

A Report on

Cryptoware: RSA (128 bits)

(Mini-Project of CSE1007-Introduction to Cryptography)

Submitted by

K.Bala Eswar- 19BCN7003

Mounika Sadineni- 19BCE7074

on

14th November 2020



School of Computer Science and Engineering

VIT-AP University, Andhra Pradesh

Abstract

This paper focuses on asymmetric Public key cryptographic method called RSA. RSA is one of the first practical public-key crypto systems and is widely used for secure data transmission. In such a cryptosystem, the encryption key is public and differs from the decryption key which is kept secret. In RSA, this asymmetry is based on the practical difficulty of factoring the product of two large prime numbers, the factoring problem. RSA is made of the initial letters of the surnames of Ron Rivest, AdiShamir, and Leonard Adleman, who first publicly described the algorithm in 1977. A user of RSA creates and then publishes a public key based on two large prime numbers, along with an auxiliary value. The prime numbers must be kept secret. Anyone can use the public key to encrypt a message, but with currently published methods, if the public key is large enough, only someone with knowledge of the prime numbers can feasibly decode the message. Hence second half of the paper discusses on security enhancement of RSA where two ciphers are generated which makes it difficult to decrypt the message as to decrypt the message attacker now need, not only the private key but also random numbers k and s from which new ciphers are calculated.

1 Introduction

Cryptography is playing a major role in data protection in applications running in a network environment. It allows people to do business electronically without worries of deceit and deception in addition to ensuring the integrity of the message and authenticity of the sender. It has become more critical to our day-to-day life because thousands of people interact electronically every day; through e-mail, e-commerce, ATM machines, cellular phones, etc.

RSA encryption and decryption algorithm talks about secrecy of messages between two users. It involves a public key and a private key generation. The public key can be known to everyone and is used for encrypting messages. Messages encrypted with the public key can only be decrypted using the private key. The private key is not publicly known, it is only known to receiver so that she/he can decrypt the encrypted message. These keys (public and private keys) for the RSA algorithm are generated using some mathematical operations.

The principal goal of design of any encryption algorithm must be security against unauthorized attacks. Within the last decade, there has been a vast increase in the accumulation and communication of digital computer data in both the private and public sectors. Much of this information has a significant value, either directly or indirectly, which requires protection. RSA is an Asymmetric key cryptography which provides security against unauthorized attacks.

1.1 About

This project creates two windows. One for Encryption and other for decryption. The user who wants to send message should enter the plain text in Encryption window. It creates a cipher text using public key(e,n) and displays the private key(d,n). Public key is available to everyone. But the private key is only accessible to the specific user. So we should note the d and n values shown under private keys. Now if a user wants to decrypt the plain text, In the decryption window he should enter the cipher text and the private keys(d,n). According to that, the respective plain text will be generated. Here the plain text can be a string, that is converted in bytes inorder to get encrypted. Similarly for decryption the generated byte message is converted into string and that will appear as our decrypted message.

The idea of an asymmetric public-private key cryptosystem is attributed to Whitfield Diffie and Martin Hellman, who published this concept in 1976. They also introduced digital signatures and attempted to apply number theory. Their formulation used a shared-secret-key created from exponentiation of some number, modulo a prime number. However, they left open the problem of realizing a one-way function, possibly because the difficulty of factoring was not well-studied at the time. Ron Rivest, Adi Shamir, and Leonard Adleman at the Massachusetts Institute of Technology, made several attempts over the course of a year to create a one-way function that was hard to invert. The algorithm is now known as RSA – the initials of their surnames in same order as their paper.

1.2 Implementation Environment

Java version 11.0.8.10 is used to implement this project. Java is a programming language and computing platform first released by Sun Micro systems in 1995. There are lots of applications and websites that will not work unless you have Java installed, and more are created every day. Java is fast, secure, and reliable. From laptops to data centers, game consoles to scientific supercomputers, cell phones to the Internet, Java is everywhere!

The Java Runtime Environment (JRE build 11.0.8.1) is what you get when you download Java software. The JRE consists of the Java Virtual Machine (JVM), Java platform core classes, and supporting Java platform libraries. The JRE is the runtime portion of Java software, which is all you need to run it in your Web browser. The installer database contains the logic and data required to install adoptOpenJDK JDK with hotspot 11.0.8.10(64 bit) .

The tables below shows the hardware and software specifications of the project.

S.No	Hardware	Specification
1	Processor	Intel Core i5
2	Ram	8 GB
3	Processor speed	1 GHz

Table 1: Hardware specifications

S.No	Software	Specification
1	JDK	To run JVM
2	IDE	Eclipse 2020-06

Table 2: Software specifications

Specification of input and output data:

1. Input: Plain text for encryption window, Cipher text and the private keys(d,n) for decryption window
2. Output: Private keys and cipher text are generated as output in encryption window, Plain text for decryption window. Public keys are generated that are available to everyone.

2 Algorithm

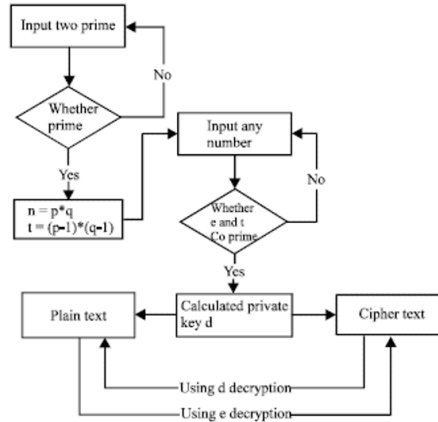


Figure 1: Flowchart for RSA Algorithm .

2.1 Generating the Keys

RSA involves a public key and a private key. The public key can be known by everyone and is used for encrypting messages. Messages encrypted with the public key can only be decrypted in a reasonable amount of time using the private key. The keys for the RSA algorithm are generated the following way:

CHOOSE two distinct prime numbers p and q . For security purposes, the integers p and q should be chosen at random, and should be of similar bit-length. In order to get a 128 bits n value, we should choose 64 bit p and q . Prime integers can be efficiently found using a primality test.

Then calculate n which is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length

$$n = p * q = pq \quad (1)$$

Calculate the totient function;

$$\phi(n) = (p - 1) * (q - 1) \quad (2)$$

Select an integer e , such that $1 < e < \phi(n)$ and e is co-prime to $\phi(n)$.

Calculate d such that

$$d = e^{-1} \text{mod} \phi(n). \quad (3)$$

d can be found using the extended euclidean algorithm. e is the public key available to everyone used for the encryption and d is the secret key used for decryption of the plain text message.

```

RSA_Key_Generation
{
  Select two large prime numbers p and q such that p≠q
  n ← p × q
  Φ(n) ← (p-1) × (q-1)
  Select e such that 1< e< Φ(n) and e is coprime to Φ(n)
  d ← e-1 mod Φ(n) // d is inverse of e modulo Φ(n)
  Public_key ← (e,n) // To be announced publicly
  Private_key ← (d,n) // To be kept secret
  Return Public_key and Private_key
}

```

2.2 Encryption

Alice transmits her public key (n, e) to Bob and keeps the private key secret. Bob then wishes to send message M to Alice. He then computes the cipher text C corresponding to

$$C = P^e \text{mod}(n) \quad (4)$$

This can be done quickly using the method of exponentiation by squaring. Bob then transmits C to Alice.

```

RSA_Encryption(P,e,n)
{
  // P is plain text in Zn and P< n
  C ← Fast_Exponentiation (P,e,n)
  return C // Calculation of (Pe mod n)
}

```

2.3 Decryption

For the generated cipher text C. Using the secret key, the plain text is generated using the equation:

$$P = C^d \bmod(n) \quad (5)$$

If the user enters the correct private key he can be able to see the correct message.

```

RSA_Decryption(C,d,n)
{
  // C is the cipher text in  $Z_n$ 
   $P \leftarrow \text{Fast\_Exponentiation}(C,d,n)$ 
  return P // Calculation of  $(C^d \bmod n)$ 
}

```

3 Major Components

1. KEY Generation saved as RSAGenerator.java

```

package rough;
import java.math.*;
import java.security.SecureRandom;
public class RSAGenerator {
  RSAGenerator()
  {
    super();
  }

  private Key publickey; //Public Key
  private Key privatekey; //Private key
  private static final BigInteger ONE = BigInteger.ONE;
  public RSAGenerator(int numbits){
    //Generating p and q
    BigInteger p = BigInteger.probablePrime(numbits, new SecureRandom());
    BigInteger q = BigInteger.probablePrime(numbits, new SecureRandom());
    //Computing modulus(n=p*q)
    BigInteger n = p.multiply(q);
    //Compute Euler's totient function, phiN

```

```

BigInteger p_minus_one = p.subtract(ONE);
BigInteger q_minus_one = q.subtract(ONE);
BigInteger phiN = p_minus_one.multiply(q_minus_one);
//Calculate public exponent
BigInteger e, d;
do {
e = BigInteger.probablePrime(numbits, new SecureRandom());
} while ((e.compareTo(ONE) == 1) && (e.compareTo(phiN) == -1) && (e.gcd(phiN).compareTo(ONE) != 1));
//Calculate private exponent
d = e.modInverse(phiN);
//Set Keys
publickey = new Key(e,n);privatekey = new Key(d,n);
System.out.println("Public key is: "+publickey);
}

//Encryption method
public BigInteger encrypt(String msg){
return (new BigInteger(msg.getBytes())).modPow(publickey.getComponent(), publickey.getModulus());
}

//Decryption method
public BigInteger decrypt(BigInteger encrypt_msg,BigInteger d,BigInteger xn){
if(d==privatekey.getComponent()&&xn==privatekey.getModulus())
return encrypt_msg.modPow(privatekey.getComponent(), privatekey.getModulus());
return encrypt_msg.modPow(d,xn);
}

//returns private key component
public String toString1(){
return (privatekey.getComponent()).toString();
}

```



```

        //returns private key modulus
        public String toString2(){
return (privatekey.getModulus()).toString();
        }
}

```

1. Encryption and Decryption saved as RSATest.java

```

package rough;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.math.BigInteger;
import javax.swing.*;
import java.awt.*;
import java.util.*;

public class RSATest extends RSAGenerator {
    public static void main(String[] args) throws Exception{
        RSAGenerator rsa_gen = new RSAGenerator(64);
        Scanner sc=new Scanner(System.in);
        JFrame f = new JFrame("Encryption");

        JFrame f1 = new JFrame("Decryption");

        JLabel l1, l2, l3,l11,l22,l33;

        JTextField t1, t2, t3,t4,t11,t22,t33,t44;

        JButton b1, b2, b3,b11,b33;

        l1 = new JLabel("Enter the message");

        l1.setBounds(270, 50, 300, 50);

        l1.setFont(new Font("TimesNewRoman", Font.BOLD, 15));

```

```
l11 = new JLabel("Enter Cipher text");

l11.setBounds(270, 50, 300, 50);

l11.setFont(new Font("TimesNewRoman", Font.BOLD, 15));

l2 = new JLabel("Encrypted message");

l2.setBounds(270, 200, 300, 50);

l2.setFont(new Font("TimesNewRoman", Font.BOLD, 15));

l22 = new JLabel("Enter private keys(d,n)");

l22.setBounds(270, 150, 300, 50);

l22.setFont(new Font("TimesNewRoman", Font.BOLD, 15));

l3 = new JLabel("Private keys(d,n)");

l3.setBounds(270, 300, 300, 50);

l3.setFont(new Font("TimesNewRoman", Font.BOLD, 15));

l33 = new JLabel("Plain text");

l33.setBounds(290, 300, 300, 50);

l33.setFont(new Font("TimesNewRoman", Font.BOLD, 15));

t1 = new JTextField();
```

```
t1.setBounds(150, 100, 350, 30);

t11 = new JTextField();

t11.setBounds(150, 100, 350, 30);

t2 = new JTextField();

t2.setBounds(150, 250, 350, 30);

t22 = new JTextField();

t22.setBounds(30, 200, 300, 30);

t44 = new JTextField();

t44.setBounds(340, 200, 300, 30);

t3 = new JTextField();

t3.setBounds(30, 350, 300, 30);

t4 = new JTextField();

t4.setBounds(340, 350, 300, 30);

t33 = new JTextField();

t33.setBounds(150, 350, 350, 30);

b1 = new JButton("Encrypt");
```

```

b1.setBounds(250, 150, 150, 30);

b11 = new JButton("Decrypt");

b11.setBounds(250, 260, 150, 30);

b3 = new JButton("close");

b3.setBounds(270, 400, 100, 30);

b33 = new JButton("close");

b33.setBounds(270, 400, 100, 30);

b1.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e)

    {
        String d = t1.getText();
        String encrypted = rsa_gen.encrypt(d).toString();
        t2.setText(encrypted);
        t3.setText(rsa_gen.toString1());t4.setText(rsa_gen.toString2());
        t11.setText(encrypted);
    }

});

b11.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e)

    {

```

```

String mn = t11.getText(); String md = t22.getText(); String nn = t44.getText();
BigInteger dd= new BigInteger(md); BigInteger n= new BigInteger(nn);
BigInteger decrypt = rsa_gen.decrypt(new BigInteger(mn), dd,n);
String decrypted = "";
for(byte b: decrypt.toByteArray()){
    decrypted += (char) b;
}

    t33.setText(decrypted);
}

});
b3.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e)

{

f.dispose();

}

});
b33.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e)

{

f.dispose();

}

});

```

```
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
f1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

f.add(l1);

f.add(t1);

f.add(l2);

f.add(t2);

f.add(l3);

f.add(t3);

f.add(t4);

f.add(b1);

f.add(b3);

f.setLayout(null);

f.setSize(750, 550);

f.setVisible(true);

f1.add(l11);

f1.add(t11);

f1.add(l22);
```

```
f1.add(t22);

f1.add(t44);

f1.add(l33);

f1.add(t33);

f1.add(b11);

f1.add(b33);

f1.setLayout(null);

f1.setSize(750, 550);


f1.setVisible(true);

}

}
```

4 Results

The output of the program are depicted in Figure 2 , 3 and 4. In 2 User encrypts message using public key and displays the private key(d,n) that should be copied. In 3 User tries to decrypt the message by entering cipher text and the private keys(d,n). In 4 we can see the public Key visible to everyone.


 Encryption
 —
□
×

Enter the message

Encrypted message

Private keys(d,n)

Figure 2: Computations at Sender-side (Alice).

 Decryption
 —
□
×

Enter Cipher text

Enter private keys(d,n)

Plain text

Figure 3: Computations at Receiver-side (Bob).

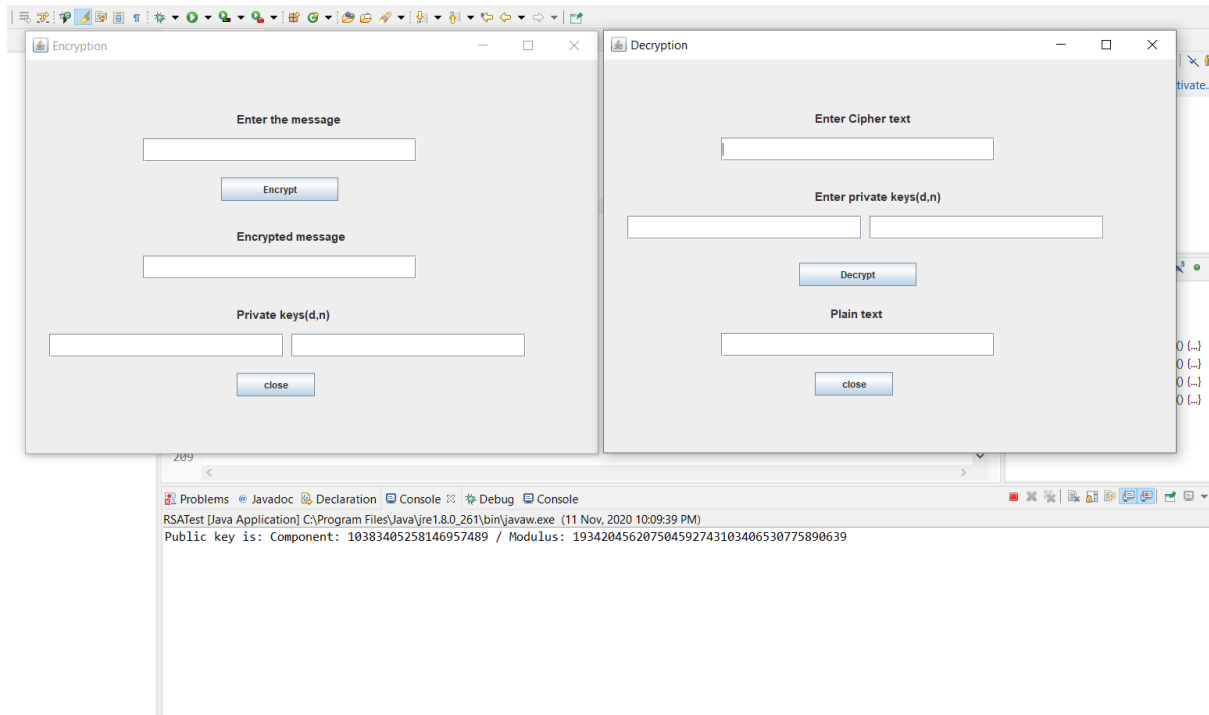


Figure 4: RSA Encryption and Decryption.

References

- [1] Behrouz A Forouzan, Debdeep Mukhopadhyay, “Cryptography and Network Security”, Mc Graw Hill, Third Edition, 2015.
- [2] William Stallings, “Cryptography and Network Security: Principles and Practice”, Pearson Education, Seventh Edition, 2017.
- [3] Overleaf: a collaborative cloud-based LaTeX editor used for writing, editing and publishing scientific documents. <http://www.overleaf.com>.
- [4] <https://www.sanfoundry.com/java-program-implement-rsa-algorithm/>
- [5] <https://www.geeksforgeeks.org/rsa-algorithm-cryptography/>