Public Key Cryptography

Das RSA-Verfahren

Autor: Jochen Ziegenbalg
Email: ziegenbalg.edu@gmail.com
Internet: https://jochen-ziegenbalg.github.io/materialien/

■ Grundbegriffe

▪ Symmetrische und asymmetrische Verschlüsselung; öffentliche Verschlüsselungssysteme (public key cryptography)

Historisch gesehen, waren zunächst *symmetrische* Verschlüsselungsverfahren in Gebrauch. Das sind Verfahren, bei denen die Kenntnis des Verschlüsselungsprinzips gleichbedeutend ist mit der Kenntnis des Entschlüsselungsprinzips (wer verschlüsseln kann, kann mit demselben Schlüssel auch entschlüsseln). Beispiele dafür sind die bereits von Caesar verwandte *Verschiebemethode* (die Kenntnis der Verschiebezahl dient der Ver- und Entschlüsselung) oder die Vigenère-Methode, bei der die Ver- und Entschlüsselung auf der Kenntnis des Schlüsselwortes beruht.

Bei *asymmetrischen* Verfahren hat man getrennte Schlüssel zum *Verschlüsseln* und zum *Entschlüsseln*. Aus der Kenntnis des Verschlüsselungs-Schlüssels können keine Schlüsse über die Entschlüsselung abgeleitet werden.

Die auf dem Prinzip des "öffentlichen Schlüssels" basierenden Verschlüsselungsverfahren (englisch "public key cryptography") stellen eine besonders wichtige Variante der asymmetrischen Verfahren dar. Dabei wird für jeden Teilnehmer ein *Schlüsselpaar* erzeugt: Es besteht aus einem Schlüssel zur Verschlüsseln und einem Schlüssel zum Entschlüsseln.
Oft wird dabei der Verschlüsselungs-Schlüssel (public key) *öffentlich* (z.B. in der Zeitung oder im Internet oder bei einer Schlüsselverwaltungsstelle) bekanntgeben. Der Entschlüsselungs-Schlüssel (private key) wird jedoch stets geheim gehalten.

■ Das Grundprinzip von öffentlichen Verschlüsselungsverfahren

Jeder Teilnehmer T hat ein Paar von Schlüsseln:

- einen *öffentlichen* Schlüssel: $E = E_T$ (E für Encryption) zur Verschlüsselung und
- einen *privaten* Schlüssel: $D = D_T$ (D für Decryption) zum Entschlüsseln

mit den folgenden Eigenschaften:
1. Für jede Nachricht m gilt: $D(E(m)) = m$ und $E(D(m)) = m$.
2. Aus der Kenntnis des öffentlichen Schlüssels E ist der private Schlüssel D (praktisch) nicht zu erschließen.

■ Ein Szenario

Vorbereitung
In einer Gruppe von Kommunikationspartnern besitze jeder Teilnehmer ein solches Paar von Schlüsseln: Öffentlicher Schlüssel E und privater Schlüssel D (für jeden Teilnehmer T fallen $E = E_T$ und $D = D_T$ anders aus). Alle zusammen veröffentlichen nun ihre öffentlichen Schlüssel E_T in einem allgemein zugänglichen Publikationsorgan (Zeitung, Internet, ...) oder übergeben ihn an eine Schlüsselverwaltungsstelle ("Zertifizierungs-Stelle"). Seinen privaten Schlüssel D_T behält jeder Teilnehmer für sich allein.

Versendung und Empfang einer Botschaft
Will A an B die Nachricht m senden, so

- sucht er aus der öffentlich zugänglichen Liste den Schlüssel E_B von B heraus,
- verschlüsselt die Nachricht m mit Hilfe des öffentlichen Schlüssels E_B und
- sendet die verschlüsselte Nachricht $E_B(m)$ an B.

Der Teilnehmer B empfängt die verschlüsselte Nachricht $E_B(m)$ und kann sie mit Hilfe seines (nur ihm bekannten) privaten Schlüssels D_B entschlüsseln: $D_B(E_B(m)) = m$.

Sicherheit
Kein anderer Teilnehmer kann die verschlüsselte Botschaft $E_B(m)$ entschlüsseln, da ihm die Kenntnis des privaten Schlüssels D_B fehlt.

Ein stark vereinfachtes Modell: System von Briefkästen
Dem öffentlichen Schlüssel entspricht der offen auf dem Briefkasten stehende Name des Empfängers; dem privaten Schlüssel entspricht der (physische) Schlüssel des Biefkasteninhabers.
Verschlüsseln: A steckt seine Nachricht in den Schlitz des (geschlossenen) Briefkastens von B.
Entschlüsseln: B hat (als einziger) den Schlüssel für seinen Briefkasten und kann ihn öffnen, um die Nachricht entgegenzunehmen.

■ "Einweg"-Funktionen

(auch *Falltür-Funktionen*, englisch: *one way functions* oder *trapdoor functions* genannt)

Eine umkehrbar eindeutige ("bijektive") Funktion $f : A \to B$ mit $x \to y = f(x)$ heißt *Einweg-Funktion*, wenn der Funktionswert y relativ leicht aus dem Argument x berechnet werden kann, wenn es aber andererseits bei Kenntnis von y nur mit sehr großem ("astronomischem") Aufwand möglich ist, das Argument x zu ermitteln, das zum Funktionswert y gehört.

Ein stark vereinfachtes Beispiel: Die Telephonbuch-Methode (aus der Zeit vor D-Info). Wir legen das Telephonbuch einer großen Stadt (z.B. Berlin, Hamburg, München, ...) zugrunde.
Die Funktion f soll zu gegebenem Namen x die Telephonnummer y ermitteln. Dies ist relativ leicht möglich; man schlage nur im alphabetisch geordneten Telephonbuch nach.
Ist umgekehrt eine Telephonnummer y gegeben, so ist es ungleich schwerer, den zugehörigen Namen zu finden.
Zum Entschlüsseln bräuchte man so etwas wie ein nach Telephonnummern sortiertes Telephonbuch, das im konkreten Fall nur mit sehr großem Aufwand herzustellen ist. In der Praxis wäre die Telephonbuch-Methode natürlich kaum praktikabel, denn für jeden Teilnehmer bräuchte man ein anderes Telephonbuch (öffentlich) und ein dazugehöriges nach Telephonnummern sortiertes (privates) Exemplar des Telephonbuchs.

Eine besonders interessante Methode zur Erzeugung von Einweg-Funktionen ist das von den Mathematikern *Rivest*, *Shamir* und *Adleman* entdeckte Verfahren, das heute zu Ehren der Entdecker als RSA-Verfahren bezeichnet wird.

■ Das RSA-Verfahren (Rivest-Shamir-Adleman)

Das RSA-Verfahren wurde etwa 1977 von den Mathematikern Rivest, Shamir und Adleman entdeckt. Es basiert auf Methoden und Gesetzen der elementaren Zahlentheorie, insbesondere dem Euklidischen Algorithmus und dem (kleinen) Satz von Fermat, sowie auf der Tatsache, daß kein "schneller" Algorithmus zur Primfaktorzerlegung "großer" natürlicher Zahlen bekannt ist. (Ein Zitat aus P. Horster "Kryptologie", BI 1985: Die Faktorisierung einer 200-stelligen Dezimalzahl benötigt mit dem schnellsten zu dem Zeitpunkt bekannten Algorithmus durchschnittlich etwa 4 000 000 000 Jahre.)

Die beim RSA-Verafhren verwendete Falltür-Funktion ist die Funktion p, die je zwei Primzahlen ihr Produkt zuordnet:

$$p : (x, y) \to p(x, y) := x \cdot y$$

Gleichgültig, wie die Zahlen x und y lauten, ihr Produkt $x \cdot y$ ist stets leicht zu berechnen. Wenn jedoch x und y grosse (etwa 100- bis 200-stellige) Primzahlen sind, dann sind die Faktoren x und y nur mit extrem hohem zeitlichen Aufwand aus dem Produkt $p(x, y)$ zu erschliessen.

■ Mathematische Grundlagen

Literaturhinweis
Ziegenbalg J.: Elementare Zahlentheorie - Beispiele, Geschichte, Algorithmen; Verlag Harri Deutsch, Frankfurt am Main, 2002

■ Euklidischer Algorithmus, Vielfachsummendarstellung des GGT

Satz (erweiterter Euklidischer Algorithmus): Zu je zwei natürlichen Zahlen a und b $(b \neq 0)$ gibt es ganze Zahlen x und y mit der Eigenschaft $\text{GGT}(a, b) = a \cdot x + b \cdot y$.

Hinweis: Die Zahlen x und y können mit Hilfe des erweiterten Euklidischen Algorithmus (Berlekamp Algorithmus) gewonnen werden (siehe auch: *Lemma von Bachet*, [Ziegenbalg 2002, Seite 39]).

Folgerung: Sind a und b teilerfremd, d.h. $\text{GGT}(a, b) = 1$, dann gilt $a \cdot x = 1 - b \cdot y$ bzw. $a \cdot x \equiv 1 \pmod{b}$.

In der Sprechweise der Algebra ausgedrückt heisst dies: x ist das *Inverse* von a modulo b. Der Euklidische Algorithmus (in seiner erweiterten Form) liefert also nicht nur den größten gemeinsamen Teiler, sondern auch, im Falle von teilerfremden Ausgangszahlen, die jeweiligen multiplikativen Inversen im Ring der Restklassen modulo b.

Beispiel: $a = 14,\ b = 51$
Dann ist $\text{GGT}(a, b) = 1$.

Mögliche Werte für x und y sind: $x = 11$, und $y = -3$
Probe (mit *Mathematica*):

```
14 * 11 + (-3) * 51
```

```
1
```

```
Mod[14 * 11, 51]
```

```
1
```

M.a.W.: 11 ist das Inverse zu 14 modulo 51.

■ Der Satz von Fermat

Satz (Fermat): Ist p eine Primzahl und a eine zu p teilerfremde natürliche Zahl, so ist $a^{p-1} \equiv 1 \pmod{p}$.

Ein Beispiel:

```
p = 7; a = 5;
```

```
a^(p-1)
```

```
15625
```

```
Mod[a^(p-1), p]
```

```
1
```

```
Quotient[a^(p-1), p]
```

```
2232
```

Probe:

```
2232 * 7 + 1
```

```
15625
```

■ Gewinnung des öffentlichen und privaten Schlüssels beim RSA-Verfahren

Zur Durchführung des RSA-Verfahrens hat jeder einzelne Teilnehmer T folgendermaßen vorzugehen:

1. T ermittelt zwei (sehr) große Primzahlen p und q.
   ("Sehr groß" heißt im Hinblick auf die heutige Computer-Technologie etwa 100- bis 200-stellig im Dezimalsystem.)

2. T berechnet die Zahlen

$$n = p \cdot q \qquad \text{(* RSA-1 *)}$$

und

$$f = (p-1) \cdot (q-1). \qquad \text{(* RSA-2 *)}$$

3. T ermittelt zwei (positive) Zahlen e und d (e für *encryption*, d für *decryption*) mit der Eigenschaft:

$$e \cdot d \equiv 1 \pmod{f}. \qquad \text{(* RSA-3 *)}$$

Dies ist stets möglich, denn:

Für e kann man z.B. eine Primzahl nehmen, die kein Teiler von f ist. Nach dem erweiterten Euklidischen Algorithmus (vgl. *Lemma von Bachet* in [Ziegenbalg 2002, Seite 39]) gibt es Zahlen d und k mit der Eigenschaft:

$$e \cdot d + f \cdot k = 1, \qquad \text{(* RSA-4 *)}$$

d.h. d ist multiplikativ invers zu e modulo f.

Bemerkung 1: Die Zahlen d und k sind nicht eindeutig bestimmt; genauer: Für jede beliebige ganze Zahl v gilt $e \cdot d + f \cdot k = e \cdot d_2 + f \cdot k_2$ mit $d_2 = d + v \cdot f$ und $k_2 = k - v \cdot e$.
Es ist also insbesondere stets möglich, d so zu wählen, daß es positiv ist.

Bemerkung 2: Die Zahlen p, q und somit auch f sind positiv. Wenn weiterhin die Zahlen e und d so gewählt werden, dass sie positiv sind, dann kann (offensichtich, wegen der durch die Formel (* RSA-4 *) gegebenen Relation) nicht k auch noch positiv sein. (D.h., bei positiven Zahlen e und f ist genau eine der Zahlen d oder k positiv und eine negativ.)

Zusammenfassung: T ermittelt (zu den gegebenen Primzahlen p und q) also Zahlen d und k (z.B. mit dem erweiterten Euklidischen Algorithmus) und wählt dabei d so, daß es positiv ist (denn mit d soll später potenziert werden); k ist dann automatisch negativ.

4. Der öffentliche Schlüssel von T besteht aus dem Zahlenpaar n und e; der private Schlüssel besteht aus der Zahl d.

5. T veröffentlicht den öffentlichen Schlüssel (n, e) an geeigneter Stelle (z.B. im Internet, in der Zeitung oder aus Gründen der Unterbindung von Manipulationen bei einer

Zertifizierungsstelle - vgl. die Ausführungen weiter unten zum Stichwort "Schlüsselverwaltung") als seinen öffentlichen Schlüssel und hält seinen privaten Schlüssel d geheim.

6. Die Zahlen p, q, f und k werden nun nicht mehr benötigt und können aus Gründen der Sicherheit gelöscht werden.

■ Versenden und Empfangen einer Nachricht

Teilnehmer A möchte die Nachricht m an Teilnehmer B senden. Die Nachricht m sei eine natürliche Zahl kleiner als n (n wie in (* RSA-1 *)). (Durch eine geeignete "Vor-Codierung" z.B. durch Aufteilung in Blöck läßt sich ggf. jede Nachricht in eine solche Zahl oder in eine Folge solcher Zahlen übersetzen.)

1. A informiert sich über den öffentlichen Schlüssel (n_B, e_B) von B.
   (Im folgenden wird kurz n für n_B und e für e_B geschrieben.)
2. A berechnet die Zahl $c \equiv m^e \pmod{n}$ und sendet die codierte Nachricht c an B.
3. B empfängt die codierte Nachricht c und berechnet mit Hilfe seines privaten Schlüssels $d = d_B$ die Zahl $m_2 \equiv c^d \pmod{n}$; d.h.: m_2 wird berechnet als der Rest bei der Division von c^d durch n. Dieser Rest (m_2) liegt also insbesondere zwischen 0 und $n - 1$.

Behauptung: $m_2 = m$.
Beweis: Es ist $m_2 \equiv c^d \equiv (m^e)^d \equiv m^{e \cdot d} \pmod{n}$

Also ist zu zeigen: $m \equiv m^{e \cdot d} \pmod{n}$.

Hilfssatz 1: Es ist $m \equiv m^{e \cdot d} \pmod{p}$, wobei p die eine der anfangs gewählten Primzahlen ist.
Beweis des Hilfssatzes:

1. Fall: m sei nicht teilerfremd zu p. Da p eine Primzahl ist, muß p also m teilen; d.h.: $m \equiv 0 \pmod{p}$. Erst recht gilt dann $m^{e \cdot d} \equiv 0 \pmod{p}$ und in diesem Fall ist also $m \equiv m^{e \cdot d} \pmod{p}$.

2. Fall: m sei teilerfremd zu p. Nach dem Satz von Fermat gilt dann $m^{p-1} \equiv 1 \pmod{p}$.
Die Zahlen d, e, f und k waren so gewählt worden, dass $e \cdot d + f \cdot k = 1$ gilt; d positiv und k negativ. Also ist $e \cdot d = 1 - k \cdot f$.
Mit $t := -k$ (also $t > 0$) ist dann $e \cdot d = 1 + t \cdot f = 1 + t \cdot (p-1) \cdot (q-1)$.
Also ist

$$m^{e \cdot d} \equiv m^{1 + t \cdot (p-1) \cdot (q-1)} \equiv m^1 \cdot m^{t \cdot (p-1) \cdot (q-1)} \equiv$$
$$m \cdot (m^{(p-1)})^{t \cdot (q-1)} \equiv m \cdot 1^{t \cdot (q-1)} \equiv m \pmod{p}$$

Damit ist Hilfssatz 1 bewiesen.

Hilfssatz 2: Es ist $m \equiv m^{e \cdot d} \pmod{q}$.
Beweis (Symmetrie-Argument): Die Primzahlen p und q sind völlig gleichwertig.

Abschluss des Beweises: Sei $z := m^{e \cdot d} - m$.

Nach den beiden Hilfssätzen gilt: z wird sowohl von p als auch von q geteilt.
Da p und q verschiedene Primzahlen sind, folgt (z.B. nach dem Fundamentalsatz der Zahlentheorie): z wird vom Produkt $p \cdot q$ geteilt. Mit anderen Worten: z wird von n $(n = p \cdot q)$ geteilt; also gilt

$$z \equiv m^{e \cdot d} - m \equiv 0 \pmod{n}.$$

Da m (nach Voraussetzung) und $m^{e \cdot d}$ (durch die Art und Weise der Berechnung - die Zahlen c und m_2 wurden ja modulo n gewählt) zwischen 0 und $n - 1$ liegen und modulo n kongruent sind, müssen sie gleich sein: $m = m^{e \cdot d}$.

■ Sicherheit des Verfahrens

Voraussetzung für die Dekodierung einer von T versandten Nachricht ist die Kenntnis der Zahl d. Diese kann leicht wie oben ermittelt werden, wenn die Ausgangs-Primzahlen p und q bekannt sind. Sie sind es aber nicht, sondern nur ihr Produkt n $(n = p \cdot q)$ ist bekannt. Um an p bzw. q zu gelangen, müsste man die Zahl n in Primfaktoren zerlegen.
Nach dem gegenwärtigen Stand der Mathematik ist jedoch kein "schnelles" Verfahren zur Faktorisierung natürlicher Zahlen bekannt. Die Faktorisierung "großer" natürlicher Zahlen mit einer Stellenzahl von etwa ab 200 Stellen aufwärts, die keine kleinen Primfaktoren besitzen, ist praktisch unmöglich.

Die Einweg-Funktion ist im Falle des RSA-Verfahrens also die Funktion, die zwei Primzahlen ihr Produkt zuordnet. Dies ist leicht möglich. Die Werte der Umkehrfunktion, welche einer natürlichen Zahl, die als das Produkt zweier grosser Primzahlen gegeben ist, ihre Primfaktorzerlegung zuordnet, sind dagegen nur mit sehr hohem Aufwand zu ermitteln.

■ Implementierung des RSA-Verfahrens in Mathematica

■ **Zunächst werden zwei "große" Zufallszahlen generiert, mit deren Hilfe dann zwei "große" Primzahlen erzeugt werden.**

```
r1=Random[Integer, {10^30, 10^40}]
r2=Random[Integer, {10^40, 10^50}]
```

```
877837557002153844494875240782597202033 7
```

```
701164268603355950268779431804403101567073415825 65
```

```
NextPrime[n_] :=   (* erste Primzahl > n *)
  Module[{a = Ceiling[n]+1},
    While[Not[PrimeQ[a]], a=a+1];
    Return[a] ]
```

```
NextPrime[12345678901234567890 1234567890]
NextPrime[98765432109876543210 9876543210]
```

```
123456789012345678901234567907
```

```
987654321098765432109876543229
```

```
p = NextPrime[r1]
```

```
877837557002153844494875240782597202036 1
```

```
q = NextPrime[r2]
```

```
701164268603355950268779431804403101567073415828 97
```

■ **Berechnung der weiteren Hilfsvariablen und Codierungs-Schlüssel**

Aus Demonstrationsgründen werden die Primzahlen p und q im folgenden (nur zur Demonstration) sehr klein gewählt.

```
(* Die Generierung der Parameter *)
p=NextPrime[12345];
q=NextPrime[23456];
n = p*q;
f = (p-1)*(q-1);  (* f = phi(n) ... Euler *)
```

```
(* ParameterAusdrucken *)
Print["p = ", p];
Print["q = ", q];
Print["n = ", n];
Print["f = ", f];
```

```
p = 12347
```

```
q = 23459
```

```
n = 289648273
```

```
f = 289612468
```

Im nächsten Schritt wählen wir e als Primzahl so, dass sie kein Teiler von f ist.

```
e = NextPrime[Random[Integer, {f/3, f/2} ]]
```

```
111120013
```

```
While[Mod[f, e] == 0,
  Print["e = ", e];
  Print["Das sehr unwahrscheinliche
     Ereignis e  teilt f  ist eingetreten. "];
  Print["Der Vorgang wird wiederholt. "];
  e = NextPrime[Random[Integer, {f / 3, f / 2} ]] ];
Print["e = ", e]
```

```
e = 111120013
```

Jetzt ist sichergestellt, dass e kein Teiler von f ist.

■ Der Berlekamp Algorithmus in Mathematica

```
?ExtendedGCD
```

```
ExtendedGCD[n1, n2, ... ] gives the extended
   greatest common divisor of the integers ni. Mehr...
```

```
ExtendedGCD[66, 42]
```

```
{6, {2, -3}}
```

Probe:

```
2 * 66 + (-3) * 42
```

```
6
```

```
BerlekampListe = ExtendedGCD[e, f];
d = First[Last[BerlekampListe]];
Print["d (vorläufig) = ", d];
k =  Last[Last[BerlekampListe]];
Print["k (vorläufig) = ", k];
```

```
d (vorläufig) = -121461607
```

```
k (vorläufig) = 46603019
```

Im folgenden wird sichergestellt, daß d positiv ist:

```
While[d<0, d=d+f; k=k-e];
Print["d (positiv) = ", d];
Print["k = ", k];
Print["Probe: e*d + f*k = ", e*d + f*k]
```

```
d (positiv) = 168150861
```

```
k = -64516994
```

```
Probe: e*d + f*k = 1
```

■ Definition der Verschlüsselungs- und Entschlüsselungs-Funktion (für Argumente, die natürliche Zahlen sind)

```
RSAenc[M_] := PowerMod[M, e, n];
RSAdec[G_] := PowerMod[G, d, n];
```

■ Ein einfaches Beispiel

Ein Test: **m** ist die ursprüngliche Botschaft und **c** die verschlüsselte Botschaft.

```
m=827
```

```
827
```

Verschlüsselung:

```
c = RSAenc[m]
```

```
117182115
```

Entschlüsselung:

```
RSAdec[c]
```

```
827
```

■ Automatisierung der Schlüssel-Generierung

```
SchluesselGenerierung[G_] :=
 (* G: Groessenordnung *)
 (r1 = Random[Integer, {G * 3 / 4, G}];
  r2 = Random[Integer, {10 * G, 10 * G * 5 / 4}];

  p = NextPrime[r1];
  q = NextPrime[r2];

  n = p * q;
  f = (p - 1) * (q - 1);   (* f=phi (n)... Euler *)

  e = NextPrime[Random[Integer, {N[(1 / 2) * f], N[(7 / 8) * f]}]];
  While[Mod[f, e] == 0,
   Print["e = ", e];
   Print["Das sehr unwahrscheinliche Ereignis "
     e teilt f " ist eingetreten. "];
   Print["Der Vorgang wird wiederholt. "];
   e = NextPrime[Random[Integer, {N[(1 / 2) * f], N[(3 / 4) * f]} ]] ];

  BerlekampListe = ExtendedGCD[e, f];
  d = First[Last[BerlekampListe]];
  If[verbose, Print["d (vorläufig) = ", d]];
  k = Last[Last[BerlekampListe]];
  If[verbose, Print["k (vorläufig) = ", k]];
  While[d <= 0, d = d + f; k = k - e];
  If[verbose, Print["d (positiv) = ", d]];
  If[verbose, Print["k = ", k]];
  If[verbose, Print["Probe: e*d + f*k = ", e * d + f * k]];
  If[verbose, Print[ "Überblick über alle Parameterwerte:"]];
  If[verbose, TableForm[Transpose[
     {{"p", "q", "n", "f", "e", "d", "k"}, {p, q, n, f, e, d, k}} ]]] )
```

```
verbose = True;
SchluesselGenerierung[10^30]
```

```
d (vorläufig) =
 1629663631585950944147289604273509037320734380156258034946141

k (vorläufig) =
 -850413604164384917312309306462263002832529312071953878218553

d (positiv) =
 1629663631585950944147289604273509037320734380156258034946141

k = -850413604164384917312309306462263002832529312071953878218553

Probe: e*d + f*k = 1

Überblick über alle Parameterwerte:

p      92973586079258490244167469361 9
q      11954028784748190830135093892551
n      11114089242127196823044381361167937703412795732374261031332069
f      11114089242127196823044381361155053938767254956641684262745900
e      57997076858148664934461651372786275097320108739649623471941 61
d      1629663631585950944147289604273509037320734380156258034946141
k      -850413604164384917312309306462263002832529312071953878218553
```

- **Abrundung: Vor-Codierung von Zeichenketten in Zahlen bzw. Zahl-Blöcke (mit Hilfe der ASCII-Werte der einzelnen Zeichen)**

■ **Falls es einmal schnell gehen muß: einige feste Werte**

```
(* Vorausermittelte Werte: Anfang *)
r1=635964323870172411566991930332;
r2=241876022493534868904579999769019205383 7;
(* daraus mit Hilfe von NextPrime, aber das kann dauern ... *)
p=63596432387017241156699193033 7
q=24187602249353486890457999976901920538 93
n=p*q
f=(p-1)*(q-1)
(* daraus mit Hilfe von NextPrime, aber das kann dauern ... *)
e=36673676417461799
BerlekampListe = ExtendedGCD[e, f];
d = First[Last[BerlekampListe]];
Print["d (vorläufig) = ", d]
k =  Last[Last[BerlekampListe]];
While[d<0, d=d+f; k=k-e]
Print["d (positiv) = ", d]
Print["Probe: e*d + f*k = ", e*d + f*k]
(* Vorausermittelte Werte: Ende *)
```

```
635964323870172411566991930337
```

```
24187602249353486890457999976901920538 93
```

```
153824521105507516573226192835556730093617061732090587995205080565195
 41
```

```
153824521105507516573226192835314854071059930430798990754279362166775
 12
```

```
36673676417461799
```

```
d (vorläufig) =
 145238759503771371316709610923796930202662011942747506604454819882 47
```

```
d (positiv) =
 145238759503771371316709610923796930202662011942747506604454819882 47
```

```
Probe: e*d + f*k = 1
```

- **Noch mal: Falls es einmal schnell gehen muß: einige feste Werte - Parameter-2**

```
p = 29
q = 43
n = p * q
f = (p - 1) * (q - 1)
e = NextPrime[f / 2]
```

```
29
```

```
43
```

```
1247
```

```
1176
```

```
593
```

```
BerlekampListe = ExtendedGCD[e, f]
d = First[Last[BerlekampListe]]
Print["d (vorläufig) = ", d]
k = Last[Last[BerlekampListe]]
While[d < 0, d = d + f; k = k - e]
Print["d (positiv) = ", d]
Print["Probe: e*d + f*k = ", e * d + f * k]
(*Vorausermittelte Werte:Ende*)
```

```
{1, {353, -178}}
```

```
353
```

```
d (vorläufig) = 353
```

```
-178
```

```
d (positiv) = 353
```

```
Probe: e*d + f*k = 1
```

```
k
```

```
-178
```

■ Digitale Signaturen

Ähnlich wichtig wie die Verhinderung von Lauschangriffen ist in Kommunikationsprozessen das Problem der *Autentifizierung*. Der Empfänger einer Nachricht sollte sicher sein können, dass die Nachricht wirklich von demjenigen Kommunikationspartner stammt, der zu sein er in der Nachricht vorgibt. Die Nachricht sollte also vom Absender in verlässlicher Form *unterschrieben* sein. Im Falle elektronischer Kommunikation kommt hierfür natürlich nur eine elektronische (bzw. digitale) Form der Unterschrift (Signatur) in Frage.

Auch hierfür eignet sich die Technik des "öffentlichen Schlüssels", an dessen Grundprinzip im folgenden zunächst noch einmal erinnert sei.

Das Grundprinzip von öffentlichen Verschlüsselungsverfahren (Wiederholung):
Jeder Teilnehmer T hat ein Paar von Schlüsseln:
einen öffentlichen Schlüssel: $E = E_T$ (E für Encryption) zur Verschlüsselung und
einen privaten Schlüssel: $D = D_T$ (D für Decryption) zum Entschlüsseln
mit den folgenden Eigenschaften:

1. Für jede Nachricht m gilt: $D(E(m)) = m$ und $E(D(m)) = m$.
2. Aus der Kenntnis des öffentlichen Schlüssels E ist der private Schlüssel D nicht zu erschließen.

Nehmen wir an, Teilnehmer A möchte eine signierte (aber zunächst noch nicht geheime) Mitteilung m an Teilnehmer B übermitteln. Dies kann folgendermassen geschehen:

- A verschlüsselt m mit Hilfe seines eigenen privaten Schlüssels $m \to m_S := D_A(m)$ und er versendet die "signierte" Mitteilung m_S an B.
- B decodiert die Nachricht m_S durch Anwendung des öffentlichen Schlüssels von A und erhält so wieder die ursprüngliche Nachricht m: $E_A(m_S) = E_A(D_A(m)) = m$.

B kann sich relativ sicher sein, dass die Nachricht von A stammt, da nur A über den privaten Schlüssel D_A verfügt. Niemand anders als A hätte die Nachricht m derart verschlüsseln können, dass sie bei Anwendung des öffentlichen Schlüssels von A wieder zum Klartext wird.

Allerdings war die Botschaft m nicht verschlüsselt. Jeder andere Teilnehmer, dem die Botschaft m_S von A in die Hände fällt, kann sie genau so leicht wie B mit dem öffentlichen Schlüssel von A entziffern.

Dies stellt jedoch kein grundlegendes Problem dar, denn die beiden geschilderten Verfahren (Geheimhaltung und Signatur) können leicht wie folgt kombiniert werden.

Szenario: A möchte B eine geheime, signierte Botschaft m schicken.
Dies kann durch die folgende Vorgehensweise erreicht werden:
1. A "signiert" die Mitteilung m durch Anwendung seines eigenen privaten Schlüssels:

$m \to m_1 = D_A(m)$

2. A verschlüsselt m_1 durch Anwendung des öffentlichen Schlüssels von B:
   $m_1 \to m_2 = E_B(m_1) = E_B(D_A(m))$
   und verschickt m_2 an B.
3. B wendet auf die Nachricht m_2 zunächst seinen eigenen privaten Schlüssel D_B und danach den öffentlichen Schlüssel E_A von A an und erhält insgesamt:
   $E_A(D_B(m_2)) = E_A(D_B(E_B(D_A(m)))) = E_A(D_A(m)) = m.$

Fazit: Die Botschaft ist geheim, da nur B über den privaten Schlüssel D_B verfügt, und sie ist autentisch, da nur A über den privaten Schlüssel D_A verfügt. Beide Schlüssel sind notwendig, und die Botschaft m zu unterzeichnen und zu entschlüsseln.

Bemerkung: A hätte auch in der folgenden Reihenfolge verfahren können: Er verschlüsselt erst die Botschaft m mit dem öffentlichen Schlüssel von B und signiert das Ergebnis dann mit seinem eigenen privaten Schlüssel. Zur Entschlüsselung müsste B dann zunächst den öffentlichen Schlüssel von A und danach seinen eigenen privaten Schlüssel anwenden.

■ Schlüsselverwaltung

Ein praktisches Problem für das Verfahren der *public key cryptography* stellt die Verwaltung der öffentlichen Schlüssel dar. Damit das Verfahren funktioniert, müssen die öffentlichen Schlüssel von einer vertrauenswürdigen "Agentur", einer sog. *Zertifizierungsstelle*, nach strengen Grundsätzen gesammelt und verwaltet werden.

Denn wenn das Verfahren für die Organisation der öffentlichen Schlüssel zu lax gehandhabt wird, kann ein Lauscher L vorgeben, B zu sein und einen falschen öffentlichen Schlüssel $\overline{E}_B$, zu dem er den passenden privaten Schlüssel $\overline{D}_B$ besitzt, in das System der öffentlichen Schlüssel einschleusen. Mit diesem falschen Schlüsselpaar könnte er dann an B gerichtete Nachrichten entschlüsseln (während B selbst das dann nicht mehr kann) oder er könnte "als B" signieren. Dies würde in der Regel nicht über einen sehr langen Zeitraum klappen. Denn B könnte dann an ihn gerichtete Nachrichten nicht mehr entschlüsseln und würde der Zertifizierungsstelle melden, dass mit seinem Schlüssel irgend etwas nicht stimmt; in kritischen Fällen kann aber auch schon eine einzige gegen den Willen von B entschlüsselte Nachricht oder eine einzige gegen seinen Willen falsch signierte Nachricht grossen Schaden anrichten.

■ Literaturhinweise

Aigner M.: Diskrete Mathematik; Vieweg Verlag, Braunschweig 1993
Bauer F.L.: Entzifferte Geheimnisse; Springer Verlag, Berlin 1997 (2. Auflage)
Beutelspacher A.: Kryptologie; Vieweg Verlag, Braunschweig 1987
Beutelspacher A. /Schwenk J. /Wolfenstetter K.-D.: Moderne Verfahren der Kryptographie; Vieweg Verlag, Braunschweig 1995
Brands G.: Verschlüsselungsalgorithmen,Vieweg Verlag,Braunschweig 2002
Buchmann J.: Einführung in die Kryptographie; Springer Verlag, Berlin 1999
DIFF-Deutsches Institut für Fernstudien: Computer im Mathematikunterricht (CM1); Tübingen 1988
DUDEN Informatik (Hrsg.V.Claus/A.Schwill): Dudenverlag Mannheim1993 (2. Auflage)
Ertel W.:Angewandte Kryptographie,Fachbuchverlag Leipzig,München 2001
Graf K.-D.: Informatik – Eine Einführung in Grundlagen und Methoden; Herder Verlag, Freiburg 1981
Grams T.: Codierungsverfahren Bibliographisches Institut (B.I.), Mannheim 1986
Hodges A.: Alan Turing: The Enigma, Walker& Company
Horster P.: Kryptologie; BI-Wissenschaftsverlag, Zürich 1985
Koblitz N.: A Course in Number Theory and Cryptography; Springer Verlag New York 1987
Rechenberg P. / Pomberger G.: Informatik-Handbuch; Carl Hanser Verlag, München 1997
Salomaa A.: Public-Key Cryptography; Springer Verlag, Berlin 1990
Schneier B.: Applied Cryptography; John Wiley& Sons, New York 1996
Schroeder M.R.: Number Theory in Science and Communication; Springer-Verlag ,Berlin Heidelberg 1984
Schulz R.-H.: Codierungstheorie-Eine Einführung; Vieweg Verlag, Braunschweig 1991
Schulz R.-H.(Hrsg.): Mathematische Aspekte der angewandten Informatik; Bibliographisches Institut (B.I.), Mannheim 1994
darin speziell:
Schulz R.-H.: Informations-und Codierungstheorie - eine Einführung (89-127)
Berendt G.: Elemente der Kryptologie (128-146)
Singh S.: The Code Book; Anchor Books, 2000
Sinkov A.: Elementary Cryptanalysis; The Mathematical Association of America 1966
Ziegenbalg J.: Algorithmen – von Hammurapi bis Gödel; Verlag Harri Deutsch, Frankfurt am Main, 2. Auflage 2007
Ziegenbalg J.: Elementare Zahlentheorie Beispiele, Geschichte, Algorithmen; Verlag Harri Deutsch, Frankfurt am Main 2002

Weitere Literaturhinweise unter: https://jochen-ziegenbalg.github.io/materialien

■ Einige Hilfsprogramme