Triple RSA Encryption

MINT CAPSTONE PROJECT 2015

2/5/2016

Ilakkiya Velusamy

Table of Contents

Table of Contents

Abstract

The **Rivest Shamir Adelman** (RSA) is a popular cryptosystem. Mainly used for its easy implementation in securing data transmissions. In RSA cryptography, two different keys are used; a private and public key. The public keys can be shared with everyone where the private key is a secret. RSA has been used for online payments, banking applications, various secure authentications processes and used by many government organizations.

This work consists of describing a new approach to enhance the security of RSA using an old approach. In this we enhanced the security feature by introducing an advance version of RSA called Triple RSA. To enhance the security and confidentiality we have implemented the Triple RSA in a similar approach to 3DES. This provides integrity and a strong authentication.

3DES is a modern variation of DES (Data Encryption Standard) which uses a block of plaintext 64 bits in length, with a 56 bit key. The actual key length equals that of the plaintext. The last bit on the right of the key is a parity bit. There were many concerns about the weakness of DES against brute force attacks due to the key length, so 3DES was developed in response to needing a stronger encryption method. 3DES works in much the same way as DES, except that goes through three cycles during the encryption process using three keys: encryption, decryption, and another encryption. It has a key length of 192 bits (64 bits x 3 keys), but its actual strength is 168 bits (56 bits x 3 keys). Encrypt with Key1, decrypt with Key2 and encrypt with Key3. There are two-key and three-key versions. This method is three times as strong as DES.

1. Cryptography

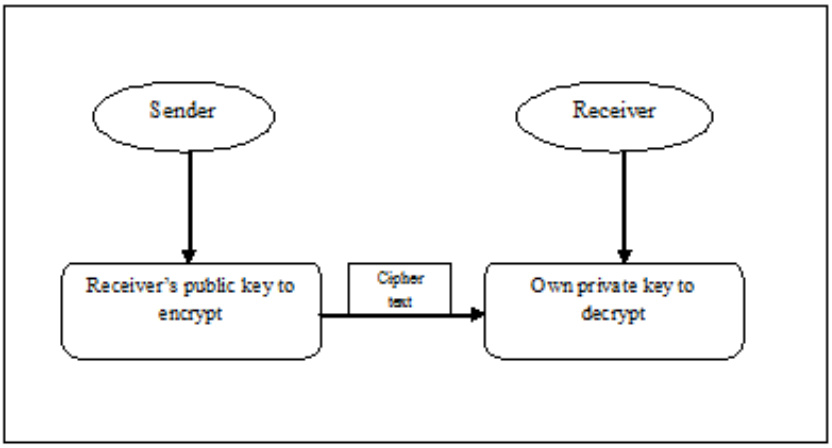
The public key cryptosystem was introduced in **1976** by **Whitfield Diffie** and **Martin Hellman**. It uses public key for encryption as well as a private key for decryption. Each user gets two keys one public and one private. The public key is published and the private key is secret. There is no need to share the private key. The challenge would be that two parties would have to agree on a secret key without anyone else finding out. The secret key method is faster but only moderately secure.

Diffie-Hellman is an asymmetric key algorithm used for public key cryptography. The Diffie-Hellman algorithm was created to address the issue of secure encrypted keys from being attacked over the internet during transmission. The process works by two peers generating a private and a public key. Peer A would send its public key to peer B and peer B would send its

public key to peer A. Peer A would then use the public key sent from peer B and its own private key to generate a symmetric key using the Diffie-Hellman algorithm. Peer B would also take the same process as peer A and in turn produce the exact same symmetric key as peer A, though enabling them to communicate securely over the in-secure internet. Both peers can now encrypt, transmit and decrypt data using their symmetric keys. DH approach is used here to securely exchange the triple RSA public keys between two parties.

2. RSA

RSA is a public key algorithm invented in **1977** and it was publicly described to everyone in 1978 by **Ron Rivest**, **Adi Shamir** and **Leonard Adleman** at MIT. Encryption is the very efficient way for data security mainly in military and in banking online transactions. It is very difficult for unauthorized access to occur when the cipher text is stronger. The encryption and decryption of RSA works as, both the sender and receiver will share their public keys with each other. The sender will send an encrypted message to the receiver, using the receiver's public key. The receiver will decrypt the message by using his/her private key.

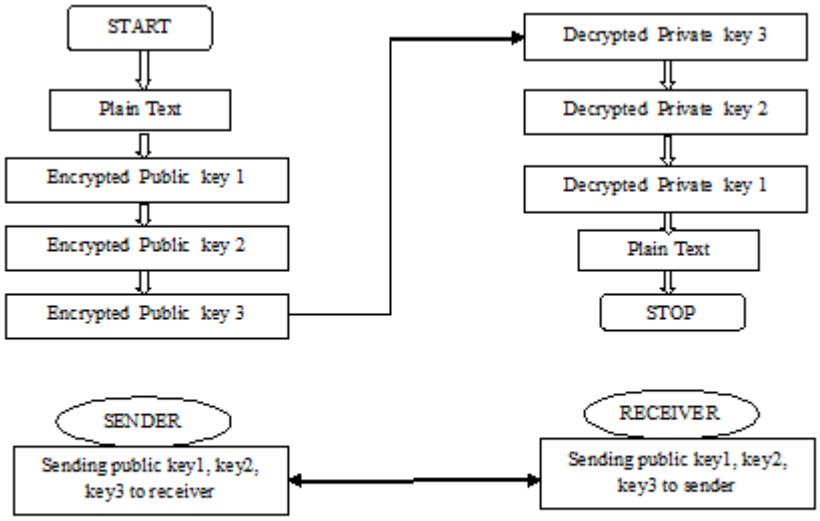


RSA Encryption decryption

3. Triple RSA encryption:

Triple RSA is an improved version of RSA. It provides more security in receiving the original private/public 3RSA key set from the issuing authority than RSA. Initially both the parties will exchange their public keys with each other. The message is first encrypted using receiver's public Key1. The receiver only has the corresponding private key to decrypt and he/she only can decrypt it. This step provides confidentiality. The encrypted message is further encrypted by using receiver's public Key2. It can be decrypted on the receiver's side using receiver's private key.

Finally, the double encrypted message is again encrypted using receiver's public Key3. The data can only be decrypted using receiver's private Key3, Key2, and Key1 respectively. Hence the message cannot be modified by unauthorized access. Hence the data integrity is achieved.



Triple RSA Model

Advantages:

Each user has sole responsibility for protecting his or her private key. The purpose of the strong algorithms and keys is to make the process of breaking the encryption take so long that the data's time value is reduced to nothing.

3a. Generating Public and private key

Explanation:

Source has only one public key and private key for RSA encryption and decryption.

Source:

```
r = new Random();
p = BigInteger.probablePrime(bitlength, r);
q = BigInteger.probablePrime(bitlength, r);
 N = p.multiply(q);
phi=p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
e = BigInteger.probablePrime(bitlength/2, r);
while (phi.gcd(e).compareTo(BigInteger.ONE) > 0 && e.compareTo(phi) < 0
)
{
   e.add(BigInteger.ONE);
 }
```

```
d = e.modInverse(phi);
```

Explanation:

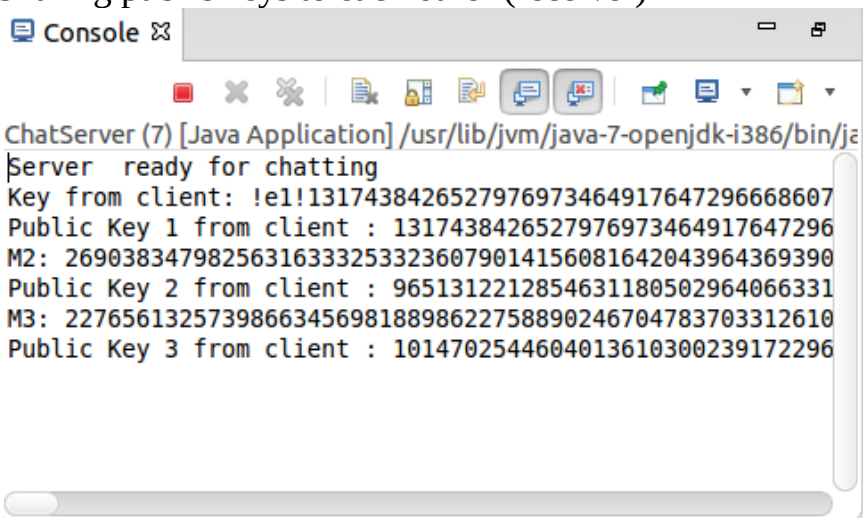
Modified code has functionality to generate 3 public keys and 3 private keys to implement triple RSA encryption and decryption

Modified:

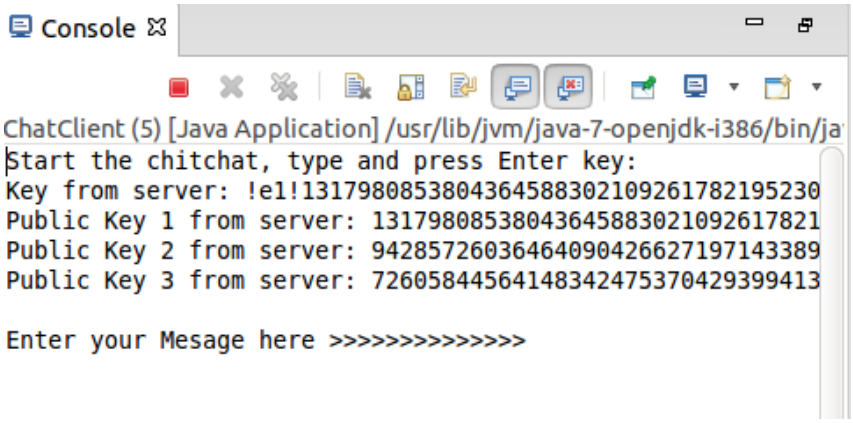
```
r = new SecureRandom();
//Generate public key and private key 1
p1 = BigInteger.probablePrime(bitlength, r);
q1 = BigInteger.probablePrime(bitlength, r);
N1 = p1.multiply (q1);
phi1=p1.subtract(BigInteger.ONE).multiply(q1.subtract(BigInteger.ONE));
e1 = BigInteger.probablePrime(bitlength/2, r);
while(phi1.gcd(e1).compareTo(BigInteger.ONE) > 0 && e1.compareTo(phi1)
< 0 )
 {
    e1.add(BigInteger.ONE);
}
d1 = e1.modInverse(phi1);
//Generate public key and private key 2
p2 = BigInteger.probablePrime(bitlength, r);
q2 = BigInteger.probablePrime(bitlength, r);
N2 = p2.multiply(q2);
phi2 = p2.subtract(BigInteger.ONE).multiply(q2.subtract(BigInteger.ONE));
e2 = BigInteger.probablePrime(bitlength/2, r);
while (phi2.gcd(e2).compareTo(BigInteger.ONE) > 0 &&
                    e2.compareTo(phi2) < 0 ) {
          e2.add(BigInteger.ONE);
}
d2 = e2.modInverse(phi2);
//Generate public key and private key 3
p3 = BigInteger.probablePrime(bitlength, r);
q3 = BigInteger.probablePrime(bitlength, r);
N3 = p3.multiply(q3);
phi3 = p3.subtract(BigInteger.ONE).multiply(q3.subtract(BigInteger.ONE));
e3 = BigInteger.probablePrime(bitlength/2, r);
while (phi3.gcd(e3).compareTo(BigInteger.ONE) > 0 &&
                                          e3.compareTo(phi3) < 0 ) {
e3.add(BigInteger.ONE);
}
d3 = e3.modInverse(phi3);
```

Output:

Sharing public keys to each other (receiver)



Sharing public keys to each other (sender)



3b. Encrypt using public key

Explanation:

Source will encrypt the plain text only once using one public key and send this cipher text for decryption

Source:

```
byte[] encrypted = rsa.encrypt(teststring.getBytes());
System.out.println("Encrypted String in Bytes: " + bytesToString(encrypted));
```

Explanation:

Modified code will encrypt the plain text 3 times using 3 different public keys

a. Plain text to Ciphertext1 using public key 1
b. Ciphertext1 to Ciphertext2 using public key 2
c. Ciphertext2 to Ciphertext3 using public key 3

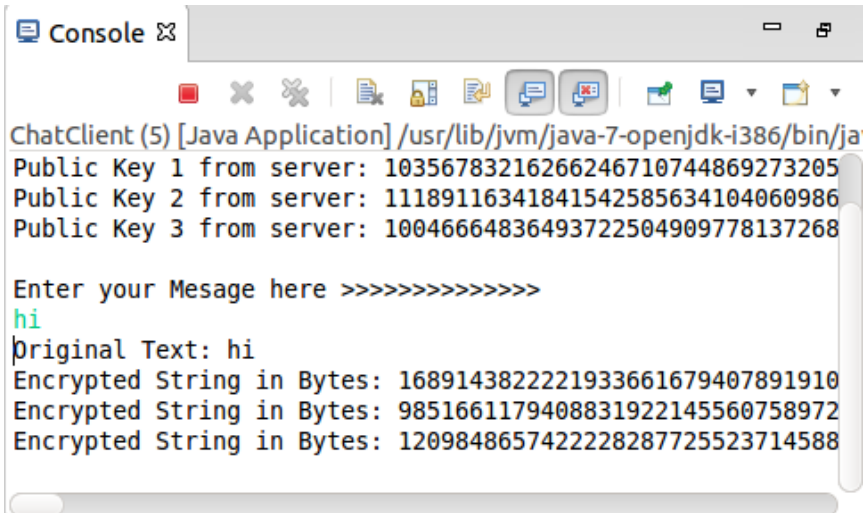
Modified code will take this Ciphertext3 for decryption

Modified:

```
// encrypt using clients publickey1
  BigInteger encrypted1 = rsa.encrypt1(new
                    BigInteger(sendMessage.getBytes()),cs.publicKey1,cs.cN1);
   System.out.println("Encrypted String in Bytes: " + encrypted1);
// encrypt using clients publickey2
  BigInteger encrypted2 = rsa.encrypt2(encrypted1,cs.publicKey2,cs.cN2);
   System.out.println("Encrypted String in Bytes: " + encrypted2);
// encrypt using clients publickey3
   BigInteger encrypted3 = rsa.encrypt3(encrypted2,cs.publicKey3,cs.cN3);
   System.out.println("Encrypted String in Bytes: " +encrypted3);
   pwrite.println(encrypted3);
    BigInteger encrypted2 = rsa.encrypt2(encrypted1,cs.publicKey2,cs.cN2);
    System.out.println("Encrypted String in Bytes: " + encrypted2);
```

Output:

Sender is sending a message which is encrypted as a string and sent to the receiver.



3c. Decrypt using private key

Explanation:

Source code will decrypt the cipher text to plain text only once using one private key

Source:

```
byte[] decrypted = rsa.decrypt(encrypted);
```

```
System.out.println("Decrypted String in Bytes: " + bytesToString(decrypted));
System.out.println("Decrypted String: " + new String(decrypted));
```

Explanation:

Modified code will decrypt the cipher text 3 times using 3 different private keys

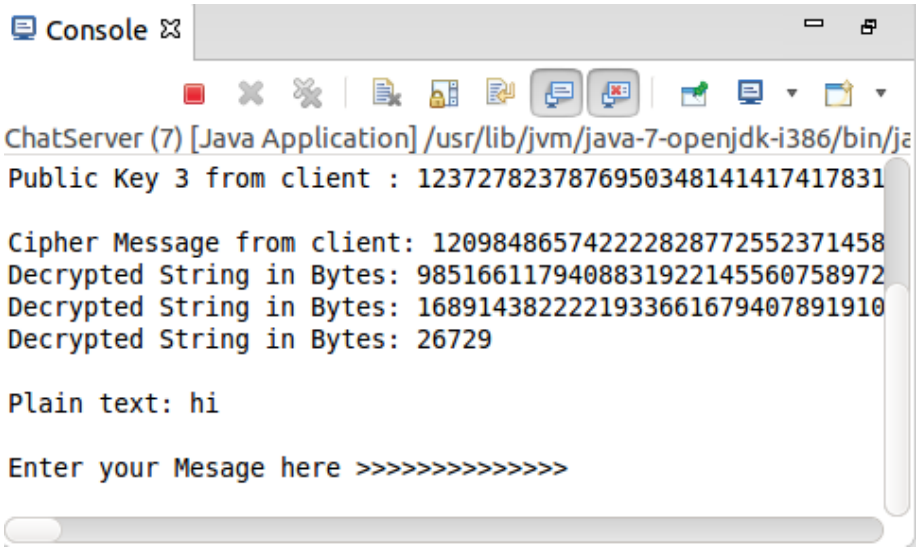
a. Ciphertext3 to Ciphertext2 using private key 3
b. Ciphertext2 to Ciphertext1 using private key 2
c. Ciphertext1 to Plain text using private key 1

Modified:

```
// decrypt using server privatekey3
BigInteger decrypted3 = rsa.decrypt3(new
        BigInteger(receiveMessage),cs.privateKey3,cs.sN3);
System.out.println("Decrypted String in Bytes: " +  decrypted3);

// decrypt using server privatekey2
BigInteger decrypted2 = rsa.decrypt2(decrypted3,cs.privateKey2,cs.sN2);
System.out.println("Decrypted String in Bytes: " +  decrypted2);
// decrypt using server privatekey1
BigInteger decrypted1 = rsa.decrypt1(decrypted2,cs.privateKey1,cs.sN1);
System.out.println("Decrypted String in Bytes: " +  decrypted1);
String text2 = new String(decrypted1.toByteArray());
        System.out.println("\nPlain text: " + text2);
```

Output:

Receiver decrypted the message by using key3, key2, key1



3d. Two way communication (Server, Client)

Explanation:

Source code has a below logics,

a. Creating connection
b. Waiting for client and accepting the connection
c. Send and receive plain message from client to server and vice versa .
d. Display messages in console

Source:

```
 // reading from keyboard (keyRead object)
   BufferedReader keyRead = new BufferedReader(new
                                                InputStreamReader(System.in));
   // sending to client (pwrite object)
   OutputStream ostream = sock.getOutputStream();
   PrintWriter pwrite = new PrintWriter(ostream, true);
  // receiving from server ( receiveRead  object)
   InputStream istream = sock.getInputStream();
   BufferedReader receiveRead = new BufferedReader(new
InputStreamReader(istream));
   String receiveMessage, sendMessage;
   while(true)
   {       if((receiveMessage = receiveRead.readLine()) != null)
     {          System.out.println(receiveMessage);
     }

     sendMessage = keyRead.readLine();
     pwrite.println(sendMessage);
     pwrite.flush();
   }
```

Explanation:

Modified code has a following logics,

a. Creating connection.
b. Waiting for client and accepting the connection .
c. Server and client will Generate 3 public and 3 private keys , both will send 3 public keys in one string for encryption (from server to client and vice versa) as soon as server accepts the connection.
d. Both server and client will parse the public key string into 3 public keys and display the keys in console.
e. When client receives plain text from user, it will encrypt the message 3 times using server's public Key1,Key2, Key3 and client will send cipher text to server

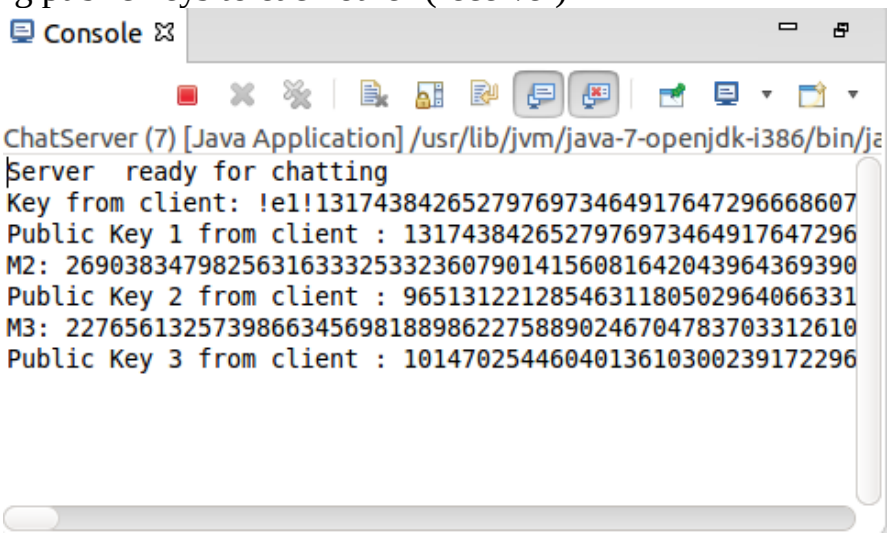
f. When server receives cipher text from client, it will decrypt the cipher text 3 times using server's own private key3,Key2,Key1 and display the plain text in console

Modified:

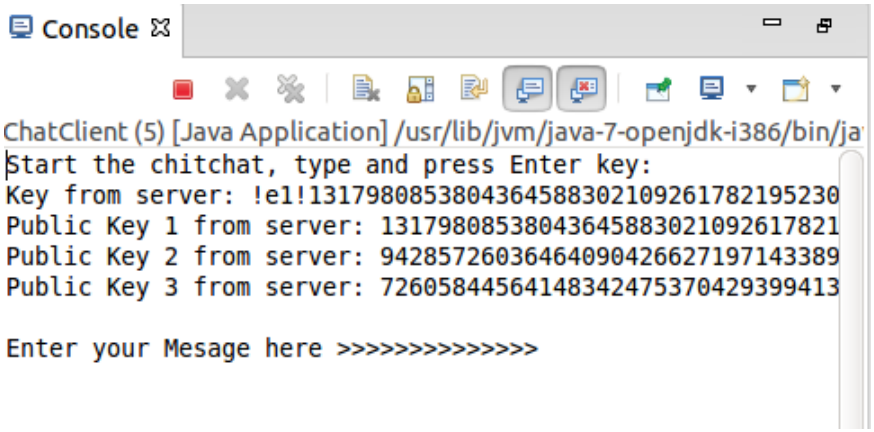
```
        PrintWriter pwrite = new PrintWriter(ostream, true);
        RSATestNew rsa = new RSATestNew();
        ChatServer cs = new ChatServer();
        String keyToClient = cs.keyGen();
        pwrite.println(keyToClient);
        pwrite.flush();
     // reading from keyboard (keyRead object)
        BufferedReader keyRead = new BufferedReader(new
    InputStreamReader(System.in));
     // receiving from server ( receiveRead  object)
        InputStream istream = sock.getInputStream();
        BufferedReader receiveRead = new BufferedReader(new
    InputStreamReader(istream));
         String receiveMessage, sendMessage;
        receiveMessage = receiveRead.readLine();
        if(receiveMessage.contains("!e1!")) {
            System.out.println("Key from client: " + receiveMessage);
            String[] textArray= receiveMessage.split("=");

            String[] key1 = textArray[0].split("!m1!");
            cs.publicKey1 = new BigInteger(key1[0].replace("!e1!", ""));
            cs.cN1 = new BigInteger(key1[1].replace("!m1!", ""));
            System.out.println("Public Key 1 from client : " + cs.publicKey1);

            String[] key2 = textArray[1].split("!m2!");
            cs.publicKey2 = new BigInteger(key2[0].replace("!e2!", ""));
            cs.cN2 = new BigInteger(key2[1].replace("!m2!", ""));
            System.out.println("M2: " + cs.cN2);
            System.out.println("Public Key 2 from client : " + cs.publicKey2);

            String[] key3 = textArray[2].split("!m3!");
       cs.publicKey3 = new BigInteger(key3[0].replace("!e3!", ""));
            cs.cN3 = new BigInteger(key3[1].replace("!m3!", ""));
            System.out.println("M3: " + cs.cN3);
            System.out.println("Public Key 3 from client : " +  cs.publicKey3);

    }
```

Output:

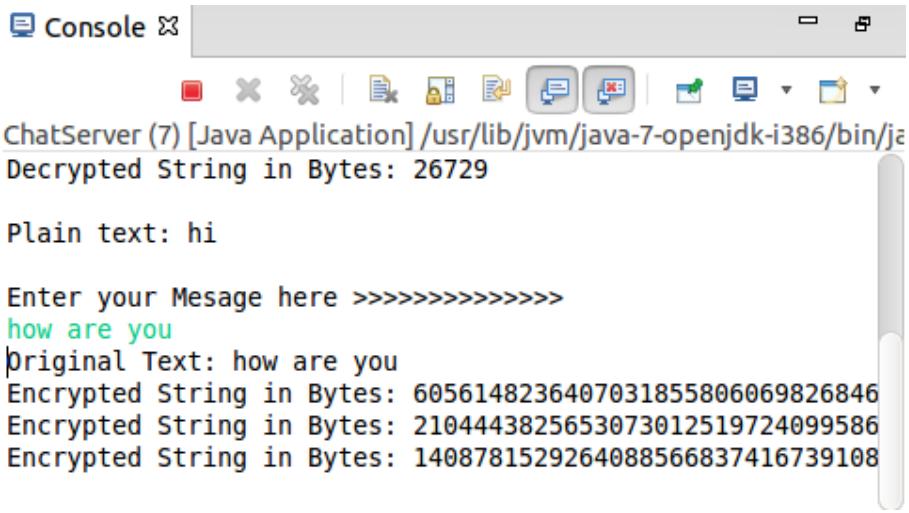
Sharing public keys to each other (receiver)



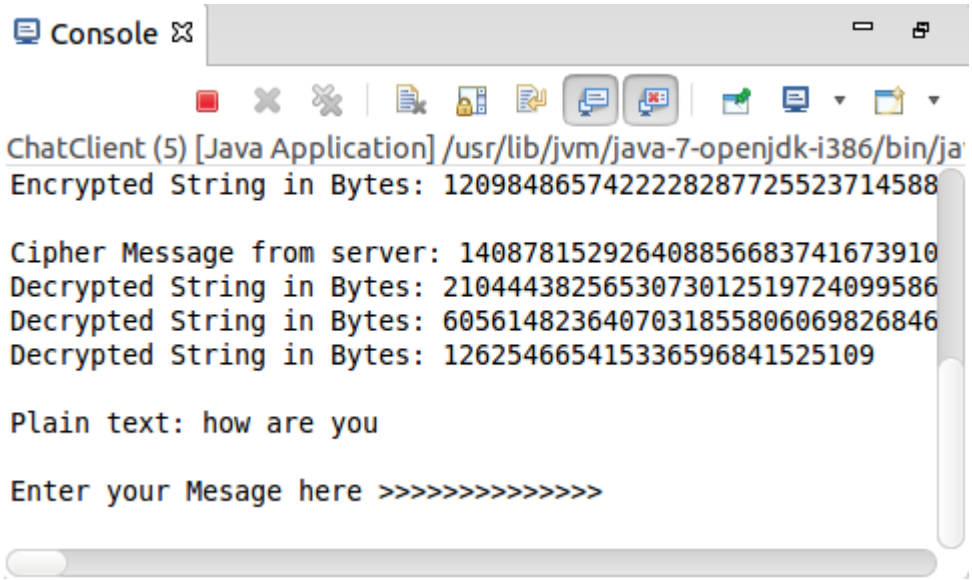
Sharing public keys to each other (sender)



Receiver is responding back to the sender. So the message is encrypted by using public key1, key2, key3 and sent.



Sender receives a message and it is decrypted again by using the key3, key2 and key1.



4. Generation of Keys:

The keys generation for the RSA algorithm are described as,

1. Consider two different prime numbers as p and q.
   - The integer p and q should be chosen at random for security reason and should be similar in magnitude and different in length.
2. To Compute "n=p*q"
   - "n" is used as the modulus for both the public and private keys.

3. Compute $\varphi(n)$
   $\varphi(n) = (p - 1)(q - 1)$
   Where φ is Euler's totient function. This value is kept private.
4. Choose an integer e such that $1 < e < \varphi(n)$
   - e is released as the public key exponent.
5. Determine d as $d = e^{-1} \pmod{\varphi(n)}$
   - d is kept as the private key exponent.

The public key consists of the modulus n and the public (or encryption) exponent e. The private key consists of the modulus n and the private (or decryption) exponent d, which must be kept secret. p, q, and $\varphi(n)$ must also be kept secret because they can be used to calculate the encryption value.

4a). Calculation Example

Example of RSA encryption and decryption:

1. Choose two distinct prime numbers, such as
   P = 7 and Q=11
2. Compute n = p*q giving
   n = P * Q = 3 * 11 = 77
3. Compute the totient of the product as $\varphi(n) = (p - 1)(q - 1)$ giving
   $\varphi(n) = (7-1)(11-1) = 60$
4. Choose any number $1 < e < 60$
   Let e = 13
5. Compute d, the modular multiplicative inverse of e (mod $\varphi(n)$) yielding,
   d = 37

   e * d mod $\varphi(n) = 1$
   13 * 37 mod 77 = 1

- Encrypt the word "art"

Note: Refer Appendix B

Let's consider **a=3**,

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28

Letter -> 'a'

$C = P^e \bmod n$
$= 3^{13} \bmod 77$
$= 1594323 \bmod 77$
$= 38$

Letter -> ‘r’

$C = P^e \bmod n$

$= 20^{13} \bmod 77$

$= 69$

Letter ->‘t’

$C = P^e \bmod n$

$= 22^{13} \bmod 77$

$= 282810057883082752 \bmod 77$

$= 22$

- Decrypt the word “art”

Let’s consider,

Letter -> ‘a’

$P = C^d \bmod n$

$= 38^{37} \bmod 77$

$=28313468473157736370011296127792731029354930758695159595008 \bmod 77$

$P = 3$

Let’s consider,

Letter -> ‘r’

$P = C^d \bmod n$

$= 69^{37} \bmod 77$

$=10899745490831950129333219729105806745673345796234565134738715329918 9 \bmod 77$

$P = 20$

Let’s consider,

Letter ->‘t’

$P = C^d \bmod n$

$= 22^{37} \bmod 77$

$= 467346710759721379726441891785142018041353921331 2 \bmod 77$

$P =22$

Note: Refer Appendix B

Let’s consider **a=0**,

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

- Encrypt the word “art”

Letter -> ‘a’

$C = P^e \bmod n$

$= 0^{13} \bmod 77$

$= 0$

Letter -> ‘r’

$C = P^e \bmod n$

$= 17^{13} \bmod 77$

$= 73$

Letter ->‘t’

$C = P^e \bmod n$

$= 19^{13} \bmod 77$

$= 61$

- Decrypt the word “art”

Letter -> ‘a’

$P = C^d \bmod n$

$= 0^d \bmod 77$

$P = 0$

Letter -> ‘r’

$P = C^d \bmod n$

$= 73^{37} \bmod 77$

$P = 17$

Letter ->‘t’

$P = C^d \bmod n$

$= 61^{37} \bmod 77$

$P = 19$

Note: Refer Appendix B

Let’s consider **a = 1**,

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

- Encrypt the word “art”

Letter -> ‘a’

$C = P^e \bmod n$

$= 1^{13} \bmod 77$

$= 1$

Letter -> ‘r’

$C = P^e \bmod n$

$= 18^{13} \bmod 77$

$= 46$

Letter ->‘t’

$C = P^e \bmod n$

$= 20^{13} \bmod 77$

$= 69$

- Decrypt the word “art”

Letter -> ‘a’

$P = C^d \bmod n$

$= 1^{37} \bmod 77$

$P = 1$

Letter -> ‘r’

$P = C^d \bmod n$

$= 46^{37} \bmod 77$

$P = 18$

Letter ->‘t’

$P = C^d \bmod n$

$= 69^{37} \bmod 77$

$P = 20$

Appendix A

1).To find the‘d’ value
p = 7
q = 11
n = 7 x 11 = 77
φn = (7-1) x (11-1) = 60

e = 13
d = 37

To find the value of “d”:

- Let’s take the value φ(n)

Step 1:
φ(n) value on top row→ 60 60
‘e’ value second row → 13 1 ← Let’s consider the value as’1’
Divide the left side value → 4

Step 2:
Multiply the quotient ‘4’ with the second row values
4 *13 = 52 4*1 =4
Step 3:
Subtract the first row value with the multiplied values
60-52=8 60-4=56

8, 56

- Repeat all 3 steps until you get the value “1”.

Step 1:
Previous calculations second row values
Should be on first row→ 13 1
Previous step3 calculation values on
Second row → 8 56

Divide the left side value → 1

Step 2:
Multiply the quotient ‘1’ with the second row values
8 *1 = 8 56*1 =56

Step 3:

Subtract the first row value with the multiplied values

13-8=8 1-56= - 55

Here we found negative value " -55". So we have to take mod φ(n) for the negative value.

-55 mod 60 = 5

5, 5

- Repeat all 3 steps until you get the value "1".

Step 1:

Previous calculations second row values

Should be on first row→ 8 56

Previous step3 calculation values on

Second row → 5 5

Divide the left side value → 1

Step 2:

Multiply the quotient '1' with the second row values

5 *1 = 5 5*1 =5

Step 3:

Subtract the first row value with the multiplied values

8-5=3 56-5= 51

3, 51

- Repeat all 3 steps until you get the value "1".

Step 1:

Previous calculations second row values

Should be on first row→ 5 5

Previous step3 calculation values on

Second row → 3 51

Divide the left side value → 1

Step 2:

Multiply the quotient '1' with the second row values

3*1 = 3 51*1 =51

Step 3:

Subtract the first row value with the multiplied values

5-3=2 5-51= - 46

Here we found negative value " -46". So we have to take mod φ(n) for the negative value.

-46 mod 60 = 14

2, 14

- Repeat all 3 steps until you get the value "1".

Step 1:

Previous calculations second row values

Should be on first row→ 3 51

Previous step3 calculation values on

Second row → 2 14

Divide the left side value → 1

Step 2:

Multiply the quotient '1' with the second row values

2*1 = 2 14*1 =14

Step 3:

Subtract the first row value with the multiplied values

3-2=1 51-14= 37

1, 37

Here we reached the value "1". Then the 'd' value is "37"

d= 37

Encrypt:

$C = P^e \text{ MOD } n$

$C = 3^{13} \text{ MOD } 77$

$C = 38$

Decrypt:
$P = C^d \text{ MOD } n$
$P = 38^{37} \text{ MOD } 77$
$P = 3$

Appendix B

If the off-set value **a=3** then the plain text will be encrypted by using public Key1, Key2, Key3 and decrypt by using private Key3, Key2, Key1. The keys will scramble the message in plaintext to cipher text and to the original message in plain text from cipher text.

Let's consider **a=3**,

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28

Encrypt:

1) Key 1

   Key 1 (7,11)

   P= 7 and Q=11 , n =77 , e=13, d=37

   - Encrypt "a"

     $C = P^e \text{ mod } n$
     $= 3^{13} \text{ mod } 77$
     $= 1594323 \text{ mod } 77$
     $= 38$

2) Key 2

   Key 2 (7,13)

   P= 7 and Q=13 , n =91 , e=11, d=59

   - Encrypt "a"

     $C = P^e \text{ mod } n$
     $= 3^{11} \text{ mod } 91$
     $= 61$

3) Key 3

   Key 3 (13, 11)

   P= 13 and Q=11, n =33, e=7, d=103

➢ Encrypt "a"

$C = P^e \bmod n$

$= 3^7 \bmod 33$

$= 9$

Decrypt:

1) Key 3

Key 3 (13, 11)

P= 13 and Q=11, n =33, e=7, d=103

➢ Decrypt "a"

$P = C^d \bmod n$

$= 9^{103} \bmod 33$

$= 3$

2) Key 2

Key 2 (7,13)

P= 7 and Q=13 , n =91 , e=11, d=59

➢ Decrypt "a"

$P = C^d \bmod n$

$= 61^{59} \bmod 91$

$= 3$

3) Key 1

Key 1 (7,11)

P= 7 and Q=11 , n =77 , e=13, d=37

➢ Decrypt "a"

$P = C^d \bmod n$

$= 38^{37} \bmod 77$

$= 3$

If off-set value **a=1,** the keys will scramble the plain text into the same text as cipher. So the message transmission will not be much stronger.

Let's consider **a=1**,

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Encrypt:

1) Key 1

Key 1 (7,11)

P= 7 and Q=11 , n =77 , e=13, d=37

- Encrypt "a"

$C = P^e \bmod n$

$= 1^{13} \bmod 77$

$= 1$

2) Key 2

Key 2 (7,13)

P= 7 and Q=13 , n =91 , e=11, d=59

- Encrypt "a"

$C = P^e \bmod n$

$= 1^{11} \bmod 91$

$= 1$

3) Key 3

Key 3 (13, 11)

P= 13 and Q=11, n =33, e=7, d=103

- Encrypt "a"

$C = P^e \bmod n$

$= 1^7 \bmod 33$

$= 1$

Decrypt:

1) Key 3

Key 3 (13, 11)

P= 13 and Q=11, n =33, e=7, d=103

- Decrypt "a"

$P = C^d \bmod n$

$= 1^{103} \bmod 33$

$= 1$

2) Key 2

Key 2 (7,13)

P= 7 and Q=13 , n =91 , e=11, d=59

- Decrypt "a"

$P = C^d \bmod n$

$= 1^{59} \bmod 91$

$=1$

3) Key 1

Key 1 (7,11)

P= 7 and Q=11 , n =77 , e=13, d=37

- Decrypt "a"

$P = C^d \bmod n$

$= 1^{37} \bmod 77$

$= 1$

If off-set value **a=0,** the keys will scramble the plain text into the same text as cipher while encryption. So the message/password will be easy to break.

Let's consider **a=0**,

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Encrypt:

1) Key 1

Key 1 (7,11)

P= 7 and Q=11 , n =77 , e=13, d=37

- Encrypt "a"

$C = P^e \bmod n$

$= 0^{13} \bmod 77$

$= 0$

2) Key 2

Key 2 (7,13)

P= 7 and Q=13 , n =91 , e=11, d=59

- Encrypt "a"

$C = P^e \bmod n$

$= 0^{11} \bmod 91$

$= 0$

3) Key 3

Key 3 (13, 11)

P= 13 and Q=11, n =33, e=7, d=103

- Encrypt "a"

$C = P^e \bmod n$

$= 0^7 \bmod 33$

$= 0$

Decrypt:

1) Key 3

Key 3 (13, 11)

P= 13 and Q=11, n =33, e=7, d=103

- Decrypt "a"

$P = C^d \bmod n$

$= 0^{103} \bmod 33$

$= 0$

2) Key 2

Key 2 (7,13)

P= 7 and Q=13 , n =91 , e=11, d=59

- Decrypt "a"

$P = C^d \bmod n$

$= 0^{59} \bmod 91$

$=0$

3) Key 1

Key 1 (7,11)

P= 7 and Q=11 , n =77 , e=13, d=37

- Decrypt "a"

$P = C^d \bmod n$

$= 0^{37} \bmod 77$

$= 0$

References

1. http://scanftree.com/programs/java/implementation-of-rsa-algorithmencryption-and-decryption-in-java/
2. http://www.gripinit.com/2015/04/03/rsa-algorithm-and-implementation/
3. http://way2java.com/networking/chat-program-two-way-communication/
4. http://www.java2s.com/Code/Java/Security/SimpleRSApublickeyencryptionalgorithmimplementation.htm
5. http://ayyostream.blogspot.com/2012/01/how-to-generate-random-prime-number-in.html
6. http://security.stackexchange.com/questions/15991/using-rsa-with-3des-instead-of-plain-3des-does-it-make-sense
7. http://www.emc.com/emc-plus/rsa-labs/standards-initiatives/triple-des.htm
8. https://en.wikipedia.org/wiki/Key_size
9. https://en.wikipedia.org/wiki/RSA_(cryptosystem) - calculation
10. https://asecuritysite.com/encryption/rsa?val=11%2C3%2C3%2C4
11. http://searchsecurity.techtarget.com/definition/asymmetric-cryptography
12. http://searchsecurity.techtarget.com/definition/RSA
13. http://www.iosrjen.org/Papers/vol2_issue7%20(part-1)/L0277277.pdf
14. http://www.di-mgt.com.au/rsa_alg.html
15. https://en.wikipedia.org/wiki/Key_generation
16. https://en.wikipedia.org/wiki/Cryptography
17. https://defuse.ca/big-number-calculator.htm
18. https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange
19. http://www.internet-computer-security.com/VPN-Guide/Diffie-Hellman.html
20. http://codenamekidnextdoor.blogspot.ca/2011/09/explaining-triple-data-encryption.html
21. http://searchsecurity.techtarget.com/tip/Expert-advice-Encryption-101-Triple-DES-explained
22. https://www.secpoint.com/what-is-diffie-hellman-encryption.html
23. http://www.math.ucla.edu/~baker/40/handouts/rev_DH/node1.html