

Dan Chirica

CS 6520

Java implementation of the RSA Algorithm  
Individual Project

This project represents an implementation of RSA algorithm using Java programming language. We choose Java because it provides support for working with big integers through the BigInteger class.

This project consists of two applications:

1. RsaKeyGen.java that generates a pair of public and private keys. Those keys are stored in two files named RsaPublic.key and RsaPrivate.key.
2. Rsa.java that performs the actual RSA encryption and decryption. This program prompts the user to choose whether she will perform encryption or decryption, enter the encryption/decryption key and provide a message (in numerical format) to be encrypted.

Following there are two output samples from running RsaKeyGen.java and Rsa.java

RsaKeyGen.java output:

```
p =
11394570107462710250178795707518144358969141224528081336615172757359642
60010119855719981778294211505498166219112449740127542626014535992734499
7842625862289
q =
87131139158353779783225637369393410129900275516859098116150368931729661
15489451212303673588522027588941128170774037235910811631714978475755532
239444499481
n =
99282187368295158964672912830993330552047442648011812524565123077718715
31051723558102315612736802666463367712141672261904975367004877677394764
59770192314906360902578814805717148864609511938138724354980712055224081
33035500471524076833817993069577185343376673956578840907187680639391701
565898507150535637972009
```

```
phi_n =
99282187368295158964672912830993330552047442648011812524565123077718715
31051723558102315612736802666463367712141672261904975367004877677394764
59770192314705284062345833923432135270164937084419132667218572143741779
23384967862808486184048489578205721200732751166216942372550494401499841
227495406620453567610240
```

Public key:

```
65537;99282187368295158964672912830993330552047442648011812524565123077
71871531051723558102315612736802666463367712141672261904975367004877677
39476459770192314906360902578814805717148864609511938138724354980712055
22408133035500471524076833817993069577185343376673956578840907187680639
391701565898507150535637972009
```

Private key:

```
23638543901839841013240705735917419624550228040337157981496165227348258
75010011053307727433680898863351914029941234020091936396642264847003492
78701406547120882135417312244094710449908504787605110794502015651173332
97473168416791486006620575116015717436199915302770178201340890102400224
644305328667860254039553;9928218736829515896467291283099333055204744264
80118125245651230777187153105172355810231561273680266646336771214167226
19049753670048776773947645977019231490636090257881480571714886460951193
81387243549807120552240813303550047152407683381799306957718534337667395
6578840907187680639391701565898507150535637972009
```

Rsa.java output:

Encrypt or decrypt? (e/d): e

Doing RSA Encryption

Enter encryption (public) key filename: rsapublic.key

Enter message (number) to be encrypted: 2101

C =

```
51915963301219667255322877611549357475420784283714213821810189949072217
60926972171318851286642265914025705464148921126761132519318163513854877
40784842025958572860354928773414628520155770123282748043614496459034686
70717252513152776972603788196205265421152729109777936425459832000085639
50856402261450435783425
```

Encrypt or decrypt? (e/d): d

Doing RSA Decryption

Enter decryption (private) key filename: rsaprivate.key

Enter message (number) to be decrypted:

```
51915963301219667255322877611549357475420784283714213821810189949072217
60926972171318851286642265914025705464148921126761132519318163513854877
40784842025958572860354928773414628520155770123282748043614496459034686
70717252513152776972603788196205265421152729109777936425459832000085639
50856402261450435783425
```

M = 2101

Encrypt or decrypt? (e/d): q

Exit!

```

/*
 * Created on Aug 12, 2004
 *
 */

/**
 * @author Dan Chirica
 *
 * Individual Project for CS 6520
 * Java implementation of RSA algorithm
 *
 * This application relies on a pair of public/private encryption keys
 * generated with the RsaKeyGen.java application
 * Those keys are stored in text files named
 * RsaPublic.key
 * and
 * RsaPrivate.key
 * The message to be encrypted is inputed in numerical format.
 */

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.math.BigInteger;

public class RSA {

    public static void main(String[] args) {
        while (true) {
            System.out.print("Encryption/Decryption/Quit? (e/d/q):
");
            try {
                BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
                String str = "";
                str = in.readLine();
                //System.out.println(str);
                if (str.equals("e"))
                    RSAEncrypt();
                else if (str.equals("d"))
                    RSADecrypt();
                else if (str.equals("q")) {
                    System.out.print("Exit!");
                    break;
                }
            }
            else
                System.out.println("I don't know what to do!");
        } catch (IOException e) {
        }
    }

    public static void RSAEncrypt(){
        System.out.println("Doing RSA Encryption");
        String publicKey = "";
        BigInteger e = new BigInteger("0");
        BigInteger n = new BigInteger("0");
    }
}

```

```

        BigInteger M = new BigInteger("0");
    try {
        BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
        String str = "";
        System.out.print("Enter encryption (public) key filename:
");
        str = in.readLine();
        publicKey = str;
    } catch (IOException f) {}
    try {
        BufferedReader in = new BufferedReader(new
FileReader(publicKey));
        BigInteger te, tn;
        te = new BigInteger(in.readLine());
        tn = new BigInteger(in.readLine());
        e = te;
        //System.out.println("e = " + e.toString());
        n = tn;
        //System.out.println("n = " + n.toString());
        in.close();
    } catch (IOException f) {}
    try {
        BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
        System.out.print("Enter message (number) to be encrypted:
");
        M = new BigInteger (in.readLine());
        //do encryption
        System.out.println("C = " + M.modPow(e, n).toString());
    } catch (IOException f) {}
}
public static void RSADecrypt() {
    System.out.println("Doing RSA Decryption");
    String publicKey = "";
    BigInteger d = new BigInteger("0");
    BigInteger n = new BigInteger("0");
    BigInteger C = new BigInteger("0");
    try {
        BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
        String str = "";
        System.out.print("Enter decryption (private) key filename:
");
        str = in.readLine();
        publicKey = str;
    } catch (IOException f) {}
    try {
        BufferedReader in = new BufferedReader(new
FileReader(publicKey));
        BigInteger td, tn;
        td = new BigInteger(in.readLine());
        tn = new BigInteger(in.readLine());
        d = td;
        //System.out.println("e = " + d.toString());
        n = tn;

```

```
        //System.out.println("n = " + n.toString());
        in.close();
    } catch (IOException f) {}
    try {
        BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
        System.out.print("Enter message (number) to be decrypted:
");
        C = new BigInteger (in.readLine());
        //do decryption
        System.out.println("M = " + C.modPow(d, n).toString());
    } catch (IOException f) {}
}
}
```

```

/*
 * Created on Aug 11, 2004
 *
 */

/**
 * @author Dan Chirica
 *
 * This application RsaGenKey is part of the individual project for
 * CS 6520 class.
 * This program generates a RSA pair of encryption/decryption keys.
 * Those keys are stored in two files RsaPublic.key and RsaPrivate.key.
 * The keys are 1024 bite long. This key pair is used by the Rsa.java
program to
 * encrypt/decrypt an message.
 */
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.math.BigInteger;
import java.security.SecureRandom;

public class RsaKeyGen {

    public static void main(String[] args) {
        int key_length = 1024; //key is hard coded at 128 bits
        BigInteger n, d, e;
        SecureRandom r = new SecureRandom();
        BigInteger p = new BigInteger(key_length / 2, 100, r);
        System.out.println("p = " + p.toString());
        BigInteger q = new BigInteger(key_length / 2, 100, r);
        System.out.println("q = " + q.toString());
        n = p.multiply(q);
        System.out.println("n = " + n.toString());
        BigInteger phi_n = (p.subtract(BigInteger.ONE))
            .multiply(q.subtract(BigInteger.ONE));
        System.out.println("phi_n = " + phi_n.toString());
        //why 65537? 'Cause is recommended being a Fermat prime
number
        e = new BigInteger("65537");
        while(phi_n.gcd(e).intValue() > 1) e = e.add(new
BigInteger("2"));
        d = e.modInverse(phi_n);
        System.out.println("Public key:");
        System.out.println(e.toString() + ";" + n.toString());
        System.out.println("Private key:");
        System.out.println(d.toString() + ";" + n.toString());
        try {
            BufferedWriter out1 = new BufferedWriter(new
FileWriter("RsaPublic.key"));
            out1.write(e.toString() + "\n" + n.toString());
            out1.close();
        } catch (IOException error) {
        }
        try {
            BufferedWriter out2 = new BufferedWriter(new
FileWriter("RsaPrivate.key"));

```

```
        out2.write(d.toString() + "\n" + n.toString());
        out2.close();
    } catch (IOException error) {
    }
}
}
```