

Praktikumsversuch 1: RSA-Verschlüsselung

Hardwarenahe Programmierung · Wintersemester 2024/25 · Prof. Dr. Peter Gerwinski

Aufgabe: Schreiben Sie ein Programm, das die Verschlüsselung nach Rivest, Shamir und Adleman (RSA) sowie die Schwierigkeiten beim Brechen („Knacken“) der Verschlüsselung demonstriert.

Schreiben Sie ein C-Programm (oder mehrere), das folgendes durchführt:

- **Schlüsselerzeugung**

Bestimmen Sie drei verschiedene Primzahlen p , q und e , wobei e kleiner als $(p-1) \cdot (q-1)$ und teilerfremd zu $(p-1) \cdot (q-1)$ sei. (Dies ist z. B. der Fall, wenn e größer als p und q ist.)

Berechnen Sie $N = p \cdot q$ sowie eine natürliche Zahl d mit der Eigenschaft:

$$(e \cdot d) \% ((p-1) \cdot (q-1)) = 1$$

(„ $x \% y$ “ wird „ x modulo y “ gesprochen und steht für den Rest, der bei Division von x durch y verbleibt.)

N und e sind der *öffentliche Schlüssel*.
 p , q und d sind der *geheime Schlüssel*.

- **Verschlüsselung**

Wählen Sie eine geheime Botschaft m eine Zahl kleiner als N , die Sie verschlüsseln wollen.

m muß teilerfremd zu N sein. (Dies ist der Fall, wenn m weder durch p noch durch q teilbar ist.)

Schreiben Sie ein Programm, das aus m die verschlüsselte Nachricht c berechnet:

$$c = m^e \% N$$

Hinweis:

$$\begin{aligned} m^e \% N &= \underbrace{(m \cdot m \cdot \dots \cdot m)}_{e \text{ Faktoren}} \% N \\ &= \underbrace{\left(\dots \left((m \cdot m) \% N \cdot m \right) \% N \cdot \dots \cdot m \right)}_{e \text{ Faktoren}} \% N \end{aligned}$$

Dies bedeutet: Multiplizieren Sie die Zahl m e -mal mit sich selbst, wobei Sie *nach jeder Multiplikation* modulo N rechnen.

- **Entschlüsselung**

Rekonstruieren Sie aus der verschlüsselten Botschaft c wieder die geheime Botschaft m :

$$m = c^d \% N$$

- **Verschlüsselung brechen**

Rekonstruieren Sie aus der verschlüsselten Botschaft c wieder die geheime Botschaft m , *ohne* den geheimen Schlüssel zu kennen, d. h. Sie kennen nur N und e , nicht jedoch p , q und d .

Hinweis: Sie können z. B. versuchen, N in seine Primfaktoren zu zerlegen. Auf diese Weise können Sie zunächst p und q berechnen und danach d .

- **Rechenzeit vergleichen**

Vergleichen Sie nun die für das Brechen der Verschlüsselung benötigte Rechenzeit mit der Zeit, die das reguläre Ver- und Entschlüsseln dauert. (Auf diesem Schwierigkeitsunterschied beruht die Sicherheit der RSA-Verschlüsselung.)

Hinweis 1: Ein einfacher Weg, die von Ihrem Programm benötigte Rechenzeit zu messen, ist die Verwendung der Funktion `clock()`. Diese gibt zurück, wieviel Rechenzeit seit Programmstart aufgewendet wurde. Der Typ dieses Rückgabewerts ist ein ganzzahliger Typ, `clock_t`, mit dem man rechnen und den man mit `%ld` ausgeben kann. Pro Sekunde wächst der Zähler um `CLOCKS_PER_SEC` Einheiten. Typischerweise hat `CLOCKS_PER_SEC` den Wert `1000000` oder `1000`, die Zeiteinheit ist also eine Mikrosekunde bzw. eine Millisekunde.

Hinweis 2: Die mit der o. a. Methode meßbaren Zeiten sind eigentlich zu ungenau, um damit die sehr kurzen Rechenzeiten erfassen zu können – etwa so, als wollten Sie mit einem Lineal mit Millimetereinteilung die Dicke eines Blatts Papier messen. Beides ist jedoch möglich.

Viel Erfolg!

Stand: 16. Oktober 2024

Copyright © 2014–2023, 2024 Peter Gerwinski
Lizenz: CC BY-SA (Version 4.0) oder GNU GPL (Version 3 oder höher)

Sie können diese Praktikumsunterlagen einschließlich \LaTeX -Quelltext herunterladen unter:
<https://gitlab.cvh-server.de/pgerwinski/hp>