Manual RSA

Hogeschool van Amsterdam

Jan Dorresteijn
500713006

Bas van der Linden
500735218

26 oktober 2020

Inhoudsopgave

1. Introduction

This document contains the manual of the third assignment.

2. Application

2.1 Introduction

This section will go into detail about the working of the system. We created an application that is able to do RSA encryption and decryption of text messages.

2.2 Application

This section will provide information about the setup of the program. The user of the application needs to make sure there are no extra spaces in their input in order to get correct results.

The user also needs to be aware that the calculation of P and Q could take some time based on the value of N. This also applies to decryption because P and Q aren't know beforehand when E and N are provided. For this reason decryption can take some time also.

2.2.1 Run

The application can be run by double clicking it.

2.3 RSA

The RSA class in our application has multiple variables relevant to the RSA encryption and decryption process which will be looked at below.

Some note worthy variables are INIT_NUMBER, this will initialize the first prime that will be used to calculate P and Q.

```
private Random rand = new Random();
public static final BigInteger INIT_NUMBER = new BigInteger("2");
private final static BigInteger one = new BigInteger("1");
private String calculateTime;
private BigInteger p, q, n, e, d, phi;
```

We've chosen to use Big Interger for the provided methods which comes with the Math package. Such methods are modPow, probablePrime and isProbablePrime.

To generate P and Q we use the method calculatePAndQ based on the input N. This method is listed below.

```
public void calculatePandQ(String nInput) {
    this.n = new BigInteger(nInput);
    long begin = System.currentTimeMillis();
    //Initialise n and p
    BigInteger p = INIT_NUMBER;
    //For each prime p
    while (p.compareTo(n.divide(INIT_NUMBER)) <= 0) {
        //If we find p
        if (n.mod(p).equals(BigInteger.ZERO)) {
            //Calculate q
            BigInteger q = n.divide(p);
            //Displays the result
            this.p = p;
            this.q = q;
            calculateTime = String.valueOf(System.currentTimeMillis() -
                begin);
            //The end of the algorithm
            return;
        }
        //p = the next prime number
        p = p.nextProbablePrime();
    }
    calculateTime = "No solution exists";
}
```

Using those steps P and Q are generated.

Generate E

To generate E we first calculate phi, then we loop and generate an E untill E is less than phi and co-prime to phi.

The code for these calculations is as follows:

```
phi = p.subtract(one).multiply(q.subtract(one));
do {
    e = new BigInteger(phi.bitLength(), rand);
} while (e.compareTo(one) <= 0 || e.compareTo(phi) >= 0 ||
     !e.gcd(phi).equals(one));
d = e.modInverse(phi);
```

2.3.1 encryption

To encrypt the message we first split the string in a char array. Every char will be cast to an integer value and that value will be encrypted. The cipherText will split each char by a comma.

```
String cipherText = "";
ArrayList<Integer> convert = convertString(message);
for(Integer i : convert){
    cipherText =cipherText + BigInteger.valueOf(i).modPow(e, n) + ",";
}
```

```
return cipherText;
```

2.3.2 Decryption

Decryption requires an e and N and the encrypted text c as input. P and Q are calculated based on the N the user has entered. Then D will be calculated with the provided input and the calculated P and Q.

```
e = new BigInteger(inputE);
calculatePandQ(inputN);
d = e.modInverse(phi);
```

Step 2

In the first step we will generate a private key, in the final step we will use this private key to decode the cipher.

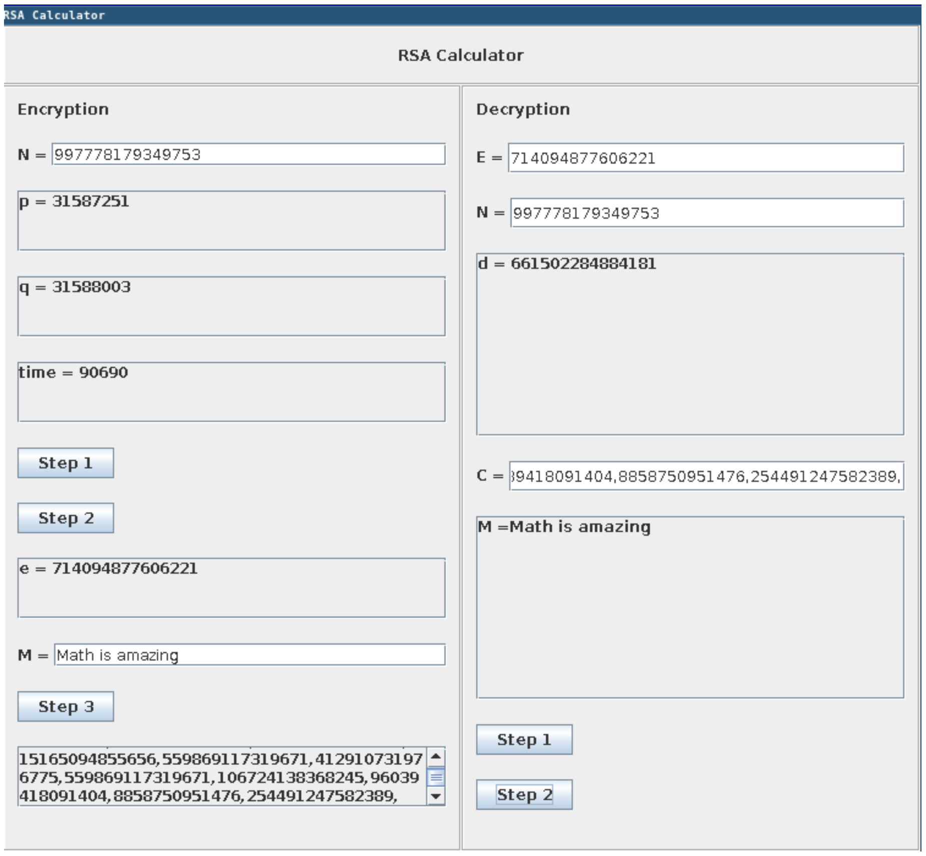
Finally every char will BigInteger will be cast to a int which will translate to the original message.

By making use of the stream method we can convert the whole array to the original message.

```
public String decodeCipher(String cipherText) {
    String[] cipherArray = cipherText.split(",");
    ArrayList<Character> mess = new ArrayList<>();
    for(String c : cipherArray){
        BigInteger letter = new BigInteger(c).modPow(d, n);
        mess.add((char)letter.intValue());
    }
    return mess.stream().map(e->e.toString()).reduce((acc, e) -> acc +
         e).get();
}
```

2.4 Examples

Below are listed some pictures that show how our application implements the encryption and decryption process. An N and a message are entered and output is generated.



N = 997778179349753

p = 31587251

q = 31588003

time = 90690

e = 714094877606221

M = Math is amazing

15165094855656,559869117319671,41291073197
6775,559869117319671,106724138368245,96039
418091404,8858750951476,254491247582389,

E = 714094877606221

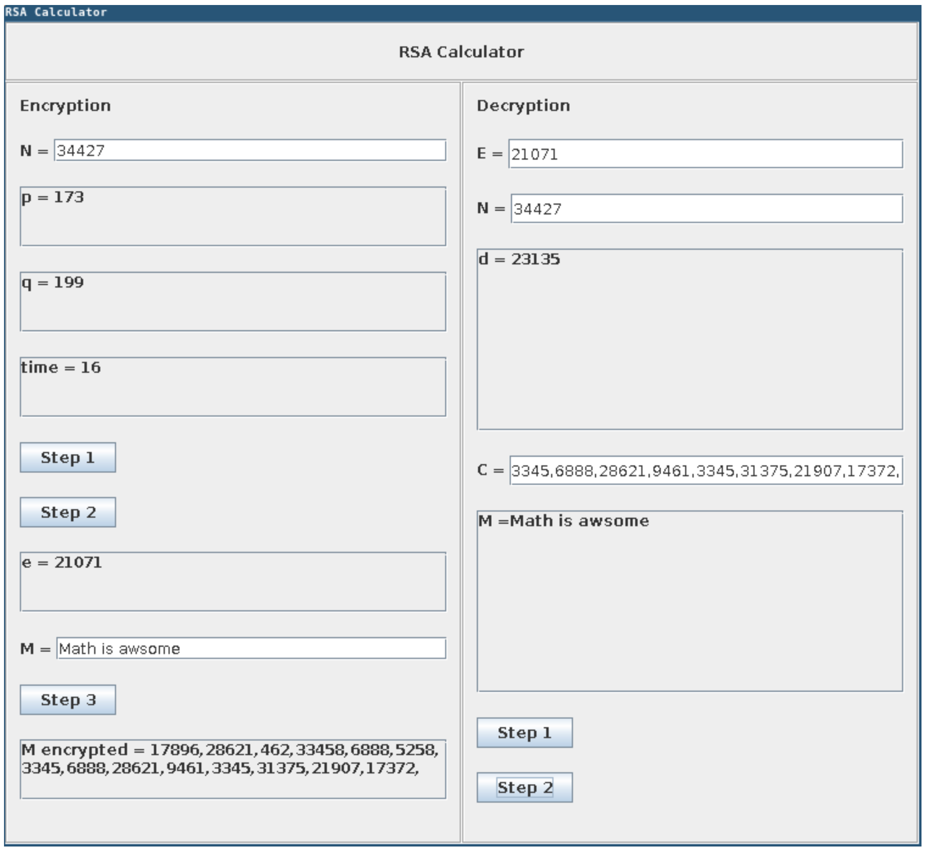
N = 997778179349753

d = 661502284884181

C = 39418091404,8858750951476,254491247582389,

M =Math is amazing

Figuur 2.1: example 1



N = 34427

p = 173

q = 199

time = 16

e = 21071

M = Math is awsome

M encrypted = 17896,28621,462,33458,6888,5258,
3345,6888,28621,9461,3345,31375,21907,17372,

E = 21071

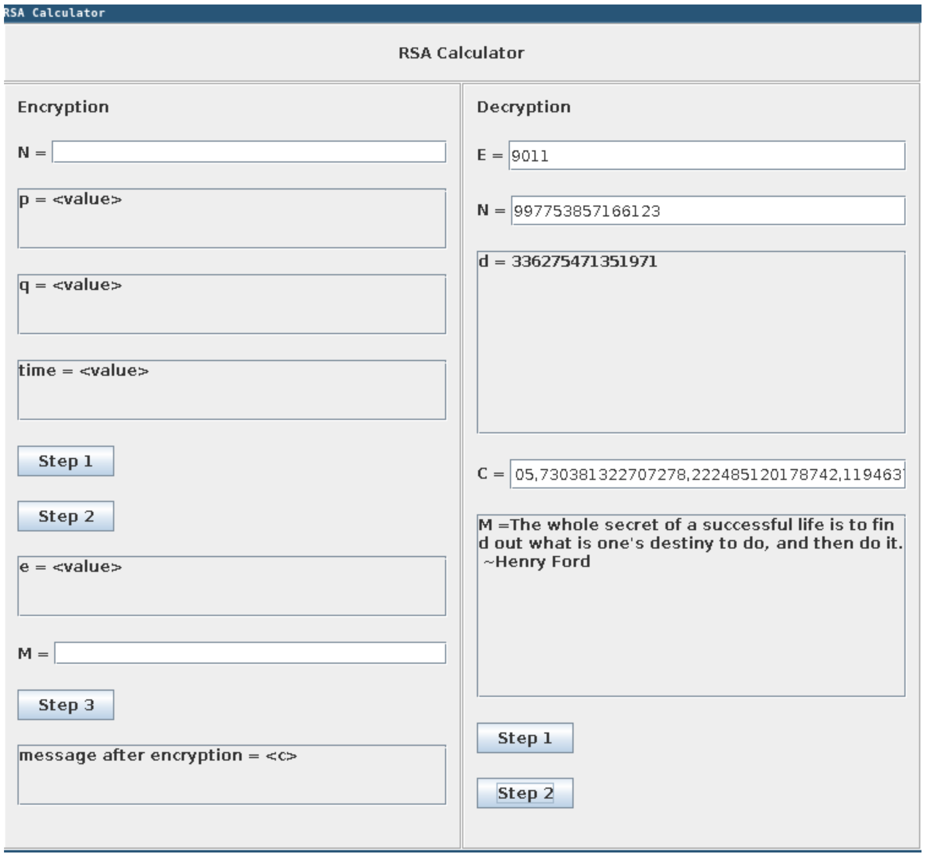
N = 34427

d = 23135

C = 3345,6888,28621,9461,3345,31375,21907,17372,

M =Math is awsome

Figuur 2.2: example 2



E = 9011

N = 997753857166123

d = 336275471351971

C = 05,730381322707278,222485120178742,119463

M =The whole secret of a successful life is to fin
d out what is one's destiny to do, and then do it.
~Henry Ford

Figuur 2.3: example 3