Anthony DeCarlo and Daniel Elice

RSA Project Report

1. Project Description:

The user will be asked to enter the value of n, the value of e, and to enter to ciphertext. The program will decrypt the ciphertext and print out the plaintext using the following methods.

1. public static int getN()

* Parameters: None
* Description: Asks the user to enter the n value

1. public static int getE()

* Parameters: None
* Description: Asks the user to enter the e value

1. public static List <Integer> factor (int x)

* Parameter: x
* Description: Uses a *while* loop inside a *for* loop to search for factors of the value, *x*.

1. public static int modInv(int u, int v)

* Parameters: u, v
* Description: Computes the inverse mod using extended Euclidean algorithm

1. public static int [] decrypt (int m)

* Parameters: m
* Description: Creates an array to store the 3 plaintext values after decrypting the ciphertext

1. public static int calcD(int e, int p, int q)

* Parameters: e, p, q
* Description: Calculates the value of d using the *modInv* method

1. public static int modExp (int x, int y, int p)

* Parameters: x, y, p
* Description: Computes modular exponentiation

1. public static int getInt()

* Parameters: None
* Description: Input validation from the user

1. Source Code:

// Anthony DeCarlo & Daniel Elice

// 05/11/2017

// MATH314

package rsa;

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class RSA {

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

int n = getN(); //Input value for n

int e = getE(); //Input value for e

List<Integer> list = factor(n); //Store the two prime factors into a list

int p = list.get(0); // First prime factor

int q = list.get(1); //Second prime factor

int d = calcD(e, p, q); // Calculates d value

System.out.println("Enter your message (c), enter a non-integer/character to quit");

while (in.hasNext()) //User inputs their ciphertext while there is input

{

if (!in.hasNextInt()) {

break;

} else {

int c = in.nextInt();

int m = modExp(c, d, n); //Modular exponentiation

int[] word = decrypt(m); //Decrypts the message and stores the block into an array

String alphabet = "abcdefghijklmnopqrstuvwxyz";

for (int i = 0; i < word.length; i++) {

System.out.print(alphabet.charAt(word[i])); //Translates the numbers into letters

}

}

}

}

public static int getN() {

System.out.println("Please enter your n value (0-100,000)");

int n = getInt();

while ((n < 0) || (n > 100000)) { //User must enter a valid integer between 0 and 100,000

System.out.println("Enter a valid n value (0-100,000)");

n = getInt();

}

return n;

}

public static int getE() {

System.out.println("Please enter your e value (0-100,000)");

int e = getInt();

while ((e < 0) || (e > 100000)) { //User must enter a valid integer between 0 and 100,000

System.out.println("Enter a valid e value (0-100,000)");

e = getInt();

}

return e;

}

public static List<Integer> factor(int x) { //Prime factorization of x values

List<Integer> factor = new ArrayList<Integer>();

int count;

for (int i = 2; i <= (x); i++) {

count = 0;

while (x % i == 0) {

x /= i;

count++;

}

if (count == 0) {

continue;

}

factor.add(i);

}

return factor;

}

public static int modInv(int x, int y) //Inverse mod using Extended Euclidian algorithm

{

int inv, x1, x3, y1, y3, z1, z3, q;

int iter;

x1 = 1;

x3 = x;

y1 = 0;

y3 = y;

iter = 1;

//Loop while y3 != 0

while (y3 != 0) {

q = x3 / y3;

z3 = x3 % y3;

z1 = x1 + q \* y1;

//Swap

x1 = y1;

y1 = z1;

x3 = y3;

y3 = z3;

iter = -iter;

}

//Make sure x3 = gcd(x,y) == 1

if (x3 != 1) {

return 0; // No inverse exists

} // Positive result

if (iter < 0) {

inv = y - x1;

} else {

inv = x1;

}

return inv;

}

public static int[] decrypt(int m) { //Inverts the process of converting blocks of three letters into a number

int[] word = new int[3]; //Create an array to store each letter

int a;

int b;

int c;

a = m % 26; //Calculates first letter

b = ((m - a) / 26) % 26; //Calculates second letter

c = (((m - a) / 26) - b) / 26; //Calculates last letter

word[0] = c;

word[1] = b;

word[2] = a; //Stores the letters into an array

return word;

}

public static int calcD(int e, int p, int q) { //Formula for calculating d

int d = modInv(e, (p - 1) \* (q - 1)); //Calls modInv method to calculate e^(-1)(mod(p-1)(q-1))

return d;

}

public static int modExp(int x, int y, int p) { //Modular exponentiation using repeated squaring method

int result = 1; //Initialize

x = x % p;

while (y > 0) {

if (y % 2 == 1) //y value is odd

{

result = (result \* x) % p;

}

// y value is even

y = y >> 1;

x = (x \* x) % p;

}

return result;

}

public static int getInt() //Looping input validation that only accepts integers from keyboard

{

Scanner inp = new Scanner(System.in);

int intInp = 0;

boolean valid;

do {

if (inp.hasNextInt()) {

intInp = inp.nextInt();

valid = true;

} else {

valid = false;

inp.next();

System.out.println("Not a valid integer! Please try again!");

}

} while (!valid);

return intInp;

}

}

1. Plaintext:

Below is an example execution of the code given in assignment instructions. At the bottom of this section, you will find the plaintext.

Please enter your n value (0-100,000)

18923

Please enter your e value (0-100,000)

1261

Enter your message (c)

12423 11524 7243 7459 14303 6127 10964 16399

9792 13629 14407 18817 18830 13556 3159 16647

5300 13951 81 8986 8007 13167 10022 17213

2264 961 17459 4101 2999 14569 17183 15827

12693 9553 18194 3830 2664 13998 12501 18873

12161 13071 16900 7233 8270 17086 9792 14266

13236 5300 13951 8850 12129 6091 18110 3332

15061 12347 7817 7946 11675 13924 13892 18031

2620 6276 8500 201 8850 11178 16477 10161

3533 13842 7537 12259 18110 44 2364 15570

3460 9886 8687 4481 11231 7547 11383 17910

12867 13203 5102 4742 5053 15407 2976 9330

12192 56 2471 15334 841 13995 17592 13297

2430 9741 11675 424 6686 738 13874 8168

7913 6246 14301 1144 9056 15967 7328 13203

796 195 9872 16979 15404 14130 9105 2001

9792 14251 1498 11296 1105 4502 16979 1105

56 4118 11302 5988 3363 15827 6928 4191

4277 10617 874 13211 11821 3090 18110 44

2364 15570 3460 9886 9988 3798 1158 9872

16979 15404 6127 9872 3652 14838 7437 2540

1367 2512 14407 5053 1521 297 10935 17137

2186 9433 13293 7555 13618 13000 6490 5310

18676 4782 11374 446 4165 11634 3846 14611

2364 6789 11634 4493 4063 4576 17955 7965

11748 14616 11453 17666 925 56 4118 18031

9522 14838 7437 3880 11476 8305 5102 2999

18628 14326 9175 9061 650 18110 8720 15404

2951 722 15334 841 15610 2443 11056 2186

Ibecameinvolvedinanargumentaboutmodernpaintingasubjectuponwhichiamspectacularlyillinformedhowevermanyofmyfriendscanbecomeheatedandevenviolentonthesubjectandienjoytheirwranglesinamodestwayiamanartistmyselfandihavesomesympathywiththeabstractionistsalthoughihavegonebeyondtheminmyownapproachtoartiamalumpisttwoorthreedecadesagoitwasquitefashionabletobeacubistandtodraweverythingincubesthentherewasarevoltbythevorticistswhodreweverythinginwhirlswenowhavetheabstractionistswhopainteverythinginaveryabstractedmannerbutmyownsmallworksdoneonmytelephonepadarecomposedofcarefullyshadedstrangelyshapedlumpswithtracesofcubismvorticismandabstractionisminthemforthosewhopossesstheseeingeyeasalumpististandalone