[**Assignment 6 RSA**](https://courseweb.pitt.edu/webapps/blackboard/execute/uploadAssignment?content_id=_16777703_1&course_id=_171948_1&assign_group_id=&mode=view)

**Part A. Small Integer RSA Cryptography**

(1) Break up the example program (SimpleCrypto.java) into three smaller programs (KeyGen.java, Encode.java, and Decode.java). Each of these programs should provide directions to the user and each should display their results so they are easy to interpret. These programs should be useful when computing your answers to questions 2 through 6.

(a) The program KeyGen.java will ask the user for two primes (p and q) with p < q. It will compute n, phi, e, and d and will display these values to the user in a nicely formatted way.

(b) The program Encode.java will ask the user for a public key pair (e,n) and a single character to be encoded. The character (m) will be entered from the keyboard and it will be used to compute c = me mod n. This encoded value (c) will then be displayed to the user as an integer.

(c) The program Decode.java will ask the user for a private key pair (d,n) and a single integer (c) to be decoded. The program will then compute m = cd mod n and display the result as a character.

(2) Send a message to Bob. The message that you want to send is the left brace character ‘{‘. You know that Bob’s (e, n) pair is (5, 437). What integer will you send? \_\_\_\_\_\_\_

(3) Bob receives a message. It is the integer 16. Bob’s (d, n) pair is (317, 437). What message did Bob

 receive? \_\_\_\_\_\_

(4) An eavesdropper is watching all communications that are destined for Ken. The eavesdropper sees the pair (9,247). He knows that the first number is an encoded ASCII value and the second number is Ken’s value for n. He also knows the algorithm that Ken uses to determine n, phi, e and d from p and q. The eavesdropper sees that Ken has chosen a rather small value for n and so decides to break this code. What ASCII character is being sent to Ken? \_\_\_\_\_\_

(5) Consider an RSA key set with p = 11, q = 29, n = 319, and e = 3. What value of d should be used

  in the secret key? \_\_\_\_\_\_\_ What is the encryption of the message m = 100 ? \_\_\_\_\_\_\_

(6) Bob receives several digitally signed messages from someone he thinks may be Alice. He knows that Alice’s public key is (e = 3, n = 391). The messages each arrive in two parts. The first part is “in the clear” and is not protected from disclosure. The second part is the first part encrypted using the signer’s secret key d. Here are the message pairs Bob receives. Which ones are actually from Alice and which one’s have been corrupted or are forged? (Hint: Alice uses her secret key to encrypt the signed part. Bob needs to use Alice’s public key to compare the clear text with the encoded text.)

<’A’, 112>

 <’L’, 359>

 <’X’, 296>

 <’B’, 113>

(7) Alice’s public key pair is ( e = 3, n = 391). Write a Java program that behaves as follows (EncodeString.java):

 java EncodeString

 Enter the encoding exponent e :  3

 Enter the modulus n : 391

 Enter the string to encode :  Hello Alice

234 16 301 301 304 315 143 301 265 228 16

(8) Alice’s private pair is (235,391). Write a Java program that behaves as follows:

 java DecodeInts.java

 Enter the decoding exponent d : 235

 Enter the modulus n : 391

 Enter the number of integers to decode : 11

 234 16  301   301 304 315 143 301 265 228 16

 Hello Alice

**Part B. BigInteger RSA Cryptography**

The public and private keys in part A is too small for secure encryption. Java provides a BigInteger class as shown in the program below – BigIntegerRSA.java.

(9) Using Java’s BigInteger class, write three small functions (KeyGen(), Encode(), and BDecode()). Each of these functions should provide directions to the user and each should display their results so they are easy to interpret.

(a) The function KeyGen() will randomly generate two large primes using an appropriate BigInteger constructor based on the number of bits entered by the user. It will compute n, phi, e, and d and will display these values to the user in a nicely formatted  
way. Finally, the values for n and e are stored in a file public.txt and the values of n and d are stored in a file private.txt. Randomly generate e as a prime greater than 1 and less than phi. You may use the modInverse() function to solve for d until you get your program working properly. In the final version of your program, **YOU MUST** replace any calls to modInverse() with calls to the XGCD() (the Extended Greatest Common Divisor algorithm discussed in class).

(b) The function Encode() will encrypt text using the public keys in public.txt..

(c) The program Decode() will decrypt using the private keys in private.txt.

Your main program should allow for the user to choose encryption or decryption of a file. You must accomplish this through command line arguments, so that your program will execute as follows (assume the program name is RSA.java and the file name is file.ext – naturally any file name can actually be used.):

 java RSA –encrypt file.ext

Encrypting a file will create a new file (and NOT delete the old file) which has the same name as the original but with the added extension of .enc (for encrypted). Thus, for the example above, the output file would have the name file.ext.enc .

java RSA –decrypt file.ext.enc

Decrypting a file will create a new file (and NOT delete the .enc file) which has the same name as the plaintext file but with the added extension of .cop (for copy). Thus, continuing the example above, the decrypted output file would have the name file.ext.cop .

// SimpleCrypto.java

// A minimum RSA public key cryptography example in Java.

// It only works for small p and q.

public class SimpleCrypto {

 public static void main(String args[]) {

 // choose two distinct primes with p < q

 long p = 13;

 long q = 19;

 System.out.println("p = " + p + " q = " + q );

 // choose n as the product of p and q

 // no known algorithm can recompute p and q from n within    
 // a reasonable period of time for large n.

 long n = p \* q;

 System.out.println("The value of n = " + n);

 // Compute phi = (p-1)\*(q-1).

 long phi = (p - 1) \* ( q - 1);

  System.out.println("The value of PHI = " + phi);

 // choose a random prime e between 1 and phi, exclusive,    
 // so that e has no common factors with phi.

 long e = findfirstnocommon(phi);

 System.out.println("The public exponent = " + e);

 // Compute d as the multiplicative inverse of e

 // modulo phi(n).

 long d = findinverse(e,phi);

 System.out.println("The private key is " + d);

 System.out.println( " (d) (e) mod phi = " + (d \* e) % phi);

 // let m be the message that needs to be encoded

 char m = 'Q';

 // encode m as c = m^e mod n using expomod

 long c = expomod(m,e,n);

 // c is sent to the receiver over an open channel

 System.out.println("Transmitting encoded " + m + " as " + c);

  // decode c to m = c^d mod n

 m = (char)expomod(c,d,n);

 System.out.println( "Decoding " + c + " to " + m);

 }

 // Let a and n be two long integers with n > 0. We wish to

 // compute x = a^n mod z.

 static long expomod(long a, long n, long z) {

 long r = a % z;

  for(long i = 1; i < n; i++) {

 r = (a \* r) % z;

  }

  return r;

 }

 static long findfirstnocommon(long n) {

  long j;

  for(j = 2; j < n; j++)

  if(euclid(n,j) == 1) return j;

  return 0;

 }

 static long findinverse(long n, long phi) {

  long i = 2;

  while( ((i \* n) % phi) != 1) i++;

  return i;

 }

 static long euclid(long m, long n) {

 // pre: m and n are two positive integers (not both 0)

 // post: returns the largest integer that divides both  
 // m and n exactly

 while(m > 0) {

 long t = m;

 m = n % m;

 n = t;

  }

  return n;

 }

}

javac SimpleCrypto.java

java SimpleCrypto

p = 11 q = 37

The value of n = 407

The value of PHI = 360

The public exponent = 7

The private key is 103

 (d) (e) mod phi = 1

Transmitting encoded A as 21

Decoding 21 to A

import java.math.BigInteger;

import java.util.Random;

public class BigIntegerRSA {

 public static void main(String args[]) {

 // get the number of bits in the primes

 int primeBits = Integer.parseInt(args[0]);

 // get a random number

 Random rnd = new Random();

 // get two distinct primes of size primeBits

 BigInteger p = new BigInteger(primeBits,128,rnd);

 BigInteger q;

 do q = new BigInteger(primeBits,128,rnd);

 while(p.compareTo(q) == 0);

 // compute the modulus

 BigInteger n = p.multiply(q);

 // compute m = phi(n)

 BigInteger pMinus1 = p.subtract(BigInteger.valueOf(1));

 BigInteger qMinus1 = q.subtract(BigInteger.valueOf(1));

 BigInteger m = pMinus1.multiply(qMinus1);

 // get e relatively prime to m

 BigInteger e = BigInteger.valueOf(3);

 while(e.gcd(m).compareTo(BigInteger.valueOf(1)) > 0)

 e = e.add(BigInteger.valueOf(2));

 // compute d the decryption exponent

 BigInteger d = e.modInverse(m);

 System.out.println("e = " + e + "\nd = " + d + "\nn=" + n);

 }

}

e = 3

d = 513296655724846623897776175084521202801633496977049328471696469765040389621375594662101307823

809605510007736357284392089852301980813490038964392320585473813556152030589812875811180568639

6758502290149009167575985629661842676915928007300006655657003766598802080552760014411

n=76994498358726993584666426262678180420245024546557399270754470464756058443206339199315196180

5714408265011604535926588134778452971220235617842090575883119316764433742542037534357090934199

7306758634433243893919381442709419309473173506472123119430950196852221386112330403387