RSA & Co. in der Schule

Moderne Kryptologie, alte Mathematik, raffinierte Protokolle

Neue Folge – Teil 1: RSA für Einsteiger

von Helmut Witten und Ralph-Hardo Schulz

In den letzten 2500 Jahren war die Kryptologie ein Werkzeug der Mächtigen – und solcher, die es werden wollten. Die Kunst des Verschlüsselns war (und ist) unerlässliches Handwerkszeug der Diplomaten, der Militärs, der politischen Führer – und natürlich auch der Spione und Verschwörer. Die Renaissance-Päpste und Michael Bakunin, Winston Churchill und Richard Sorge waren von der Wichtigkeit geheimer Botschaften überzeugt.

In den letzten 30 Jahren ist die Kryptologie durch die Verbreitung vernetzter Computersysteme zum Gegenstand öffentlicher Diskussionen geworden: Politiker streiten über Krypto-Politik („Soll jedermann effektiv verschlüsseln dürfen oder muss sichergestellt werden, dass die Polizei im Zweifelsfall mitlesen kann?“), Kryptologen diskutieren auf öffentlichen Kongressen über die neuesten Algorithmen zur Chiffrierung (und ob man sie brechen könnte), „Cypherpunks“ (eine abgewandelte Zusammensetzung aus *Cipher* – Geheimschrift – und *Punk* – Strolch) haben mit ihren kryptologischen Kenntnissen dazu beitragen, die Privatsphäre der Bürger vor dem „Big Brother“ zu schützen. Und schließlich ist Datensicherheit ohne Kryptologie nicht zu erreichen.

Kryptologie und Schule

An der Schule hat sich dieses spannende Thema inzwischen ebenfalls etabliert. In zahlreichen Artikeln wurden Unterrichtseinheiten vorgestellt, auch die Autoren dieses Artikels haben sich daran beteiligt. In drei Folgen haben wir zusammen mit Irmgard Letzner (Witten/Letzner/Schulz, 1998 f.) wichtige „klassische“ Verschlüsselungsverfahren behandelt, die auch Schülerinnen und Schülern verständlich gemacht werden können: von Caesar über Vigenère bis zum „one-time-pad“ (letzteres ist auch als das *Verfahren von Vernam* bekannt). Gleichzeitig haben wir uns immer auch mit den Möglichkeiten der Entschlüsselung ohne Kenntnis des Schlüssels (der so genannten Kryptoanalyse) beschäftigt, da nur so die Qualität einer Chiffre beurteilt werden kann. Dabei hat sich gezeigt, dass zurzeit einzig das Verfahren von Vernam wirkliche Sicherheit bietet, die sich sogar mathematisch beweisen lässt („perfekte Sicherheit“).

Diese Sicherheit hat allerdings ihren Preis: Es ist sehr aufwändig, wirklich zufällige Schlüssel zu erzeugen, die dann nur einmal verwendet werden dürfen – sonst ist die Sicherheit nicht mehr „perfekt“! Zum Zweiten besteht die Schwierigkeit, diese Schlüssel zu verteilen – bei Millionen von Kommunikationsteilnehmern ein praktisch unlösbares Problem, weil die Zahl der benötigten Schlüssel quadratisch mit der Zahl der Kommunikationsteilnehmer wächst. Und zum Dritten kann auch dieses Verfahren nicht garantieren, dass die Nachricht wirklich von dem Absender stammt, der angegeben wurde – das legt die Forderung nach Authentifikation nahe.

Alle bislang vorgestellten Verfahren wurden bereits vor der Erfindung der Computernetze entwickelt (siehe Bild 1, nächste Seite). Mit der zunehmenden Verwendung dieser Netze für den elektronischen Zahlungsverkehr und zur Kommunikation z. B. per E-Mail entstand ein dringender Bedarf nach kryptologischen Verfahren, die auch zivilen Nutzern zur Verfügung stehen. Dazu mussten völlig neue Wege beschritten werden.

Neue Wege in der Kryptografie

Kennzeichnend für die moderne Kryptografie sind die erst Ende der 70er-Jahre entwickelten asymmetrischen oder öffentlichen Chiffriersysteme. Solche Verschlüsselungsverfahren wurden 1976 von Whitfield Diffie und Martin E. Hellmann vorgeschlagen und unter dem Titel „New Directions in Cryptography“ veröffentlicht. Eine sichere Implementation eines asymmetrischen Kryptosystems wurde zuerst 1978 von **R**ivest, **S**hamir und **A**dleman realisiert (daher RSA-Algorithmus). Inzwischen wurden weitere zuverlässige asymmetrische Kryptosysteme entwickelt (El Gamal und als Weiterentwicklung davon elliptische Kurven), in der

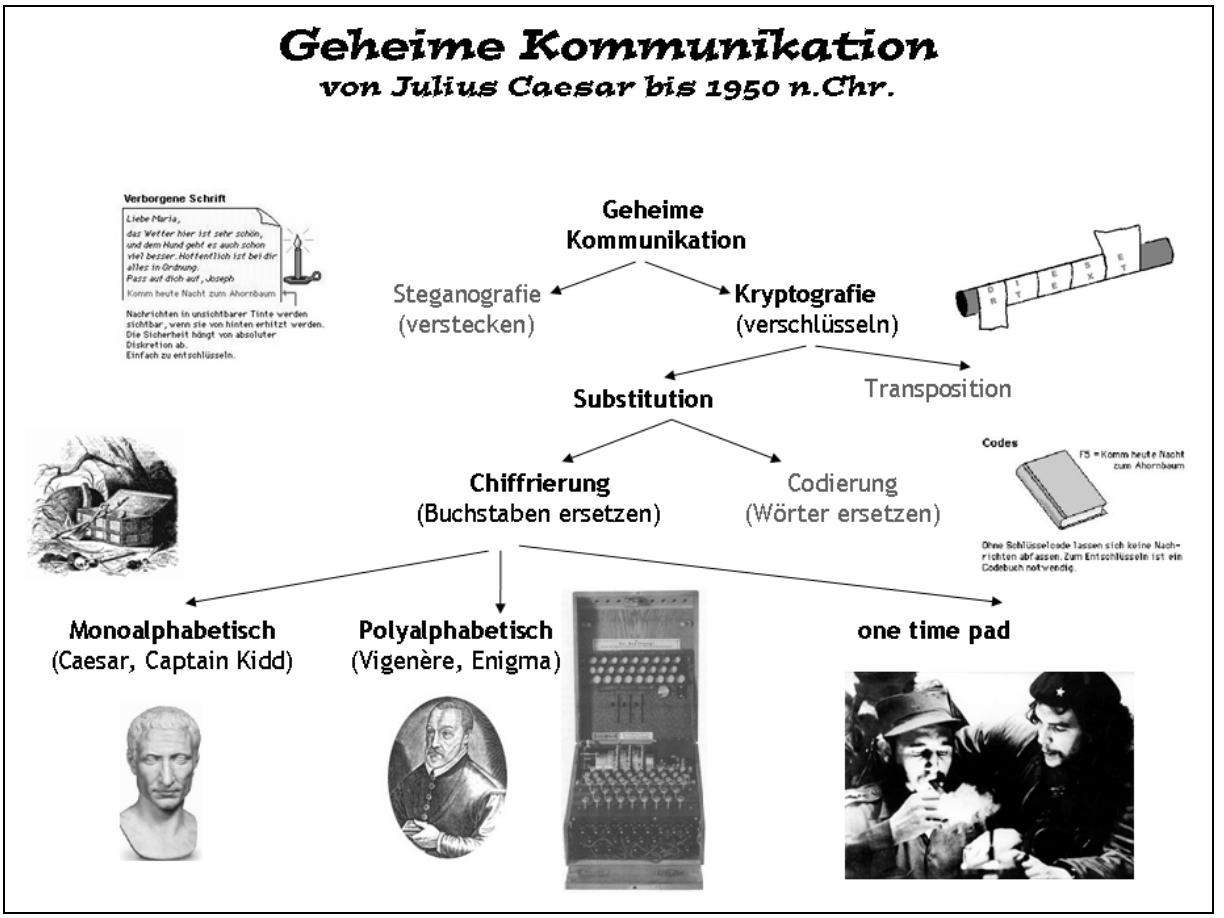


Bild 1.

Praxis ist aber RSA nach wie vor das am häufigsten eingesetzte System (siehe Bild 2, nächste Seite).

Bei der öffentlichen Verschlüsselung erhält jeder Teilnehmer an der Kommunikation ein eigenes Schlüsselpaar: einen öffentlichen Schlüssel, der z. B. in einer Art Telefonbuch verzeichnet ist, und einen privaten Schlüssel, den nur der Teilnehmer selbst kennt und für andere unzugänglich aufbewahrt. Damit braucht man nur noch so viele Schlüsselpaare wie Teilnehmer.

Will man eine Nachricht übermitteln, die vor unbefugtem Lesen geschützt sein soll, so verschlüsselt man diese mit dem öffentlichen Schlüssel des Empfängers; nur dieser kann sie dann mit seinem privaten Schlüssel entschlüsseln. Umgekehrt kann man mit den asymmetrischen Systemen auch das Problem der elektronischen Unterschrift lösen: Der Teilnehmer verschlüsselt einen Teil seiner Nachricht mit seinem geheimen Schlüssel, und jeder kann mit dem öffentlichen Schlüssel nachprüfen, dass diese wirklich von dem Betreffenden stammt. Kombiniert man beide Methoden, erhält man eine vertrauliche und authentische Kommunikation, die vor Abhören und Verfälschen sicher ist, wenn man weitere Vorsichtsmaßnahmen berücksichtigt.

Wie kann man nun asymmetrische Kryptosysteme realisieren? Da die Dechiffrierfunktion die Inverse der Chiffrierfunktion ist, bedeutet dies, dass die Chiffrierung mit Funktionen erfolgen muss, deren Inverse nur mit einem unakzeptabel hohen Aufwand bestimmt werden kann. Solche Funktionen werden auch als Einweg- oder Falltür-Funktionen bezeichnet: Der Informationsgehalt ist für den unberechtigten Lauscher nach der Verschlüsselung wie bei einem Schnappschloss oder einer Falltür verschwunden und kann nur mit einer zusätzlichen Information (dem privaten Schlüssel oder bei der Falltür mit einer passenden Leiter) wieder ans Tageslicht befördert werden.

Hierbei kommt die „alte“ Mathematik ins Spiel: Die Erfinder des RSA-Verfahrens haben ein Stück klassischer elementarer Zahlentheorie verwendet, auf die wir in dieser Artikelserie noch genauer eingehen werden (es handelt sich im Wesentlichen um das Potenzieren in endlichen Ringen oder Gruppen, wobei dem so genannten „kleinen Satz von Fermat“ eine zentrale Rolle zukommt). Um die Falltürfunktion zu konstruieren, werden bei RSA Produkte sehr großer Primzahlen verwendet, die zwar schnell zu berechnen sind, sich bei einer entsprechenden Länge aber nur mit ungeheuer großem Rechenaufwand wieder in ihre Faktoren zerlegen lassen (vgl. Schulz, 1993; Kippenhahn, 1997, S. 277 ff., sowie Singh, 2000, S. 329 ff.).

Neben den asymmetrischen Verfahren spielen aber auch im Coumputerzeitalter symmetrische Verfahren nach wie vor eine große Rolle (DES, AES u. a.). Dies liegt darin begründet, dass mit den genannten Verfahren größere Datenmengen schneller verschlüsselt werden können. In der Praxis kommen daher häufig so genannte Hybridverfahren zum Einsatz, so z. B. bei PGP (***P**retty **G**ood **P**rivacy*). Dabei wird zunächst ein Sitzungsschlüssel für die symmetrische Verschlüsselung mit einem asymmetrischen Kryptosystem sicher übermittelt; die eigentliche Verschlüsselung wird dann mit einem schnellen symmetrischen Verfahren durchgeführt.

RSA für Eilige

Wenn nur wenig Unterrichtszeit zur Verfügung steht, wird man das RSA-Verfahren direkt vorstellen. Hierbei hat sich der Arbeitsbogen von Frau Letzner bewährt, den wir im Unterricht und in Fortbildungen schon häufig eingesetzt haben und der für den Unterricht in der Sekundarstufe I entwickelt wurde (Kasten: „Das RSA-Verfahren“, nächste Seite). Da das modulare Rechnen meist nicht bekannt ist, weil es im Mathematikunterricht normalerweise nicht vorkommt, muss zunächst das aus der Grundschule bekannte Teilen mit Rest wiederholt werden. Die weiteren mathematischen Grundlagen des modularen Rechnens sind zur Information der Leserinnen und Leser im Kasten „Regeln für modulares Rechnen“ (Seite 49) zusammengestellt.

Modulares Rechnen ist den Lernenden aus dem Alltag vertraut. So werden beispielsweise Uhrzeiten modulo 12 oder modulo 24 angegeben, Wochentage modulo 7, Kilometerzähler bei Kraftfahrzeugen modulo 100000. Zur Übung können Aufgaben der folgenden Art dienen:

- ▷ Es ist jetzt 11 Uhr. Wieviel Uhr ist es nach 100 Stunden?
- ▷ Heute ist Donnerstag. Welcher Wochentag ist nach 47 Tagen?
- ▷ Es ist der Monat August. Welcher Monat ist nach 50 Monaten?

Wenn die Begriffe *Rest* und *Kongruenz* eingeführt wurden, wird als Einstieg die Aufgabe formuliert, ein eigenes RSA-System nach der Vorlage von Kasten „Das RSA-Verfahren“ (nächste Seite) zu bilden. Erlaubtes Hilfsmittel ist zunächst nur ein einfacher Taschenrechner. Dieses Vorgehen bietet zwei didaktische Vorteile: Die Lernenden machen sich einerseits mit den Handhabung des RSA-Systems vertraut. Andererseits stoßen sie dabei auf Probleme, die im weiteren Unterrichtsgang geklärt und nicht extra motiviert werden müssen.

Als Arbeitsform bieten sich Partner- oder Gruppenarbeit an. Die einzelnen Teams können sich nach berühmten Vorbildern benennen: Room 40, Black Chamber, Bletchley Park, NSA, BSI, CCC, … Da mit asymmetrischer Kryptografie ja in aller Öffentlichkeit chiffriert werden kann, erhalten die Gruppen den Auftrag, ihren öffentlichen Schlüssel (n, d) und eine chiffrierte Botschaft c zu veröffentlichen. Wenn die Gruppen damit fertig sind, haben sie die Aufgabe, die RSA-Systeme der anderen Gruppen zu „knacken“. Bei Schwierigkeiten zeigt sich dann meistens, dass die jeweilige Gruppe noch Fehler in ihrem System hat. Besonders häufig werden die Module n (zum Ver- und Entschlüsseln) und $\varphi(n)$ (zur Erzeugung des Schlüsselpaares) verwechselt.

Dabei ist es nach unserer Erfahrung kein Problem, dass die Botschaft m und die verschlüsselte Nachricht c zunächst nur einfache Zahlen sind. Auf das Problem der Umwandlung von Texten in Zahlen und umgekehrt kommen wir später zurück.

Die Berechnung zum Beispiel des ganzzahligen Restes bei der Division von *6241* mit *85* kann mit einem Taschenrechner auf zwei Arten vorgenommen werden:

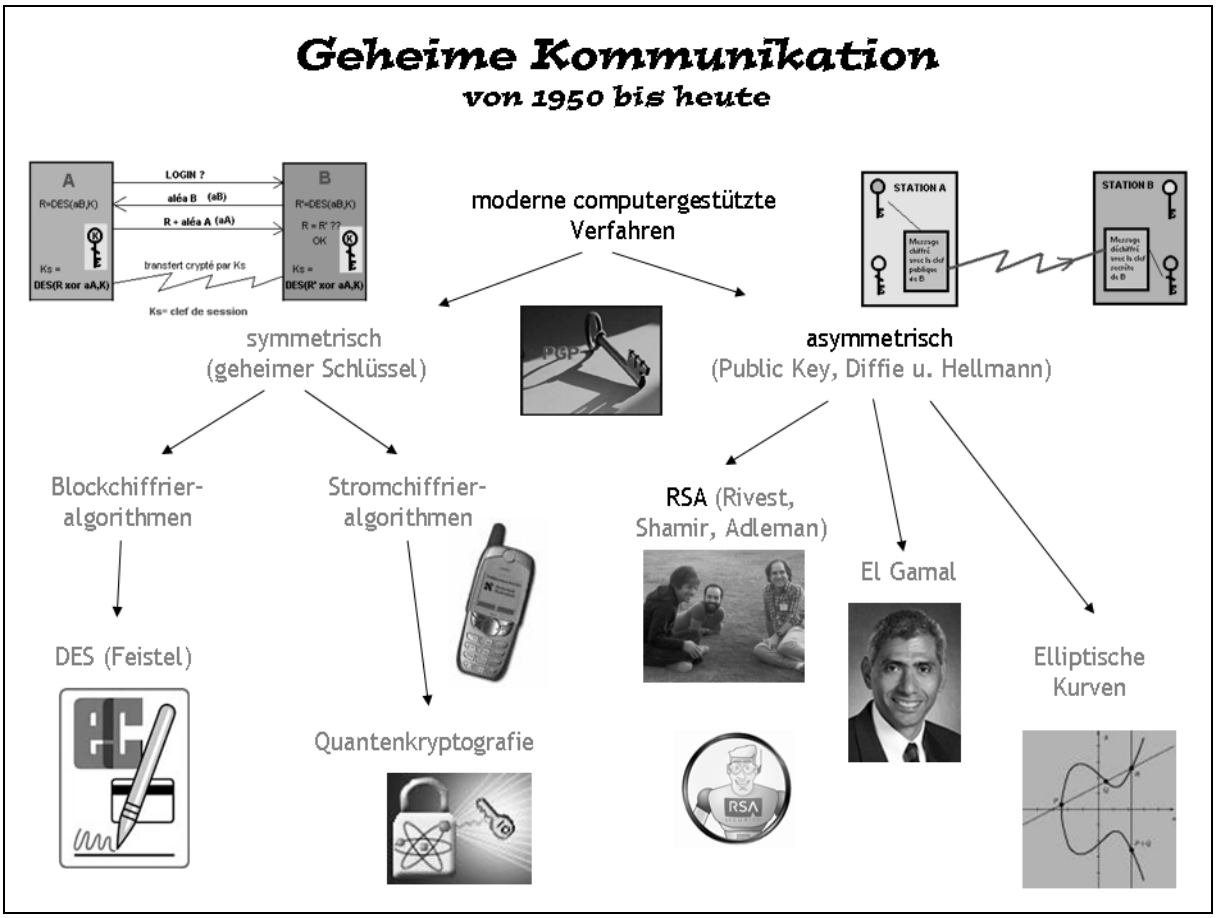


Bild 2.

1. Man dividiert: $6241 : 85 = 73{,}4235$
   Man multipliziert den ganzzahligen Anteil mit dem Modul: $73 \cdot 85 = 6205$
   Man bildet die Differenz: $6241 - 6205 = 36$
   Ergebnis: $6241 \equiv 36 \bmod 85$
2. Man dividiert: $6241 : 85 = 73{,}4235$
   Man multipliziert die Nachkommanstellen mit dem Modul: $0{,}4235 \cdot 85 = 36$
   Ergebnis: $6241 \equiv 36 \bmod 8$

Wenn aber die Zahlen größer werden (z. B. Berechnung von 85^{10} modulo 26), versagen beide Verfahren, weil übliche Taschenrechner in Gleitkommadarstellung übergehen und weder die Vor- noch die Nachkommastellen genau abzulesen sind: $85^{10} : 26 = 7{,}572092 \cdot 10^{17}$.

Bei der Entwicklung eines eigenen RSA-Systems werden durch die Potenzierung Kongruenzen für große Zahlen benötigt. Zur Reduktion der Zahlen ist die folgende Regel hilfreich (vgl. Kasten, S. 49, zur Wahl der Schreibweise siehe Witten/Letzner/Schulz, 1999, S. 54):

$$R_m(a \cdot b) = R_m[R_m(a) \cdot R_m(b)]$$
mit $[R_m(a) = c \Leftrightarrow a \equiv c \bmod m$ und $0 \le c < m]$

Diese Regel lässt sich leicht durch Nachrechnen beweisen. Im Informatikunterricht werden evtl. Zahlenbeispiele genügen, um sie plausibel zu machen.

Die Regel eröffnet Möglichkeiten, die Zahlen zu verkleinern, sodass sie auch mit dem Taschenrechner bearbeitet werden können. Man sagt dann, dass die Zahlen modulo m reduziert werden.

Beispiel:

$$\begin{aligned} R_{26}(85^{10}) &= R_{26}(85^3 \cdot 85^3 \cdot 85^4) \\ &= R_{26}[R_{26}(85^3) \cdot R_{26}(85^3) \cdot R_{26}(85^4)] \\ &= R_{26}(5 \cdot 5 \cdot 9) \\ &= R_{26}(225) = 17 \end{aligned}$$

oder auch

$$\begin{aligned} R_{26}(85^{10}) &= R_{26}((85^3)^3 \cdot 85^1) \\ &= R_{26}[(R_{26}(85^3))^3 \cdot R_{26}(85^1)] \\ &= R_{26}(5^3 \cdot 7) \\ &= R_{26}(875) = 17 \end{aligned}$$

Es gilt also $85^{10} \equiv 17 \bmod 26$.

Nach diesen Vorbereitungen bzw. Einhilfen wird sich die erste Stufe des Wettbewerbs ohne Weiteres durchführen lassen.

Bei der Auswertung ergeben sich dann weiterführende Fragen:

1. Wie kann ich das Verfahren zur Bildung der modularen Potenzen so vereinfachen, dass es sich programmieren lässt?
2. Wie erhalte ich genügend große Primzahlen?
3. Wie kann ich die Produkte wieder in die einzelnen Faktoren zerlegen?

Die erste Frage führt auf den bekannten „square and multiply"-Algorithmus, der im Folgenden vorgestellt wird. Die Beantwortung der beiden anderen Fragen wird zunächst zurückgestellt.

Von den Ägyptern zu „Square and Multiply"

Als Einstieg für diesen Algorithmus verwenden wir die „ägyptische" oder „äthiopische" oder auch „russische Bauern-Multiplikation", die aus der Unterhaltungsmathematik bekannt ist. Wir zitieren aus dem

Das RSA-Verfahren

Das Verfahren wurde 1978 von **R**ivest, **S**hamir und **A**dleman entwickelt.

Schlüsselvergabe	**Simulation**
▷ Wähle zwei große Zahlen p und q!	p = 5 und q = 11
▷ Berechne n = p · q!	n = 5 · 11 = 55
▷ Bestimme zwei Zahlen d und e so, dass d · e ≡ 1 MOD φ(n), d. h. d · e ≡ 1 MOD (p – 1) (q – 1)!	φ(n) = (p – 1) (q – 1) = 4 · 10 Kandidaten für d · e: 41 ≡ 1 MOD 40 (prim!) 81 = 27 · 3

ÖFFENTLICHER SCHLÜSSEL: (n, e)	ÖFFENTLICH: n = 55 und e = 27
GEHEIMER SCHLÜSSEL: d	GEHEIM: d = 3

Verschlüsselung

▷ (durch beliebigen Teilnehmer mithilfe des öffentlichen Schlüssels (n, e))	(durch beliebige Teilnehmer mithilfe des öffentlichen Schlüssels (55; 27))
$c \equiv m^e \bmod n$ ($c = R_n(m^e)$)	Verschlüsselung von 2: $2^{27} \equiv 18 \bmod 55$

Entschlüsselung

▷ (durch Empfänger mithilfe des geheimen Schlüssels d)	(durch Empfänger mithilfe des geheimen Schlüssels d = 3)
$m' \equiv c^d \bmod n$ ($m' = R_n(c^d)$) $\Rightarrow m' \equiv c^d \equiv m^{ed} \equiv m \bmod n$	$18^3 \equiv 2 \bmod 55$

Regeln für modulares Rechnen

1 Kongruenz modulo *n*

Es bezeichne $R_n(m)$ den Rest zwischen 0 und $n-1$ von m bei Division durch n. Andere Bezeichnungen: m mod n, $m\%n$. Wir definieren:

$a \equiv b \pmod n$ (a kongruent b modulo n)
$:\Leftrightarrow n$ teilt $a-b$ (in Zeichen: $n \mid (a-b)$)
$:\Leftrightarrow \exists t \in \mathbb{Z} : a - b = t \cdot n$
$\Leftrightarrow a$ und b haben bei Division durch n den gleichen Rest
$\Leftrightarrow R_n(a) = R_n(b)$.

Wir vermerken, dass u. a. $R_n(a) \equiv a \pmod n$ folgt.
Es gelten folgende Rechenregeln:

2 Rechenregeln

$a_i \equiv b_i \pmod n$ für $i = 1,2 \quad \Rightarrow \quad a_1 + a_2 \equiv b_1 + b_2 \pmod n$
$a_1 \cdot a_2 \equiv b_1 \cdot b_2 \pmod n$
(und damit) $a \equiv b \pmod n \quad \Rightarrow \quad a^k \equiv b^k \pmod n$.

Division einer Kongruenz durch t ist nur erlaubt, wenn t und n teilerfremd sind (vgl. Beispiel 3 c):

Für ggT(t, n) = 1 folgt [$at \equiv bt \pmod n \Rightarrow a \equiv b \pmod n$].

3 Beispiel für das Rechnen mit Resten

a) $R_{12}(15) + R_{12}(11) = 3 + 11 = 14 \equiv 2 \pmod{12}$
$\neq 2$
Man beachte also: $R_n(a_1+a_2) = \boldsymbol{R_n}(R_n(a_1) + R_n(a_2))$

b) Definiert man die Verknüpfung $\oplus_n$ und $\odot_n$ durch
$a \oplus_n b := R_n(a+b)$ und $a \odot_n b := R_n(a \cdot b)$,
so ergeben sich für $n = 5$ folgende Verknüpfungstafeln:

$\oplus_5$	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

$\odot_5$	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

c) Beispiele von Produktbildungen (bei Modul 10) mit und ohne Umkehrbarkeit:

a	0	1	2	3	4	5	6	7	8	9
$3 \cdot a \pmod{10}$	0	3	6	9	2	5	8	1	4	7

Die Umkehrung (Division durch 3) ist möglich,
da ggT(3, 10) = 1 gilt.

a	0	1	2	3	4	5	6	7	8	9
$4 \cdot a \pmod{10}$	0	4	8	2	6	0	4	8	2	6

Division durch 4 ist nicht möglich, da ggT(4, 10) $\neq$ 1 gilt.

4 Restklassen

Die Kongruenz mod n ist eine Äquivalenzrelation. Jede Äquivalenzklasse (*Restklasse*) ist von der Form $\overline{a} = \{a + kn \mid k \in \mathbb{Z}\} = a + n\mathbb{Z}$, besteht also aus ganzen Zahlen, die bei Division durch n den Rest a haben. Je zwei Restklassen mod n sind gleich oder disjunkt. Es gilt

$$\mathbb{Z} = \bigcup_{a=0}^{n-1} \overline{a}\,.$$

Beispiel: $\mathbb{Z} = \overline{0} \cup \overline{1}$ für $\overline{0} = 2\mathbb{Z}$ und
$\overline{1} = \{..., -7, -5, -3, -1, 1, 3, 5, 7, \ldots\}$

Die Restklassen modulo n fassen wir zusammen in der Menge

$\mathbb{Z}_n := \{\overline{0}, \overline{1}, ..., \overline{n-1}\}$.

Jedes Element $\overline{a}$ dieser Menge wird repräsentiert durch den Rest $R_m(a)$, der oft mit $\overline{a}$ identifiziert wird.

Verknüpfungen auf $\mathbb{Z}_n$ erhält man durch die Definition
$\overline{a} + \overline{b} := \overline{a+b}$ und $\overline{a} \cdot \overline{b} := \overline{a \cdot b}$;
diese Addition und Multiplikation entsprechen bei dem Übergang zu den Resten den Verknüpfungen $\oplus_n$ und $\odot_n$.
Identifiziert man z. B. bei $\mathbb{Z}_5$ die Restklassen $\overline{a}$ mit den Resten a (für $a \in \{0, 1, \ldots, 4\}$), so erhält man die beiden Verknüpfungstafeln von 3 b).

Anmerkung: $(\mathbb{Z}_5, +, \cdot)$ ist dabei ein endlicher Körper (mit 5 Elementen), auch GF (5) genannt. Analog erhält man mit $(\mathbb{Z}_p, +, \cdot)$ für eine Primzahl p einen Körper mit p Elementen, als GF (p) bezeichnet.

schönen Buch „Das chinesische Dreieck“ von Dominic Olivastro (1995, S. 20):

> In Äthiopien erzählt man sich die Geschichte von einem Oberst, der sieben Stiere kaufen wollte, von denen jeder 22 Maria-Theresien-Taler kostete. Der Besitzer der Tiere rief den örtlichen Priester, der die erforderliche Multiplikation ausführte, indem er in den Erdboden eine Reihe von Löchern (sogenannten Häusern) grub, die in zwei parallelen Spalten angeordnet waren. Ins oberste Haus der ersten Spalte legte er 7 Kiesel (die Zahl der Stiere), ins oberste Haus der zweiten Spalte 22 (der Kaufpreis für jeden Stier). Hier der Bericht des Obersts:
>
> „Man erklärte mir, dass die erste Spalte zum Multiplizieren mit zwei verwendet wird, das heißt: Doppelt so viele Kiesel wie im ersten Haus werden ins zweite getan, dann zweimal soviel ins dritte und so weiter. Die zweite Spalte dient dem Dividieren durch zwei: Halb so viele Kiesel, wie im ersten Haus sind, werden ins zweite getan und so weiter, bis im letzten Haus nur noch ein Kiesel ist. Brüche werden dabei nicht berücksichtigt. Dann sieht man nach, in welchen Mulden der Teilungsspalte sich gerade und in welchen sich ungerade Zahlen von Kieseln befinden. Alle geraden Häuser gelten als böse, alle ungeraden Häuser als gut. Sobald man ein böses Haus ausgemacht hat, werden die Kiesel (auch aus dem daneben lie-

genden Haus der ersten Spalte) hinausgeworfen und nicht gezählt. Dann werden alle Kiesel in den übrigen Mulden der Multiplikationsspalte gezählt, und die Endsumme ist das Ergebnis."

Das Problem des Obersten sah also folgendermaßen aus:

Erste Spalte Verdoppelung	Zweite Spalte Halbierung
~~7~~	~~22~~
14	11
28	5
~~56~~	~~2~~
112	1
154	

Mit anderen Worten: $7 \cdot 22 = 154$. Man wird an dieser Stelle den Schülerinnen und Schülern die Aufgabe geben, dieses Verfahren mit anderen Zahlen nachzuvollziehen. Bei genauerer Untersuchung dieses Beispiels stellt man fest, dass hier eigentlich mit einer Summe von Zweierpotenzen multipliziert wird: $7 \cdot 22 = 14 + 28 + 112 = 7 \cdot (2 + 4 + 16)$. Notiert man hinter der zweiten Spalte die Reste der Division durch 2, sieht man, dass gerade die Zeilen mit dem Rest Null die „bösen Häuser" kennzeichnen. Liest man diese Reste von unten nach oben, erhält man mit *10110* die Dualzahl, die im Dezimalsystem *22* ist. Das Verfahren läuft also auf den bekannten Restwertalgorithmus hinaus (vgl. Olivastro, 1995, S. 21 f.).

Hier bietet sich ein kleiner Exkurs zur Multiplikation bei Computern an. Wenn man alle Zahlen aus der obigen Tabelle in Dualzahlen übersetzt ergibt sich das folgende Muster:

Erste Spalte Verdoppelung	Zweite Spalte Halbierung
~~111~~	~~10110~~
1110	1011
11100	101
~~111000~~	~~10~~
1110000	1
10011010	

In der Tat ist es so, dass Computer im Grundsatz die Multiplikation wie die alten Ägypter oder Äthiopier oder einige russische Bauern durchführen, da Verdoppeln und Halbieren im Dualsystem besonders einfach sind. Nicht ohne Grund verfügen moderne Prozessoren über entsprechende Shift-Befehle (vgl. F. L. Bauer, 1994).

Nach diesen Überlegungen sind wir in der Lage, die ägyptische Multiplikation als Programm zu formulieren. Wir verwenden wegen der guten Lesbarkeit dazu die Skript-Sprache PYTHON, deren Programme dem Pseudocode sehr ähnlich sind, sodass man sie einfach in beliebige andere Programmiersprachen übertragen kann. Ein weiterer wichtiger Grund ist die eingebaute Langzahlarithmetik, von der wir im Folgenden noch ausgiebig Gebrauch machen werden. PYTHON läuft auf den gängigen Betriebssystemen und kann kostenfrei unter http://www.python.org/ bezogen werden; für den Einstieg ist das mitgelieferte Tutorial hervorragend geeignet (vgl. auch Arnhold, 2001).

Natürlich kann man RSA mit jeder Programmiersprache implementieren, die über eine Langzahlarithmetik verfügt. Eine Alternative ist der Einsatz von Computer-Algebra-Systemen (CAS), z. B. DERIVE. Näheres dazu wird weiter unten ausgeführt.

Die PYTHON-Funktion lautet folgendermaßen:

```
def multi(x,y):
    'Multiplizieren durch Addieren und Verdoppeln'
    prod = 0
    while y > 0:
        if  y % 2 == 1:        # Zweiter Faktor ist
                               # ungerade
            prod = prod + x    # Ein "gutes" Haus:
                               # Addieren!
            y = y - 1          # Nun ist er wieder
                               # gerade!
        else:
            x = x + x          # Erster Faktor wird
                               # verdoppelt
            y = y / 2          # Zweiter Faktor wird
                               # halbiert
    return prod
```

Wird nun das Verdoppeln durch Quadrieren und das Addieren durch Multiplizieren ersetzt, hat man den angekündigten „Square and Multiply"-Algorithmus:

```
def pot(x,y):
    'Potenzieren durch Quadrieren und Multiplizieren'
     potenz = 1
    while y > 0:
      if y%2 == 1:
          potenz = potenz * x
          y = y - 1
      else:
          y = y / 2
          x = x * x
    return potenz
```

Wenn jetzt noch die Zwischenergebnisse bei jedem Schritt modulo *n* reduziert werden, hat man eine erstaunlich einfache Funktion, mit der man nach dem RSA-Algorithmus Ver- und Entschlüsseln kann:

```
def modpot(x,y,n):
    '''
    Schnelles (modulares) Potenzieren x^y (mod n)
    nach der Methode "square and multiply".
    Bei jedem Schritt wird modulo n reduziert! '''
    pot = 1
    while y > 0:
        if  y % 2 == 1:          # Falls Exponent
                                 # ungerade:
            pot = (pot * x) % n  # Multiplizieren!
            y = y - 1            # Nun ist der
                                 # Exponent wieder
                                 # gerade
        else:
            x = (x * x) % n      # Quadrieren und
            y = y / 2            # Exponent
    return pot                   # halbieren
```

An dieser Stelle soll nicht unerwähnt bleiben, dass CAS-Programme wie DERIVE über eingebaute zahlentheoretische Funktionen verfügen, die u. a. auch das modulare Potenzieren ermöglichen. Die Langzahlarithmetik steht dort ebenfalls zur Verfügung. Der Vorteil des von uns beschriebenen Weges liegt darin, dass der Algorithmus für die Schülerinnen und Schüler keine „Black-Box" bleibt, sondern nachvollzogen werden kann. Das gilt nach unseren Erfahrungen auch für die Sekundarstufe I.

Die Ver- und Entschlüsselung im RSA-System ist mit der Funktion modpot erschöpfend behandelt. Mit der Langzahlarithmetik können realistisch große Zahlen mit hunderten von Stellen verarbeitet werden; Beispiele dazu werden wir noch anführen.

Schwieriger ist das Problem der Schlüsselerzeugung für RSA.

„Der Zahlenteufel" von Hanns Magnus Enzensberger.

Ein Rezept zur Erzeugung von Schlüsseln für RSA

Das Rezept zur Erzeugung eines Schlüsselpaares für RSA lautet (siehe Kasten, S. 48):

1. Wähle zwei (große) Primzahlen p und q.
2. Bilde das Produkt $n = p \cdot q$.
3. Berechne $\varphi(n) = (p - 1) \cdot (q - 1)$.
4. Finde zwei Zahlen d und e, für deren Produkt gilt: $d \cdot e \equiv 1 \bmod \varphi(n)$, d. h. der Rest von $d \cdot e$ bei Division durch $\varphi(n)$ ist 1 (d. h. d und e sind modular invers bezüglich $\varphi(n)$).
5. Vernichte p, q und $\varphi(n)$, denn sie werden zum Ver- und Entschlüsseln nicht mehr benötigt und würden unerwünschte Entschlüsselungen ermöglichen, wenn sie in die falschen Hände geraten.
6. Verschließe d im Tresor.
7. Veröffentliche n und e.

Die meisten Punkte sollten keine Schwierigkeiten bereiten, nur 1. und 4. verdienen eine genauere Betrachtung. Wir betrachten zunächst den Punkt 1. und wenden wir uns daher der Suche nach Primzahlen zu.

Prima Zahlen

Der Zahlenteufel in dem gleichnamigen Buch von Hanns Magnus Enzensberger erscheint dem Jungen Robert zwölfmal im Traum, um ihm die Angst vor der Mathematik zu nehmen. In der dritten Nacht flüstert er ihm ins Ohr (um Herrn Enzensberger nicht unnötig zu ärgern, zitieren wir wie im Original in alter deutscher Rechtschreibung):

> „Du mußt wissen, es gibt da diese hundsgewöhnlichen Zahlen, die sich teilen lassen, und dann gibt es die anderen, bei denen es nicht geht. Die sind mir lieber. Weißt du, warum? Weil sie prima sind. An denen haben sich die Mathematiker schon seit über tausend Jahren die Zähne ausgebissen. Wunderbare Zahlen sind das. Zum Beispiel die Elf oder die Dreizehn oder die Siebzehn."
>
> Robert wunderte sich, denn der Zahlenteufel sah plötzlich ganz verzückt aus, so, als ließe er sich einen Leckerbissen auf der Zunge zergehen. (Enzensberger, 1997, S. 55).

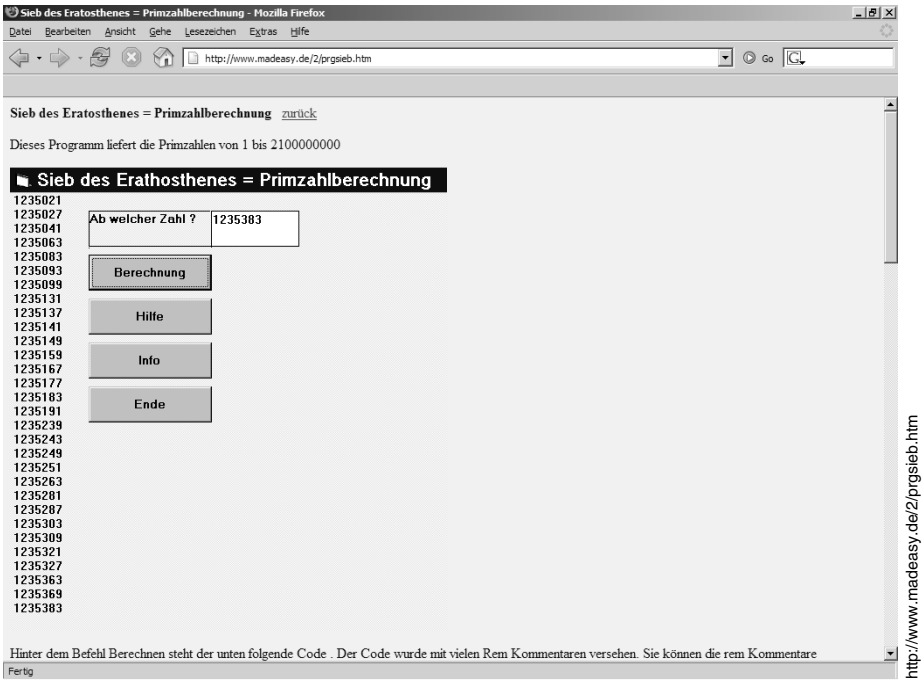
Diese prima Zahlen sind also nichts anderes als die Primzahlen: Alle Zahlen größer gleich zwei, die sich nur durch eins oder durch sich selber teilen lassen, die Null und die Eins sind nach Definition nicht prima.

Über zweitausend Jahre lang wusste man keinen praktischen Nutzen aus dem Wissen über die Primzahlen zu ziehen. Dies änderte sich erst mit dem Aufkommen elektronischer Rechenmaschinen, da die Primzahlen jetzt in der Kryptografie eine zentrale Rolle spielen.

Für die Schlüsselerzeugung bei RSA stellt sich zunächst die Frage, ob es überhaupt genügend Primzahlen gibt, mit anderen Worten: ob die Folge der Primzahlen abbricht oder ob es unendlich viele Primzahlen gibt. Diese Frage wurde bereits von Euklid geklärt: Die Folge der Primzahlen bricht niemals ab (siehe z. B. http://de.wikipedia.org/wiki/Satz_von_Euklid oder Olivastro, 1997, S. 23). Dieser Beweis ist auch für Schülerinnen und Schüler der Sekundarstufe I verständlich. In einem berühmten Aufsatz von Martin Wagenschein wird beschrieben, wie der Satz von den Lernenden selbst gefunden wird („Ein Unterrichtsgespräch zu dem Satz von Euklid über das Nicht-Abbrechen der Primzahlenfolge" in: Wagenschein, 1980, S. 228–236).

Nach dem Fundamentalsatz der Algebra lässt sich jede natürliche Zahl durch Multiplikation von Primzahlen eindeutig darstellen (Zerlegung in Primfaktoren). Aufgrund dieser Eigenschaft nehmen die Primzahlen eine besondere atomare Stellung in der Mathematik ein. Alexander K. Dewdney bezeichnete diese als den Elementen der Chemie weitgehend ähnlich (http://de.wikipedia.org/wiki/Primzahlen).

Das Sieb des Eratosthenes wird im Internet häufig zum Experimentieren angeboten.



http://www.madeasy.de/2/prgsieb.htm

Das Sieb des Eratosthenes

Wie findet man Primzahlen? Um noch einmal den Zahlenteufel zu zitieren: „Der Witz ist nämlich der: Man sieht einer Zahl nicht an, ob sie prima ist oder nicht. Kein Mensch kann das vorher wissen. Man muß es ausprobieren“ (Enzensberger, 1997, S. 57).

Ein sehr altes und effektives Verfahren wird dem Mathematiker Eratosthenes von Kyrene zugeschrieben (siehe z. B. Olivastro, 1997, S. 28 ff., bzw. Enzensberger, 1997, S. 57 ff.). Zuerst werden alle Zahlen 2, 3, 4, … bis zu einem frei wählbaren Maximalwert *S* aufgeschrieben. Die zunächst unmarkierten Zahlen sind potenzielle Primzahlen. Die kleinste unmarkierte Zahl ist immer eine Primzahl. Nachdem eine Primzahl gefunden wurde, werden alle Vielfachen dieser Primzahl als zusammengesetzt markiert. Es genügt dabei, mit dem Quadrat der Primzahl zu beginnen, da alle kleineren Vielfachen bereits markiert sind. Sobald das Quadrat der Primzahl größer als die Schranke *S* ist, sind alle Primzahlen kleiner oder gleich *S* bestimmt: Es sind die nicht markierten Zahlen (http://de.wikipedia.org/wiki/Sieb_des_Eratosthenes).

Es bietet sich an, diesen Algorithmus in ein Programm umzusetzen. Im Internet findet man zahlreiche Animationen, z. B. http://www.faust.fr.bw.schule.de/mhb/eratosib.htm.

Einen Quelltext für das Sieb kann man unter http://de.wikipedia.org/wiki/Primzahltest#Sieb_des_Eratosthenes nachlesen. Die in der *Wikipedia* in der Programmiersprache PASCAL angegebene Implementierung könnte in PYTHON folgendermaßen aussehen:

```
from math import sqrt
N=1000000
Primzahlfeld = range(N)  # Liste der Zahlen von
                         # 0..N-1
                         # Alle Zahlen > 0 werden
                         # zunächst als Primzahlen
                         # angenommen,
Primzahlfeld[1] = 0      # Ausnahme: 1 ist keine
                         # Primzahl!

for i in range(2,int(sqrt(N))): # Man muss nur bis
                                # sqrt(n) sieben!
    if Primzahlfeld[i] <> 0:
        for j in range(i*i,N,i):
            Primzahlfeld[j] = 0 # Hier wird
                                # gesiebt!
```

PYTHON verfügt über einen eingebauten Datentyp Liste, der hier aber nur wie ein Array in PASCAL verwendet wird, wobei das erste Element den Index *0* und das letzte den Indes *N–1* hat. Die PYTHON-Funktion range erzeugt Listen von Zahlen, wobei Start- und Endwert sowie die Schrittweite angegeben werden können. Die for-Schleifen arbeiten dann die in den Listen angegebenen Zahlen ab. Wenn man zum Schluss alle Zahlen von Primzahlfeld ausgeben lässt, die größer als Null sind, erhält man alle Primzahlen kleiner *N*. Mit diesem schlichten Programm lassen sich erstaunlich große Primzahlen erzeugen; so erhält man auf einem nicht mehr ganz neuem Laptop nach zwei Sekunden die Zahlen *999907*, *999917*, *999931*, *999953*, *999959*, *999961*, *999979* und *999983* als letzte Zahlen der Liste, wenn man die Ausgabe der kleineren Primzahlen unterdrückt. Wird die Liste noch länger, muss die Auslagerungsdatei verwendet werden. Damit können größere Primzahlen nicht mehr effektiv berechnet werden. Die Ausgabe aller Primzahlen einer großen Liste dauert sehr lange. Das liegt aber nicht am Algorithmus, sondern an dem Zeitaufwand für I/O-Operationen.

Für eine weitere Runde im RSA-Wettkampf der Schülerinnen und Schüler sind diese Primzahlen aber sicherlich groß genug. Jetzt müssen noch *d* und *e* bestimmt werden; erlaubtes Hilfsmittel ist dabei der PYTHON-Interpreter.

Die nächste Runde: Bestimmung der Schlüsselzahlen d und e

Wir wählen z. B. $p = 999907$ und $q = 999983$. Dann ist $n = p \cdot q = 999890001581$ und $\varphi(n) = (p - 1) \cdot (q - 1) = 999888001692$, wobei die Multiplikationen nicht mehr mit einem normalen Taschenrechner durchgeführt werden können, da dieser auf die Gleitkommadarstellung umschaltet. Hier bietet sich der PYTHON-Interpreter mit eingebauter Langzahlarithmetik an. Man gibt dazu einfach die Zahlen am Prompt ein, die Produkte werden dann nach Eingabe von <RETURN> ausgegeben:

```
>>> p = 999907
>>> q = 999983
>>> p*q
999890001581L
>>> (p-1)*(q-1)
999888001692L
>>>
```

Mit dem nachgestellten L gibt der Interpreter an, dass auf Langzahl-Arithmetik umgeschaltet wurde.

Um die Zahlen *e* und *d* bei großen *p* und *q* zu finden, wird der erweiterte Euklidische Algorithmus (EEA) benötigt, den wir in der nächsten Folge dieser Beitragsreihe vorstellen werden. Eine weitere Schwierigkeit besteht darin, dass *e* und *d* teilerfremd zu $\varphi(n)$ sein müssen (siehe Kasten „Regeln für modulares Rechnen“, S. 49). Auch hier würde der Euklidische Algorithmus helfen, das wird den Schülerinnen und Schülern aber an dieser Stelle noch nicht verraten. Immerhin kann man schon so ablesen, dass *e* und *d* ungerade sein müssen, da das Produkt $e \cdot d$ bei der Division durch $\varphi(n)$ den Rest *1* haben soll.

Für kleinere Zahlen kann man die Schlüsselzahlen – wie im Kasten „Das RSA-Verfahren“ (S. 48) dargestellt – auch durch einfaches Probieren finden. Bei den gewählten Zahlen, die im Vergleich zu den bei RSA real verwendeten Zahlen immer noch lächerlich klein sind, funktioniert diese „Brute-Force“-Methode aber nur bei großem Glück bzw. sehr viel Geduld. Solche fehlgeschlagenen Experimente können sehr lehrreich sein, weil sie ein Gefühl für die Komplexität der verwendeten Algorithmen geben („Brute-Force“ hat lineare Laufzeit, der EEA logarithmische).

Wir kommen daher auf das oben gewählte Zahlenbeispiel später zurück und verwenden zunächst kleinere Primzahlen.

```
>>> p=9091
>>> q=701
>>> p*q
6372791
>>> (p-1)*(q-1)
6363000
```

Das folgende PYTHON-Programm liefert rasch ein Ergebnis:

```
e = 1111111
phi =  6363000
d = 1
while d < phi:
    if e*d % phi == 1:
        print d
        break
    d = d + 2
if d > phi:
    print "Pech gehabt, anderes e versuchen!"
```

Mit *e = 1111111* erhält man *d = 696991*, für *e = 1111112* hat man erwartungsgemäß Pech.

Damit kann man das folgende RSA-Kryptosystem angeben:

▷ öffentlicher Schlüssel: *n = 6372791, e = 1111111*
▷ privater Schlüssel: *d = 696991*

Jede Gruppe erhält wiederum den Auftrag, nach diesem Muster ein RSA-System zu entwickeln, damit eine Botschaft *m* zu verschlüsseln und *n*, *e* sowie die verschlüsselte Botschaft *c* zu veröffentlichen. Bei dem Versuch, das Kryptosystem der anderen Gruppen zu „knacken“, wird deutlich, dass dazu *n* faktorisiert werden muss.

Faktorisieren mit Eratosthenes

Das Sieb des Eratosthenes kann mit einer geringen Modifikation auch zum Faktorisieren verwendet werden. Man muss sich dazu lediglich merken, mit welcher Zahl jeweils zuerst gesiebt wurde. Dies ist nämlich der kleinste nichttriviale Primteiler; bei Primzahlen ist es die Primzahl selbst. Im PYTHON-Programm zum Sieb lautet die modifizierte geschachtelte Schleife:

```
for i in range(2,int(sqrt(N))):
    if Primzahlfeld[i] == i:
        for j in range(i*i,N,i):
            Primzahlfeld[j] = i
```

Für *N* muss eine Zahl gewählt werden, die etwas größer als die zu untersuchende Zahl ist, in unserem Zahlenbeispiel etwa *6400000*. Die Ausgabe von Primzahlfeld[6372791] ist *701*, mit der Division *6372791/701* liefert der Interpreter den zweiten Faktor *9091*. Damit kann das gegnerische Krypto-Team den geheimen Schlüssel *d* wie oben beschrieben ermitteln und den vorgegebenen Geheimtext *c* entschlüsseln.

RSA mit CAS am Beispiel DERIVE

Natürlich kann man an Stelle von PYTHON ein Computer-Algebra-System (CAS) einsetzen, da dieses ebenfalls über eine Langzahlarithmetik verfügt. Wir beschreiben das Vorgehen am Beispiel von DERIVE, das an deutschen Schulen am stärksten verbreitet ist. Alternativ kann z. B. das kostenfreie System KASH verwendet werden, das über http://www.math.tu-berlin.de/~kant/ bezogen werden kann. Auf dieser Seite findet man auch die Materialien zum Krypto-Labor aus der „Langen Nacht der Wissenschaften“ 2005 und 2006 in Berlin, mit denen sich Interessierte an RSA beschäftigen konnten.

Wir hatten weiter oben für *p* und *q* die Zahlen *999907* und *999983* ermittelt. Diese Zahlen sind für ein echtes RSA-System natürlich immer noch viel zu klein, weil dort Primzahlen mit 100 oder mehr Stellen verwendet werden. Für Primzahlen dieser Größe müssen effektivere Primzahltests entwickelt werden, die wir in einer weiteren Folge dieser Artikelserie vorstellen werden. Solche effektiven Tests sind in CAS bereits implementiert, bei DERIVE z. B. die Funktion next_prime, die aber nur als „Black Box“ verwendet werden kann.

Mit DERIVE können auch für das oben angegebene Zahlenbeispiel die Zahlen *d* und *e* gefunden werden. Da diese beiden Zahlen zueinander modular invers bezüglich des Moduls $\varphi(n)$ sein müssen, kann man zur Bestimmung von *d* die Funktion inverse_mod verwenden,

die z. B. für inverse_mod(1234567, 999888001692) $d = 342228226579$ liefert. Für die eigentliche Verschlüsselung eignet sich die ebenfalls eingebaute Funktion zum modularen Potenzieren power_mod(n, d, m), wobei n die Basis, d der Exponent und m der Modul ist.

Der öffentliche Schlüssel lautet in diesem Fall also (n, e) mit $n = 999890001581$ und $e = 1234567$. Die Faktorisierung von n gelingt mit DERIVE und diesen Zahlen in 0,37 Sekunden. Damit wird nochmals deutlich, dass man auch hiermit noch weit von einem sicheren RSA-Kryptosystem entfernt ist.

Eine reizvolle Aufgabe ist es, den schon mehrmals erwähnten Verschlüsselungswettbewerb jetzt mit dem Hilfsmittel DERIVE durchführen zu lassen. Auch dabei wird man an Grenzen stoßen, die durch den Aufwand zur Faktorisierung gegeben sind (Schulz, 1996). Diese Grenzen werden auch durch die Leistungsfähigkeit der Rechner bestimmt, sodass die in dem zitierten Artikel genannten Zahlenwerte deutlich erhöht werden sollten. Aus dem gleichen Grund müssen die für RSA vorgeschlagenen Schlüsselgrößen von Zeit zu Zeit heraufgesetzt werden.

Fazit und Ausblick

Wir haben in diesem Beitrag einen „experimentellen" Zugang zu RSA beschrieben, der die Schülerinnen und Schülern die Probleme beim Einsatz von RSA „am eigenen Leibe" spüren lässt. Zusätzliche Motivation bieten die mit unterschiedlichen Hilfsmitteln durchgeführten RSA-Wettbewerbe.

Bei diesem Zugang mussten wir den RSA-Algorithmus als gegeben voraussetzen. In der nächsten Folge werden wir einen Weg beschreiben, auf dem RSA „nacherfunden" werden kann. Dabei wird der erweiterte Euklidische Algorithmus im Zentrum stehen.

Andere offene Probleme – wie die Korrektheit von RSA, schnelle Primzahltests und die Schwierigkeiten bei der Faktorisierung, die für die Sicherheit von RSA fundamental sind – werden in weiteren Folgen behandelt.

(wird fortgesetzt)

Prof. Dr. Ralph-Hardo Schulz
Freie Universität Berlin
Fachbereich Mathematik und Informatik
Institut für Mathematik II
Arnimallee 3
14195 Berlin

E-Mail: schulz@math.fu-berlin.de

StD Helmut Witten
Fachseminar für Informatik
1. Schulpraktische Seminar
Charlottenburg-Wilmersdorf (S)
Walther-Rathenau-Schule (Gymnasium)
Herbertstraße 4
14193 Berlin

E-Mail: helmut@witten-berlin.de

Wir danken Frau Irmgard Letzner für Hinweise und Materialien zum Einsatz von RSA in der Sekundarstufe I.

Literatur

Arnhold, W.: Lieben Sie PYTHON? In: LOG IN, 21. Jg. (2001), H. 2, S. 18–24.

Bauer, F. L.: Multiplikation und Dualsystem. In: Informatik Spektrum, 17. Jg. (1994), H. 4, S. 245–250.

Enzensberger, H. M.: Der Zahlenteufel – Ein Kopfkissenbuch für alle, die Angst vor der Mathematik haben. München; Wien: Carl Hanser, 1997.

Kippenhahn, R.: Verschlüsselte Botschaften – Geheimschrift, Enigma und Chipkarte. Reinbek bei Hamburg: Rowohlt, 1997.

Olivastro, D.: Das chinesische Dreieck – die kniffligsten mathematischen Rätsel aus 10.000 Jahren. Frankfurt/Main: Zweitausendeins, o. J. (Nachdruck des Buches aus dem Verlag Droemer Knaur, München, 1995).

Schulz, R.-H.: Primzahlen im öffentlichen Chiffrierverfahren. In: mathematik lehren, 11 Jg. (1993), H. 61, S. 56–64.

Schulz, R.-H.: Primfaktorzerlegung – Experimente zum Zeitaufwand. In: LOG IN, 16. Jg. (1996), H. 5/6, S. 22–26.

Singh, S.: Geheime Botschaften – Die Kunst der Verschlüsselung von der Antike bis in die Zeiten des Internet. München; Wien: Hanser, 2000.

Wagenschein, M.: Naturphänomene sehen und verstehen – Genetische Lehrgänge. Stuttgart: Klett, 1980.

Witten, H.; Letzner, I.; Schulz, R.-H.: RSA & Co. in der Schule, Teil 1 – Statistik und Sprache. In: LOG IN, 18. Jg. (1998a), H. 3/4, S. 57–65.

Witten, H.; Letzner, I.; Schulz, R.-H.: RSA & Co. in der Schule, Teil 2 – Von Cäsar über Vigenère zu Friedman. In: LOG IN, 18. Jg. (1998b), H. 5, S. 31–39.

Witten, H.; Letzner, I.; Schulz, R.-H.: RSA & Co. in der Schule, Teil 3 – Flußchiffren, perfekte Sicherheit und Zufall per Computer. In: LOG IN, 19. Jg. (1999), H. 2, S. 50–57.