Belegaufgabe 3

# RSA Chiffre

Stefan Schumacher
(162659) <stefan@net-tex.de>

19. April 2003

## Inhaltsverzeichnis



## 1 RSA Verfahren

Aufgabenstellung ist die Implementierung des bekannten kryptographischen RSA Verfahrens, welches nach seinen Erfindern Rivest, Shamir und Adleman benannt ist. Durch das entwickelte Prinzip des öffentlichen und privaten Schlüssels ist es möglich geworden verschlüsselte Daten und öffentlichen Schlüssel über ungesicherte Kanäle auszutauschen und so starke kryptographische Verfahren quasi für Jederman zugänglich zu machen. RSA wird unter anderem in `OpenSSH`, `GnuPG` oder dem `NetBSD-cgd` verwendet und ist offiziell in `RFC3447`[1] definiert.

Grundlage des Verfahrens ist das Erzeugen eines Schlüsselpaares welches dann zur Verschlüsselung der Daten verwendet wird. Erzeugt werden die Schlüssel in dem die zufälligen Primzahlen $p$ und $q$ generiert werden. Aus diesen Primzahlen wird die Modulzahl $modulus$ gemäß Vorschrift $modulus = p * q$ sowie $phi$ mit $phi = (p-1) * (q-1)$ erzeugt. Danach generiert man eine große (empfohlen werden vom US DoD mindestens 1024 Bit) Primzahl $e$ welche als privater Schlüssel genutzt wird und daher geheim zu

[1] ftp://ftp.rfc-editor.org/in-notes/rfc3447.txt

halten ist. Abschließend wird der öffentliche Schlüssel $d$ mittels Auflösen der Gleichung $(e*d) mod\ phi\ =\ 1$ nach $d$ erzeugt. Da mit der Kenntnis von $p$ und $q$ die Zahlen $modulus$ und $phi$ berechnet werden können ist es somit auch möglich den privaten Schlüssel zu generieren, daher ist es notwendig diese Zahlen entsprechend zu schützen bzw. nach Erzeugung des Systems zu vernichten.
Die Verschlüsselung eines Textes erfolgt mittels

$$Chiffre\ =\ (Klartext^{\ öffentlicher\ Schlüssel}) mod\ modulus$$

Die Entschlüsselung vollzieht sich also folgendermaßen:

$$entschlüsselt\ =\ (Chiffre^{\ privater\ Schlüssel}) mod\ modulus$$

Die Implementierung des RSA Algorithmus ermöglicht es eine gewünschte Bitlänge zur Erzeugung von $p$ und $q$ zu übergeben. Dies ist wünschenswert, da nur eine entsprechend hohe Bitlänge (mindestens 1024 Bit, besser noch 2048) die Sicherheit des Schlüsselsystems ermöglicht, denn der Aufwand zur Zerlegung von $phi$ in die entsprechenden Primfaktoren ist zu hoch.

## 2 Realisierung

Java bietet mit der Klasse `java.math.BigInteger` eine Klasse die alle gewünschten mathematischen Funktionen bereitstellt. So ist es möglich die Zufallsprime mittels
`BigInteger p = BigInteger.probablePrime(bitLength, new Random());`
zu erzeugen und die Berechnung des öffentlichen Schlüssels über die Funktion
`BigInteger.modInverse` zu realisieren.
Die Ver- und Entschlüsselung kann mit `modPow(exponent,modulus)` implementiert werden. Hierzu ist es allerdings erforderlich den Eingabetext erst nach Integer (ASCII Code) und dann nach BigInteger zu casten und nach der Chiffrierung buchstabenweise in ein Array zu setzen das danach in einer Schleife wieder zu einem String konvertiert wird. In der Entschlüsselungsfunktion wird das selbe Verfahren, nur umgekehrt, verwendet. Die GUI lässt sich aus den Funktionen der `java.awt.*` komfortabel Ableiten, allerdings ist es notwendig dies als Applikation zu gestalten, da Java-Applets aus Sicherheitsgründen das Schreiben in Dateien nicht erlauben.

## 3 Nutzerbeschreibung

Der Nutzer kann in der grapischen Oberfläche ein Schlüsselpaar erzeugen lassen, dazu sind weitere Eingaben nicht notwendig, da $p$ und $q$ zufällig erzeugt werden. Nach Erzeugung des Schlüsselpaares ist es möglich dieses als einfache ASCII Zeichenketten in eine Datei zu schreiben, sollte diese Datei schon existieren wird sie gelöscht und mit dem frisch generierten Schlüsselpaar neu erzeugt.

Danach ist es möglich in einem Eingabefeld einen beliebigen Text einzugeben der verschlüsselt und wieder entschlüsselt in der Oberfläche ausgegeben wird.

## 4 Klassenstruktur

Wie gefordert wurde die Oberfläche in einer eigenen Klasse namens `MyRSAGUI` implementiert. Die GUI verwendet zur Eingabe ein TextField, zur Ausgabe Labels und Buttons zur Erzeugung der Bedienoberfläche und Steuerung der Applikation.
Die `MyRSA`-Klasse implementiert die nötigen Methoden um das Schlüsselpaar zu erzeugen und Daten zu verschlüsseln und zu entschlüsseln. Diese sind unter anderem:
`erzeugeHilfszahlen()` die $p$, $q$, *modulus* und *phi* generiert und zurückliefert, dabei wird sichergestellt das $p \neq q$ gilt, da es sonst zu fehlerhaften Schlüsseln kommen kann.
`oeffentlicherSchluessel()` returned per Zufallsgenerierung den öffentlichen Schlüssel in gewünschter Bitlänge
`privatSchluessel()` berechnet aus $e$ und *phi* den entsprechenden privaten Schlüssel, welcher ebenfalls zurückgegeben wird
`verschluessele()` liest einen String ein und casted die Zeichen über int (ASCII-Code) nach BigInteger um sie anschließend per Modulopotenzierung zu verschlüsseln und wieder in ein Array zusammenzusetzen.
`entschluessele()` funktioniert prinzipiell wie `verschluessele()`, entschlüsselt allerdings den übergebenen String
`speichereDatei(String name)` dient dazu den öffentlichen und privaten Schlüssel sowie *modulus* in eine Datei zu schreiben. Dazu werden die entsprechenden BigInteger aneinandergehängt, mittels Leerzeichen getrennt und in eine Datei geschrieben. Dazu werden die Methoden aus java.io.* verwendet um einen Stream in eine Datei zu schreiben. Sollte die Datei mit dem gewählten Namen schon existieren wird sie gelöscht und mit dem aktuellen Schlüsseln neu geschrieben.