Table of Content

Introduction

Communicating over the network is facing more and more security challenges nowadays. In this coursework, several encryption and authentication methods are inspected and implemented, in order to develop a safe, trustable tool for local area network communications. Some relevant concepts are introduced below.

Symmetric Encryption

Symmetric encryption is the oldest and most-used technique; it uses one key for encryption and decryption. To make sure the encrypted massage is safe, the key should be long enough and should be kept secret to anyone besides the sender and the receiver. Caesar Cipher is probably one of the most famous and simplest examples of how such technique is implemented; it simply shifts the letters by 3 places to produce the cipher text.

Asymmetric Encryption

The problem that lies in symmetric encryption is how to get the key exchanged. This problem is especially serious when exchanging the secret key over an open channel, such as the Internet. It is likely that the key gets intercepted and therefore exposes the encrypted data. Asymmetric encryption solves the problem by using two keys in a pair: one of them is public - it is available to anyone who wishes to send encrypted data; another one is private, it is used to decrypt the date so only the receiver holds it. However, asymmetric encryption does not solve the problem perfectly, it is reported that asymmetric encryption could be up to 1000 times slower than symmetric encryption.

Digital Signature

Digital signature is used in a very similar way as the traditional signature, but the former could be much more secure as it can be carefully designed and therefore, can be quite difficult to copy. Digital signature is implemented in a public key/private key fashion. Usually, instead of applying the keys on documents directly, signature is signed and verified on their fingerprint (hash), because one document has a unique hash, and it is impossible to compute the original document by using the hash.

MIC

MIC stands for Message Integrity Code; it is the checksum of a given file and it

should be the only checksum of that file, assuming that it is produced by the same algorithm. On Microsoft's TechNet, MIC is said to be equivalent to hash, such statement is valid because the integrity code is usually generated by using Message Digest 5 or Secure Hash Algorithm (SHA), which are all widely used hash algorithm.

AES

Advanced Encryption Standard (AES) is a block-based symmetric encryption method; it can be seen as the next generation of Data Encryption Standard (DES). It has a fix-sized (4 by 4 bytes) block and a key of 128, 192 or 256 bits long. According to Federal Information Processing Standards Publication 197, AES takes four steps to produce the encrypted data, namely: SubBytes() Transformation, ShiftRows() Transformation, MixColumns() Transformation and AddRoundKey() Transformation.

RSA

RSA is a well-known asymmetric algorithm that was first published in 1977. It uses two rather large prime numbers and a third number, whose value is less than and relatively prime to the multiplication of previous two prime numbers, to calculate its public key and private key. Although the keys are related and one of them has to be available to everyone, using the public key to determine the private key is believed to be very difficult. Still, it is recommended to use longer keys to reduce the risk. In 2005, the US National Institute of Standards and Technology claims that "2048 bit keys for RSA will remain resistant to cracking until about 2030".

DSA

The Digital Signature Algorithm (DSA), just as its name implies, is an algorithm for digital signature (but it is still possible to use it for encryption). Similar to RSA, it relies on a public and a private key to achieve verification and signing respectively. DSA also requires SHA for hashing, as both public key and private key operate on the hash.

Objectives

For Server

- Deal with the connection request from client;
- Generate DSA key pairs for digital signature and send the pubic key to client;
- Generate secret key for AES encryption;
- Receive the RSA private key from client, use it to encrypt the secret key and send the key to client;
- Receive the DSA pubic key from client for digital signature;

- Apply AES encryption algorithm on outgoing message;
- Hash the outgoing message with MIC;
- Sign the hash with DSA private key;
- Send the genuine message, encrypted message and the hash to client;
- Receive the genuine message, encrypted message and the hash from client;
- Use secret key to decrypt the message;
- Use client's DSA public key to verify the message;
- Display genuine message, decrypted message and signature verification result on screen;

For Client

- Be able to connect to the server over LAN;
- Generate DSA key pairs for digital signature and send the pubic key to server;
- Generate RSA key pairs for key encryption and send the public key to server;
- Receive the encrypted AES key from server and decrypted using RSA private key;
- Receive the DSA public key from server;
- Apply AES encryption algorithm on outgoing message;
- Hash the outgoing message with MIC;
- Sign the hash with DSA private key;
- Send the genuine message, encrypted message and the hash to server;
- Receive the genuine message, encrypted message and the hash from server;
- Use secret key to decrypt the message;
- Use client's DSA public key to verify the message;
- Display genuine message, decrypted message and signature verification result on screen;

Software Requirement Specification

Overview

Secomolan is developed for CSM029 Network Security, Assessment 2. Its name comes from “Secure Communication over LAN”, which indicates its purpose: to set up a secure, trustable connection between the client and the server on a local area network. The programme is written in Java.

Functionality

The server contains the Client Listener thread to process connection request from the client, hence, it should be started first. The client reaches the server by calling its IP address. Both of them use port 3000 for connection.

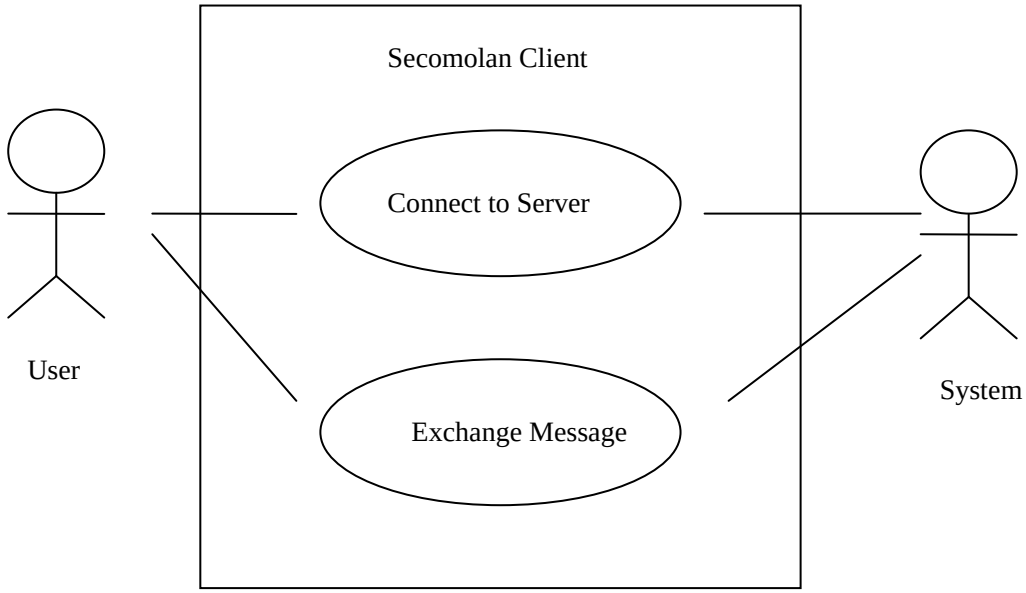
Before communication begins, the server and the client generate several keys and exchange them. First of all, both of them generate DSA key pairs and the exchange the public keys for digital signature; Client then generates RSA key pairs for key encryption and send the public key to the server; Server generates AES key and use the RSA public key from server to encrypted it, then the encrypted key is sent to the client. Key exchange stage finishes after this step.

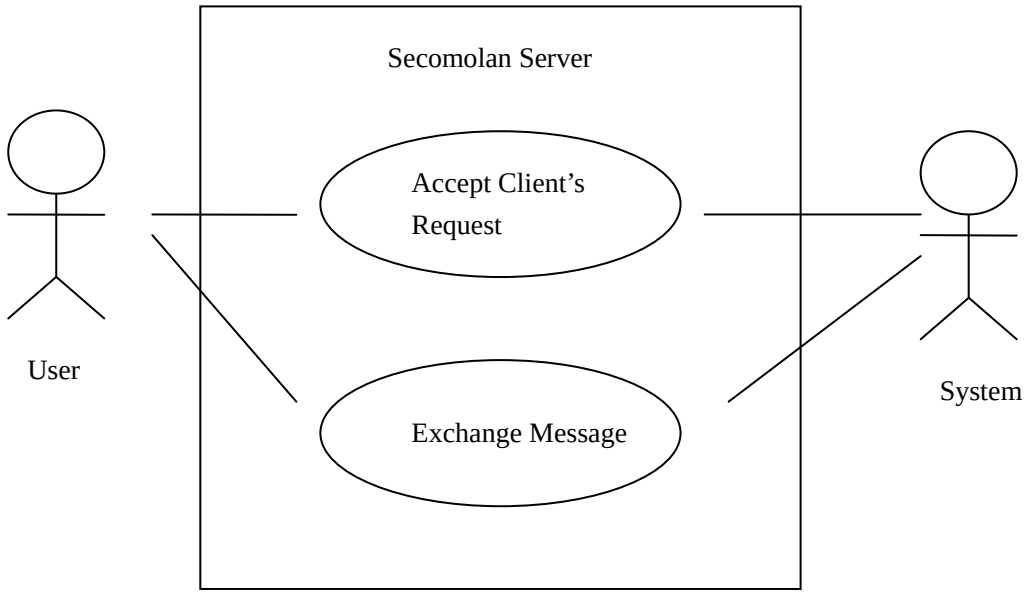
The client and the server use AES key to encrypt outgoing messages; they compute the message integrity code for encrypted messages and apply digital signature using DSA public keys on the hash. The client and the server send a packet, which contains the genuine clear text, encrypted text and the signature, to each other and they verify the encrypted message to make sure that it is not tempered with, then decrypt the message using the same AES key. The genuine clear text, signature verification result and decrypted message are all kept for user’s inspection.

External Interface

The programme is designed for ordinary audience, therefore, it uses graphic user interface to interact with the user. Messages are gathered and display on text areas; a button labeled “Send” invokes send message command. The user is able to see the AES key, DSA public key generated locally and DSA public from the other party from the “Operation” menu. Also, the user can choose whether or not to display genuine clear text and signature verification result on the screen from the same menu.

Use Case Diagram





Screenshots

Server

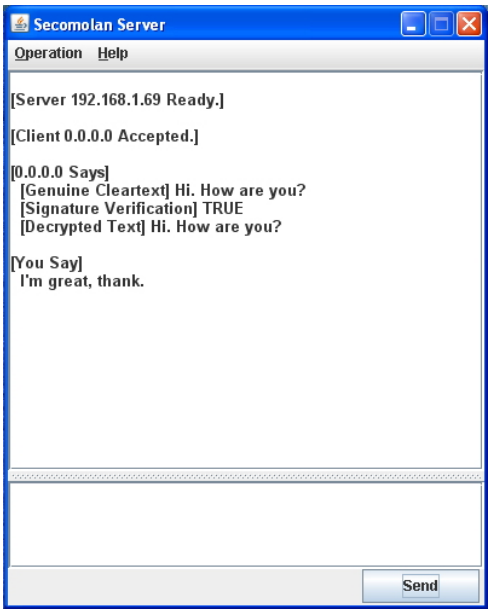


Fig.1 Server GUI

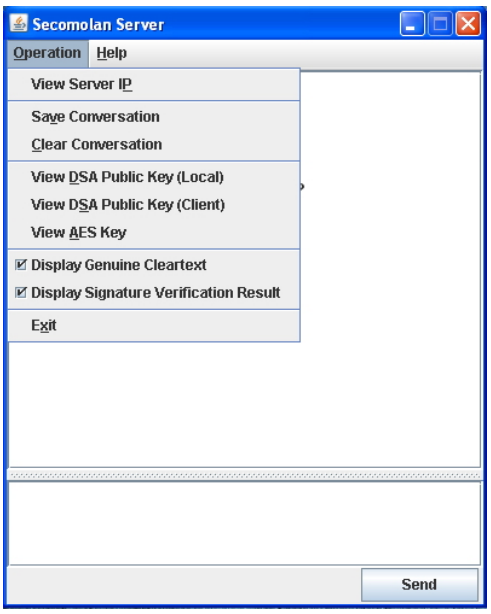


Fig.2 Options in Server Operation Menu

Client

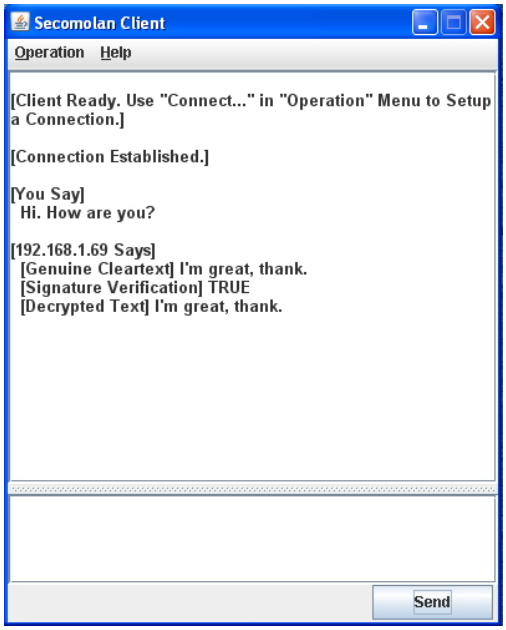


Fig.3 Client GUI

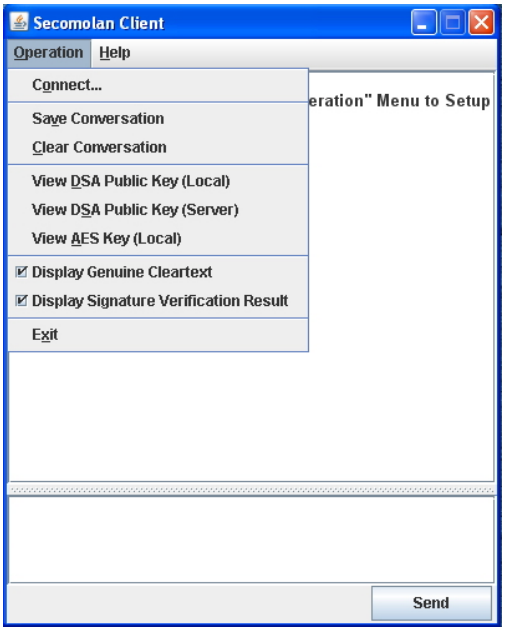


Fig.4 Options in Client Operation Menu

Appendix: Source Code

ServerActivator.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import java.util.Calendar;
import java.io.File;
import java.net.InetAddress;

public class ServerActivator extends JFrame implements ActionListener,
        ItemListener, WindowListener{
    private JMenuItem viewServerIP, viewDSAKey, viewClientDSAKey, viewAESKey, saveConv,
                                clearConv, exit, help, about;
    private JCheckBoxMenuItem dispGenCleartextMi, dispSigverMi;
    private JTextArea textDispPane, textInputPane;
    private JButton send;
    private boolean dispGenCleartext, dispSigver;
    private Server server;

    public ServerActivator() {
        super("Secomolan Server");

        JMenuBar menuBar = new JMenuBar();
        JMenu operationMenu = new JMenu("Operation");
        operationMenu.setMnemonic(KeyEvent.VK_O);
        viewServerIP = new JMenuItem("View Server IP", KeyEvent.VK_P);
        viewServerIP.addActionListener(this);
        operationMenu.add(viewServerIP);
        operationMenu.addSeparator();
        saveConv = new JMenuItem("Save Conversation", KeyEvent.VK_V);
        saveConv.addActionListener(this);
        operationMenu.add(saveConv);
        clearConv = new JMenuItem("Clear Conversation", KeyEvent.VK_C);
        clearConv.addActionListener(this);
        operationMenu.add(clearConv);
        operationMenu.addSeparator();
        viewDSAKey = new JMenuItem("View DSA Public Key (Local)", KeyEvent.VK_D);
        viewDSAKey.addActionListener(this);
        operationMenu.add(viewDSAKey);
        viewClientDSAKey = new JMenuItem("View DSA Public Key (Client)", KeyEvent.VK_S);
```

```
        viewClientDSAKey.addActionListener(this);
        operationMenu.add(viewClientDSAKey);
        viewAESKey = new JMenuItem("View AES Key", KeyEvent.VK_A);
        viewAESKey.addActionListener(this);
        operationMenu.add(viewAESKey);
        operationMenu.addSeparator();
        dispGenCleartext = true;
        dispGenCleartextMi = new JCheckBoxMenuItem("Display Genuine Cleartext",
dispGenCleartext);
        dispGenCleartextMi.addItemListener(this);
        operationMenu.add(dispGenCleartextMi);
        dispSigver = true;
        dispSigverMi = new JCheckBoxMenuItem("Display Signature Verification Result",
dispSigver);
        dispSigverMi.addItemListener(this);
        operationMenu.add(dispSigverMi);
        operationMenu.addSeparator();
        exit = new JMenuItem("Exit", KeyEvent.VK_X);
        exit.addActionListener(this);
        operationMenu.add(exit);

        JMenu helpMenu = new JMenu("Help");
        helpMenu.setMnemonic(KeyEvent.VK_H);
        help = new JMenuItem("Help", KeyEvent.VK_E);
        help.addActionListener(this);
        helpMenu.add(help);
        helpMenu.addSeparator();
        about = new JMenuItem("About...", KeyEvent.VK_B);
        about.addActionListener(this);
        helpMenu.add(about);
        menuBar.add(operationMenu);
        menuBar.add(helpMenu);
        setJMenuBar(menuBar);

        textDispPane = new JTextArea("\n");
        textDispPane.setEditable(false);
        textDispPane.setLineWrap(true);
        textDispPane.setWrapStyleWord(true);
        textDispPane.setFont(new Font("Arial", Font.BOLD, 13));
        JScrollPane textDispScrollPane = new JScrollPane(textDispPane);
        textDispScrollPane.setPreferredSize(new Dimension(380, 330));
        textInputPane = new JTextArea();
        textInputPane.setLineWrap(true);
        textInputPane.setWrapStyleWord(true);
```

```
    textInputPane.setFont(new Font("Arial", Font.PLAIN, 13));
    JScrollPane textInputScrollPane = new JScrollPane(textInputPane);
    JSplitPane splitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT,
                                          textDispScrollPane, textInputScrollPane);

    send = new JButton("Send");
    send.addActionListener(this);
    JPanel buttomPanel = new JPanel(new GridLayout(1, 5));
    buttomPanel.setBorder(new EmptyBorder(0, 2, 2, 2));
    for (int i = 1; i < 4; i++)
       buttomPanel.add(new JPanel());
    buttomPanel.add(send);

    getContentPane().add(splitPane, BorderLayout.CENTER);
    getContentPane().add(buttomPanel, BorderLayout.PAGE_END);
    setSize(400, 500);
    setResizable(false);
    setLocationRelativeTo(null);
    setVisible(true);
    setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
    addWindowListener(this);

    server = new Server(this);
}

// Implement ItemListener
public void itemStateChanged(ItemEvent e) {
    Object source = e.getSource();
    if (source == dispGenCleartextMi) {
       dispGenCleartext = dispGenCleartextMi.isSelected();
    }
    else if (source == dispSigverMi) {
       dispSigver = dispSigverMi.isSelected();
    }
}

// Implement ActionListener
public void actionPerformed (ActionEvent e) {
    Object source = e.getSource();
    if (source == viewServerIP) {
       try {
          JOptionPane.showMessageDialog(null,
                                        server.getServerIP(), "Server IP",
                                        JOptionPane.PLAIN_MESSAGE);
```

```
            }
            catch (Exception exc) {
            }
        }
        else if (source == clearConv) {
            textDispPane.setText("");
        }
        else if (source == saveConv) {
            String path = "";
            JFileChooser fileChooser = new JFileChooser();
            fileChooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
            fileChooser.setMultiSelectionEnabled(false);
            if (fileChooser.showSaveDialog(null) == JFileChooser.APPROVE_OPTION) {
                File selectedFile = fileChooser.getSelectedFile();
                if (selectedFile.isDirectory()) {
                    Calendar cal = Calendar.getInstance();
                    path = selectedFile.getAbsolutePath() + "\\ConversationHistory_" +
                            cal.getTimeInMillis() + ".txt";
                }
                else
                    path = selectedFile.getAbsolutePath();
                server.saveConversation(path, textDispPane.getText());
            }
        }
        else if (source == viewDSAKey) {
            JOptionPane.showMessageDialog(null, server.getDSAPublicKey(), "Local DSA Key",
JOptionPane.PLAIN_MESSAGE);
        }
        else if (source == viewClientDSAKey) {
            JOptionPane.showMessageDialog(null, server.getClientDSAPublicKey(), "Client DSA
Key", JOptionPane.PLAIN_MESSAGE);
        }
        else if (source == viewAESKey) {
            JOptionPane.showMessageDialog(null, server.getAESKey(), "AES Key",
JOptionPane.PLAIN_MESSAGE);
        }
        else if (source == help) {
            new HelpDocViewer();
        }
        else if (source == about) {
            JOptionPane.showMessageDialog(null, "Secomolan Server\nCSM029 Network
Security\nAssessment 2\n\nRuoxi Li",
                                                "About", JOptionPane.PLAIN_MESSAGE);
        }
```

```
    else if (source == exit) {
        server.closeConnection();
        System.exit(0);
    }
    else if (source == send) {
        server.sendMessage(textInputPane.getText());
        textInputPane.setText("");
    }
}
public void windowClosing(WindowEvent e) {
    server.closeConnection();
    System.exit(0);
}
public void windowActivated(WindowEvent e) {
}
public void windowDeactivated(WindowEvent e) {
}
public void windowClosed(WindowEvent e) {
}
public void windowOpened(WindowEvent e) {
}
public void windowIconified(WindowEvent e) {
}
public void windowDeiconified(WindowEvent e) {
}

public void updateStatus(String status) {
    textDispPane.append("[" + status + "]\n\n");
    textDispPane.setCaretPosition(textDispPane.getDocument().getLength()-1);
}

public void updateIncomingMessage(String[] messagePackage) {
    textDispPane.append("[" + server.getCurrentClientIP() + " Says]");
    if (dispGenCleartext) {
        textDispPane.append("\n    [Genuine Cleartext] " + messagePackage[0]);
    }
    if (dispSigver) {
        textDispPane.append("\n    [Signature Verification] " + messagePackage[1]);
    }
    if (dispSigver || dispGenCleartext) {
        textDispPane.append("\n    [Decrypted Text] " + messagePackage[2] + "\n\n");
    }
    else
        textDispPane.append("\n    " + messagePackage[2] + "\n\n");
```

```
        textDispPane.setCaretPosition(textDispPane.getDocument().getLength()-1);
    }

    public void updateOutgoingMessage(String message) {
        textDispPane.append("[You Say]\n    " + message + "\n\n");
        textDispPane.setCaretPosition(textDispPane.getDocument().getLength()-1);
    }

    public static void main (String[] _) {
        new ServerActivator();
    }
}
```

Server.java

```
import java.net.*;
import java.io.*;
import java.security.*;
import javax.crypto.*;

public class Server extends Thread {
    private ServerSocket serverSocket;
    private ObjectInputStream in;
    private ObjectOutputStream out;
    private PublicKey DSAPublicKey, clientDSAPublicKey, clientRSAPublicKey;
    private PrivateKey DSAPrivateKey;
    private SecretKey secretKey;
    private Cipher cipher;
    private MessageDigest digester;
    private Signature signature;
    private ServerActivator gui;

    public Server(ServerActivator gui) {
        try {
            // Initialise Sockets
            serverSocket = new ServerSocket(3000);
            this.gui = gui;
            new ClientListener(this).start();
            gui.updateStatus("Server " + getServerIP() + " Ready.");
        }
        catch (Exception e) {
        }
    }
```

```
public void run() {
   while(true) {
      try {
         Object[] packet = (Object[])in.readObject();
         String genuineText = new String((String)packet[0]);
         byte[] ciphertextByte = (byte[])packet[1];
         byte[] signatureByte = (byte[])packet[2];

         digester.update(ciphertextByte);
         signature.initVerify(clientDSAPublicKey);
         signature.update(digester.digest());
         boolean isVerified = signature.verify(signatureByte);
         String sigVer = ("" + isVerified).toUpperCase();
         cipher = Cipher.getInstance("AES");
         cipher.init(Cipher.DECRYPT_MODE, secretKey);
         String decryptedText = new String(cipher.doFinal(ciphertextByte));

         gui.updateIncomingMessage(new String[]{genuineText, sigVer, decryptedText});
      }
      catch(Exception e) {
      }
   }
}

class ClientListener extends Thread {
   Server server;
   public ClientListener(Server server) {
      this.server = server;
   }
   public void run() {
      try {
         Socket socket = serverSocket.accept();
         gui.updateStatus("Client " + getCurrentClientIP() + " Accepted.");
         in = new ObjectInputStream(socket.getInputStream());
         out = new ObjectOutputStream(socket.getOutputStream());

         // Generate DSA Keys for Digital Signature
         KeyPairGenerator keypairGenerator = KeyPairGenerator.getInstance("DSA");
         keypairGenerator.initialize(1024);
         KeyPair keypair = keypairGenerator.generateKeyPair();
         DSAPublicKey = keypair.getPublic();
         DSAPrivateKey = keypair.getPrivate();

         // Generate AES Secret Key
```

```
            KeyGenerator keygen = KeyGenerator.getInstance("AES");
            keygen.init(128);
            secretKey = keygen.generateKey();

            // Exchange Keys
            clientDSAPublicKey = (PublicKey) in.readObject();
            out.writeObject(DSAPublicKey);
            clientRSAPublicKey = (PublicKey) in.readObject();
            byte[] secretKeyByte = secretKey.getEncoded();
            cipher = Cipher.getInstance("RSA");
            cipher.init(Cipher.ENCRYPT_MODE, clientRSAPublicKey);
            byte[] encryptedSecretKeyByte = cipher.doFinal(secretKeyByte);
            out.writeObject(encryptedSecretKeyByte);

            // Initialise Hash & Signature
            digester = MessageDigest.getInstance("SHA-256");
            signature = Signature.getInstance("DSA");
        }
        catch (Exception e) {
        }
        server.start();
    }
}

public void sendMessage(String message) {
    try {
        cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        byte[] ciphertextByte = cipher.doFinal(message.getBytes());
        digester.update(ciphertextByte);
        signature.initSign(DSAPrivateKey);
        signature.update(digester.digest());
        byte[] signatureByte = signature.sign();

        out.writeObject(new Object[]{message,ciphertextByte,signatureByte});
        gui.updateOutgoingMessage(message);
    }
    catch (Exception e) {
    }
}

public String getDSAPublicKey() {
    String key = "";
    if (DSAPublicKey == null)
```

```
        key = "Client DSA Public Key Has Not Been Initialised.";
    else
        key = DSAPublicKey.toString();
    return key;
}

public String getClientDSAPublicKey() {
    String key = "";
    if (clientDSAPublicKey == null)
        key = "Client DSA Public Key Has Not Been Initialised.";
    else
        key = clientDSAPublicKey.toString();
    return key;
}

public String getAESKey() {
    String key = "";
    if (secretKey == null)
        key = "AES Key Has Not Been Initialised.";
    else
        key = secretKey.toString();
    return key;
}

public String getCurrentClientIP() {
    return serverSocket.getInetAddress().getHostAddress();
}

public String getServerIP() {
    String IP = "";
    try {
        IP = InetAddress.getLocalHost().getHostAddress();
    }
    catch (Exception e) {
    }
    return IP;
}

public void saveConversation(String path, String message) {
    try {
        if (!path.endsWith(".txt"))
            path += ".txt";
        File conv = new File(path);
        BufferedWriter out = new BufferedWriter(new FileWriter(conv, true));
```

```
            out.write(message);
            out.close();
            gui.updateStatus("Conversation History Has Been Saved to: " + path);
        }
        catch (Exception e) {
        }
    }

    public void closeConnection() {
        try {
            serverSocket.close();
        }
        catch (Exception e) {
        }
    }
}
```

ClientActivator.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import java.util.Calendar;
import java.io.File;

public class ClientActivator extends JFrame implements ActionListener,
        ItemListener, WindowListener {
    private JMenuItem connect, viewDSAKey, viewServerDSAKey, viewAESKey, saveConv,
                                clearConv, exit, help, about;
    private JCheckBoxMenuItem dispGenCleartextMi, dispSigverMi;
    private JTextArea textDispPane, textInputPane;
    private JButton send;
    private boolean dispGenCleartext, dispSigver, isConnected;
    private Client client;

    public ClientActivator() {
        super("Secomolan Client");

        JMenuBar menuBar = new JMenuBar();
        JMenu operationMenu = new JMenu("Operation");
        operationMenu.setMnemonic(KeyEvent.VK_O);
        connect = new JMenuItem("Connect...", KeyEvent.VK_O);
        connect.addActionListener(this);
```

```
        operationMenu.add(connect);
        operationMenu.addSeparator();
        saveConv = new JMenuItem("Save Conversation", KeyEvent.VK_V);
        saveConv.addActionListener(this);
        operationMenu.add(saveConv);
        clearConv = new JMenuItem("Clear Conversation", KeyEvent.VK_C);
        clearConv.addActionListener(this);
        operationMenu.add(clearConv);
        operationMenu.addSeparator();
        viewDSAKey = new JMenuItem("View DSA Public Key (Local)", KeyEvent.VK_D);
        viewDSAKey.addActionListener(this);
        operationMenu.add(viewDSAKey);
        viewServerDSAKey = new JMenuItem("View DSA Public Key (Server)", KeyEvent.VK_S);
        viewServerDSAKey.addActionListener(this);
        operationMenu.add(viewServerDSAKey);
        viewAESKey = new JMenuItem("View AES Key (Local)", KeyEvent.VK_A);
        viewAESKey.addActionListener(this);
        operationMenu.add(viewAESKey);
        operationMenu.addSeparator();
        dispGenCleartext = true;
        dispGenCleartextMi = new JCheckBoxMenuItem("Display Genuine Cleartext",
dispGenCleartext);
        dispGenCleartextMi.addItemListener(this);
        operationMenu.add(dispGenCleartextMi);
        dispSigver = true;
        dispSigverMi = new JCheckBoxMenuItem("Display Signature Verification Result",
dispSigver);
        dispSigverMi.addItemListener(this);
        operationMenu.add(dispSigverMi);
        operationMenu.addSeparator();
        exit = new JMenuItem("Exit", KeyEvent.VK_X);
        exit.addActionListener(this);
        operationMenu.add(exit);

        JMenu helpMenu = new JMenu("Help");
        helpMenu.setMnemonic(KeyEvent.VK_H);
        help = new JMenuItem("Help", KeyEvent.VK_E);
        help.addActionListener(this);
        helpMenu.add(help);
        helpMenu.addSeparator();
        about = new JMenuItem("About...", KeyEvent.VK_B);
        about.addActionListener(this);
        helpMenu.add(about);
        menuBar.add(operationMenu);
```

```
    menuBar.add(helpMenu);
    setJMenuBar(menuBar);

    textDispPane = new JTextArea("\n");
    textDispPane.setEditable(false);
    textDispPane.setLineWrap(true);
    textDispPane.setWrapStyleWord(true);
    textDispPane.setFont(new Font("Arial", Font.BOLD, 13));
    JScrollPane textDispScrollPane = new JScrollPane(textDispPane);
    textDispScrollPane.setPreferredSize(new Dimension(380, 330));
    textInputPane = new JTextArea();
    textInputPane.setLineWrap(true);
    textInputPane.setWrapStyleWord(true);
    textInputPane.setFont(new Font("Arial", Font.PLAIN, 13));
    JScrollPane textInputScrollPane = new JScrollPane(textInputPane);
    JSplitPane splitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT,
                                          textDispScrollPane, textInputScrollPane);

    send = new JButton("Send");
    send.addActionListener(this);
    JPanel buttomPanel = new JPanel(new GridLayout(1, 5));
    buttomPanel.setBorder(new EmptyBorder(0, 2, 2, 2));
    for (int i = 1; i < 4; i++)
        buttomPanel.add(new JPanel());
    buttomPanel.add(send);

    getContentPane().add(splitPane, BorderLayout.CENTER);
    getContentPane().add(buttomPanel, BorderLayout.PAGE_END);

    setSize(400, 500);
    setResizable(false);
    setLocationRelativeTo(null);
    setVisible(true);
    setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
    addWindowListener(this);

    client = new Client(this);
}

public void itemStateChanged(ItemEvent e) {
    Object source = e.getSource();
    if (source == dispGenCleartextMi) {
        dispGenCleartext = dispGenCleartextMi.isSelected();
    }
```

```
        else if (source == dispSigverMi) {
            dispSigver = dispSigverMi.isSelected();
        }
    }

    public void actionPerformed (ActionEvent e) {
        Object source = e.getSource();
        if (source == connect) {
            String IPAddress = JOptionPane.showInputDialog(null, "Server IP Address:");
            if (IPAddress != null) {
                if (IPAddress.toLowerCase().equals("localhost") ||
                        client.isValidIPAddress(IPAddress))
                    client.connectServer(IPAddress);
                else
                    JOptionPane.showMessageDialog(null, "Invalid IP Address.", "Error",
                                                          JOptionPane.ERROR_MESSAGE);
            }
        }
        else if (source == clearConv) {
            textDispPane.setText("");
        }
        else if (source == saveConv) {
            String path = "";
            JFileChooser fileChooser = new JFileChooser();
            fileChooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
            fileChooser.setMultiSelectionEnabled(false);
            if (fileChooser.showSaveDialog(null) == JFileChooser.APPROVE_OPTION) {
                File selectedFile = fileChooser.getSelectedFile();
                if (selectedFile.isDirectory()) {
                    Calendar cal = Calendar.getInstance();
                    path = selectedFile.getAbsolutePath() + "\\ConversationHistory_" +
                            cal.getTimeInMillis() + ".txt";
                }
                else
                    path = selectedFile.getAbsolutePath();
                client.saveConversation(path, textDispPane.getText());
            }
        }
        else if (source == viewDSAKey) {
            JOptionPane.showMessageDialog(null, client.getDSAPublicKey(), "Local DSA Key",
JOptionPane.PLAIN_MESSAGE);
        }
        else if (source == viewServerDSAKey) {
            JOptionPane.showMessageDialog(null, client.getServerDSAPublicKey(), "Server DSA
```

```
Key", JOptionPane.PLAIN_MESSAGE);
        }
        else if (source == viewAESKey) {
            JOptionPane.showMessageDialog(null, client.getAESKey(), "AES Key",
JOptionPane.PLAIN_MESSAGE);
        }
        else if (source == help) {
            new HelpDocViewer();
        }
        else if (source == about) {
            JOptionPane.showMessageDialog(null, "Secomolan Client\nCSM029 Network
Security\nAssessment 2\n\nRuoxi Li", "About", JOptionPane.PLAIN_MESSAGE);
        }
        else if (source == exit) {
            client.closeConnection();
            System.exit(0);
        }
        else if (source == send) {
            client.sendMessage(textInputPane.getText());
            textInputPane.setText("");
        }
    }
    public void windowClosing(WindowEvent e) {
        client.closeConnection();
        System.exit(0);
    }
    public void windowActivated(WindowEvent e) {
    }
    public void windowDeactivated(WindowEvent e) {
    }
    public void windowClosed(WindowEvent e) {
    }
    public void windowOpened(WindowEvent e) {
    }
    public void windowIconified(WindowEvent e) {
    }
    public void windowDeiconified(WindowEvent e) {
    }

    public void updateStatus(String status) {
        textDispPane.append("[" + status + "]\n\n");
        textDispPane.setCaretPosition(textDispPane.getDocument().getLength()-1);
    }
```

```
    public void updateIncomingMessage(String[] messagePackage) {
        textDispPane.append("[" + client.getServerIP() + " Says] ");
        if (dispGenCleartext) {
            textDispPane.append("\n    [Genuine Cleartext] " + messagePackage[0]);
        }
        if (dispSigver) {
            textDispPane.append("\n    [Signature Verification] " + messagePackage[1]);
        }
        if (dispSigver || dispGenCleartext) {
            textDispPane.append("\n    [Decrypted Text] " + messagePackage[2] + "\n\n");
        }
        else
            textDispPane.append("\n    " + messagePackage[2] + "\n\n");
        textDispPane.setCaretPosition(textDispPane.getDocument().getLength()-1);
    }

    public void updateOutgoingMessage(String message) {
        textDispPane.append("[You Say]\n    " + message + "\n\n");
        textDispPane.setCaretPosition(textDispPane.getDocument().getLength()-1);
    }

    public static void main (String[] _) {
        new ClientActivator();
    }
}
```

Client.java

```
import java.net.*;
import java.io.*;
import java.security.*;
import javax.crypto.*;
import javax.crypto.spec.*;
import java.util.StringTokenizer;

public class Client extends Thread {
    private Socket clientSocket;
    private ObjectOutputStream out;
    private ObjectInputStream in;
    private PublicKey RSAPublicKey, DSAPublicKey, serverDSAPublicKey;
    private PrivateKey RSAPrivateKey, DSAPrivateKey;
    private SecretKey secretKey;
    private Cipher cipher;
    private MessageDigest digester;
```

```
    private Signature signature;
    private ClientActivator gui;
    private String serverIP;

    public Client(ClientActivator gui) {
        try {
            this.gui = gui;

            // Generate RSA Keys for En/Decryption
            KeyPairGenerator keypairGen = KeyPairGenerator.getInstance("RSA");
            keypairGen.initialize(1024);
            KeyPair keypair = keypairGen.generateKeyPair();
            RSAPublicKey = keypair.getPublic();
            RSAPrivateKey = keypair.getPrivate();

            // Generate DSA Keys for Digital Signature
            keypairGen = KeyPairGenerator.getInstance("DSA");
            keypairGen.initialize(1024);
            keypair = keypairGen.generateKeyPair();
            DSAPublicKey = keypair.getPublic();
            DSAPrivateKey = keypair.getPrivate();

            // Initialise Hash & Signature
            digester = MessageDigest.getInstance("SHA-256");
            signature = Signature.getInstance("DSA");

            gui.updateStatus("Client Ready. Use \"Connect...\" in \"Operation\" Menu to Setup a
Connection.");
        }
        catch (Exception e) {
        }
    }

    public void connectServer(String serverIP) {
        try {
            // Initialise Socket & I/O Stream
            clientSocket = new Socket(serverIP, 3000);
            out = new ObjectOutputStream(clientSocket.getOutputStream());
            in = new ObjectInputStream(clientSocket.getInputStream());

            // Exchange Keys
            out.writeObject(DSAPublicKey);
            serverDSAPublicKey = (PublicKey) in.readObject();
            out.writeObject(RSAPublicKey);
```

```
            byte[] EncryptedAESKeyByte = (byte[]) in.readObject();
            cipher = Cipher.getInstance("RSA");
            cipher.init(Cipher.DECRYPT_MODE, RSAPrivateKey);
            byte[] AESKeyByte = cipher.doFinal(EncryptedAESKeyByte);
            secretKey = new SecretKeySpec(AESKeyByte, "AES");

            gui.updateStatus("Connection Established.");
            this.serverIP = serverIP;
        }
        catch(Exception e) {
        }
        start();
    }

    public void sendMessage(String message) {
        try {
            cipher = Cipher.getInstance("AES");
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);
            byte[] ciphertextByte = cipher.doFinal(message.getBytes());
            digester.update(ciphertextByte);
            signature.initSign(DSAPrivateKey);
            signature.update(digester.digest());
            byte[] signatureByte = signature.sign();

            out.writeObject(new Object[]{message,ciphertextByte,signatureByte});
            gui.updateOutgoingMessage(message);
        }
        catch (Exception e) {
        }
    }

    public void run() {
        while(true) {
            try {
                Object[] packet = (Object[])in.readObject();
                String genuineText = new String((String)packet[0]);
                byte[] ciphertextByte = (byte[])packet[1];
                byte[] signatureByte = (byte[])packet[2];

                digester.update(ciphertextByte);
                signature.initVerify(serverDSAPublicKey);
                signature.update(digester.digest());
                boolean isVerified = signature.verify(signatureByte);
                String sigVer = ("" + isVerified).toUpperCase();
```

```
            cipher = Cipher.getInstance("AES");
            cipher.init(Cipher.DECRYPT_MODE, secretKey);
            String decryptedText = new String(cipher.doFinal(ciphertextByte));

            gui.updateIncomingMessage(new String[]{genuineText, sigVer, decryptedText});
        }
        catch(Exception e) {
        }
    }
}

public boolean isValidIPAddress(String IP) {
    StringTokenizer st = new StringTokenizer(IP, ".");
    String IPSegment;
    boolean isValid = true;
    int loopCounter = 0;

    if (IP.length() == 0)
        isValid = false; // Invalid If IP is Empty
    else {
        SegmentLoop: while (st.hasMoreTokens()) {
            IPSegment = st.nextToken();
            for (int i = 0; i < IPSegment.length(); i++) {
                if (!Character.isDigit(IPSegment.charAt(i))) {
                    isValid = false; // Invalid If IP Contains Non-Digit Char
                    break SegmentLoop;
                }
            }
            if (Integer.parseInt(IPSegment) > 255) {
                isValid = false; // Invalid If IP Segment is Greater Than 255
                break SegmentLoop;
            }
            loopCounter++;
        }
    }
    if (loopCounter < 4)
        isValid = false; // Invalid If IP isn't in Form of X.X.X.X
    return isValid;
}

public String getDSAPublicKey() {
    return DSAPublicKey.toString();
}
```

```
public String getServerDSAPublicKey() {
    String key = "";
    if (serverDSAPublicKey==null)
        key = "Server DSA Public Key Has Not Been Initialised.";
    else
        key = serverDSAPublicKey.toString();
    return key;
}

public String getAESKey() {
    String key = "";
    if (secretKey == null)
        key = "AES Key Has Not Been Initialised.";
    else
        key = secretKey.toString();
    return key;
}

public void closeConnection() {
    if (in!=null && out!=null && clientSocket!=null) {
        try {
            in.close();
            out.close();
            clientSocket.close();
        }
        catch (Exception e) {
        }
    }
}

public void saveConversation(String path, String message) {
    try {
        if (!path.endsWith(".txt"))
            path += ".txt";
        File conv = new File(path);
        BufferedWriter out = new BufferedWriter(new FileWriter(conv, true));
        out.write(message);
        out.close();
        gui.updateStatus("Conversation History Has Been Saved to: " + path);
    }
    catch (Exception e) {
    }
}
```

```
   public String getServerIP() {
      return serverIP;
   }
}
```

HelpDocViewer.java

```
// THIS CLASS IS IDENTICAL IN BOTH CLIENT AND SERVER
import java.awt.*;
import javax.swing.*;
import java.net.*;
import javax.swing.text.html.*;
import javax.swing.event.HyperlinkListener;
import javax.swing.event.HyperlinkEvent;

public class HelpDocViewer extends JFrame implements HyperlinkListener {
   public HelpDocViewer () {
      super("Help");
      setLayout(new BorderLayout());

      try {
         URLClassLoader loader = (URLClassLoader)this.getClass().getClassLoader();
         URL url = loader.findResource("HelpDoc.htm");
         JEditorPane manual = new JEditorPane();
         manual.setEditable(false);
         manual.setPage(url);
         manual.addHyperlinkListener(this);

         JScrollPane scrollPane = new JScrollPane(manual);
         scrollPane.setHorizontalScrollBarPolicy(JScrollPane.

HORIZONTAL_SCROLLBAR_NEVER);
         scrollPane.setVerticalScrollBarPolicy(JScrollPane.

VERTICAL_SCROLLBAR_AS_NEEDED);
         add(scrollPane, BorderLayout.CENTER);
      }
      catch (Exception e) {
      }

      this.setSize(500, 600);
      this.setResizable(false);
      this.setLocationRelativeTo(null);
      this.setVisible(true);
```

```
    }

    public void hyperlinkUpdate(HyperlinkEvent e) {
        if (e.getEventType() == HyperlinkEvent.EventType.ACTIVATED) {
            JEditorPane pane = (JEditorPane) e.getSource();
            if (e instanceof HTMLFrameHyperlinkEvent) {
                HTMLFrameHyperlinkEvent evt = (HTMLFrameHyperlinkEvent) e;
                HTMLDocument doc = (HTMLDocument) pane.getDocument();
                doc.processHTMLFrameHyperlinkEvent(evt);
            }
            else {
                try {
                    pane.setPage(e.getURL());
                }
                catch (Throwable t) {
                    t.printStackTrace();
                }
            }
        }
    }
}
```