# Eine Klasse für den RSA-Algorithmus

Michael Dienert

7. Mai 2012

## 1 Der RSA-Algorithmus

Der RSA-Algorithmus (RSA steht für die Anfangsbuchstaben von Rivest, Shamir und Adlemann) ist ein *asymmetrisches* Verschlüsselungsverfahren. Das bedeutet, daß zum Verschlüsseln ein anderer Schlüssel verwendet wird wie zum Entschlüsseln.

Die Verwendung von zwei Schlüsseln beseitigt dabei das Problem der Schlüsselübertragung: Das Dokument wird mit dem *öffentlich* zugänglichen Schlüssel des Empfängers *verschlüsselt* und kann dann ausschließlich mit dem zugehörigen *privaten* und geheimen Schlüssel vom Empfänger entschlüsselt werden. Man nennt derartige Verfahren deshalb auch *Public Key* Verfahren.

## 2 Die Verschlüsselungsfunktion

Kern des Algorithmus ist eine Funktion, die den Ordnungswert eines Zeichens mit einem **Verschlüsselungsexponenten** potenziert und anschließend eine **Modulodivision** durchführt. Die Zahl, mit der die Modulodivision durchgeführt wird, nennt man *Hauptmodul*.

Hier ein Beispiel: das Zeichen 'P' soll verschlüsselt werden:

Der Ordnungswert von 'P' im Alfabet [1] ist **16**. Dieser Wert lässt sich ausrechnen, indem man vom ASCII-Code des Zeichens 64 subtrahiert.

Der Verschlüsselungsexponent sei **151**, das Hauptmodul sei **681**. Dann erhält man den Chiffre-Wert von 'P' mit folgender Formel:

$$chiffre = (16^{151}) \,\%\, 681 \tag{1}$$

Das Problem bei dieser Formel ist, daß die Potenz riesige Werte annehmen kann, die auch der Datentyp `long` nicht mehr aufnehmen kann. Nach der Modulodivision bleibt aber nur noch eine Zahl übrig, die kleiner als der Hauptmodul ist.

Die obige Formel lässt sich aber umschreiben:

$$chiffre = (16 \cdot 16 \cdot 16 \ldots \cdot 16) \,\%\, 681 \tag{2}$$

Und das wiederum liefert das gleiche Ergebnis wie:

$$chiffre = (\ldots((16 \,\%\, 681) \cdot 16) \,\%\, 681) \cdot 16) \,\%\, 681) \ldots) \tag{3}$$

Auf diese Weise lässt sich die Chiffreberechnung natürlich sehr leicht programmieren. Die Methode `potMod` aus dem UML-Diagramm Abb. 1 soll genau diese Berechnung durchführen.

## 3 Die Schlüsselerzeugung

Mit folgendem Algorithmus lassen sich die beiden Schlüssel des RSA-Systems erzeugen:

[1] vgl. erweiterte Rechtschreibreform nach Dienert, 2004

- wähle zwei beliebige, große Primzahlen $p$ und $q$ [2].
- berechne das Hauptmodul $n$:
$$n = p \cdot q \tag{4}$$
- berechne das Nebenmodul $m$:
$$m = (p-1) \cdot (q-1) \tag{5}$$
- wähle als Chiffrierexponent eine *beliebige* Zahl $e$, die keine gemeinsamen Teiler mit $m$ hat.
- wähle als Dechiffrierexponent eine beliebige Zahl $d$, für die folgende Gleichung gilt:
$$(d \cdot e) \% \, m = 1 \tag{6}$$

Ein Beispiel mit kleinen Primzahlen:

$$\begin{aligned} p = 5, \;\; q &= 11 \\ n = p \cdot q &= 55 \\ m = (5-1) \cdot (11-1) &= 40 \\ \text{wähle z.B. } \; e &= 3 \\ (3 \cdot d) \% \, 40 \doteq 1 \;\rightsquigarrow \;\; d &= 27 \end{aligned}$$

# 4 Methoden der RSA-Klasse

Die restlichen Methoden des UML-Diagramms sind im Folgenden kurz beschrieben:

**`doRSA`** Diese Methode berechnet für jeden Wert eines Startfelds das Ergebnis der Funktion `potMod()` und schreibt dies in ein Zielfeld.

**`setHauptmodul`** Einfache set-Methode

**`setExponent`** Einfache set-Methode

**`setChiffre`** set-Methode für die Eigenschaft `startFeld`

**`setKlartext`** die Ordnungswerte der Zeichen des übergebenen Strings sollen im Array `startFeld` abgelegt werden.

**`getKlartext2Chiffre`** liefert ein Array mit Chiffre-Werten, die aus dem Feld ( `startFeld` ) mit den Ordnungswerten des Klartexts errechnet werden.

**`getChiffre2Klartext`** liefert einen String, der die Klartextzeichen enthält. Das Chiffre-Array muss zuvor in der Eigenschaft `startFeld` abgelegt sein.

# 5 Aufgaben

1. Schreibe eine RSA-Klasse, die zunächst nur die Eigenschaften aus Abb. 1 sowie die Methoden `setExponent(int)`, `setHauptmodul(int)` und die Verschlüsselungsmethode `potMod(int)` enthält. Um die Klasse testen zu können, soll sie auch eine main-Methode enthalten, die das Beispiel von oben (Kap. 2) berechnet. Hierzu ist die Methode vorübergehend als `public` zu deklarieren.
2. Ergänze die Klasse um die Methode `doRSA()`.
3. vervollständige die Klasse schrittweise mit den set- und get-Methoden für Chiffre und Klartext und teste die Klasse mit den Code-Werten aus der ct'-Geschichte.

[2] damit die Verschlüsselung sicher wird, wählt man die Primzahlen so groß, daß ihr Produkt eine Länge von z.B. 1024 bit hat.

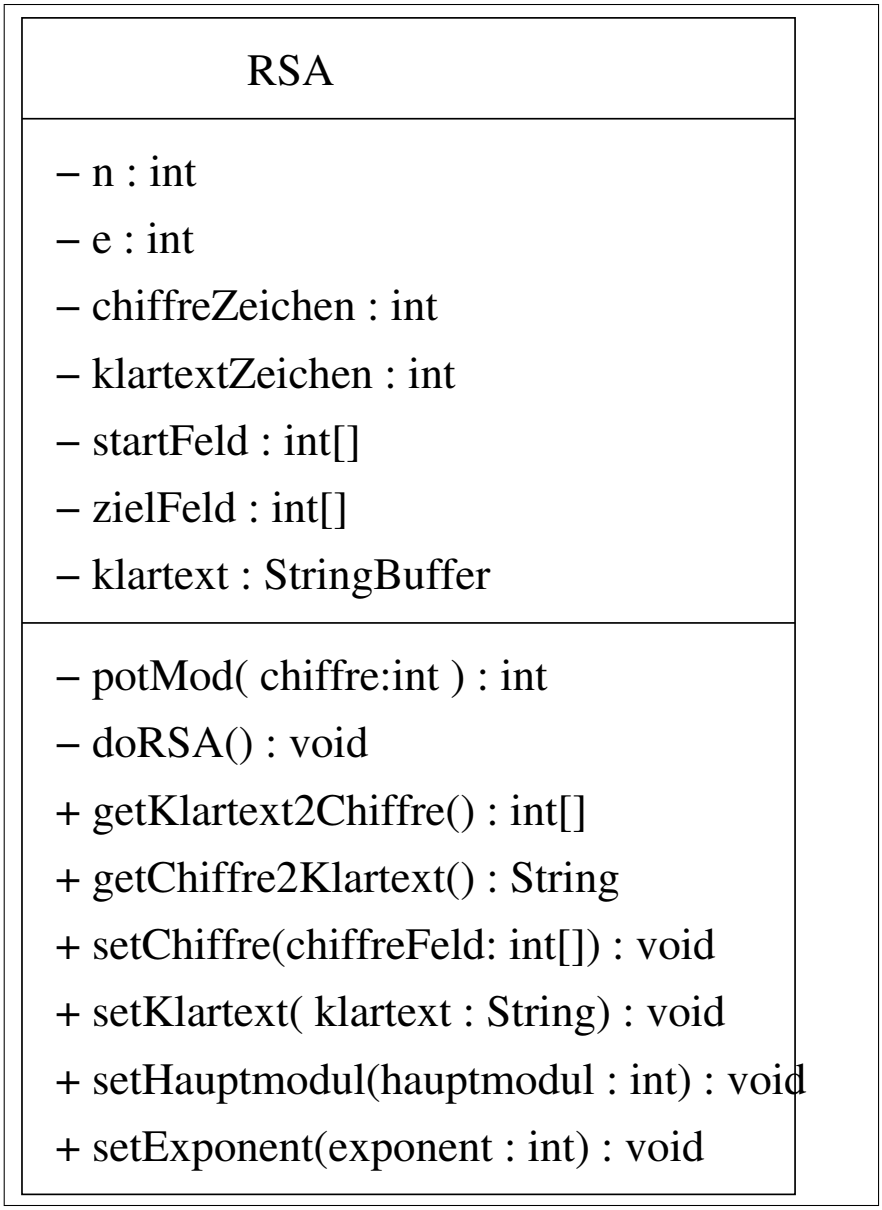


Abbildung 1: Ein Klassendiagramm