

**Übung zur Vorlesung
„Einsatz und Realisierung von Datenbanksystemen“
im Sommersemester 2007**

Richard Kuntschke (richard.kuntschke@in.tum.de)

Lösungen zu Blatt 4

Aufgabe 1

In beiden Fällen stellt das Entziehen des Rechts ein Problem dar:

Fall (a) In Fall (a) darf S_3 das Recht nicht einfach entzogen werden, da es das Recht zweimal erhalten hat – einmal von Subjekt S_1 und unabhängig davon ein andermal von S_2 . Man muss sich also merken, wie oft (und von wem) S_3 das Recht erhalten hat. In 3. wird dann nur eine der beiden Zuweisungen des Rechts (nämlich die von S_2) entzogen. Erst wenn auch die Zuweisung des Rechts von S_1 entzogen wird, verliert S_3 das Recht.

Fall (b) Die Subjekte S_2 und S_3 haben von S_1 das Recht zugewiesen bekommen und dieses ihrerseits weitergegeben. Im vorliegenden Beispiel tritt zudem ein Zyklus der Rechtedelegation zwischen S_2 und S_3 auf. Die Frage ist nun, wie die Aufhebung der Rechtezuweisung von S_1 an S_2 und S_3 erfolgen soll. Hier unterscheidet man zwischen zwei alternativen Vorgehensweisen, nämlich dem kaskadierenden und dem nicht-kaskadierenden Rechteentzug:

Kaskadierendes Rücksetzen Bei kaskadierendem Rechteentzug werden die ursprüngliche Zuweisung (also S_1 an S_2 und ebenso S_1 an S_3) und alle davon abhängigen weiteren Zuweisungen aufgehoben. Dies bedeutet, dass auch die Delegation des Rechts an S_3 durch S_2 aufgehoben wird. Genauso wird die Rechtezuweisung von S_3 an S_2 aufgehoben. Zusätzliche Schwierigkeit bei der algorithmischen Realisierung bereiten zyklische Zuweisungen wie im vorliegenden Beispiel. Der Algorithmus stellt eine rekursive Abarbeitung dar. Für jeden Empfänger von Rechten – S_2 und S_3 im Beispiel – müssen Delegationen rekursiv weiterverfolgt werden, wobei natürlich die in Fall (a) angesprochene Problematik berücksichtigt werden muss. Im Beispiel verlieren folglich S_2 und S_3 das Recht.

Nicht-kaskadierendes Rücksetzen Bei nicht-kaskadierendem Rechteentzug werden nur die unmittelbaren Delegationsstufen aufgehoben. Dies bedeutet, S_2 und S_3 verlieren die von S_1 ausgesprochene Rechtezuweisung. Sie behalten aber weiterhin die gegenseitig zugewiesenen Berechtigungen. Wie bei (a) handelt es sich dabei zwar prinzipiell um dasselbe Privileg, aber auch hier wird bezüglich der „Herkunft“, also von wem das Recht zugewiesen wurde, unterschieden.

Kaskadierender Rechteentzug ist oftmals unerwünscht. Zwar sollen die Berechtigungen für einen Benutzer entzogen werden, nicht jedoch die von ihm oder ihr ausgesprochenen Rechtezuweisungen. Man betrachte zum Beispiel den Fall, dass S_1 Administrator eines DBMS ist. Während der Zeit der Anstellung hat S_1 zahlreichen anderen Benutzern des DBMS (also beispielsweise auch S_2 und S_3) Rechte zugewiesen, die sie zur Erledigung ihrer Arbeiten benötigen.

Gibt S_1 die Administratortätigkeit auf, so sollen zwar S_1 alle Administrator-Berechtigungen entzogen werden, die von S_1 ausgesprochenen Zuweisungen müssen aber weiterhin ihre Gültigkeit bewahren.

Aufgabe 2

Um das Gehalt des Professors / der Professorin zu ermitteln, der / die von allen „C4“-Professoren am meisten verdient, kann man wie folgt vorgehen: Zuerst bestimmt man die Anzahl der „C4“-Professoren und das Gesamtgehalt:

```
select count(*) as AnzahlProfsC4, sum(Gehalt) as SumGehaltC4
from Professoren
where Rang = 'C4';
```

Diese Anfrage erweitert man nun um die zusätzliche Einschränkung auf „C4“-Professoren, die weniger als das *Faktor*-fache des Durchschnittsgehalts verdienen. Die Funktion **avg** steht nicht zur Verfügung, kann aber mittels **sum** und **count** bekanntermaßen einfach umgesetzt werden:

```
select count(*) as AnzahlProfs, sum(Gehalt) as SumGehalt
from Professoren
where Rang = 'C4'
and Gehalt < %Faktor% *
      (select sum(Gehalt) / count(*)
       from Professoren
       where Rang = 'C4');
```

Um nun das gesuchte Gehalt zu bestimmen, muss man den *Faktor* nur schrittweise erhöhen – und zwar so lange, bis $AnzahlProfs = AnzahlProfsC4 - 1$ ist. Die Differenz von $SumGehaltC4$ und $SumGehalt$ ist dann der gesuchte Betrag.

Aufgabe 3

In Abschnitt 12.6.2 des Datenbankbuchs werden die notwendigen Berechnungsschritte des RSA-Verfahrens vorgestellt. Der Algorithmus arbeitet wie folgt:

1. Wähle zwei zufällige große Primzahlen p und q und berechne $r = p \cdot q$.
2. Wähle eine zufällige große Zahl v , die relativ prim zu $(p - 1) \cdot (q - 1)$ ist, also

$$\text{ggT}(v, (p - 1) \cdot (q - 1)) = 1.$$

3. Berechne e , so dass gilt

$$e \cdot v = 1 \text{ mod } (p - 1) \cdot (q - 1).$$

(v, r) bildet den öffentlichen Schlüssel und (e, r) den privaten Schlüssel. Eine Verschlüsselung (**encrypt()**) einer Nachricht B erfolgt dann durch folgende Berechnung:

$$C = B^v \text{ mod } r.$$

Aus C kann die Nachricht durch **decrypt()** rückberechnet werden durch

$$B = C^e \text{ mod } r.$$

Abbildung 1 zeigt eine Realisierung des RSA-Verfahrens in Java – angelehnt an <http://pajhome.org.uk/crypt/rsa/implementation.html>.

Die Bibliotheken von Java erleichtern die Implementierung des RSA-Verfahrens wesentlich und ermöglichen eine (verblüffend kurze – aber vermutlich auch ineffiziente) Beschreibung, wie sie in Listing 1 dargestellt ist:

- Modellierung großer Zahlen (*BigInteger*). Elementare Datentypen wie *short* und *integer* würden nicht ausreichen, um genügend große Zahlen zu repräsentieren, die eine ausreichende Sicherheit für RSA-Verschlüsselung bieten.

Die Klasse *BigInteger* unterstützt dies und bietet zudem folgende Methoden an:

- Berechnung von Primzahlen, bzw. Durchführung von Primzahltests. Der für die Variablen p und q angewendete Konstruktor erzeugt mit einer Wahrscheinlichkeit $> 1 - \frac{1}{2^{100}}$ Primzahlen.
- Berechnung des multiplikativen Inversen e zu $v \pmod{(p-1) \cdot (q-1)}$.
- Neben grundlegenden Rechenoperationen wie Addition, Subtraktion und Multiplikation, stellt *BigInteger* Funktionalität zur Exponentenbildung bereit, die für die Verschlüsselung und Entschlüsselung von Nachrichten essentiell ist.

```
import java.math.BigInteger;
import java.security.SecureRandom;

class Rsa {

    private BigInteger r, v, e;

    public Rsa(int bitlen) {
        SecureRandom rand = new SecureRandom();
        // Erzeugen großer Primzahlen mit bitlen Bit-Stellen
        BigInteger p = new BigInteger(bitlen / 2, 100, rand);
        BigInteger q = new BigInteger(bitlen / 2, 100, rand);
        r = p.multiply(q);
        BigInteger m = (p.subtract(BigInteger.ONE)).multiply(q
            .subtract(BigInteger.ONE));
        // berechne v, relativ prim zu (p-1)(q-1)
        v = new BigInteger("3");
        while (m.gcd(v).intValue() > 1)
            v = v.add(new BigInteger("2"));
        // berechne e, das multiplikativ Inverse zu v (mod (p-1)(q-1))
        e = v.modInverse(m);
    }

    public BigInteger encrypt(BigInteger message) {
        return message.modPow(v, r);
    }

    public BigInteger decrypt(BigInteger message) {
        return message.modPow(e, r);
    }
}
```

```

}
}

```

Listing 1: Java-Implementierung des RSA-Algorithmus

Aufgabe 4

Abbildung 1 zeigt eine UML-Modellierung einer objektorientierten Datenbank für das in Aufgabe 2.6 des Übungsbuchs Datenbanksysteme vorgestellte Zugauskunftssystem.

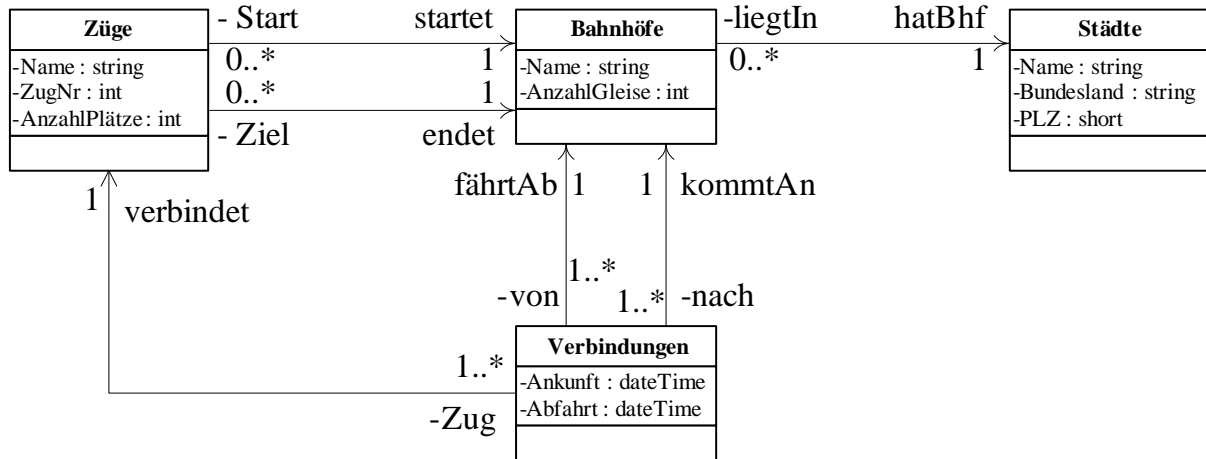


Abbildung 1: UML-Objektmodell für die Beziehungen des Zugauskunftssystems