Algorithmische Anwendungen

RSA – Verschlüsselung

Projekt – Dokumentation

26.01.2006

Gruppe F_pink_Ala0506

Martin Zipzer Mat.nr.:11017809

Wassilios Argiridis Mat.nr.:11018827

Inhaltsverzeichnis

1. Aufgabenstellung

Erstellung eines Java Programms in Projektarbeit zur Lösung eines Algorithmischen Problems mit folgenden Anforderungen.

- Das Thema darf noch nicht in einem anderen Kurs während Ihres Studiums behandelt worden sein.
- Führen Sie zu dem ausgewählten Thema eine intensive Literaturrecherche durch. Zu vielen Themen gibt es Primärliteratur. In der Regel handelt es sich dabei um einen Beitrag in einer Fachzeitschrift. Diese Artikel müssen Sie sich unbedingt z.B. per Fernleihe in der Bibliothek besorgen und im Praktikum vorlegen.
- Recherchieren Sie auch in Sekundärliteratur, z.B. auch im Internet.
- Stellen Sie das Thema und die Problemstellung detailliert im Praktikum und in der schriftlichen Ausarbeitung vor.
- Das von Ihnen ausgesuchte Thema soll einen Bezug zu praktischen Anwendungen haben. Beispiel: Themengebiet Lineare Programmierung, Thema Simplex Algorithmus.
- Die Idee des Algorithmus muss ausführlich erklärt werden. Ein kommentierter Programmcode reicht nicht aus.
- Implementieren Sie den Algorithmus in Java.
- Schreiben Sie ein Anwendungsprogramm in Java, das den Algorithmus benutzt, um praktische Probleme damit zu lösen.
- Für das Anwendungsprogramm ist eine ordentliche GUI zu erstellen.
- Stellen Sie das komplette Programm als lauffähiges jar-File zur Verfügung, so dass bei der Präsentation im Praktikum nur dieses jar-File auf einem beliebigen Rechner gestartet werden muss.
- Untersuchen Sie das Laufzeitverhalten des Algorithmus. Führen Sie dazu eine asymptotische und eine experimentelle Laufzeitanalyse durch. Dokumentieren Sie die Bedingungen der Analysen und die Ergebnisse.
- Zu allen genannten Punkten ist eine ausführliche Dokumentation im PDF-Format zu erstellen und zu jedem Praktikumstermin mitzubringen.
- Am Ende des Praktikums muss jedes Team eine vollständige Dokumentation im PDF-Format und ein lauffähiges jar-File abgeben. Die Dokumentationen aller Teams werden auf meiner Homepage veröffentlicht.

2. RSA – Algorithmus

2.1. Geschichte

Der RSA - Algorithmus wurde 1983 von Ronald Rives, Adi Shamir und Leonard Adleman am Massachusetts Institute of Technology (MIT) als Weiterentwicklung des Public Key Verfahren entwickelt. Mit diesen Algorithmus sollte es möglich sein, Texte zu ver- und entschlüsseln oder auch nur zu signieren. Zuvor veröffentlichten sie 1978 den Artikel „*A method for obtaining Digital Signatures and Public Key Cryptosystems"* in dem sie den Existenzbeweis derartiger Funktionen erbrachten.

Zu Beginn wollten die drei eigentlich beweisen, dass es solche Funktion gar nicht gibt. Bei ihrer Arbeit stießen sie jedoch auf die Zahlentheorie von Leonhard Euler (1707 - 1783) und benutzten diese um das Problem der asymmetrischen Kryptologie zu lösen.

2.2. Theoretische Grundlagen

Der RSA – Algorithmus ist ein asymmetrischer Verschlüsselungsalgorithmus mit einer so genannten „Trapdoor" - Eigenschaft. Die Asymmetrie entsteht durch die verschiedenen Schlüssel zur Ver- und Entschlüsselung. So wird der so genannte „Public Key" jedem möglichen Kommunikationspartner mitgeteilt, damit dieser dann Nachrichten mit diesem Schlüssel verschlüsseln kann. Der so genannte „Private Key" wird geheim gehalten und vom Empfänger zur Entschlüsselung gebraucht.
Die Sicherheit des Algorithmus beruht auf seiner „Trapdoor" – Eigenschaft. Diese Eigenschaft sagt aus, das der Algorithmus recht schnell und unkompliziert in eine Richtung funktioniert, der Ablauf in die entgegengesetzte Richtung überhaupt nicht oder nur mit immensem Zeit- und Rechenaufwand möglich ist.

Die Sicherheit des RSA – Algorithmus beruht auf der Schwierigkeit, eine sehr große Zahl n in ihre Primfaktoren zu zerlegen. Man kann relativ schnell überprüfen ob eine gegebene Zahl prim ist (kleiner Fermatscher Satz, Sieb des Eratosthenes) oder nicht. Von einer nicht-Primzahl n die Primfaktorenzerlegung zu bestimmen ist mit heute bekannten Mathematischen Methoden nicht ohne weiteres möglich. Man kann zwar alle möglichen Faktoren durchprobieren, was aber dazu führt, das man den gesamten Bereich von $1...\sqrt{n}$ in Betracht ziehen muss. Die in der Praxis verwendeten Zahlen sind aber so groß, das dies in annehmbarer Zeit nicht möglich ist.

2.3.Ablauf des Algorithmus

2.3.1. Primzahlerzeugung

Der erste Schritt der RSA – Algorithmus ist die Erzeugung zweier möglichst großer Primzahlen p und q. Dazu erzeugt man eine Zufallszahl und bedient sich dem „Fermatschen Primzahltest" welcher aber unter umständen auch eine sogenannte „Pseudoprimzahl" erzeugen kann, oder man erzeugt mit dem „Sieb des Eratosthenes" eine Reihe von Primzahlen, was aber bei großen Primzahlen unverhältnismäßig lange dauern kann.

2.3.1.1.Fermatscher Primzahltest

Mit dem Fermatschen Primzahltest kann man Primzahlen von zusammengesetzten Zahlen uterscheiden. Der Test erhält eine Zahl n und eine Basis a wobei gelten muss das n > 3 und ungerade, sowie 1 < a < n-1 ist.

Der eigentliche Test besteht aus 2 Schritten.

1. Schritt:
   Berechne $b = a^{n-1} \bmod n$
2. Schritt:
   Prüfe ob b = 1.
   Falls b = 1 ist n bezüglich a prim. Ansonsten ist n auf jeden Fall nicht prim.

Den Fermatschen Primzahltest setzt man ein, wenn man eine (große) Zahl n gegeben hat, von der man herausfinden möchte ob sie eine Primzahl ist. In den meisten Fällen reicht dann ein Primzaltest mit a = 2. Durch den kleinen Fermatschen Satz ist aber nicht garantiert das es sich bei einer gefunden möglichen Primzahl auch um eine echte Primzahl handelt. So ist die Zahl 341 die erste sogenannte Pseudoprimzahl die zwar laut Test bezüglich 2 eine Primzahl sein sollte $2^{341-1} \bmod 341 = 1$ aber in wirklichkeit eine zusammengesetzte Zahl, nämlich *11*31* ist. Aus diesem Grund wird dieses Verfahren auch PRP-Test (probable prime test) genannt.
Da laut Paul Erdös die Anzahl der Pseudoprimzahlen zu einer festen Basis a im gegensatz zur Anzahl der gefunden Primzahlen aber recht gering ist, kann dieser Fall vernachlässigt werden.

2.3.1.2.Sieb des Eratosthenes

Beim Sieb des Eratosthenes werden ausgehend von der Startzahl 2 und einem Maximum m alle Primzahlen ermittelt. Dazu werden als erstes alle Zahlen von 2 bis m erzeugt. Danach wird eine Hilfvariable n = 2 initialisert. Nun werden alle vielfache von n aus der liste Entfernt. Ist man am Ende der Liste angelangt, wird die Hilfsvariable mit der nächsten noch verfügbaren Zahl belegt und der Prozess der Streichung wiederholt. Der Prozess kann bei der Wurzel aus m abgebrochen werden. Alle nun noch in der Liste vorhandenen Zahlen sind Prim.

2.3.2 Schlüsselberechnung

Aus den nun gefunden Primzahlen p und q wird als erstes das Produkt dieser beiden Zahlen $n = p * q$ gebildet. Für weitere Berechnungen errechnen wir auch $\Phi(n) = (p - 1) * (q - 1)$.
Nun wählt man den öffentlichen Exponenten e, so das der größte gemeinsame Teiler von e und Φ(n) gleich 1 ist.

Zur Bestimmung des größten gemeinsamen Teilers (ggT) wird in der Regel der euklidischer Algorithmus verwendet. Diese Prinzip wird auch gegenseitige Wechselwegnahme genannt. Eingangsgrößen sind 2 natürliche Zahlen a und b die nach folgendem Schema verglichen werden.

1. Setze m = a; n = b
2. Ist m < n, so vertausche m und n
3. Berechne r = m – n
4. Setze m = n; n = r
5. Ist r ≠ 0, so fahre fort mit Schritt 2

Mit ablauf des Verfahrens hat man mit m den ggT von a und b gefunden.

Ist die Differenz von a und b sehr groß, sind unter Umständen sehr viele Subtraktionsschritte notwendig. Aus diesem Grund wird heutzutage hauptsächlich der Divisions – Algorithmus statt des Subtraktions - Algorithmus verwendet.

Subtraktions – Algorithmus:

```
While a ≠ b
   If a > b
   Then a = a – b
   Else b = b – a
End While
ggT = a
```

Divisions – Algorithmus:

```
While (b > 0)
   r = a mod b
   a = b
   b = r
End While
ggT = a
```

Um eine schnelle Verschlüsselung zu gewährleisten wird der Exponent e meist niedrig gewählt.

Als letztes wird noch der geheime Exponent d über die Formel $d = e - 1 (mod\ \Phi(n))$ berechnet.

Nun haben wir mit (N E) und (N D) den Public- bzw. Private Key erstellt. Aus sicherheitsgründen sollten die beiden Primzahlen p und q vernichtet werden, da sie bei einmal erzeugten Schlüsseln nicht mehr benötigt werden.

2.3.3. Verschlüsselung

Bevor nun ein Text verschlüsselt werden kann, muss dieser zunächst in eine Zahlenkette umgewandelt werden. Dazu wird den Buchstaben im Text einfach der zugehörige ASCII - Code zugeordnet.

Nachfolgend muss noch die so erhaltene Zahlenkette in Blöcke aufgeteilt werden wobei die Blockgröße n nicht überschritten werden darf.

Nun wird jeder Block K über die Formel $C = K\ E\ mod\ N$ in einen verschlüsselten Text C umgewandelt und kann nun verschickt werden.

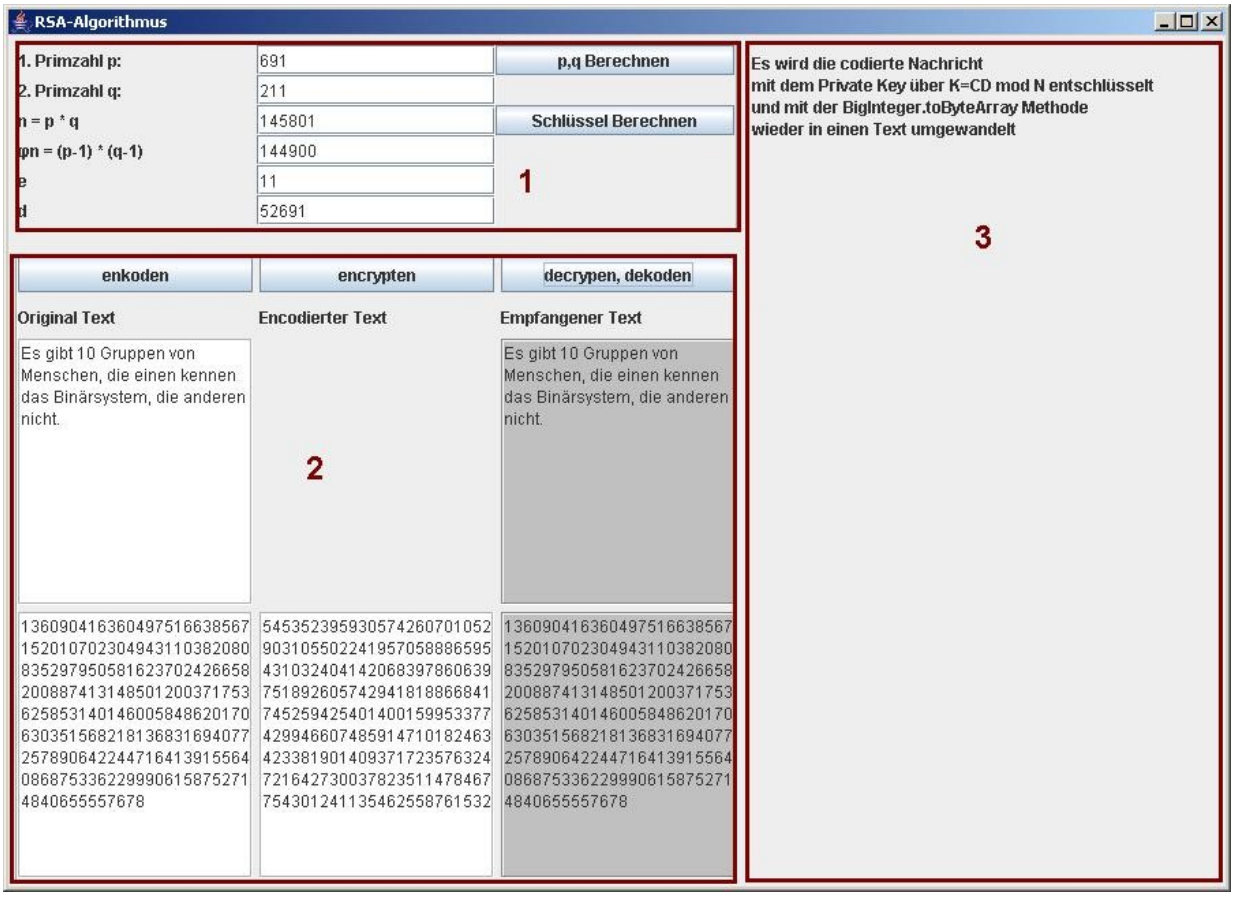
2.3.4. Entschlüsselung

Der empfangen Text C kann nun mit dem geheim gehaltenen Private Key über die Formel $K=CD\ mod\ N$ wieder in die Klartextzahlenkette K entschlüsselt werden.

Anschließend wird die Zahlenkette wieder mit Hilfe der ASCII - Tabelle in einen Text umgewandelt..

3. Dokumentation des Programms

3.1 Screenshot



Das Programm zur Demonstration des RSA-Algorithmus ist in 3 große Bereiche aufgeteilt.

1. Der Bereich für die Eingabe/Erzeugung der Primzahlen und der für die Ver- und Entschlüsselung notwendigen Schlüssel.
2. Der Bereich in dem ein Text in eine Dezimalkette umgewandelt, mit RSA verschlüsselt und dann wieder entschlüsselt und in einen Text rückgewandelt wird.
3. Der Bereich in dem zu jedem Schritt der Schlüsselerzeugung bzw. der Ver- und Entschlüsselung ein Informationstext angezeigt wird.

3.2.Dokumentation der einzelnen Programmteile

Durch betätigen des Buttons „p,q Berechnen“ werden die Primzahlen p und q mit Hilfe der Java eigenen Klasse BigInteger errechnet. In unserem Fall wird eine Zahl mit 10 bzw. 8 Bit Länge und die Wahrscheinlichkeit das wir keine Primzahl erhalten ist $1-2^{20}$.

```
pqbutton.addActionListener(new ActionListener(){        // p und q berechnen
    public void actionPerformed(ActionEvent e){
            tp.setText(new BigInteger(9 + 1, 20, new Random()).toString());
            tq.setText(new BigInteger(9 - 1, 20, new Random()).toString());

            lerk.setText("<html> Es werden die 2 Primzahlen p und q berechnet<br><br>" +
                         "Nächster Schritt: Schlüssel berechnen");

    }
});
```

Durch betätigen des Buttons „Schlüssel Berechnen“ werden aus den Primzahlen p und q die notwendigen Schlüsselelemente

$n = p*q$

$\varphi n = (p-1)*(q-1)$

berechnet.

Das Element e wird über den euklidischen Algorithmus bestimmt.

```
public static int findeE(int internp, int internq, int internphi){
        int erg;
        int ggt =0;
        erg = 2;
        while (ggt != 1){
               erg++;
               ggt = findeggt(erg, internphi);
        }

        return erg;
}

public static int findeggt(int m, int n){
        if (m == 0) {
                   return n;
                }
                if (n == 0) {
                   return m;
                }
                if (m < 0) {
                   return findeggt(-m,n);
                }
                if (n < 0) {
                   return findeggt(m,-n);
                }
                if (m > n) {
                   return euklid(n,m);
                }
                return euklid(m,n);
}

public static int euklid(int m, int n){
         int a = m%n;
         while (a != 0) {
            m = n;
            n = a;
            a = m%n;
         }
         return n;
}
```

Das Schlüsselelemnt d wird nun mit dem erweiterten Euklidischen Algorithmus berechnet.

```
public static double finded(int internE,int internPHI) {
        double u1, u2, u3,  g, inverse, uu, vv;
        double q,t1,t2,t3,v1,v2,v3,z;
        u1 = 1;
        u2 = 0;
        u3 = internPHI;
        v1 = 0;
        v2 = 1;
        v3 = internE;

        while (v3 != 0) {
             q = Math.floor(u3/v3);
             t1 = u1 - q * v1;
             t2 = u2 - q * v2;
             t3 = u3 - q * v3;

             u1 = v1;
             u2 = v2;
             u3 = v3;

             v1 = t1;
             v2 = t2;
             v3 = t3;
             z = 1;
        }
        uu = u1;
        vv = u2;

        if (vv < 0) {
                  inverse = vv + internPHI;
        } else {
             inverse = vv;
        }
        return inverse;
```

Nach betätigen des Buttons „encoden“ wird der Text aus dem Textfeld in eine Dezimalkette umgewandelt

```
         encodebutton.addActionListener(new ActionListener(){    // Text in Zahlen
                                                                  umwandeln
      public void actionPerformed(ActionEvent e){
             try{
             tnachrichtnum.setText(new
                           BigInteger(tnachricht.getText().getBytes()).toString());
             lerk.setText("<html> Der Text wurde über die BigInteger.toString methode <br>"
                                  +"in eine Zeichenkette umgewandelt<br><br>" +
                           "Nächster Schritt: encrypten");
             }catch(NumberFormatException nfe){
                    JOptionPane.showMessageDialog(null,"Bitte einen Text zum enkoden
                           eingeben",nfe.getMessage(),JOptionPane.INFORMATION_MESSAGE);
             }}
   });
```

Danach kann die Dezimalkette über „encrypten“ in den verschlüsselten Text umgewandelt werden. Da die größe von n variabel ist und bei der verschlüsselung die Dezimalkettenlänge die gleichzeitig verschlüsselt wird nicht größer als n sein darf, arbeiten wir mit einer Bitmaske die geshiftet wird mit deren Hilfe wir die einzelnen „Chunks“ erhalten, welche wir dann mit C=KE mod N encrypten.

```
   encryptbutton.addActionListener(new ActionListener(){  // Zahlenwerte mit dem public key
                                                 verschlüsseln
      public void actionPerformed(ActionEvent e){
 try {
             tencoded.setText(encrypt(new BigInteger(tnachrichtnum.getText()), new
                    BigInteger(te.getText()), new BigInteger(tn.getText())).toString());
             lerk.setText("<html> Die Zahlenwerte werden in Blöcke mit der <br>" +
```

```
                                "größe n-1 aufgeteilt und mit dem Public Key<br>" +
                                "über C=KE mod N berechnet<br><br>" +
                        "Nächster Schritt: decrypten, dekoden");
        }catch(NumberFormatException nfe){
                JOptionPane.showMessageDialog(null,"Bitte einen Text enkoden, oder bei
manueller eingabe nur Zahlenwerte eingeben und eventuell manuel eingegebene Schlüsselwerte ü
berprüfen.",nfe.getMessage(),JOptionPane.INFORMATION_MESSAGE);
        }}
    });


 public static BigInteger encrypt(BigInteger m, BigInteger e, BigInteger n) {
        BigInteger c, bitmask;
        c = new BigInteger("0");      //erstellen einer bitmaske für die chunks
        int i = 0;
        bitmask = (new BigInteger("2")).pow(n.bitLength()-1).subtract(new BigInteger("1"));
//setzen der          bitmask größe auf n-1
        while (m.compareTo(bitmask) == 1) {
                c = m.and(bitmask).modPow(e,n).shiftLeft(i*n.bitLength()).or(c);   //encrypten
                                des chunks und schreiben in c, danach linksschift
                m = m.shiftRight(n.bitLength()-1); //nachricht rechtsschiften
                i = i+1;
        }
        c = m.modPow(e,n).shiftLeft(i*n.bitLength()).or(c);
        return c;
}
```

Analog dazu wird über den Button „decrypten, dekoden“ der verschlüsselte Text wieder mit K=CD mod N entschlüsselt und wieder in einen Klartext umgewandelt.

```
  decodebutton.addActionListener(new ActionListener(){     // Verschlüsselte Nachricht
                        entschlüsseln und dann Zahlen in Text umwandeln
        public void actionPerformed(ActionEvent e){
                try {
                tempfangennum.setText(decrypt(new BigInteger(tencoded.getText()), new
                BigInteger(td.getText()),     new BigInteger(tn.getText())).toString());
                tempfangen.setText(new String(new
                                BigInteger(tempfangennum.getText()).toByteArray()));

                lerk.setText("<html> Es wird die codierte Nachricht <br>" +
                                "mit dem Private Key über K=CD mod N entschlüsselt<br>" +
                                "und mit der BigInteger.toByteArray Methode <br>" +
                                "wieder in einen Text umgewandelt");
                }catch(NumberFormatException nfe){
                        JOptionPane.showMessageDialog(null,"Bitte einen Text encrypten oder
                        bei manueller Eingabe nur Zahlenwerte
                        eingeben",nfe.getMessage(),JOptionPane.INFORMATION_MESSAGE);
                }}
    });
```

4. Literatur

Primärliteratur:

R.L. Rivest, A. Shamir and L. Adleman

MIT Loboratory for Computer Science and Department of Mathematics

„A Method for Obtaining Digital Signatures and Public Key Cryptosystems“

1978

Sekundärliteratur:

„Sieb des Eratosthenes“, Euklidischer Algorithmus und Fermatscher Primzahltest auf:

de.wikipedia.org

Der RSA – Algorithmus

http://www.wiwi.unibielefeld.de/StatCompSci/lehre/material_spezifisch/statalg00/rsa/node9.html

KRYPTOGRAPHIE

http://www.kuno-kohn.de/crypto/crypto/rsa.htm

Asymmetrische Verfahren: RSA

http://www.regenechsen.de/phpwcms/index.php?krypto_asym_rsa