Pragmatic Implementation of RSA-1024 Cryptography

Sanjeev Karmakar
Bhilai Institute of Technology, Durg, Bhilai House, 491001, Chhattisgarh, India

Siddhartha Choubey
Shri Shankaracharya Technical Campus / Shri Shankaracharya group of Institutions, Bhilai, Durg, India

ABSTRACT
Contemporary RSA-1024 system is used in the current world’s computer networks for secure communication and transmission. The RSA-1024 system is absolutely secure from the intruders. However, it required huge computational effort during encryption and decryption. This can be solved by the fast exponential method however again required mathematical space and time complexity. Java mathematical tool provides BigInteger APIs are significantly suitable to solve this complexity problem. In this paper RSA public key (e, n) and private key (d, n) used as 1024 bits large prime product. The encryption and decryption is accomplished through the methods of BigInteger class. In this paper, pragmatic RSA-2048 cryptosystem is presented for e-mail service. The entire design, implementation and deployment is presented in this paper.

Keywords
RSA, RMI, Cryptography, Encryption, Decryption, Network, Security, RSA-1024, NFS, ECM

1. INTRODUCTION
The RSA cryptosystem is a public-key cryptosystem that provides both encryption and digital signatures (authentication). Ronald Rivest, Adi Shamir, and Leonard Adleman developed the RSA system in 1977; RSA stands for the first letter in each of its inventors’ last names. Its security is supported on the intractability of the integer factorization on the problem [1, 2] which will be discussed in this chapter afterward sections. Surveys by Rivest [3] and Brassard [4, 5] form an excellent introduction to modern cryptography. Some textbook treatments are provided by Stinson [6] and Stallings [7], while Simmons provides an in-depth coverage of the technical aspects of cryptography [8]. A comprehensive review of modern cryptography can also be found in Applied Cryptography [9]; Ford [10,11] provided detailed reporting of subject for example cryptography standards and secure communication.

In RSA algorithm, to generate public keys, encryption RSA algorithm selects two large prime numbers p, q such that n = p·q and ψ = (p – 1)·(q – 1), and e ∈ 1, ϕ<e<ψ, such that gcd (e, ψ) = 1. After that algorithm used Euclidean algorithm to compute the unique integer d, 1<d<ψ, such that e·d = 1 mod ψ. Through that e public key (n, e) and private key is d. These e and d RSA key generation are called the encryption exponent and the decryption exponent respectively, while n is called the modulus.

To send message (m), where m ∈ 1, e·m ∈ 0, n – 1 interval, compute cipher text c = m^e (mod n) at sender end this task is known as encryption. At the receiver and again compute c^d (mod n) = m to get the original message (m), this task is known as decryption. To proof decryption task, since e·d = 1 (mod ψ), £ k ∈ 1 such that e·d = 1 + k·ψ. Now if gcd (m, p) = 1 then by Fermat’s theorem mp-1 = mod p. Raising both side of this congruence to the power of k(q-1) and then multiplying both side by m yields

\[ m^{1+k·(p-1)(q-1)} = m \pmod{p} \]

On the other hand, if gcd(m,p) = p, then this last congruence is again valid since each side congruence to 0 modulo p.

Hence in all cases

\[ m^{ed} = m \pmod{n} \]

By same argument \( m^{ed} \equiv m \pmod{q} \). Finally, since q and q are distinct primes, it follows that \( (m^q)^d \equiv m \pmod{n} \); and hence \( e^d \equiv (m^q)^d \equiv m \pmod{n} \).

Here, single client is connected with the server. In general, in the for all computer network applications, often multiple clients are connected with sever. For multiple client server applications it is noted that server computes unique public as well as private key for each connected clients. Following Figure 1 shows how server will send unique public key for each connected clients and maintain unique private key for each client as well. At this juncture, three clients are connected with the server is shown. Server will generate three set of public and private key for each connected clients. It is cleared that, each client have their own pair of key for the secure communication. This feature produces security potency of the system as well. In the subsequent sections of the chapter the detail design and Java code of a multi client server RSA cryptography system is also discussed. Readers of this book may use this code and exhibit the system in their machine or LAN. In the next section, an example of RSA cryptography is explained.

Fig.1. Multi-client Server based RSA-1024 Cryptography System
2. JAVA BIG INTEGER
Java BigInteger provided analogues to all of Java's primitive integer operators, and all relevant methods from java.lang.Math. Additionally, BigInteger provides operations for modular arithmetic, GCD calculation, primality testing, prime generation, bit manipulation, and a few other miscellaneous operations. Modular arithmetic operations are provided to compute residues, perform exponentiation, and compute multiplicative inverses. It has been found the implementation of RSA algorithm can be easily obtained from following code segment. Detail of implementation procedure as well as code is given in the subsequent sections of this chapter.

```java
int size1 = size/2;
int size2 = size1;
int offset1 = (int)(5.0*(rnd.nextDouble()) + 5.0);
int offset2 = -offset1;
if (rnd.nextDouble() < 0.5) {
    offset1 = -offset1;
    offset2 = -offset2;
}
size1 += offset1; size2 += offset2;
// generate two random primes, so that p*q = n
has size bits
BigInteger p1= new BigInteger(size1, rnd); // random int
p= nextPrime(p1);
BigInteger phi1 = p.subtract(BigInteger.ONE);
BigInteger q1 = new BigInteger(size2, rnd);
q= nextPrime(q1);
BigInteger phi2 = q.subtract(BigInteger.ONE);
phi = phi1.multiply(phi2); // (p-1)*(q-1)
BigInteger e= THREE;
d= e.modInverse(phi);
```

Where rnd is random number is obtained by Random class which is defined in java.util package. And next prime is obtained from following piece of code.

```java
public BigInteger nextPrime(BigInteger x)
{
    if (!(x.isProbablePrime(10)) break;
    x = x.add(TWO);
    return x;
}
```

Encryption and Decryption is obtained from following code segment

```java
public BigInteger RSAEncrypt(BigInteger c)
{
    return c.modPow(d, n);
}
public BigInteger RSADecrypt(BigInteger c)
{
    return m.modPow(e, n);
}
```

An application has been developed to demonstrate RSA cryptography in multi-client server environment as shown in above Figure 1. Wherein, multiple clients are connected with the server to send text message to the server in specific user id. For that server generates unique public key as well as private key for each client as discussed above in the section 2 and depicted in Figure 1. Encrypted text message is transferred to the server from the client. Server accepts the cipher text. Decrypts it to converts original message by using corresponding private key of the client. And save into the mail box (server database). To obtain this objective RSA algorithm has been implemented by using Java. It is found that Java math package supports public class BigInteger class (for detail see package java.math package) that is appropriate for such big numbers calculations. BigInteger provides analogues to all of Java's primitive integer operators, and all relevant methods from java.lang.Math. Additionally, BigInteger provides operations for modular arithmetic, GCD calculation, primality testing, prime generation, bit manipulation, and a few other miscellaneous operations.

3. REMOTE METHOD INVOCATION (RMI)
Java Remote Method Invocation (RMI) technology is used to create multi client server based network system. Input and output interfaces are developed by using Java swing. Another alternative is Java socket is not adopted to develop distributed nature of the system because the problems with sockets essentially stem from the fact that they are very primitive.

Not part of Object Model. They don't fit in with the object model - we are explicitly creating and using [low-level] communications links. This is at odds with our use of an object-oriented programming language, based around the invocation of methods belonging to objects. RMI allows us to continue to use our object-oriented paradigm even though we are dealing with distributed systems. (On the other hand, there is a growing view that distributed programming is not well-served by the object model - driven by the web services model.)

Point-to-point. Each socket only represents a communication link between two hosts. There is no mechanism to, say, find out where specific services are offered - you must simply know.

Data only. On the face of it, you can only send data over socket connections (and you would have to go to some trouble to send more than simple data). There is in fact a way of sending more complex objects - methods as well as data - known as serialization. Serialization with RMI or with 'raw' sockets can be use. However, there is no clear, standard mechanism to, for example, invoke methods over sockets.

We could, of course, construct a software layer on top of simple sockets. This would allow us to locate hosts offering particular services, using some kind of 'clearing house' system. It would then allow us to invoke methods directly on remote objects. This is precisely what Java RMI does for us.

The basic structure of an RMI-based method call involves a client, a server and a registry. To make a call to a remote object, the client first looks up the object it wishes to invoke a method on in the registry. The registry returns a reference to the
object (assuming it exists) on the server, which the client can use to invoke any methods that the remote object implements. The client communicates with the remote object via a User-defined interface that is actually implemented by the remote object. The client actually does not deal directly with the remote object at all, but with a code stub that deals with the process of communication between client and server (using, in the default case, sockets).

At the server end, in pre-Java 2 JDKs (before JDK 1.2), a code skeleton dealt with client communication. In Java 2, the skeleton is no longer needed. Fortunately, the code stub (and skeleton if necessary) are generated automatically by the RMI Compiler rmic. Remote objects can be invoked with parameters, naturally - these parameters can be entire objects, complete with methods that are passed using serialization. The basic structure is shown in Figure 2.

The RMI server must implement the remote object’s interface, and hence it must implement the methods in that interface. (It can in fact implement other methods as well, but such additional methods will only be available to other objects on the server to call - only those methods declared in the interface will be accessible remotely.) In addition, it is commonly the case that remote objects extend the class java.rmi.server.UnicastRemoteObject, which provides the default behavior required for RMI-based communication in 'typical' cases. It is also possible for you to define your own such behavior, but that is beyond the scope of this module. In addition to the methods of the interface, the server must also undertake a certain amount of initialization - make the remote object(s) available by binding them into the registry.

The basic structure is shown in Figure 2. At the server end, in pre-Java 2 JDKs (before JDK 1.2), a code skeleton dealt with client communication. In Java 2, the skeleton is no longer needed. Fortunately, the code stub (and skeleton if necessary) are generated automatically by the RMI Compiler rmic. Remote objects can be invoked with parameters, naturally - these parameters can be entire objects, complete with methods that are passed using serialization. The basic structure is shown in Figure 2.

The RMI server must implement the remote object’s interface, and hence it must implement the methods in that interface. (It can in fact implement other methods as well, but such additional methods will only be available to other objects on the server to call - only those methods declared in the interface will be accessible remotely.) In addition, it is commonly the case that remote objects extend the class java.rmi.server.UnicastRemoteObject, which provides the default behavior required for RMI-based communication in 'typical' cases. It is also possible for you to define your own such behavior, but that is beyond the scope of this module. In addition to the methods of the interface, the server must also undertake a certain amount of initialization - make the remote object(s) available by binding them into the registry.

The RMI server must implement the remote object's interface, and hence it must implement the methods in that interface. (It can in fact implement other methods as well, but such additional methods will only be available to other objects on the server to call - only those methods declared in the interface will be accessible remotely.) In addition, it is commonly the case that remote objects extend the class java.rmi.server.UnicastRemoteObject, which provides the default behavior required for RMI-based communication in 'typical' cases. It is also possible for you to define your own such behavior, but that is beyond the scope of this module. In addition to the methods of the interface, the server must also undertake a certain amount of initialization - make the remote object(s) available by binding them into the registry.

The RMI server must implement the remote object's interface, and hence it must implement the methods in that interface. (It can in fact implement other methods as well, but such additional methods will only be available to other objects on the server to call - only those methods declared in the interface will be accessible remotely.) In addition, it is commonly the case that remote objects extend the class java.rmi.server.UnicastRemoteObject, which provides the default behavior required for RMI-based communication in 'typical' cases. It is also possible for you to define your own such behavior, but that is beyond the scope of this module. In addition to the methods of the interface, the server must also undertake a certain amount of initialization - make the remote object(s) available by binding them into the registry.

The RMI server must implement the remote object's interface, and hence it must implement the methods in that interface. (It can in fact implement other methods as well, but such additional methods will only be available to other objects on the server to call - only those methods declared in the interface will be accessible remotely.) In addition, it is commonly the case that remote objects extend the class java.rmi.server.UnicastRemoteObject, which provides the default behavior required for RMI-based communication in 'typical' cases. It is also possible for you to define your own such behavior, but that is beyond the scope of this module. In addition to the methods of the interface, the server must also undertake a certain amount of initialization - make the remote object(s) available by binding them into the registry.

The RMI server must implement the remote object's interface, and hence it must implement the methods in that interface. (It can in fact implement other methods as well, but such additional methods will only be available to other objects on the server to call - only those methods declared in the interface will be accessible remotely.) In addition, it is commonly the case that remote objects extend the class java.rmi.server.UnicastRemoteObject, which provides the default behavior required for RMI-based communication in 'typical' cases. It is also possible for you to define your own such behavior, but that is beyond the scope of this module. In addition to the methods of the interface, the server must also undertake a certain amount of initialization - make the remote object(s) available by binding them into the registry.

The RMI server must implement the remote object's interface, and hence it must implement the methods in that interface. (It can in fact implement other methods as well, but such additional methods will only be available to other objects on the server to call - only those methods declared in the interface will be accessible remotely.) In addition, it is commonly the case that remote objects extend the class java.rmi.server.UnicastRemoteObject, which provides the default behavior required for RMI-based communication in 'typical' cases. It is also possible for you to define your own such behavior, but that is beyond the scope of this module. In addition to the methods of the interface, the server must also undertake a certain amount of initialization - make the remote object(s) available by binding them into the registry.

The RMI server must implement the remote object's interface, and hence it must implement the methods in that interface. (It can in fact implement other methods as well, but such additional methods will only be available to other objects on the server to call - only those methods declared in the interface will be accessible remotely.) In addition, it is commonly the case that remote objects extend the class java.rmi.server.UnicastRemoteObject, which provides the default behavior required for RMI-based communication in 'typical' cases. It is also possible for you to define your own such behavior, but that is beyond the scope of this module. In addition to the methods of the interface, the server must also undertake a certain amount of initialization - make the remote object(s) available by binding them into the registry.

The RMI server must implement the remote object's interface, and hence it must implement the methods in that interface. (It can in fact implement other methods as well, but such additional methods will only be available to other objects on the server to call - only those methods declared in the interface will be accessible remotely.) In addition, it is commonly the case that remote objects extend the class java.rmi.server.UnicastRemoteObject, which provides the default behavior required for RMI-based communication in 'typical' cases. It is also possible for you to define your own such behavior, but that is beyond the scope of this module. In addition to the methods of the interface, the server must also undertake a certain amount of initialization - make the remote object(s) available by binding them into the registry.

The RMI server must implement the remote object's interface, and hence it must implement the methods in that interface. (It can in fact implement other methods as well, but such additional methods will only be available to other objects on the server to call - only those methods declared in the interface will be accessible remotely.) In addition, it is commonly the case that remote objects extend the class java.rmi.server.UnicastRemoteObject, which provides the default behavior required for RMI-based communication in 'typical' cases. It is also possible for you to define your own such behavior, but that is beyond the scope of this module. In addition to the methods of the interface, the server must also undertake a certain amount of initialization - make the remote object(s) available by binding them into the registry.

4. ARCHITECTURE OF RSA-1024
Object oriented analysis (OOD) is concerned here with developing system engineering requirements and specifications that spoken as a system's class/object model, functional model. Class description or class models have been prepared for the server and client. These are shown in Table 1 and 2 respectively. And prepared object model and functional model is depicted in the Figure 3 and 4 respectively.

4.1 Centralized database
Microsoft access database system is used to create database for the system wherein following database tables (DB_Table) are defined and used. Java JDBC-ODBC Bridge (see following code segment) is used to provide connection between server interface and database. Wherein, ServerDatabase is the Data Source Name (DSN). Users of this system have to create a DSN from Microsoft Windows control panel Administrative Tool. Administrative Tool provides a shortcut Data Source (ODBC) will provide following interface to create DSN.

```java
try {
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    con= DriverManager.getConnection("jdbc:odbc:ServerDatabase");
    st= con.createStatement();
} catch(Exception e) {
    System.out.println("Error:"+e);
}
```
5. DEPLOYMENT
To install the system in personal computer or local network carefully check following checksum.

1. All machine in network and connected through TCP/IP protocol, having unique InetAddress.
2. Jdk1.5.0 is already installed in the server as well as server machine.
3. Set path in autoexec.bat file path= “C:\jdk1.5.0\bin” in server as well as client machine.

5.1 Server Installation
Following steps are needed to install the server in server machine.

1. Create a directory in server machine say C:\Server.
2. Copy following files in C:\Server.
3. Start rmiregistry by command- C:\Server\start rmiregistry
4. Create a DSN named by ServerDatabase in server machine and set path “C:\Server\ImapRSAmailBox.mdb”.
5. Start server by command- C:\Server\StartServer.
6. Server interface is depicted in Figure 5 (a).

5.2 Client Installation
Following steps are needed to install the client program in the client machine.

1. Create directory in client machine C:\Client.
2. Copy following class file in the directory C:\Client.
3. Make sure that your server machine InetAddress has been taken in client program name by class ImapRSAMailWin to lookup servers object.
4. Start client by command- C:\Client\StartClient.

Client interface is depicted in Figure 5 (b)

6. DISCUSSIONS
The security strength of RSA is based on integer factorization problem. In mathematics, the RSA numbers are a set of large semiprimes (numbers with exactly two prime factors) that are
part of the RSA Factoring Challenge. The challenge was to find the prime factors but it was declared inactive in 2007 [12]. It was created by RSA Laboratories in March 1991 to encourage research into computational number theory and the practical difficulty of factoring large integers. RSA Laboratories published a number of semiprimes with 100 to 617 decimal digits. Cash prizes of varying size were offered for factorization of some of them. The smallest RSA number was factored in a few days. Most of the numbers have still not been factored and many of them are expected to remain unfactored for quite some time. As of November 2010, 16 of the 54 listed numbers have been factored: the 15 smallest from RSA-100 to RSA-200, plus RSA-768. The RSA challenge officially ended in 2007 but people can still attempt to find the factorizations. According to RSA Laboratories, “Now that the industry has a considerably more advanced understanding of the cryptanalytic strength of common symmetric-key and public-key algorithms, these challenges are no longer active.” [13] Some of the smaller prizes had been awarded at the time. The remaining prizes were retracted.

The first RSA numbers generated, from RSA-100 to RSA-500, were labeled according to their number of decimal digits. Later, beginning with RSA-576, binary digits are counted instead. An exception to this is RSA-617, which was created prior to the challenge in the numbering scheme. The numbers are listed in increasing order below.

<table>
<thead>
<tr>
<th>Table 3. RSA Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA-100</td>
</tr>
<tr>
<td>RSA-170</td>
</tr>
<tr>
<td>RSA-230</td>
</tr>
<tr>
<td>RSA-360</td>
</tr>
<tr>
<td>RSA-450</td>
</tr>
</tbody>
</table>

In this paper, above code is given for RSA-1024 has 1,024 bits (309 decimal digits) cryptographic system. The system generates RSA-1024 bits prime product ‘n’, and has not been factored so far. US$100,000 was previously offered for factorization. Successful factorization of RSA-1024 has important security implications for many users of the RSA public-key authentication algorithm, as the most common key length currently in use is 1024 bits. To know detail of RSA challenges and related research area visit url http://www.rsa.com/rsalabs/. Another research area of RSA algorithm is that it is quite slow for large volume of data has huge time and space complexity (see Table 3) during the encryption and decryption process.

Example of RSA-1024 =

619123598472123699712880624626013812212052
3872891516526642427575355377715314956133303
740165861254483744006193147926873462965364
465180800893886149043261507645110125287810
1228221166215805514188295353950264428180250
8065076685490427759522543449885511466585297
0853971989413138402658275762711728119244056
9395083

7. CONCLUSIONS

In this paper, comprehensive implementation of RSA-1024 cryptography with a case as e-mail service in the multi-client server environment on centralized database has been discussed. Wherein, RSA modulus ‘n’ of size 1024 bits is used. The java BigInteger class has provided sufficiently suitable methods to overcome the calculations dilemma during the encryption and decryption operations on communication and transformation of message. It is concluded that BigInteger may be applied for the RSA-1024 system for increasing speed of process. This implementation is very constructive when readers are eager to use RSA-1024 algorithm in their own applications without desire to implement fast exponential algorithm in their applications. The BigInteger may also be used in RSA-2048 system however here implementation for RSA-1024 system is discussed.

8. REFERENCES

9. APPENDIX

Table 1. Class Description of Server

<table>
<thead>
<tr>
<th>No.</th>
<th>Class/Interface</th>
<th>Constructor(s)</th>
<th>Methods</th>
</tr>
</thead>
</table>
| 1   | ImapRSAMailServerIntf extends Remote | -No-                             | Method have been declared as per the server functionality implemented in ImapRSAMailServerImpl class. The methods are-  
|     |                                  |                                  |  
|     |                                  |                                  | public String [] getConnection(String IpAddress) throws RemoteException;  
|     |                                  |                                  | public String closeConnection(String IpAddress) throws RemoteException;  
|     |                                  |                                  | public String submitNewUserInfo(String UID, String Pass, String RePass, String Name, String FName, String DOB, String Deptt, String Add, String Phone, String Email, String DOJ) throws RemoteException;  
|     |                                  |                                  | public String userVerification(String UID, String Pass) throws RemoteException;  
|     |                                  |                                  | public String submitMail(String From, String To, String CC, String sub, String Date, String Message) throws RemoteException;  
|     |                                  |                                  | public String[][] readMail(String Id) throws RemoteException;  
|     |                                  |                                  | public String[] readMailContent(String Sender, String Subject) throws RemoteException;  
| 2   | ImapRSAMailServerImpl extends UnicastRemoteObject implements ImapRSAMailServerIntf | public ImapRSAMailServerImpl() throws RemoteException | Used to provide  
|     |                                  |                                  |  
|     |                                  |                                  | 1. Connection between client and server.  
|     |                                  |                                  | 2. Create RSA public key (e,n) and private key (d,n) and send public key (e,n) to the client by creating object of class Random, RSAPrivateKey1.  
|     |                                  |                                  | Public String closeConnection(String Ip) throws RemoteException  
|     |                                  |                                  | Used to close connection and freeing the socket.  
|     |                                  |                                  | Public String submitNewUserInfo(String UID, String Pass, String RePass, String Name, String FName, String DOB, String Org, String Add, String Mob, String Phone, String Fax) throws RemoteException  
|     |                                  |                                  | Used to submit user information in database and send acknowledgment to client.  
|     |                                  |                                  | public String userVerification(String UID, String Pass) throws RemoteException  
<p>|     |                                  |                                  | Used to check authentication of client request and sent response to the client.  |</p>
<table>
<thead>
<tr>
<th>No.</th>
<th>Class/Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>ImapRSAMailServer()</td>
<td>Used to display server GUI, create server object, and rebind server in port 1025 by rebind() method of the Naming class.</td>
</tr>
<tr>
<td>4</td>
<td>ClientMachineID</td>
<td>Used to display connected users' machines user ID plus IP Address.</td>
</tr>
<tr>
<td>5</td>
<td>StartServer</td>
<td>Used to display initial window of server. Duration: 1000 ms.</td>
</tr>
<tr>
<td>6</td>
<td>RSAPrivateKey1(int size, Random rnd, String name)</td>
<td>Used to create two large (512 byte long) prime integer numbers, calculate n = p.q, calculate si = (p-1).(q-1), calculate public key (e,n), and create private key (d,n).</td>
</tr>
<tr>
<td>7</td>
<td>RSAPublicKey(String name)</td>
<td>Used to create RSA Public Key.</td>
</tr>
</tbody>
</table>

**Java Code Snippets**

```java
public String submitMail(String From, String To, String CC, String Sub, String Date, String Message) throws RemoteException
    Used to submit mail from the client in mailbox after decryption using RSA algorithm.

public String[][] readMail(String Id) throws RemoteException
    Used to send mailing list to the client.

public String[] readMailContent(String Sender, String Subject) throws RemoteException
    Used to send mail content to the client.
```
<table>
<thead>
<tr>
<th>No.</th>
<th>Class/Interface</th>
<th>Constructor(s)</th>
<th>Method(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ImapRSAMailWin extends JFrame implements ActionListener</td>
<td>ImapRSAMailWin()</td>
<td>-Nil-&lt;br&gt;Used to provide&lt;br&gt;1. GUI -Desktop Pane with internal frames.&lt;br&gt;2. Connection by using lookup() method of Naming class. And by calling getConnection() method of server.&lt;br&gt;3. create object of NewUser class&lt;br&gt;4. Object of Login class.&lt;br&gt;5. Used to close connection by calling closeConnection() method of the server.&lt;br&gt;All the server function call achieved by either in constructor or public void actionPerformed(ActionEvent) method of ActionListener. No user defined function created.</td>
</tr>
<tr>
<td>2</td>
<td>NewUser extends JInternalFrame implements ActionListener</td>
<td>NewUser(ImapRSAMailWin)</td>
<td>No user defined function is created however, server method called or functionality achieved in the method&lt;br&gt;public void actionPerformed(ActionEvent) method of ActionListener.</td>
</tr>
<tr>
<td>3</td>
<td>Login extends JInternalFrame implements ActionListener</td>
<td>Login(ImapRSAMailWin)</td>
<td>No user defined methods are created however, server method called in the method&lt;br&gt;public void actionPerformed(ActionEvent) method of ActionListener.</td>
</tr>
</tbody>
</table>
| 4   | MailBox extends JInternalFrame implements ActionListener, MouseListener, ItemListener | MailBox (Login) | No user defined function is created however, server method called or functionality achieved in the method<br>public void actionPerformed(ActionEvent) method of ActionListener.<br>public void mouseClicked(MouseEvent me)<br>public void mouseEntered(MouseEvent me)<br>public void mouseExited(MouseEvent me)<br>public void mousePressed(MouseEvent me) public void mouseEntered(MouseEvent me) public void mouseExited(MouseEvent me) public void mousePressed(MouseEvent me)
| 5 | MailBoxContent extends JInternalFrame | MailBoxContent(MailBox)  
Used to display mail contents. | public void mouseReleased(MouseEvent me)  
of MouseListener interface  
public void itemStateChanged(ItemEvent i)  
of ItemListener interface.  
No user-defined function is created  
functionality achieved in constructor only. |