE MAGIC WORDS ARE SQUEAMISH OSSIFRAGE

dezimal codiert, indem A durch 01, B durch 02, C durch 03 usw. und der Zwischenraum jeweils durch 00 ersetzt wurde. Dies ergab schließlich die Dezimalzahl

m := 2008050013010709030023151804190001180500191721050113091908001519
19090618010705

welche sichtlich kleiner als n ist, sodaß eine Unterteilung in mehrere Blöcke also nicht erforderlich war. Als e wurde schließlich

e:=9007

gewählt, was wegen

GCD(e, (p - 1)(q - 1)) = 1

sicher eine geeignete Wahl ist. Als Chffrat ergab sich somit

(c := MOD(m^e, n)) =
968696137546220614771409222543558829057599911245743198746951209308162
98225145708356931476622883989628013391990551829945157815154

Der öffentliche Schlüssel (e,n), sowie natürlich c wurden dann in besagter Kolumne veröffentlicht.

Außerdem war in dem Artikel noch als "Signatur" diejenige Zahl angegeben, welche sich nach der Verschlüsselung des Textes

THE FIRST SOLVER WINS ONE HUNDRED DOLLARS

in derselben Weise wie oben, aber unter Benutzung des privaten Schlüssel der MIT-Gruppe ergab, nämlich

t := 1671786115038084424601527138916839824543690103235831121783503844 6
92906265544879223711449050957860865566249657797484000405702037 3

Tatsächlich ergibt die Überprüfung

(s := = MOD(t^e, n)) = 609181920001915122205180023091419001514050008211404180504000415121201181 9

d.h. wenn man s noch eine führende 0 voranstellt, um auf eine gerade Ziffernanzahl zu kommen, gerade obigen Text: 06→F, 09→I, 18→R, 19→S,20→T,usw.

Wir wollen nun auch die Entschlüsselung des Chiffrats im Jahre 1994 rekonstruieren, nachdem also die Faktorisierung von n in seine Primfaktoren gefunden worden war. Dazu nehmen wir wieder an, daß NUMBER.MTH als Utility-File vorher geladen wurde, da wir die Hilfsfunktion INVERSE_MOD daraus benötigen.

d := INVERSE_MOD(e, LCM(p - 1, q - 1))= 209123950501613736909419363468101957730461840930060908793048423220456085696971214722578758536822031722587178886785573767357802 71

(mm := MOD(c^d, n)) = 200805001301070903002315180419000118050019172105011309190800151919090618010705

mm = m = true

Vorsichtigerweise haben wir das Ergebnis der Dechiffrierung mm genannt, doch war diese Vorsicht unbegründet: mm stimmt tatsächlich mit unserem m von oben überein.

5. Das ElGamal Kryptosystem oder das Problem des diskreten Logarithmus

Als letztes Beispiel eines Chiffriersystems mit öffentlichem Schlüssel soll noch kurz auf das ElGamal-Kryptosystem eingegangen werden, da dieses vor allem in den USA sehr beliebt ist und es auch in PGP neben RSA verwendet wird.

Entscheidend für seine Sicherheit ist hier die angenommmene Schwierigkeit "diskrete Logarithmen" zu berechnen. Darunter versteht man allgemein folgendes Problem: Ist G eine Gruppe und U die von einem festen Element $g \in G$ erzeugte Untergruppe, wobei G und U als sehr groß vorausgesetzt werden, so soll für jedes vorgegebene $a \in U$ die Gleichung $a = g^k$ gelöst werden.

Beim klassischen ElGamal-Kryptosystem ist G die multiplikative Gruppe eines großen Körpers $\mathbf{F}_q$ und g ein Element von G, dessen Ordnung möglichst groß sein soll (im Idealfall ein erzeugendes Element!)

Jeder Anwender A wählt nun eine ganze Zahl a mit $0 < a < q - 1$. Es ist dann a sein geheimer Schlüssel und $g^a \in \mathbf{F}_q$ sein öffentlicher Schlüssel, welcher zusammen mit den Daten über die Gruppe G und das gewählte Element g publik gemacht wird. Nach obigem ist es trotzdem "praktisch unmöglich", a daraus zu errechnen.

Will nun jemand an A eine Nachricht schicken, die durch ein $m \in \mathbf{F}_q$ repräsentiert sei, so wählt er eine ganze Zufallszahl k und schickt an A das Paar

$$(g^k, mg^{ak}) \in \mathbf{F}_q^2$$

d.h. die Nachricht m wird im Produkt mg^{ak} gewissermaßen "versteckt". A kann jedoch trotzdem mit Hilfe von a, das nur er kennt, m mittels der Formel

$$m = (mg^{ak})(g^{k})^{q-1-a}$$

zurückrechnen.

Es ist dabei sehr wichtig, daß für k immer andere Werte genommen werden. Werden nämlich zwei Nachrichten m_1 und m_2 mit dem gleichen k verschlüsselt, so könnte man durch Quotientenbildung der jeweils zweiten Komponenten

$$\frac{m_1}{m_2} = \frac{m_1 g^{ak}}{m_2 g^{ak}}$$

und damit bei bekannten m_2 auch m_1 berechnen.

Das ElGamal-Kryptosystem hat gegenüber RSA den Vorteil, daß auch gleiche Nachrichten durch die Zufallszahl k immer verschiedene Chiffrate ergeben. Dem steht allerdings der Nachteil gegenüber, daß das Chiffrat durch die zweikomponentige Darstellung gegenüber der Nachricht um den Faktor 2 aufgebläht wird.

Für G können auch andere Gruppen genommen werden, insbesondere Gruppen über elliptischen Kurven bieten sich an. Der einfachste und zugleich häufigst verwendete Fall ist jedoch sicher der, daß G die multiplikative Gruppe eines Restklassenrings mod p für eine große Primzahl p ist. In diesem Fall ist das "Problem des diskreten Logarithmus" in etwa vergleichbar mit dem Faktorisierungsproblem für den Modul n in RSA, wo n die gleiche Größenordnung wie p hat. Daraus ergibt sich, daß p derzeit mindestens 512 bits haben sollte.

Nachfolgend wieder ein mit DERIVE gerechnetes Beispiel (die Rechenzeiten sind zwar durch den Zufallscharakter der Zahlen Schwankungen unterworfen, liegen jedoch immer im Zehntelsekundenbereich)

(p := NEXT_PRIME(RANDOM(2^512))) =
8286480867606574331875035298097192022140655830210928861603106683490810821505092875703471116556201534751394398620798025885454433090431933967035660268589691

FLOOR(LOG(p, 2)) + 1 = 512

g := 2

(a := RANDOM(p - 1)) =
5540779571969209169890558949569211541190580284135335411700169175685316026135148341859342677243027677994338790611389233135452833692496130077694303570770384

(alpha := MOD(g^a, p)) =
3068020664751838857817053291221346786002944047694073461836032269592240560206072867036365955413842015265232827106689531994331569719007356945617366043448810

p, g und alpha bilden nun den öffentlichen Schlüssel und a den privaten Schlüssel des Benutzers. Die Verschlüsselung geht dann z.B. so vor sich:

m := 123456789

(k := RANDOM(p - 1)) =
4243695947141676247314946550984174018039004413452092364244954653594614

558816492254931419764622449971454951028777912137779195885578801094796
3294855954259350

y := [MOD(g^k, p), MOD(m MOD(alpha^k, p), p)]=
[207703671704478666225822364765843110195675175779167168040059508631518146395108935104010151841861096444718892694930653259446861013320509959708643801974714 2,
230036094571616327791690153019127085789574442013650089495559287731092538805813173731064270263817677311387727210394338135587459168488706907637541863075524 2]

Und hier die Entschlüsselung mit Hilfe von a:

(mm := MOD(ySUB2 MOD(ySUB1^(p - 1 - a), p), p)) = 123456789

m = mm = true

6. Einige Fragen zur Sicherheit von Public Key Kryptosystemen am Beispiel von RSA

Wie sicher sind Public Key Kryptosysteme? Wir werden am Beispiel von RSA, das wohl bisher in dieser Hinsicht am meisten untersucht wurde, dieser Frage etwas nachgehen.

Zunächst sollte aus den bisherigen Ausführungen über RSA klar sein, daß ein ganz entscheidender Faktor für seine Sicherheit die Größe des Moduls n ist. Jeder, dem es gelingt die Faktorisierung n = pq zu finden, kann daraus in der gleichen Weise, wie wir dies gemacht haben, den privaten Schlüssel d ermitteln. Untere Grenze für die Länge von n in bits ist 512, es gibt aber heute schon Fachleute, die meinen, dies könnte bald nicht mehr sicher sein und größere n mit z.B. 768,1024 oder 2048 bits vorschlagen. (In PGP 6.0 kann man sogar bis 4096 bits gehen, was dann wohl mehr Ausdruck von Gigantomanie als echtes Sicherheitsbedürfnis ist.)

Hat man sich für eine Größe von n entschieden, so ergeben sich daraus automatisch die ungefähren Größen der Primfaktoren p und q, da diese etwa halb so viele Stellen haben sollten. Für ein n mit 1024 bits wären dies z.B. also je ca. 512 bits, was etwa 155 Dezimalstellen entspricht. Was übrigens auf den ersten Blick wie ein Problem aussieht, nämlich die Generierung von Primzahlen dieser Größenordnung, ist in der Praxis überhaupt keines. So dauerte etwa die Ausführung der Zeile

VECTOR(NEXT_PRIME(RANDOM(2^512)), k_, 1, 10)

welche mittels DERIVE 10 derartige Primzahlen erzeugt auf meinem PC nur 37s, also durchschnittlich 3.7s für eine einzelne.

Allerdings muß gesagt werden, daß nach strengen Maßstäben die so erhaltenen Zahlen nur mit einer für praktische Zwecke vernachlässigbar kleinen Irrtumswahrscheinlichkeit prim sind. Konkret wird in DERIVE ein auf Primalität zu testendes N zunächst auf Teilbarkeit durch kleine Primzahlen (bis max. 1021) überprüft und anschließend, falls notwendig, defaultmäßig bis zu 6 sog. Rabin-Miller-Tests unterworfen. Die Anzahl der Rabin-Miller-Tests kann aber auch als 2.Parameter von NEXT-PRIME manuell eingegeben werden. So dauert nun z.B. die Ausführung von

VECTOR(NEXT_PRIME(RANDOM(2^512), 100), k_, 1, 10)

wo zur Erhöhung der Sicherheit jede der als "prim" ausgewiesenen Zahlen nun 100 Rabin-Miller-Tests bestehen muß, etwas länger, nämlich 107s (d.h. 10.7s im Durchschnitt für eine einzelne Zahl), dafür ist aber auch die Irrtumswahrscheinlichkeit, wie man zeigen kann, auf weniger als 4^{-100} gesunken.

Obwohl es also bei sorgfältiger Auswahl so gut wie ausgeschlossen ist, daß eine der beiden so erhaltenen Zahlen p oder q nicht prim ist, kann man doch die folgende interessante Frage stellen: Würde man es spätestens bei der Anwendung von RSA mit diesen Parametern, d.h. bei der Verschlüsselung und Entschlüsselung von Nachrichten merken, wenn p oder q nicht prim wären? Die überraschende Antwort lautet: Nicht notwendigerweise.

Sieht man sich nämlich die Herleitung der grundlegenden mathematischen Gleichungen zu RSA in Kapitel 3 noch einmal an, so wird man feststellen, daß wir dabei nur zwei Tatsachen über p und q wirklich benötigt haben:

1. p und q sind teilerfremd.
2. p und q erfüllen beide den "Kleinen Fermatschen Satz", d.h. für alle $a \in \mathbb{Z}$ gilt

$$a^p \equiv a \mod p \text{ bzw. } a^q \equiv a \mod q.$$

Wie die nachfolgende DERIVE-Rechnung zeigt, müssen p und q dazu nicht notwendigerweise prim sein:

GCD(561, 1729) = 1

SELECT(MOD(a^561, 561) /= a, a, 0, 560) = [ ]

SELECT(MOD(a^1729, 1729) /= a, a, 0, 1728) = [ ]

FACTOR([561, 1729]) = [3·11·17, 7·13·19]

Tatsächlich gibt es, wie 1994 gezeigt wurde, sogar unendlich viele dieser Zahlen, welche also den "Kleinen Fermatschen Satz" erfüllen ohne prim zu sein. Sie werden in der Literatur Carmichaelzahlen genannt und unser Beispiel 561 ist zugleich das kleinste. (Es wäre in diesem Zusammenhang übrigens eine reizvolle Aufgabe, mit Hilfe von DERIVE eine Liste dieser interessanten Zahlen z.B. bis 10000 zu erstellen und einige ihrer Eigenschaften selbst herauszufinden.) Wie der Leser selbst nachprüfen möge, "funktioniert" die Verschlüsselung und Entschlüssselung mit diesen p und q für wie üblich gewählte Exponenten e und d tatsächlich, auch wenn sie sonst natürlich denkbar ungeeignet sind.

Welche Dinge hat man bei der Auswahl von p und q sonst noch zu beachten? Nachfolgend einige Punkte dazu:

- p und q sollten nicht zu nahe beieinander liegen

"Nahe" ist hier natürlich bezogen auf die Größe von p und q zu verstehen. Stimmen zum Beispiel die Hälfte der führenden Stellen von p und q überein, so kann ihre Absolutdifferenz noch immer sehr groß sein, relativ gesehen sind sie sich aber berreits viel zu nahe.

Das folgende Beispiel, in dem p und q beide 100-stellige Primzahlen sind, welche so konstruiert wurden, daß sie in den ersten 50 Stellen übereinstimmen, sei eine eindringliche Warnung in diesem Zusammenhang!

(r := RANDOM(10^50)) =
86941828992926916514847590040509253067992624121437

(p := NEXT_PRIME(10^50·r + RANDOM(10^50))) =
86941828992926916514847590040509253067992624121437086675501314336696961212113088196936618083298157 59

(q := NEXT_PRIME(10^50·r + RANDOM(10^50))) =
86941828992926916514847590040509253067992624121437982253367377240473020343455255097549539951936187 79

(n := p·q) =
75588816286353473703123387787160528682701504703205382587474241000390082989153169732118607454126699990117730076355813401333017753422571972200623657781234778569080549539408109043232969238758161552538261

Wir bestimmen zunächst die nächstgrößere ganze Zahl u zu $\sqrt{n}$:

(u := FLOOR(√n) + 1) =
86941828992926916514847590040509253067992624121437534464434345788584990777784171647243079017617172 69

Unter den angegebenen Voraussetzungen ist dann $u^2 - n$ stets ein Quadrat v^2 für eine natürliche Zahl v:

NUMBER(SQRT(u^2 - n)) = true

(v := SQRT(u^2 - n)) =
44778893303145188802956567108345030646093431901510

Wegen $n = u^2 - v^2 = (u + v)(u - v)$ hat man dann eine nichttriviale Faktorisierung von n gefunden, d.h. u+v und u-v müssen bis auf ev. die Reihenfolge mit p und q übereinstimmen. Tatsächlich gilt in unserem Beispiel

{p, q} = {u + v, u - v} = true

Wenn der Leser nun meint, daß angesichts dieser jedem Schüler der Unterstufe zugänglichen Faktorisierungsmethode nur ein Verrückter auf die Idee kommen könne, die Primzahlen so nahe beieinander zu wählen, so liegt er damit der Wahrheit näher, als er vielleicht ahnt.

Tatsächlich gab es im Oktober 1996 einen aufsehenerregenden Fall, wo ein inzwischen rechtskräftig verurteilter Verbrecher in einem zum Großteil mit RSA verschlüsselten Brief an ein österreichisches Nachrichtenmagazin eine Serie von Briefbomben ankündigte. Er verwendete dazu zwei Primzahlen p und q mit je 122 Stellen, welche sich jedoch nur in den 13 letzten Stellen unterschieden! Natürlich ist es dann, wie oben gezeigt, buchstäblich "ein Kinderspiel" das daraus resultierende 243-stellige n, nämlich

n :=
63054821507012954715671833249588963223443414541197127588837698760326022525278792613527673894410568910003629553586814142438653640364957870769912818949143213863190059077472921499001536910276096488477634484971781148430952891504011795209806188688 1

wieder in seine Primfaktoren zu zerlegen, wovon sich der Leser sich selbst überzeugen möge. (Interessanterweise spielte in dem Indizienprozeß auch eine Rolle, daß auf dem

Computer des Täters ein Programm zum Generieren von großen Primzahlen gefunden wurde - ein weiter Punkt in der Liste, die angibt, wozu Primzahlen gut sind!)

Übrigens ist der von uns hier betrachtetete Fall nur ein Spezialfall in einer etwas allgemeineren nach Fermat benannten Faktorisierungsmethode. Dabei startet man mit demselben u, wie oben, erhöht jedoch, falls es damit nicht klappt, d.h. $u^2 - n$ kein Quadrat ist, dessen Wert jeweils um 1. Eine entsprechende DERIVE-Routine könnte z.B. so aussehen

FERMAT(n) := (ITERATE(IF(NUMBER(SQRT(b_)), [a_, b_, [a_ - SQRT(b_), a_ + SQRT(b_)]], [a_ + 1, b_ + 2·a_ + 1, c_]), [a_, b_, c_], [- FLOOR(- SQRT(n)), FLOOR(- SQRT(n))^2 - n, []]))SUB3

(In obiger Schreibweise ist dabei a_ mit u und b_ mit $u^2 - n$ zu identifizieren. Die Erhöhung von a_ um 1 führt dann zu dem neuen Wert b_ + 2·a_ + 1 für b_.)

- p±1, q±1 sollten große Primfaktoren enthalten, andernfalls die (p - 1)-Methode von Pollard bzw. die (p + 1)-Methode von Williams anwendbar sind.

Wir gehen nachfolgend nur auf die (p - 1)-Methode von Pollard etwas näher ein. Die Grundidee ist, daß unter der Voraussetzung, daß ein Primfaktor p von n die Eigenschaft hat, daß p - 1 in viele relativ kleine Primfaktoren zerfällt (was in der Praxis allerdings nur selten zutrifft!), es dann nicht schwer ist ein ganzes e zu finden, daß ein Vielfaches von p - 1 ist. Man könnte für e etwa das Produkt aller Primzahlpotenzen unterhalb einer geeignet gewählten Schranke s nehmen. (Die gemachte Voraussetzung über p besagt dann gerade, daß s relativ klein sein kann!)

Hat man nun tatsächlich so ein e gefunden und ist a mit $1 < a < n$ zu n teilerfremd (man beachte, daß diese Bedingung über a sicher keine Einschränkung darstellt, da ja sonst schon ggT(a,n) ein nichttrivialer Teiler von n wäre!), so gilt wieder aufgrund des "Kleinen Fermatschen Satzes" (diesmal allerdings in der Form

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

wo also durch a "gekürzt" wurde), daß

$$a^e = (a^{p-1})^{e/(p-1)} \equiv 1 \bmod p\,.$$

Daraus folgt nun aber sofort, daß der $\mathrm{ggT}(a^e - 1, n)$ durch p teilbar, also jedenfalls nicht 1 ist. Außer in dem extrem unwahrscheinlichen Fall, daß er n ist, hätten wir also so einen nichttrivialen Teiler von n gefunden.

Die nachfolgende DERIVE-Implementierung verwendet die gleiche Idee, doch wurden der Einfachheit halber die Primzahlen > 7 in gewisser Weise ersetzt durch Zahlen, die zu 210 teilerfremd sind und diese bis zu einer Schranke s in e aufakkumuliert. Dabei wird in gewissen Abständen, nämlich jeweils nach Abarbeitung eines Intervals der Länge 210 mit dem momentanen e nachgesehen, ob $\mathrm{ggT}(a^e - 1, n) > 1$ ist. Ist dies der Fall oder wird die Schranke s erreicht, so wird der Wert dieses ggT retourniert. (Die Werte von a und s können beim Aufruf auch weggelassen werden, worauf dann deren Defaultwerte 3 bzw. ∞ genommen werden.)

PMINUS1(n, a, s) := GCD((ITERATE(IF(GCD(a_ - 1, n) > 1 OR IF(s < b_SUB2, true, false, false), [a_, b_], [MOD(a_^PRODUCT(b_), n), b_ + [0, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210,

210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210, 210]]), [a_, b_], [MOD(IF(a, 3, a, 3)^160030080000, n), [210, 1, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 121, 127, 131, 137, 139, 143, 149, 151, 157, 163, 167, 169, 173, 179, 181, 187, 191, 193, 197, 199, 209]]))SUB1 - 1, n)

Im nachfolgenden Beispiel hat der Primfaktor p = 73! +1 von n nur kleine Primfaktoren und wird daher sehr schnell gefunden, obwohl er immerhin 106 Stellen hat!

(n := (73! + 1)·(10^100 + 267)) =
44701154615126843408912571381250511100768007002829050158190800923704221040671833170169036800000001193520838223886719017965655879388646390505786975535639223694384662902701785937945643513282560000000000000267

APPROX(n) = 4.47011·10^205

APPROX(73! + 1) = 4.47011·10^105

PMINUS1(n) = 73! + 1 = true (13.7s !)

Hier schließlich noch einige weitere Punkte, auf die man achten sollte, auf die wir aber nicht mehr im Detail eingehen:

- Die multiplikative Ordnung r von e mod kgV(p-1,q-1) sollte nicht zu klein sein, andernfalls könnte man (r-1)-maliges Iterieren der Chiffrierabbildung auf $c = m^e \bmod n$, d.h. durch Berechnung von

  $$c^e \bmod n, (c^e)^e \bmod n, ((c^e)^e)^e \bmod n, \ldots$$

  den Klartext m erhalten. (Dies wird übrigens wieder durch die spezielle Wahl von p und q sichergestellt: Sind r und s große Primfaktoren, welche p - 1 bzw. q - 1 nach obigen enthalten sollten, so sollten auch sie wieder die Eigenschaft haben, daß r - 1 bzw. s - 1 einen großen Primfaktor besitzen.)

- Derer ggT(p-1,q-1) sollte nicht zu groß sein, da damit das kgV(p-1,q-1) und damit auch d, das ja von derselben Größenordnung ist, zu klein wird. (Besonders kritisch wird es, wenn d weniger als ein Viertel der Stellen von n hat: In diesem Fall gibt es sogar einen Polynomialzeitalgorithmus zu seiner Berechnung; siehe [1]).

- Die Anzahl der Fixpunkte von $x^e \bmod n$, nämlich

  $$(1+ggT(p-1,e-1))(1+ggT(q-1,e-1))$$

  sollte ebenfalls nicht zu groß sein. (Pikanterweise dürfen die Fixpunkte der Chiffrierabbildung, für die also das Chiffrat Klartext darstellt, nicht bekanntgemacht werden, da man daraus sofort p und q berechnen könnte!)

Abschließend noch einige grundsätzliche Bemerkungen zur Frage: Wie sicher ist RSA? Wie schon erwähnt, besteht eine latente Gefahr darin, daß sich das Faktorisierungsproblem wider Erwarten als "einfach" herausstellen könnte, z.B. indem man einen Polynomialzeitalgorithmus dafür findet. (Im konkreten Fall bedeutet dies, daß der Aufwand zum Faktorisieren von N, gemessen z.B. an der Anzahl der Bitoperationen, durch einen Ausdruck der Form $C(\ln N)^k$ mit positiven Konstanten C und k nach oben beschränkt ist und damit insbesondere relativ langsam wächst.) Leider ist aber für den allgemeinen Fall von RSA noch nicht einmal bewiesen, daß die Faktorisierung des

Moduls n überhaupt benötigt wird für die Lösung des eigentlichen "RSA-Problems" nämlich die Berechnung der mod n eindeutigen Lösung m der Kongruenz

$$x^e \equiv c \bmod n$$

Um wenigstens dies zu gewährleisten, betrachtet man auch spezielle Varianten von RSA, für die sich dies streng beweisen läßt. Die wichtigste ist die sog. Rabin-Variante, wo stets e=2 genommen wird, allerdings um den Preis, daß obige Kongruenz dann i.allg. 4 Lösungen mod n besitzt, unter denen man dann unter Zuhilfenahme von Redundanz die "richtige" auswählen muß.

Eine Angst, welche auch immer wieder geäußert wird, ist allerdings unbegründet, daß nämlich durch den Fortschritt in der Computertechnologie faktorisieren leichter und RSA damit unsicherer werden könnte. Tatsächlich ist es sogar so, daß technologischer Fortschritt es sogar sicherer macht, da bei gleichbleibenden Datendurchsatz die Parameter sehr viel größer gewählt werden können, womit die Möglichkeit, größere Zahlen zu faktorisieren, weit mehr als nur wettgemacht wird. (Der letzte Stand der Dinge auf diesem Gebiet ist übrigens die Faktorsierung von RSA-140 im Februar 1999 und über Internet wurde inzwischen auch bereits ein Projekt zur Faktorsierung von RSA-155 gestartet, was genau der Modullänge von 512 bits bei RSA entspricht.)

Was die Frage der Sicherheit von RSA betrifft, kommt D. Boneh, der in [1] eine ganze Reihe von Attacken gegen dieses Kryptosystem unter die Lupe nimmt, zu folgendem Schluß. "Two decades of research into inverting the RSA function produced some insightful attacks, but no devastating attack has ever been found. The attacks so far mainly illustrate the pitfalls to be avoided when implementing RSA. At the moment it appears that proper implementation can be trusted to provide security in the digital world." Wenn wir's auch nicht beweisen können, so glauben wir's doch gerne!

Literatur

[1] D. Boneh, Twenty Years of Attacks on the RSA Cryptosystem, Notices of the AMS, Vol.46, Nr.2 (203-213)

[2] W.Diffie and M.Hellman, New directions in cryprography, IEEE Transactions on information theory, IT-22(1976),644-654.

[3] A. Menezes, P. van Ooorschot and S. Vanstone, Handbook of Applied Cryptography, CRC Press, Boca Raton, FL, 1996.

[4] R.L.Rivest, A.Shamir and L.Adleman, A method for obtaining digital signatures and public key cryptosystems, Commun. ACM 21 !978), 120-126.

Anschrift des Autors

Ao Prof. Dr. Johann Wiesenbauer
(email: j.wiesenbauer@tuwien.ac.at)
Institut f. Algebra u. Computermath.
TU Wien
Wiedner Hauptstraße 8-10
A-1040 Wien