Manual Expert System

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.

2.2.1 Run

To run the application run the following command in a terminal.
java -cp pa3.jar main

2.2.2 RSA

To let the program Handel we decided to use the BigInterger() class. This isn't a primitive type but an object made to handle Big Intergers.

The following block of code will show all the objects the RSA class has. These are used to show on the front-end such as calculateTime.

```java
public class Rsa {
    private final static SecureRandom random = new SecureRandom();
    private final static BigInteger one = new BigInteger("1");

    private int bLength;

    private BigInteger modulus;

    private BigInteger publicKey;
    private BigInteger privateKey;
    private long calculateTime;
}
```

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 a prime we make use of the method probablePrim and isProbablePrime. This is show in the following block of code.

The probablePrime method takes the lengt in bits and a random to generate a candidate that it will return.

Then we will validate the prime with the method isProbablePrime. This will return a boolean if the prime is validated.

With those steps P and q are generated.

Also thanks to the usage of Big Intergers we are able to work with numbers bigger then 32 bits.

```java
begin = System.currentTimeMillis();
BigInteger p = BigInteger.probablePrime(bLength / 2, new Random());
BigInteger q = BigInteger.probablePrime(bLength / 2, new Random());
while(!p.isProbablePrime(100)) {
    p = BigInteger.probablePrime(bLength / 2, new Random());
}
while (!q.isProbablePrime(100)) {
    q = BigInteger.probablePrime(bLength / 2, new Random());
}
end = System.currentTimeMillis();
this.calculateTime = end - begin;
BigInteger Ophase = (p.subtract(one)).multiply(q.subtract(one));

this.modulus = p.multiply(q);
this.publicKey = new BigInteger("65537");
this.privateKey = publicKey.modInverse(Ophase);
```

Thanks to the modpow method we are able to encrypte the message.

```java
public BigInteger encrypt(BigInteger message){
    return message.modPow(publicKey, modulus);
}
```

The method to decrypte the message.

```java
public BigInteger decrypt(BigInteger encrypted){
    return encrypted.modPow(privateKey, modulus);
}
```