Praktische Informatik 3 WS 04/05

1. Übungsblatt

Ausgabe: 01.11.04

Bearbeitungszeit: Zwei Wochen

Christoph Lüth <cxl>
Shi Hui <shi>
Klaus Lüttich <luettich>
Wolfgang Machert <wmachert>
Christian Maeder <maeder>
Hennes Märtins <maertins>
Kai-Florian Richter <richter>
Dennis Walter <dw>
Diedrich Wolter <dwolter>

In diesen Zeiten kann man einfach nicht wachsam genug sein. Terroristen sind überall, und der Große Bruder hört mit. Deshalb beschäftigen wir uns in diesem Übungsblatt mit Verschlüsselungstechniken.

1 *Rhabarbertee* *5 Punkte*

Die erste betrachtete Verschlüsselungstechnik ist nicht vollständig sicher, und erfreut sich besonders in den Kindergärten der Republik großer Beliebtheit. Sie funktioniert ganz einfach: nach jedem Vokal wird der Buchstabe 'b' angehängt, und der Vokal wiederholt. Aus "Rhabarbertee" wird somit "Rhababababarbebertebeebe".

Implementieren Sie zwei Funktionen

```
encrypt :: Char-> String-> String
decrypt :: Char-> String-> String
```

die eine Zeichenkette auf diese Art und Weise ver- und wieder entschlüsseln. Das erste Argument ist der Konsonant, der zwischen die wiederholten Vokale eingefügt wird (also `'b'` oben).

Hinweis: Diese Verschlüsselung ist nicht besonders sicher.

2 *Asymmetrische Verschlüsselung* *15 Punkte*

Im Gegensatz zu dem Verfahren aus der ersten Aufgabe sind die sogenannten asymmetrischen Verschlüsselungsverfahren nach dem gegenwärtigen Stand der Kunst mit hinreichend langen Schlüsseln sicher.

Bei einem asymmetrischen Schlüsselverfahren hat jeder Teilnehmer X einen öffentlichen Schlüssel P_X, den jeder andere Teilnehmer kennt, und einen geheimen Schlüssel S_X, den nur X kennt. Eine Nachricht M von A ("Alice") an B ("Bob") mit dem öffentlichen Schlüssel P_B von B verschlüsselt, $C = P_B(M)$. B kann die Nachricht dann mit seinem geheimen Schlüssel S_B entschlüsseln, wenn folgendes gilt:

$$M = S_B(P_B(M)) \tag{1}$$

Entscheidend für die Sicherheit ist hier, dass aus dem öffentlichen Schlüssel P_X der geheime Schlüssel S_X nicht errechnet werden kann, oder nur mit sehr großem Aufwand.

Das *RSA-System* ist eine Übung in angewandter Algebra (die mathematischen Grundlagen finden sich im Anhang, sind aber nicht Bestandteil der Übung). Ein *Schlüssel* besteht immer

aus zwei Zahlen (k, n) mit $k <= n$, wobei n die Schlüsselgröße ist. Die Verschlüsselungsfunktion für eine Nachricht M und einen öffentlichen Schlüssel (e, n) ist

$$(e, n)(M) = M^e \; mod \; n,$$

und als Entschlüsselungsfunktion ergibt sich für einen privaten Schlüssel (d, n) und eine Nachricht C,

$$(d, n)(C) = C^d \; mod \; n.$$

Die Implementation benutzt als Schlüssel bestimmte Paare aus beliebig großen, ganzen Zahlen (siehe auch Anhang), und besteht aus zwei Funktionen:

```
type Key = (Integer, Integer)

encrypt :: Key-> Integer-> Integer
decrypt :: Key-> Integer-> Integer
```

die eine Nachricht (eine Zahl m) mit einem öffentlichen Schlüssel (e, n) verschlüsselt bzw. einem privaten Schlüssel (c, n) entschlüsselt. Dabei muss gelten, dass $m < n$. Auf der Webseite sind geeignete Schlüsselpaare zu finden.

Um die Verschlüsselung hinreichend effizient zu gestalten, implementieren wir eine schnelle Version der Exponentation:

```
modexp :: Integer-> Integer-> Integer-> Integer
```

Diese reduziert Exponentation auf Quadrierung und nutzt die Tatsache, dass $a^{2n} = a^{n2}$ d.h. für n gerade ist

$$a^n \; mod \; p = (a^{n \div 2} \; mod \; p)^2 \; mod \; p$$

und für n ungerade ist

$$a^n \; mod \; p = (a(a^{n \div 2} \; mod \; p)^2) \; mod \; p$$

wobei $a^0 \; mod \; p = 1$ und $a^1 \; mod \; p = a$. *(5 Punkte)*

Nun ist die Verschlüsselung von ganzen Zahlen, auch wenn diese sehr groß werden können (bei den typischerweise verwendenten 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 wandeln wir dazu jedes Zeichen in eine Zahl um, und verschlüsseln dieses. Damit ergibt sich als verschlüsselter Text eine Liste von (sehr großen) ganzen Zahlen:

```
encryptMsg :: Key -> String-> [Integer]
decryptMsg :: Key-> [Integer]-> String
```

Der Nachteil dieser Methode ist, dass sich die Größe der Nachrichten enorm vergrößert, da jedes Zeichen in einen Zahl zwischen 1 und der (sehr großen) Schlüsselgröße n abgebildet wird. *(5 Punkte)*

In einem zweiten Schritt unterteilen wir deshalb die Nachricht in Blöcke zu k Zeichen, wobei dann ein Block von k Zeichen eine Zahl zu der Basis 256 darstellt.

Dazu implementieren wir zuerst zwei Funktionen:

```
convertToBase :: Integer-> Integer-> [Integer]
convertFromBase :: Integer-> [Integer]-> Integer
```

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

```
> convertToBase 10 1984
[4,8,9,1]
>  convertFromBase 10 [4,8,9,1]
1984
```

Mit diesen Funktionen werden jetzt zwei Funktionen implementiert, die eine Zeichenkette als eine Zahl zur Basis 256 auffasst:

```
stringAsNumber :: String-> Integer
numberAsString :: Integer-> String

> stringAsNumber "foo"
7303014
> numberAsString 7496034
"bar"
```

Zur Verschlüsselung wird jetzt noch eine Funktion benötigt, die einen String in Teilstücke (*chunks*) gegebener Länge aufspaltet:

```
chunks :: Int-> String-> [String]

> chunks 4 "open the box!"
["open"," the"," box","!"]
```

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

```
length (convertToBase 256 n) -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 jeder entschlüsselt;
2. in der entschlüsselten Liste wird jeder `Integer` mittels `numberAsString` zu einer Zeichenkette konvertiert;
3. die Listen von Zeichenketten wird zu einer einzigen Zeichenkette zusammengefügt.

(*5 Punkte*)

Anhang: *Mathematische Grundlagen des RSA-Systems.*

Das *RSA-System* benutzt für die Verschlüsselung multiplikative Restgruppenkörper, also Z_n^* für $n \in \mathbb{N}$. $\phi(n)$ ist die Größe von Z_n^* und berechnet sich für Primzahlen p und q als $\phi(pq) = (p-1)(q-1)$. Zur Berechnung eines Schlüsselpaares (e, n) und (d, n) gehen wir wie folgt vor:

1. Gegeben seien zwei (große) Primzahlen p und q,
2. Dann ist $n = pq$, und $\phi(n) = (p-1)(q-1)$.
3. Als öffentlichen Schlüssel wählen wir eine (kleine) ungerade Zahl e so dass e und $\phi(n)$ teilerfremd sind.
4. Der geheime Schlüssel d ist jetzt das multiplikative Inverse von e module $\phi(n)$. Dieses existiert immer und ist eindeutig bestimmt, weil e und $\phi(n)$ teilerfremd sind.
5. Als Verschlüsselungsfunktion ergibt sich für eine Nachricht M und einen Schlüssel (e, n),
$$(e, n)(M) = M^e \; mod \; n$$
und als Entschlüsselungsfunktion ergibt sich für (d, n) und eine Nachricht C,
$$(d, n)(C) = C^d \; mod \; n$$

Um zu zeigen, dass RSA korrekt ist, müssen wir (1) zeigen, also

$$S(P(M)) = M^{ed} \; mod \; n = M \tag{2}$$

Nun sind e und d multiplikative Inverse bezüglch $\phi(n) = (p-1)(q-1)$ sind, also $ed = 1 + k(p-1)(q-1)$, und nach Fermat's Theorem gilt für jede Primzahl p, $a^{p-1} \equiv_p 1$, damit:

$$\begin{aligned} M^{ed} &\equiv_p M(M^{p-1})^{k(q-1)} \\ &\equiv_p M(1)^{k(q-1)} \\ &\equiv_p M \end{aligned}$$

Damit ist $M^{ed} \equiv_p M$ und $M^{ed} \equiv_q M$, also $M^{ed} \equiv_n M$ (chinesischer Restwertesatz), und daraus folgt (2).

Dies ist Version 1.8 vom 2004/11/15 13:50:12.