Manual RSA

Hogeschool van Amsterdam

Jan Dorresteijn
500713006

Bas van der Linden
500735218

26 oktober 2020

Inhoudsopgave

1. Introductie

This document contains the manual of the third assignment.

2. Application

2.1 Introduction

This section will go in detail about the working of the system. We created an application that is able to generate a RSA encryption.

2.2 Application

This section will provide information about the setup of the program. Make sure there are no spaces in the input. This will trip the application.

Be also aware that the calculation of P and Q could take a time based on the N. This is also applicable to decryption because P and Q aren't know if E and N is provided. This is way it takes a time to decrypt.

2.2.1 Run

The application can be run by clicking the application.

2.3 RSA

The program differs from an original RSA structure. This is thanks the the input N that will generate the P and Q.

The rsa object has multiple variables that will be shown below.

The note worthy variables are INIT_NUMBER, this will init 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 choose to use Big Interger for the provided methods that come with the Math package. Such methods are modPow, probablePrime and isProbablePrime.

To generate P and Q we use te method calculatePAndQ based on the input N.

```
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";
}
```

With those steps P and q are generated.

Generate E

To generate E we first calculate phi, then we loop and generate an E till E ¡ phi and co-prime to phi.

The code looks as following

```
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 with an ",".

```
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

To Decode we will handle input e and N and calculate P and q based on input. 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

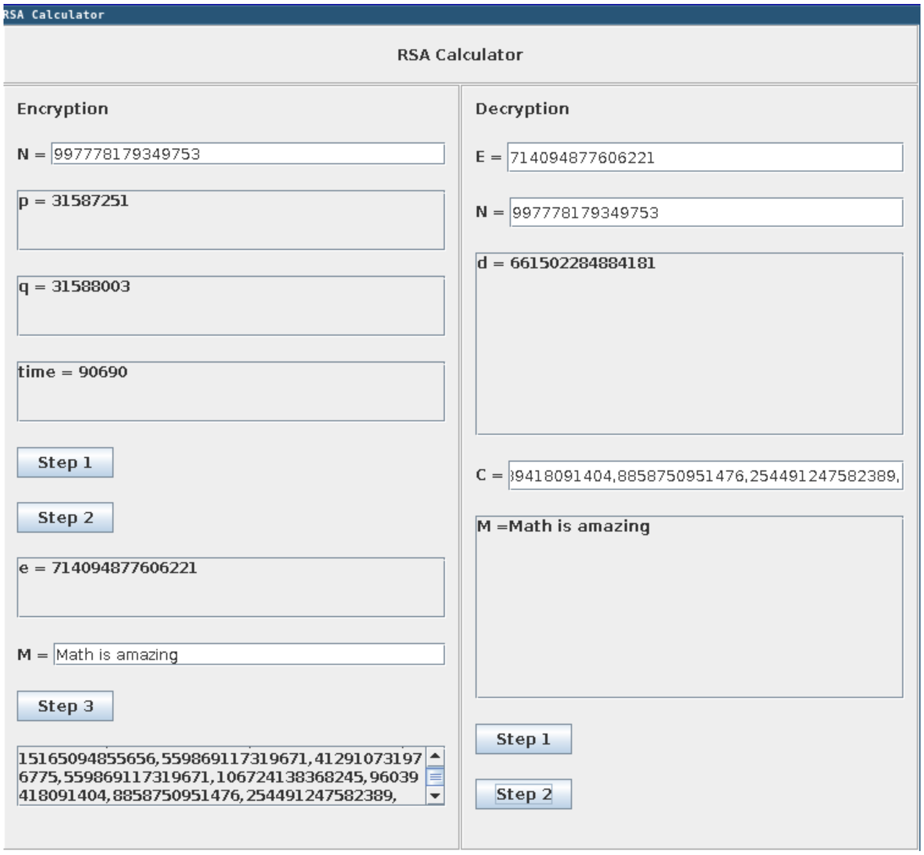
In the first step we will generate a private key, in the final step we will use this private key to decode the cipher.

Finally every char will BigInteger will be cast to a int that will translate to the 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



Encryption

N = 997778179349753

p = 31587251

q = 31588003

time = 90690

e = 714094877606221

M = Math is amazing

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

Decryption

E = 714094877606221

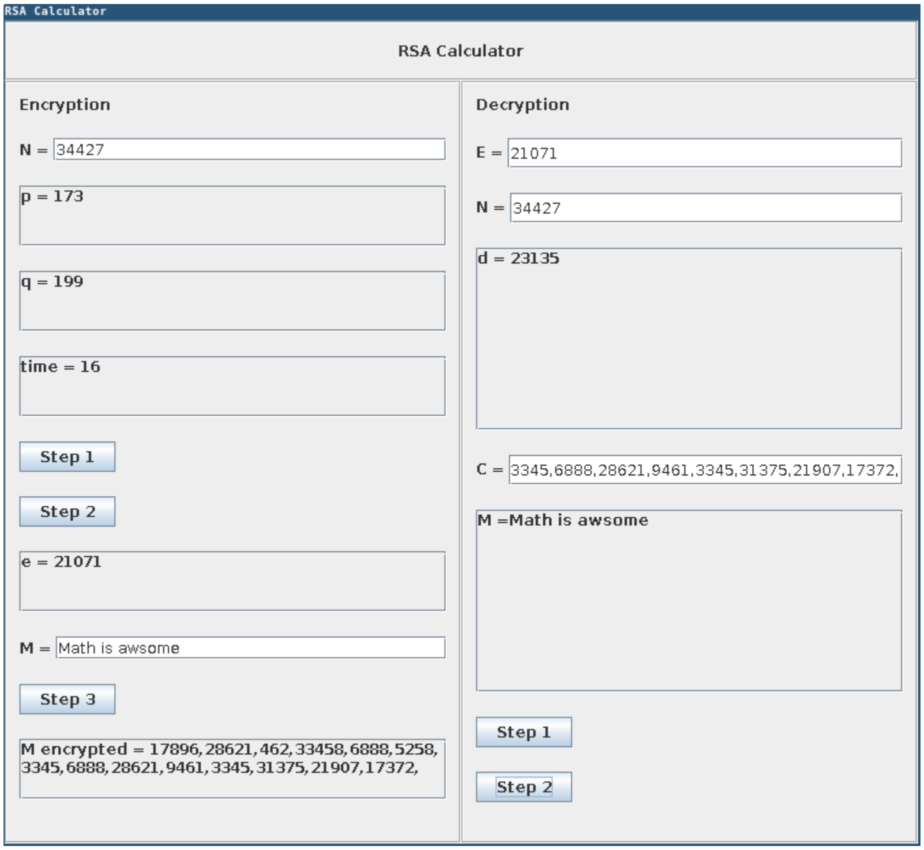
N = 997778179349753

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

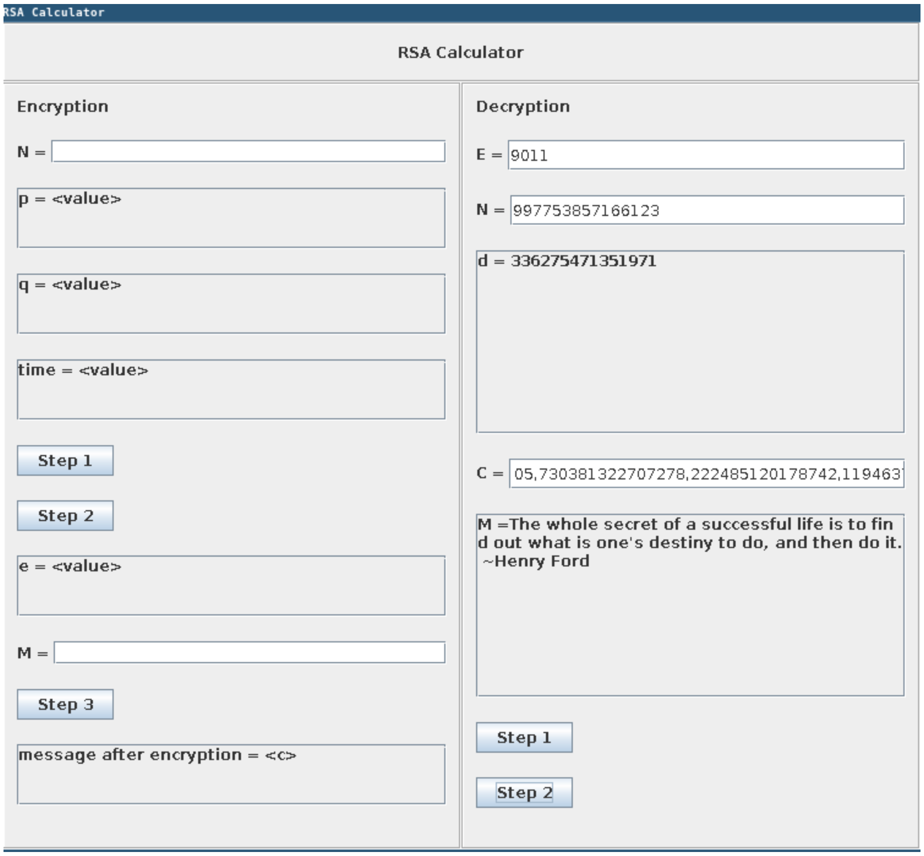
N = 34427

d = 23135

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

M =Math is awsome

Figuur 2.2: example 2



N =

p = <value>

q = <value>

time = <value>

e = <value>

M =

message after encryption = <c>

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