Praktische Informatik 3 WS 2013/2014

# 1. Übungsblatt

Ausgabe: 22. 10. 2013
Abgabe: 01.11.2013

Berthold Hoffmann <hof>
Christian Maeder <maeder>
Daniel Müller <dmueller>
Sören D. Schulze <sschulze>
Jasper van de Ven <jasper>

## 1.1 Asymmetrische Verschlüsselung (10 Punkte)

Die sogenannten asymmetrischen Verschlüsselungsverfahren gelten – mit hinreichend langen Schlüsseln – nach dem gegenwärtigen Stand der Kunst als sicher. Dabei hat jeder Teilnehmer einen *öffentlichen Schlüssel* $P$, den jeder andere Teilnehmer kennt, und einen *geheimen Schlüssel* $S$, den nur er selber kennt. Eine Nachricht $m$ an den Teilnehmer wird mit dessen öffentlichen Schlüssel $P$ verschlüsselt: $C = encoding_P(m)$. Der Teilnehmer kann die Nachricht dann mit seinem geheimen Schlüssel $S$ entschlüsseln, wenn folgendes gilt:

$$m = decoding_S(encoding_P(m)) \tag{1}$$

Entscheidend für die Sicherheit ist, dass der geheime Schlüssel $S$ nicht zum Absender transportiert werden muss und er aus dem öffentlichen Schlüssel $P$ nur mit sehr großem Aufwand errechnet werden kann.

Das von R.L. Rivest, A. Shamir und L. Adleman am MIT entwickelte *RSA-System* basiert auf zwei großen Primzahlen $p$ und $q$. Daraus wird die *Schlüsselgröße* $k = pq$ bestimmt. Dann wird eine (ungerade) Zahl $e$ zwischen 1 und $n = (p-1)(q-1)$ gewählt so dass $e$ und $n$ teilerfremd sind, sowie eine positive Zahl $d$, so dass gilt: $e \cdot d \bmod n = 1$. Das Paar $P = (e, k)$ wird dann als öffentlicher Schlüssel bekannt gemacht, während das Paar $S = (d, k)$ als geheimer Schlüssel bewahrt wird. Die Verschlüsselungsfunktion für eine Zahl $m$ (die kleiner als $k$ sein muss) und einen öffentlichen Schlüssel $(e, k)$ ist

$$encoding_{(e,k)}(m) = m^e \bmod k,$$

Als Entschlüsselungsfunktion ergibt sich für einen geheimen Schlüssel $(d, k)$ und eine codierte Zahl $c$,

$$decoding_{(d,k)}(c) = c^d \bmod k.$$

Die Verschlüsselungsfunktionen erfüllen die Korrektheitsbedingung (1). Weshalb, kann man nachlesen.[1] Für die Bearbeitung des Blattes ist es aber nicht nötig, weil wir geeignete Schlüsselpaare zusammen mit diesem Blatt zur Verfügung stellen.

In Haskell soll die Verschlüsselung realisiert werden mit zwei Funktionen:

```
type Key = (Integer, Integer)
encoding, decoding :: Key → Integer → Integer
```

Um die Verschlüsselung hinreichend effizient zu gestalten, müssen wir eine schnelle Version der *Exponentiation* implementieren:

```
modExp :: Integer → Integer → Integer → Integer
```

[1] Z. B. unter `http://www.uni-giessen.de/~g013/code/rsa6.pdf`.

Diese reduziert Exponentation auf Quadrierung und nutzt die Eigenschaften, dass $a^{2n} = (a^n)^2$ und $(a^n \bmod p) * (a^m \bmod p) = a^{n+m} \bmod p$.

$$a^n \bmod p = \begin{cases} 1 & \text{wenn } n = 0 \\ (a^{n \operatorname{div} 2} \bmod p)^2 \bmod p & \text{wenn } n \text{ gerade ist} \\ (a(a^{n \operatorname{div} 2} \bmod p)^2) \bmod p & \text{wenn } n \text{ ungerade ist} \end{cases}$$

Nun ist die Verschlüsselung von ganzen Zahlen, auch wenn diese sehr groß werden können (bei den typischerweise verwendeten Schlüssel mehr als 30 Stellen), auf die Dauer nur für Zahlenfetischisten spannend. Wir wollen natürlich Textdateien, Bilder oder zumindest Zeichenketten verschlüsseln.

In einem ersten Schritt entwickeln wir zwei Funktionen:

*encryption* :: *Key* → *String* → [*Integer*]
*decryption* :: *Key* → [*Integer*] → *String*

Sie wandeln jedes Zeichen mit der Funktion *ord*:: *Char* →*Int* in eine Zahl um, und verschlüsseln diese. Zum Entschlüsseln verwenden wir die duale Funktion *chr*:: *Int* → *Char*. Damit ergibt sich als verschlüsselter Text eine Liste von (sehr großen) ganzen Zahlen.

**Tipps für die Implementierung**:

1. Die Funktionen (*Integral* $\alpha$) ⇒ *even*, *odd* :: $\alpha$ → *Bool* überprüfen, ob eine ganze Zahle gerade oder ungerade ist (für *expMod*).
2. Mit den Funktionen

   *toInteger* :: (*Integral* a) ⇒ a → *Integer*
   *fromInteger* :: (*Num* a) ⇒ *Integer* → a

   können dann *Int*-Werte in *Integer* gewandelt werden und zurück.
3. In der vorgegebenen Datei RSA.hs sind geeignete Schlüsselpaare verschiedener Länge vorgegeben.

## 1.2 Nachteile der zeichenweisen Verschlüsselung (4 Punkte)

Ein Nachteil der oben implementierten Verschlüsselung ist, dass die Nachrichten sehr viel länger werden, da jedes Zeichen in eine Zahl zwischen 1 und der (sehr großen) Schlüsselgröße $k$ abgebildet wird.

Nennen Sie einen viel schwerwiegenderen Nachteil, der die Sicherheit der Verschlüsselung betrifft.

## 1.3 Kompakte Verschlüsselung von Texten (6 Punkte)

In einem zweiten Schritt unterteilen wir deshalb die Nachricht in Blöcke zu $n$ Zeichen, wobei dann ein Block von $n$ Zeichen eine Zahl zu der Basis 256 darstellt. Dazu implementieren wir zuerst zwei Funktionen:

*conversionToBase* :: *Int*→ *Integer*→ [*Integer*]
*conversionFromBase* :: *Int*→ [*Integer*]→ *Integer*

Das erste Argument dieser Funktionen ist eine Basis $b$. Die erste Funktion konvertiert in die Darstellung in eine Liste von Ziffern zu dieser Basis, die zweite zurück. Beispielsweise ist

$$\begin{aligned} \mathit{conversionToBase}\ 10\ 1984 &\rightsquigarrow [4,8,9,1] \\ \mathit{conversionFromBase}\ 10\ [4,8,9,1] &\rightsquigarrow 1984 \end{aligned}$$

Mit diesen Funktionen werden jetzt zwei Funktionen implementiert, die eine Zeichenkette als eine Zahl zur Basis 256 aufffassen: [2]

```
stringAsNumber :: String→ Integer
numberAsString :: Integer→ String
```

$$\begin{aligned} \mathit{stringAsNumber}\ \text{"foo"} &\rightsquigarrow 7303014 \\ \mathit{numberAsString}\ 7496034 &\rightsquigarrow \text{"bar"} \end{aligned}$$

Zur Verschlüsselung können Sie folgende Funktion beutzen, die eine Zeichenkette in Teilstücke (*chunks*) gegebener Länge aufspaltet:

```
— Zeichenketten in Teilketten der Laenge n aufspalten
chunks :: Int→ String→ [String]
chunks n s = if null s then [] else (take n s : chunks n (drop n s))
```

Die Länge der Teilstücke muss so gewählt werden, dass die damit zu der Basis 256 dargestellten Zahlen immer kleiner als $k$ sind, also wählen wir als Länge eine Ziffer weniger als $k$ zur Basis 256:

```
length (conversionToBase 256 k) –1
```

Damit wird eine Nachricht wie folgt verschlüsselt:

1. Die zu verschlüsselnde Botschaft wird in chunks der berechneten Länge aufgeteilt.
2. Jeder dieser Zeichenketten wird mittels *stringAsNumber* in einen Integer konvertiert.
3. Jede dieser Zahlen wird verschlüsselt.

Die Entschlüsselung läuft entsprechend umgekehrt:

1. In der Liste von Integer wird zuerst jede Zahl entschlüsselt.
2. In der entschlüsselten Liste wird jede Zahl mit *numberAsString* in eine Zeichenkette konvertiert.
3. Die Listen von Zeichenketten wird zu einer einzigen Zeichenkette zusammengefügt.

[*Die Idee zu dieser Aufgabe stammt von* **Christoph Lüth**.]

Dies ist Fassung 1.1 vom 23. Oktober 2013.

[2] Wir verschlüsseln hier nur 8-Bit-Zeichen, nicht *UniCode*. Für den asiatischen Markt ist die Lösung also nicht geeignet.