Technische Universität München
Institut für Informatik
Lehrstuhl für Effiziente Algorithmen
Prof. Dr. Ernst W. Mayr
Dmytro Chibisov
Hanjo Täubig

WS 2006/07
Aufgabenblatt 6
27. Oktober 2006

Praktikum Algorithmen-Entwurf

(Abgabetermin: Montag, den 4.12.2006, 14.^{00} Uhr)

Aufgabe 1 Randomisierter Primzahltest

Implementieren Sie den im Skript beschriebenen Algorithmus zur randomisierten Erkennung von Primzahlen. Ihr Programm sollte folgende Benutzer-Schnittstelle zur Verfügung stellen:

- Aufruf mit einem Kommandozeilen-Parameter, der die Anzahl der gewünschten Wiederholungen des Tests angibt.
- Die zu testende Primzahl wird von der Standard-Eingabe eingelesen (als zusammenhängender String von Ziffern). Führende Leerzeichen sollten hierbei ignoriert werden. Ferner wird angenommen, daß die Eingabe weniger als 1000 Stellen besitzt.
- Auf der Standard-Ausgabe wird `yes` oder `no` ausgegeben, je nachdem, ob die eingelesene Zahl eine Primzahl ist oder nicht.

Implementieren Sie den Primzahltest als eigenständige Funktion und nicht in `main()`, da Sie diese Funktion in der nächsten Aufgabe wiederverwenden werden.
Als Eingabe können Sie die Dateien `p10.dat` bis `p500.dat`, sowie `np10.dat` bis `np500.dat` verwenden. Hierbei steht `p` für Primzahl (Die Bedeutung von `np` stellen wir als Übungsaufgabe). Die Zahl danach gibt an, wie viele Stellen die Binär-Darstellung der Zahl ungefähr besitzt. Die Dateien `cm1.dat` und `cm2.dat` enthalten Carmichael-Zahlen. Ihr Programm rufen Sie dann beispielsweise so auf:

```
primetest 10 < p300
```

Aufgabe 2 RSA-Verfahren

Implementieren Sie das RSA-Verfahren zum Verschlüsseln und digitalen Signieren von Files. Die Funktionalität soll hierbei auf zwei eigenständige Programme verteilt werden.

a) Das Programm `rsainit` zur Initialisierungsphase soll folgende Kommandozeilen-Schnittstelle realisieren:

`rsainit <Anzahl Bits von` p`> <Anzahl Bits von` q`> <Filename>`

Der Aufruf `rsainit 10 20 foo` soll die Dateien `foo.public`, `foo.private` und `foo.log` erzeugen. `foo.public` und `foo.private` bestehen aus zwei Zeilen, wobei die erste Zeile die Zahl n und die zweite Zeile die Zahl e bzw. d enthalten soll. `foo.log` besteht aus vier Zeilen, welche die Zahlen n, $\lfloor \log_2(n) \rfloor$, p und q enthalten.

Beim randomisierten Primzahltest muß festgelegt werden, wie viele Wiederholungen durchgeführt werden sollen. Bitte verwenden Sie bei Ihrer Implementierung mindestens zehn Wiederholungen. Dadurch wird eine für unsere Zwecke akzeptable Fehler-Wahrscheinlichkeit von unter 0.1% erreicht.

b) Das Programm `rsamain` zur Verschlüsselung bzw. digitalen Signatur erhält folgende Kommandozeilen-Schnittstelle:

`rsamain <Schlüssel-Datei>`

Als Schlüssel-Datei gibt man die `.public`-Datei an, wenn man verschlüsseln will bzw. die `.private`-Datei zum Entschlüsseln. Zum Signieren werden die Rollen von `.public`- und `.private`-Datei vertauscht.

`rsamain` soll von der Standard-Eingabe lesen und auf die Standard-Ausgabe schreiben. Hierbei besteht jede Zeile aus einem RSA-Block, der getrennt zu behandeln ist. Das Ergebnis der Ver- bzw. Entschlüsselung ist wiederum Zeile für Zeile auszugeben (Das File-Format vor und nach der Behandlung mit `rsamain` bleibt also gleich).

Die beiden Perl-Skripten `numencode` und `numdecode` dienen dazu, daß Sie mit `rsamain` auch Text-Dateien ver- und entschlüsseln können[1]. Durch

```
numencode 100 foo.txt > numfoo.txt
```

wird die Datei `foo.txt` in eine Zahl-Darstellung konvertiert, die als Eingabe für `rsamain` verwendet werden kann, und diese in der Datei `numfoo.txt` gespeichert. Das erste Argument steht hierbei für die maximale Länge von Bits, die ein einzelner Block (entspricht einer Zeile) umfassen darf. Man muß hier eine Zahl angeben, die kleiner ist als der Wert $\lfloor \log_2(n) \rfloor$, der von `rsainit` ausgegeben wurde.

[1] In Zukunft steht also Ihrer hoch-geheimen Korrespondenz nichts mehr im Wege…

Mit

```
numdecode 100 numfoo.txt > foo.txt
```

wird die Zahl-Darstellung wieder zurück-transformiert in die ursprüngliche Datei `foo.txt`. Beachten Sie, daß beim Ausführen der Perl-Skripten der in den Skripten angegebene Pfad zum Interpreter `perl` stimmen muß. Diesen Pfad müssen Sie anpassen, wenn Sie die Skripten beispielsweise unter Linux verwenden möchten.

Wenn `foo.txt` den Inhalt

```
Yet another stupid demo.
```

besitzt, so sieht `numfoo.txt` folgendermaßen aus:

```
089101116032097110111116104101
114032115116117112105100032100
101109111046010032032032032032
```

Solche Dateien sollte Ihr Programm Zeile für Zeile ver- und entschlüsseln können.

Hinweis

Versuchen Sie, möglichst viele Teile des Programms zum Primzahltest wiederzuverwenden. Die Datei `cryexa.cry` enthält einen verschlüsselten Text, den Sie zum Testen Ihres Programmes verwenden können. Wenn Sie die Datei mit `prakkey.public` entschlüsseln und `numdecode 550` auf das Ergebnis anwenden, sollte ein mehr oder minder sinnvoller Klartext zum Vorschein kommen.