Geheimschriften und Computeralgebra: Das RSA-Verfahren

Prof. Dr. Wolfram Koepf
Universität Kassel

koepf@mathematik.uni-kassel.de

http://www.mathematik.uni-kassel.de/~koepf

Schülerprojektwoche Kryptographie
19. Februar 2004

UNIKASSEL VERSITÄT

Rechnerisches Caesar-Verfahren

- Als Appetithäppchen betrachten wir zunächst eine Implementierung des Caesar-Verfahrens.
- Ersetzt man jeden Buchstaben durch seine Nummer $A \to 0, B \to 1, C \to 2, D \to 3, \ldots, Z \to 25$, so läuft das Ersetzen durch den drittnächsten Buchstaben auf eine Addition hinaus: $E(x) = x + 3$.
- Damit aber auch X, Y und Z richtig ersetzt werden, müssen wir modulo 26 rechnen:

$$E(x) = x + 3 \pmod{26} .$$

Entschlüsselung bei Caesar

- Die Entschlüsselung beim Caesar-Verfahren ist herzlich einfach. Die Verschiebung um e Buchstaben wird durch die Verschlüsselungsfunktion

$$E_e(x) = x + e \pmod{26}$$

beschrieben und hat die Entschlüsselungsfunktion

$$D_e(x) = E_{-e}(x) = x - e \pmod{26}\ .$$

- Vorführung mit *Mathematica*

Was brauchen wir für RSA?

Zur Erinnerung ist hier eine Liste der Ingredienzien, die wir für das RSA-Verfahren benötigen:

- Wir müssen möglichst effizient Potenzen $x^e \bmod m$ berechnen.
- Wir benötigen die Berechnung des modularen Inversen d mit der Eigenschaft $e \cdot d \equiv 1 \bmod m$.
- Außerdem: Mit geeigneten Hilfsfunktionen wollen wir unsere Nachrichten erst in Zahlen umwandeln und diese am Ende wieder zurücktransformieren.

Schnelles Potenzieren

Die modulare Potenz `powermod[a,n,p]`$:= a^n \bmod p$ berechnet man am effizientesten durch Zurückführen auf Exponenten der Größe $\frac{n}{2}$ (Divide-and-Conquer-Algorithmus):

- $a^0 \bmod p := 1$ (Abbruchbedingung)
- $a^n \bmod p := (a^{\frac{n}{2}} \bmod p)^2 \bmod p$ (für gerade n)
- $a^n \bmod p := (a^{n-1} \bmod p) \cdot a \bmod p$ (sonst)

UNIKASSEL VERSITÄT

Modulares Inverses

Als nächstes müssen wir das modulare Inverse finden.

- Hierzu wird – wie behandelt – der erweiterte euklidische Algorithmus genutzt.
- In *Mathematica* wird dies ebenfalls von der Funktion `PowerMod` erledigt:

$$e \cdot d \equiv 1 \bmod p \quad \Leftrightarrow \quad d = \texttt{PowerMod}[\texttt{e}, -1, \texttt{p}]\ .$$

Hilfsfunktionen

Um einen (nicht zu langen) Text in eine ganze Zahl umzuwandeln, wandeln wir

- zuerst jedes Zeichen in seinen (dezimal) maximal dreistelligen ASCII-Code um
- und setzen die ASCII-Codes zu einer langen ganzen Zahl zusammen.

Beispiel: "TEXT"
$\to \{84, 69, 88, 84\} \to \{84.069.088.084\}$

Daten für RSA

Für das RSA-Verfahren benötigen wir

- zwei große Primzahlen $\boldsymbol{p}$ und $\boldsymbol{q}$. Diese erzeugen wir mit `NextPrime`.
- $\boldsymbol{m} := p \cdot q \quad$ sowie $\quad \boldsymbol{\varphi} := (p-1)(q-1)$
- den öffentlichen Schlüssel $\boldsymbol{e}$, der keinen gemeinsamen Teiler mit φ haben darf
- den privaten Schlüssel $\boldsymbol{d}$ mit $e \cdot d \equiv 1 \mod m$.

Rechenvorschrift beim RSA-Verfahren

- Die Verschlüsselung zum öffentlichen Schlüssel e funktioniert gemäß der Vorschrift

$$E_e(x) = x^e \pmod{m} \, .$$

- Die Entschlüsselung zum privaten Schlüssel d (Achtung: Geheimhaltung sicherstellen!) funktioniert gemäß der analogen Vorschrift

$$D_d(x) = x^d \pmod{m} \, .$$