CRYPTOGRAPHY FUNDAMENTALS

J COMPONENT REVIEW – 3

Submitted to: Prof Madhu Viswanatham V

Submitted by:

Ishita Gupta 17BCI0084

Charizma Gupta 17BCI0157

Rohith Rajeev Pillai 17BCI0181

Course Code: CSE1011

Slot: C1

RSA ENCRYPTED MULTI-USER CHAT

ABSTRACT

We aim to create a multi-user chat based on the client-server model that sends and receives messages. Each user will be able to chat to multiple other users simultaneously (many-to-many) and share details and information amongst themselves. We plan to do this in Java using socket programming and telnet as our connection protocol and plan to encrypt the sensitive information using RSA cryptographic algorithm. Over an intranet, corporations can establish a telnet server and use it as a means to communicate within their office building. This idea of an encrypted centralized chat server can serve as a startup step for many up and coming companies and also has a crucial role in business as communication between employees is essential.

APPROACH

1. Creating the Basic Chat Server
2. Handling User Presence
3. Sending Direct Messages
4. Creating the Chat Client API
5. Building the Client GUI
6. Adding encryption and decryption functions

TCP/IP

Short for Transmission Control Protocol/Internet Protocol, TCP/IP is a set of rules (protocols) governing communications among all computers on the Internet. TCP/IP specifies how data is exchanged over the internet by providing end-to-end communications that identify how it should be broken into packets, addressed, transmitted, routed and received at the destination. The TCP part has to do with the verifying delivery of the packets. The IP part refers to the moving of data packets between nodes. TCP/IP can be used as a communications protocol in a private network

JAVA

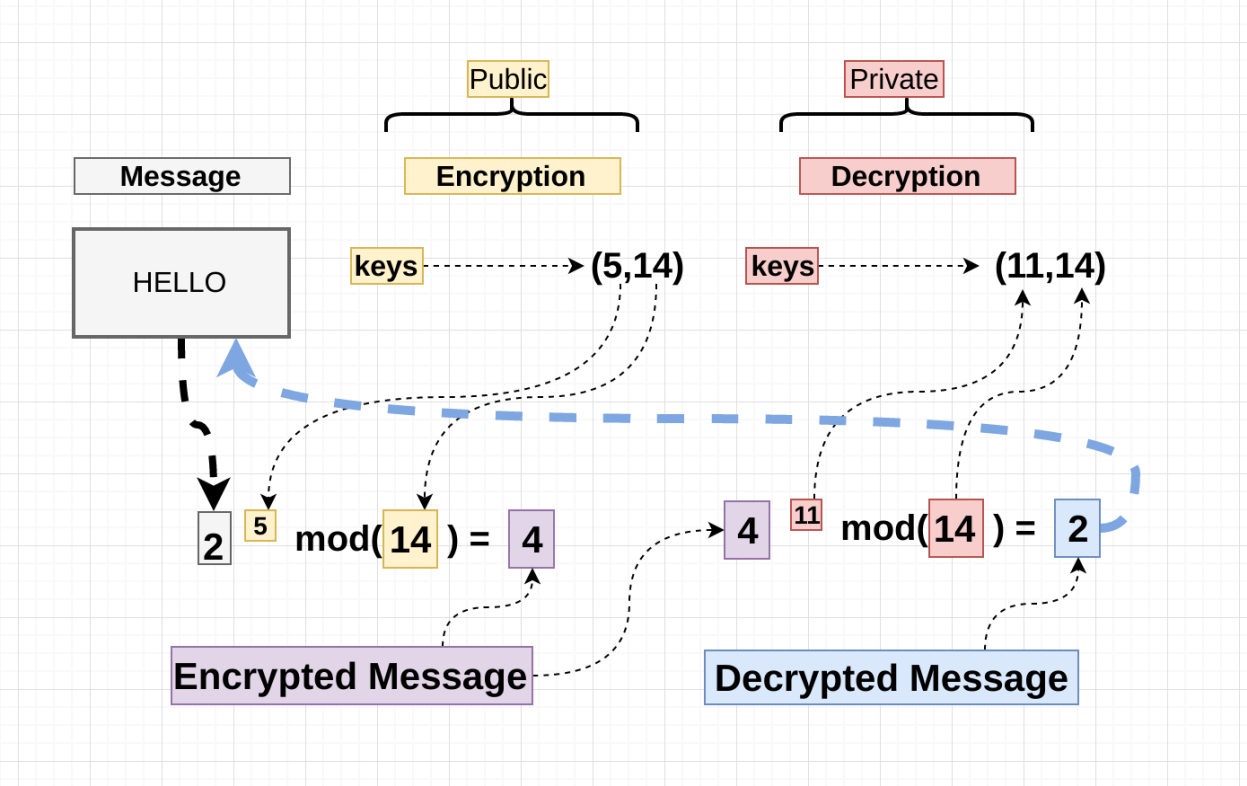
The “Multi-User” component of our project will require us to use multi-threading to be able to create a functioning chat application. Java is known to excel at multi-thread-based programming and is also powerful and efficient when handling TCP/IP sockets. Java is extremely popular among coders to be used to program server-side because of its scalability, efficient memory management and cross-platform nature, to list just a few.

RSA

Rivest–Shamir–Adleman is one of the first public-key (asymmetric) cryptosystems that was used for data transmission. Two (large) prime numbers are chosen and used to generate the public and private keys through a specific algorithm (which includes the multiplication of both as its first step). RSA works on the infeasibility of factoring that product. With sufficiently large enough primes, the factorization

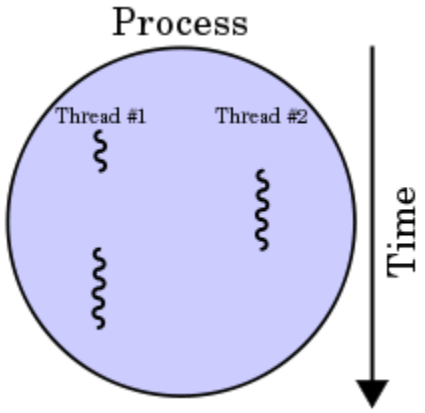
becomes too computationally taxing and nearly impossible. Once the Private and Public keys are generated, the public key is sent to the party who will send the message. That party will encrypt their plaintext into ciphertext using that public key and send it (plaintext) back to the one who generated the keys in the first place (the recipient). The recipient will then use his secret private key to decrypt the ciphertext back to plaintext and see the original message.

Key generation	$n = P * Q$ $d * e = 1 \bmod \Phi(n)$
Encryption	$c = m^e \bmod n$ Public Key(n,e)
Decryption	$m = c^d \bmod n$ private key (d)



MULTI THREADING

In computer architecture, multithreading is the ability of a central processing unit (CPU) (or a single core in a multi-core processor) to execute multiple processes or threads concurrently



A process with two threads of execution, running on a single processor.

Similarly multithreading in Java allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are lightweight processes within a process.

Threads can be created by using two mechanisms:
1. Extending the Thread class
2. Implementing the Runnable Interface

We focus on thread creation by extending the Thread class in our project.

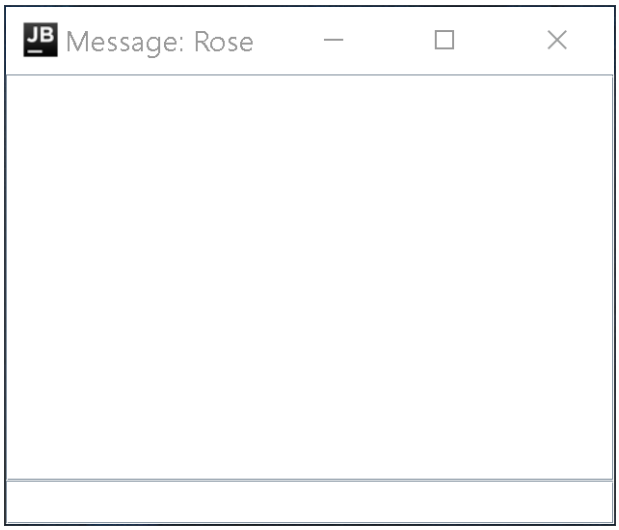
We create a class that extends the java.lang.Thread class. This class overrides the run() method available in the Thread class. A thread begins its life inside run() method. We create an object of our new class and call start() method to start the execution of a thread. Start() invokes the run() method on the Thread object.

TELNET

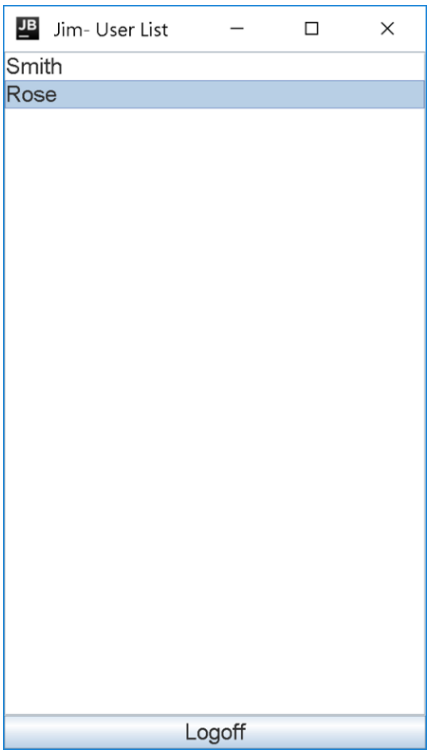
Telnet is a protocol that allows you to connect to remote hosts over a TCP/IP network (such as the internet). Using telnet client software on your computer, you can make a connection to a telnet server. Once your telnet client establishes a connection to the remote host, your client becomes a virtual terminal, allowing you to communicate with the remote host from your computer. This remote host can then serve as the central point for the chat. A user that intends to send a message should login to the server, that message should then be sent to the host, the host should then send that same message to all other users connected to it. This leads to a “Group Chat” dynamic where the host is constantly checking for message requests from users and sending messages between all users.

IMAGES

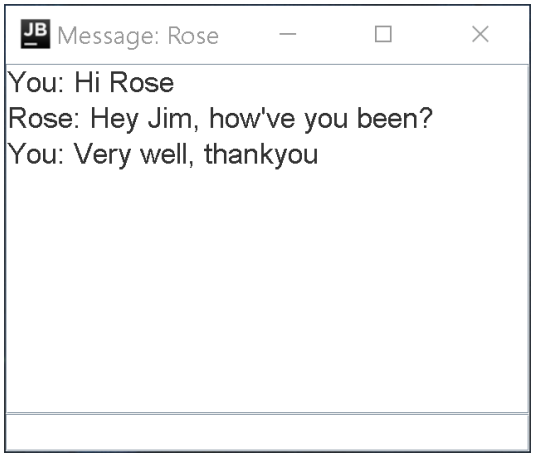
MessagePane

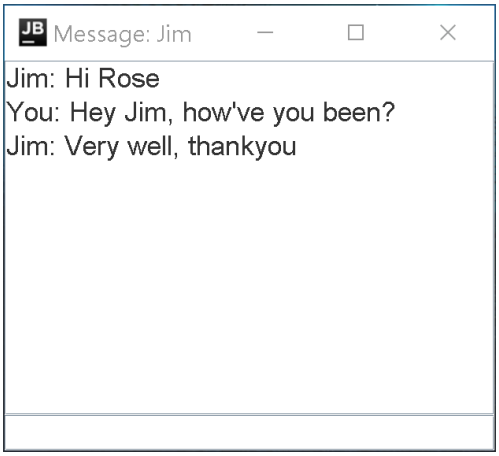


UserListPane

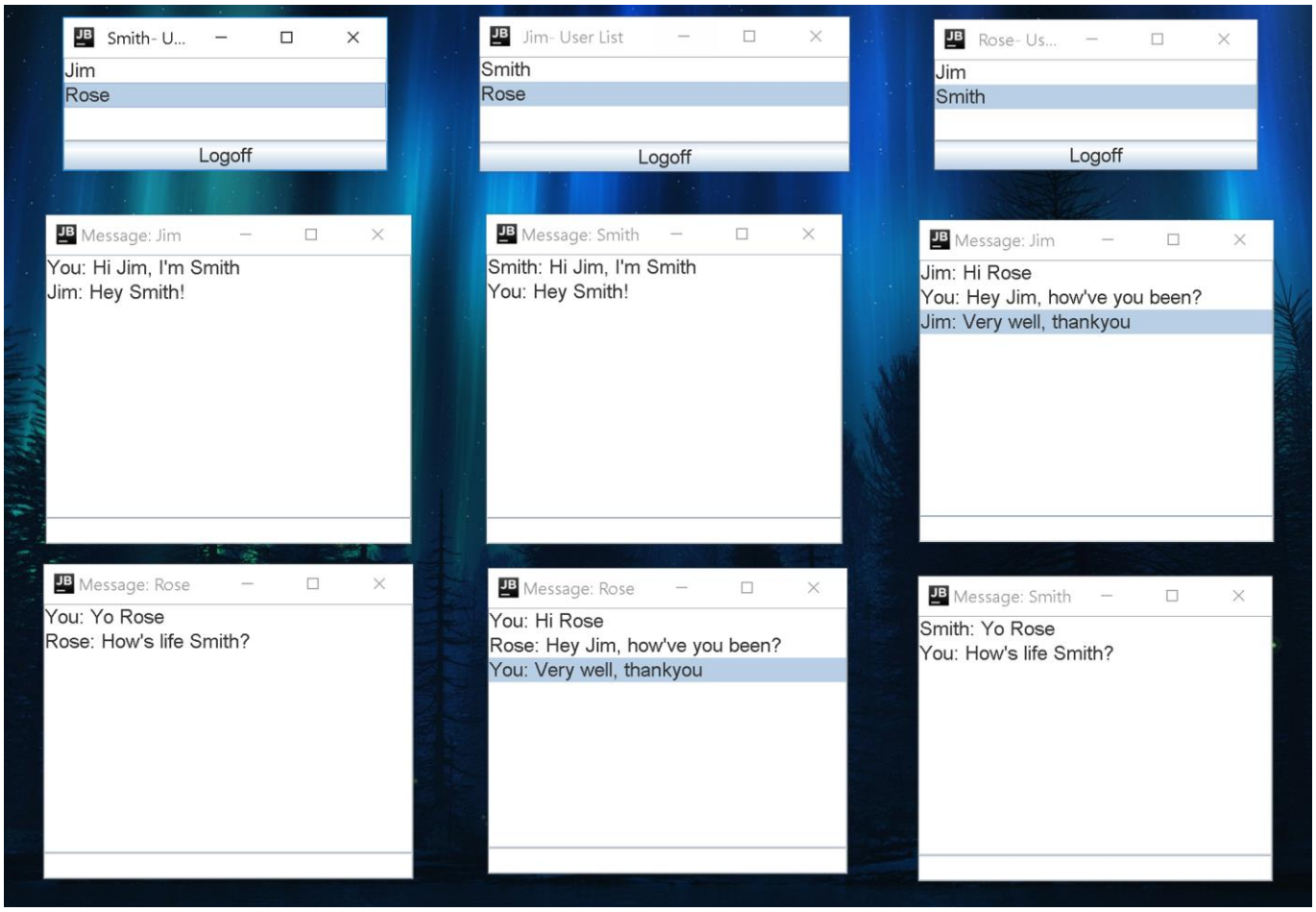


Conversation (One-on-One)

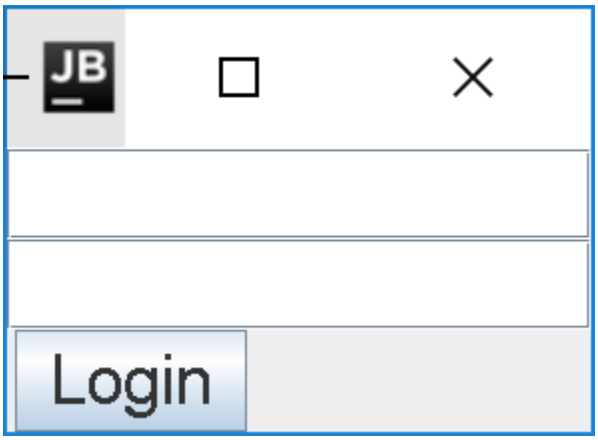




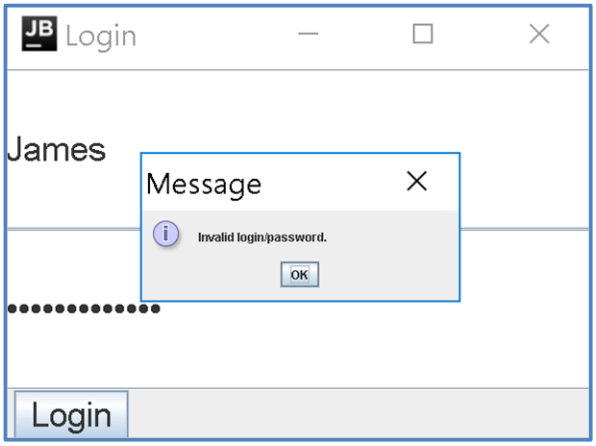
Conversation (Multiple)



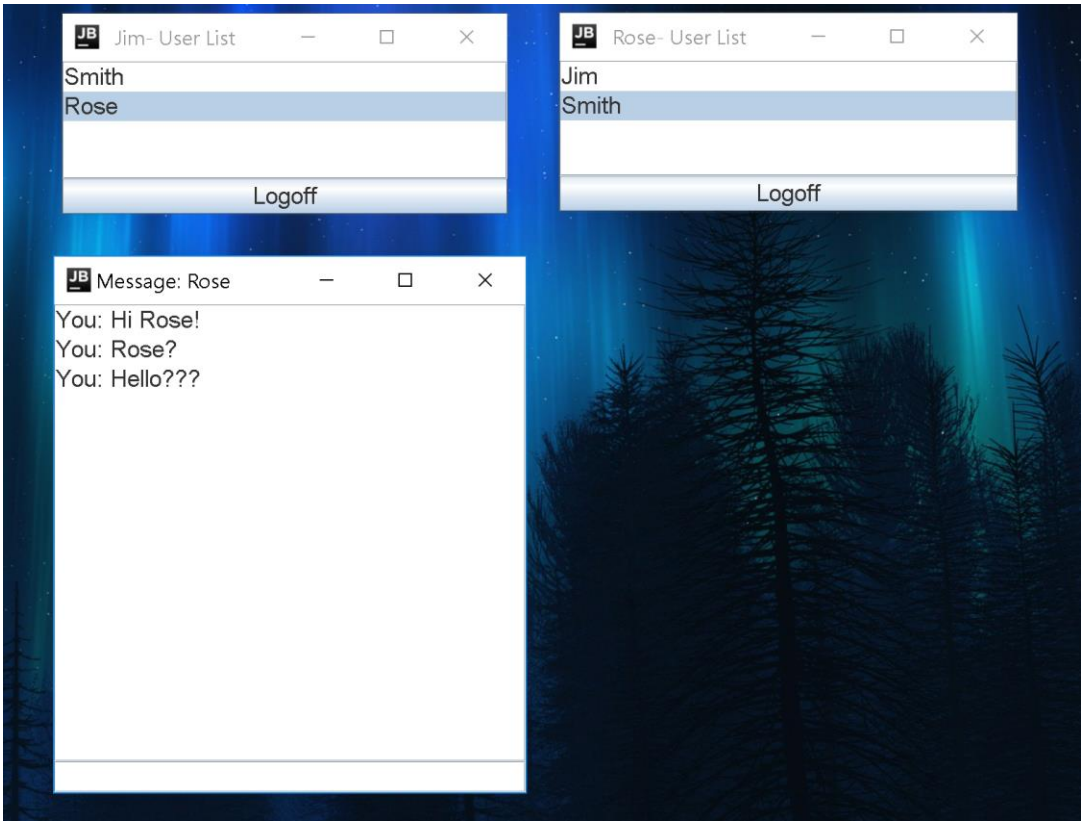
LoginWindow



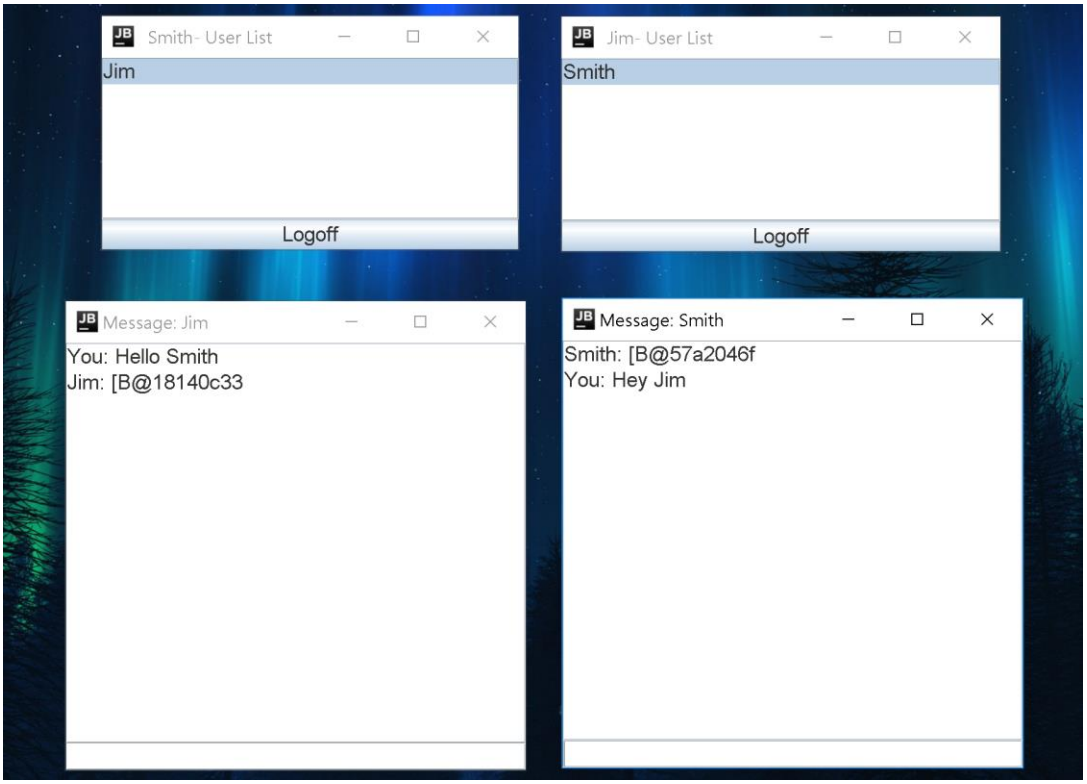
LoginWindow (Invalid Details)



One Sided Chat (Displaying the Secure Consent System)



If the Decryption method wasn't called (What the Server Sees- Encrypted chat)



CODE

Server Main

This class prepares the port selected (in our case, 8818) to accept connections and starts the server class, the thread that accepts client connections.

```
public class ServerMain {
   public static void main(String[] args) {
       int port = 8818;
       Server server = new Server(port);
       server.start();
   }
}
```

Server

This class is called upon by Server Main, it accepts client connections and starts the threads that handle each of these client connections (ServerWorkers).

```
package com.muc;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;

public class Server extends Thread {
    private final int serverPort;
    private ArrayList<ServerWorker> workerList = new ArrayList<>();

    public Server(int serverPort) {
        this.serverPort = serverPort;
    }

    public List<ServerWorker> getWorkerList() {
        return workerList;
    }

    @Override
    public void run() {
        try {
            ServerSocket serverSocket = new ServerSocket(serverPort);
            while(true) {
                System.out.println("About to accept client connection...");
                Socket clientSocket = serverSocket.accept();
                System.out.println("Accepted connection from " + clientSocket);
                ServerWorker worker = new ServerWorker(this, clientSocket);
                workerList.add(worker);
                worker.start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
```

```
    }

    public void removeWorker(ServerWorker serverWorker) {
        workerList.remove(serverWorker);
    }
}
```

Server Worker

Instances of the ServerWorker class are created in Server. Each instance acts as a thread whose job is to handle each client connection, waiting for inputs and executing the correct operation depending on the input, this includes sending messages. The job of these threads is everything from logging a user in to informing other instances of the ServerWorker class about which users are online and which aren't.

```
package com.muc;

import org.apache.commons.lang3.StringUtils;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.Socket;
import java.util.HashSet;
import java.util.List;



public class ServerWorker extends Thread {

    private final Socket clientSocket;
    private final Server server;
    private String login = null;
    //private String password = null;
    private OutputStream outputStream;
    private HashSet<String> topicSet = new HashSet<>();


    public ServerWorker(Server server, Socket clientSocket) {
        this.server = server;
        this.clientSocket = clientSocket;
    }

    @Override
    public void run() {
        try {
            handleClientSocket();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    private void handleClientSocket() throws IOException, InterruptedException {
        InputStream inputStream = clientSocket.getInputStream();
        this.outputStream = clientSocket.getOutputStream();
```

```
        BufferedReader reader = new BufferedReader(new
InputStreamReader(inputStream));
        String line;
        while ((line = reader.readLine()) != null) {
            String[] tokens = StringUtils.split(line);
            if (tokens != null && tokens.length > 0) {
                String cmd = tokens[0];
                if ("logoff".equalsIgnoreCase(cmd) || "quit".equalsIgnoreCase(cmd)) {
                    handleLogoff();
                    break;
                } else if ("login".equalsIgnoreCase(cmd)) {
                    handleLogin(outputStream, tokens);
                } else if ("msg".equalsIgnoreCase(cmd)) {
                    String[] tokensMsg = StringUtils.split(line, null, 3);
                    handleMessage(tokensMsg);
                } else if ("join".equalsIgnoreCase(cmd)) {
                    handleJoin(tokens);
                } else if ("leave".equalsIgnoreCase(cmd)) {
                    handleLeave(tokens);
                } /*else if ("signup".equalsIgnoreCase(cmd)) {
                    handleSignup(tokens);
                }*/ else {
                    String msg = "unknown " + cmd + "\n";
                    outputStream.write(msg.getBytes());
                }
            }
        }

        clientSocket.close();
    }

    /*private void handleSignup(String[] tokens) {

        if (tokens.length == 3) {
            this.login = tokens[1];
            this.password = tokens[2];
        }

    }*/

    private void handleLeave(String[] tokens) {
        if (tokens.length > 1) {
            String topic = tokens[1];
            topicSet.remove(topic);
        }
    }

    public boolean isMemberOfTopic(String topic) {
        return topicSet.contains(topic);
    }

    private void handleJoin(String[] tokens) {
        if (tokens.length > 1) {
            String topic = tokens[1];
            topicSet.add(topic);
        }
    }

    // format: "msg" "login" body...
    // format: "msg" "#topic" body...
    private void handleMessage(String[] tokens) throws IOException {
        String sendTo = tokens[1];
```

```
        String body = tokens[2];

        boolean isTopic = sendTo.charAt(0) == '#';

        List<ServerWorker> workerList = server.getWorkerList();
        for (ServerWorker worker : workerList) {
            if (isTopic) {
                if (worker.isMemberOfTopic(sendTo)) {
                    String outMsg = "msg " + sendTo + ":" + login + " " + body + "\n";
                    worker.send(outMsg);
                }
            } else {
                if (sendTo.equalsIgnoreCase(worker.getLogin())) {
                    String outMsg = "msg " + login + " " + body + "\n";
                    worker.send(outMsg);
                }
            }
        }
    }

    private void handleLogoff() throws IOException {
        server.removeWorker(this);
        List<ServerWorker> workerList = server.getWorkerList();

        // send other online users current user's status
        String onlineMsg = "offline " + login + "\n";
        for (ServerWorker worker : workerList) {
            if (!login.equals(worker.getLogin())) {
                worker.send(onlineMsg);
            }
        }
        clientSocket.close();
    }

    public String getLogin() {
        return login;
    }

    private void handleLogin(OutputStream outputStream, String[] tokens) throws
IOException {
        if (tokens.length == 3) {
            String login = tokens[1];
            String password = tokens[2];

            if ((login.equals("Smith") && password.equals("smith")) ||
(login.equals("Jim") && password.equals("jim")) || (login.equals("Rose") &&
password.equals("rose")) ) {
                String msg = "ok login\n";
                outputStream.write(msg.getBytes());
                this.login = login;
                System.out.println("User logged in succesfully: " + login);

                List<ServerWorker> workerList = server.getWorkerList();

                // send current user all other online logins
                for(ServerWorker worker : workerList) {
                    if (worker.getLogin() != null) {
                        if (!login.equals(worker.getLogin())) {
                            String msg2 = "online " + worker.getLogin() + "\n";
                            send(msg2);
                        }
                    }
                }
```

```
                // send other online users current user's status
                String onlineMsg = "online " + login + "\n";
                for(ServerWorker worker : workerList) {
                    if (!login.equals(worker.getLogin())) {
                        worker.send(onlineMsg);
                    }
                }
            } else {
                String msg = "error login\n";
                outputStream.write(msg.getBytes());
                System.err.println("Login failed for " + login);
            }
        }
    }

    private void send(String msg) throws IOException {
        if (login != null) {
            try {
                outputStream.write(msg.getBytes());
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }
}
```

ChatClient

The server accepts inputs of a certain format and only responds to them; however, a user will only type the message they want to send, this is where the ChatClient comes in. The input fed into the user interface is extracted by this class and converted into proper commands that can be interpreted by the server. It also receives data and instructions from the server which it interprets and prints to the UI. The chat client also contains both the encryption and decryption methods, and encrypts any message the user wants to send to ensure that private conversations aren't read at the server end or anywhere in between before being passed to the intended party.

```
package com.muc;

import org.apache.commons.lang3.StringUtils;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.Socket;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.util.ArrayList;

import javax.crypto.Cipher;
import javax.crypto.NoSuchPaddingException;
```

```
public class ChatClient {
    private final String serverName;
    private final int serverPort;
    private Socket socket;
    private InputStream serverIn;
    private OutputStream serverOut;
    private BufferedReader bufferedIn;

    private ArrayList<UserStatusListener> userStatusListeners = new ArrayList<>();
    private ArrayList<MessageListener> messageListeners = new ArrayList<>();

    KeyPair keyPair;
    PublicKey pubKey;
    PrivateKey privateKey;

    public ChatClient(String serverName, int serverPort) throws
NoSuchAlgorithmException, NoSuchPaddingException {
        this.serverName = serverName;
        this.serverPort = serverPort;
        keyPair = buildKeyPair();
        pubKey = keyPair.getPublic();
        privateKey = keyPair.getPrivate();
    }

    public static void main(String[] args) throws IOException, NoSuchPaddingException,
NoSuchAlgorithmException {
        ChatClient client = new ChatClient("localhost", 8818);
        client.addUserStatusListener(new UserStatusListener() {
            @Override
            public void online(String login) {
                System.out.println("ONLINE: " + login);
            }

            @Override
            public void offline(String login) {
                System.out.println("OFFLINE: " + login);
            }
        });

        client.addMessageListener(new MessageListener() {
            @Override
            public void onMessage(String fromLogin, byte[] msgBody) {
                System.out.println("You got a message from " + fromLogin + " ===>" +
msgBody);
            }
        });

        if (!client.connect()) {
            System.err.println("Connect failed.");
        } else {
            System.out.println("Connect successful");

            if ((client.login("Rose","rose")) || (client.login("Jim","jim")) ||
(client.login("Smith","smith"))) {
                System.out.println("Login successful");

            } else {
                System.err.println("Login failed");
            }

            //client.logoff();
        }
    }
```

```java
public void msg(String sendTo, String msgBody) throws IOException {
    String cmd = "msg " + sendTo + " " + msgBody + "\n";
    serverOut.write(cmd.getBytes());
}

public boolean login(String login, String password) throws IOException {
    String cmd = "login " + login + " " + password + "\n";
    serverOut.write(cmd.getBytes());

    String response = bufferedIn.readLine();
    System.out.println("Response Line:" + response);

    if ("ok login".equalsIgnoreCase(response)) {
        startMessageReader();
        return true;
    } else {
        return false;
    }
}

public void logoff() throws IOException {
    String cmd = "logoff\n";
    serverOut.write(cmd.getBytes());
}

private void startMessageReader() {
    Thread t = new Thread() {
        @Override
        public void run() {
            readMessageLoop();
        }
    };
    t.start();
}

private void readMessageLoop() {
    try {
        String line;
        while ((line = bufferedIn.readLine()) != null) {
            String[] tokens = StringUtils.split(line);
            if (tokens != null && tokens.length > 0) {
                String cmd = tokens[0];
                if ("online".equalsIgnoreCase(cmd)) {
                    handleOnline(tokens);
                } else if ("offline".equalsIgnoreCase(cmd)) {
                    handleOffline(tokens);
                } else if ("msg".equalsIgnoreCase(cmd)) {
                    String[] tokensMsg = StringUtils.split(line, null, 3);
                    handleMessage(tokensMsg);
                }
            }
        }
    } catch (Exception ex) {
        ex.printStackTrace();
        try {
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
    private void handleMessage(String[] tokensMsg) throws Exception {
        String login = tokensMsg[1];
        String msgBody = tokensMsg[2];

        byte[] encrypted = encrypt(pubKey, msgBody);

        for(MessageListener listener : messageListeners) {
            listener.onMessage(login, encrypted);
        }
    }

    public static KeyPair buildKeyPair() throws NoSuchAlgorithmException {
        final int keySize = 2048;
        KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
        keyPairGenerator.initialize(keySize);
        return keyPairGenerator.genKeyPair();
    }

    public static byte[] encrypt(PublicKey publicKey, String message) throws Exception
{
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.ENCRYPT_MODE, publicKey);

        return cipher.doFinal(message.getBytes());
    }
    public static byte[] decrypt(PrivateKey privateKey, byte [] encrypted) throws
Exception {
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.DECRYPT_MODE, privateKey);

        return cipher.doFinal(encrypted);
    }

    private void handleOffline(String[] tokens) {
        String login = tokens[1];
        for(UserStatusListener listener : userStatusListeners) {
            listener.offline(login);
        }
    }

    private void handleOnline(String[] tokens) {
        String login = tokens[1];
        for(UserStatusListener listener : userStatusListeners) {
            listener.online(login);
        }
    }

    public boolean connect() {
        try {
            this.socket = new Socket(serverName, serverPort);
            System.out.println("Client port is " + socket.getLocalPort());
            this.serverOut = socket.getOutputStream();
            this.serverIn = socket.getInputStream();
            this.bufferedIn = new BufferedReader(new InputStreamReader(serverIn));
            return true;
        } catch (IOException e) {
            e.printStackTrace();
        }
        return false;
    }

    public void addUserStatusListener(UserStatusListener listener) {
        userStatusListeners.add(listener);
```

```
    }

    public void removeUserStatusListener(UserStatusListener listener) {
        userStatusListeners.remove(listener);
    }

    public void addMessageListener(MessageListener listener) {
        messageListeners.add(listener);
    }

    public void removeMessageListener(MessageListener listener) {
        messageListeners.remove(listener);
    }

    /*public void signup(String login, String password) throws IOException {
        String cmd = "signup" + login + " " + password + "\n";
        serverOut.write(cmd.getBytes());
        startMessageReader();
    }*/
}
```

MessagePane

The MessagePane class is what determines the UI of the message window – the window that pops up when two users are conversing. This is also where the decryption function is executed to decrypt the ciphertext the ChatClient sends over.

```
package com.muc;

import java.awt.BorderLayout;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;

import javax.swing.DefaultListModel;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextField;

public class MessagePane extends JPanel implements MessageListener {

    private final ChatClient client;
    private final String login;

    private DefaultListModel<String> listModel = new DefaultListModel<>();
    private JList<String> messageList = new JList<>(listModel);
    private JTextField inputField = new JTextField();

    public MessagePane(ChatClient client, String login) {
        this.client = client;
        this.login = login;
        inputField.setFont(new Font("Arial",Font.PLAIN,35));
        messageList.setFont(new Font("Arial",Font.PLAIN,35));

        client.addMessageListener(this);

        setLayout(new BorderLayout());
```

```
        add(new JScrollPane(messageList), BorderLayout.CENTER);
        add(inputField, BorderLayout.SOUTH);

        inputField.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    String text = inputField.getText();
                    client.msg(login, text);
                    listModel.addElement("You: " + text);
                    inputField.setText("");
                } catch (IOException e1) {
                    e1.printStackTrace();
                }
            }
        });
    }

    @Override
    public void onMessage(String fromLogin, byte[] msgBody) throws Exception {
        if (login.equalsIgnoreCase(fromLogin)) {

            byte[] decrypted = client.decrypt(client.privateKey, msgBody);
            String msg= new String(decrypted);
            String line = fromLogin + ": " + msg;
            listModel.addElement(line);
        }
    }
}
```

LoginWindow

This is what is run to open up the login window. Users sign in using their given username and password and when given correct inputs (a valid username and password) the userlist pane is opened. Otherwise, an error pops up.

```
package com.muc;

import java.awt.BorderLayout;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.security.NoSuchAlgorithmException;

import javax.crypto.NoSuchPaddingException;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;

public class LoginWindow extends JFrame {
    private final ChatClient client;
    JTextField loginField = new JTextField();
    JPasswordField passwordField = new JPasswordField();
    JButton loginButton = new JButton("Login");
    //JButton signupButton = new JButton("Signup");
```

```
public LoginWindow() throws NoSuchPaddingException, NoSuchAlgorithmException {
    super("Login");

    this.client = new ChatClient("localhost", 8818);
    client.connect();

    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    loginField.setFont(new Font("Arial",Font.PLAIN,35));
    passwordField.setFont(new Font("Arial",Font.PLAIN,35));
    loginButton.setFont(new Font("Arial",Font.PLAIN,35));
    //signupButton.setFont(new Font("Arial",Font.PLAIN,35));

    JPanel p = new JPanel();
    p.setLayout(new BoxLayout(p, BoxLayout.Y_AXIS));
    p.add(loginField);
    p.add(passwordField);
    p.add(loginButton);
    //p.add(signupButton);

    loginButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            doLogin();
        }
    });

    /*signupButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {doSignup();}
    });*/

    getContentPane().add(p, BorderLayout.CENTER);

    pack();

    setVisible(true);
}

/*private void doSignup() {
    setVisible(false);
    SignupWindow signupWindow = new SignupWindow();
    JFrame frame = new JFrame("Signup");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(600, 1000);

    frame.getContentPane().add(signupWindow, BorderLayout.CENTER);
    frame.setVisible(true);
}*/

private void doLogin() {
    String login = loginField.getText();
    String password = passwordField.getText();

    try {
        if (client.login(login, password)) {
            // bring up the user list window
            UserListPane userListPane = new UserListPane(client);
            JFrame frame = new JFrame("  "+login + "- User List");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(600, 1000);
```

```
                frame.getContentPane().add(userListPane, BorderLayout.CENTER);
                frame.setVisible(true);

                setVisible(false);
            } else {
                // show error message
                JOptionPane.showMessageDialog(this, "Invalid login/password.");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) throws NoSuchAlgorithmException,
NoSuchPaddingException {
        LoginWindow loginWin = new LoginWindow();
        loginWin.setVisible(true);
    }
}
```

UserListPane

When a user logs in through the LoginWindow, the UserListPane is opened. The UserListPane displays all other online users and allows the user to choose who to converse to. It listens for a double click on a username and opens the respective chat box (MessagePane).

```
package com.muc;

import java.awt.BorderLayout;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.io.IOException;
import java.security.NoSuchAlgorithmException;

import javax.crypto.NoSuchPaddingException;
import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JScrollPane;

public class UserListPane extends JPanel implements UserStatusListener {

    private final ChatClient client;
    private JList<String> userListUI;
    private DefaultListModel<String> userListModel;

    public UserListPane(ChatClient client) {
        this.client = client;
        this.client.addUserStatusListener(this);

        userListModel = new DefaultListModel<>();
        userListUI = new JList<>(userListModel);
        userListUI.setFont(new Font("Arial",Font.PLAIN,35));
        JButton logoffButton = new JButton("Logoff");
```

```
        logoffButton.setFont(new Font("Arial",Font.PLAIN,35));

        setLayout(new BorderLayout());
        add(new JScrollPane(userListUI), BorderLayout.CENTER);
        add(logoffButton, BorderLayout.SOUTH);

        logoffButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    doLogoff();
                } catch (IOException e1) {
                    e1.printStackTrace();
                }
            }
        });

        userListUI.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                if (e.getClickCount() > 1) {
                    String login = userListUI.getSelectedValue();
                    MessagePane messagePane = new MessagePane(client, login);

                    JFrame f = new JFrame("Message: " + login);
                    f.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
                    f.setSize(800, 800);
                    f.getContentPane().add(messagePane, BorderLayout.CENTER);
                    f.setVisible(true);
                }
            }
        });
    }

    private void doLogoff() throws IOException {
        client.logoff();
        System.exit(0);
    }

    public static void main(String[] args) throws NoSuchPaddingException,
NoSuchAlgorithmException {
        ChatClient client = new ChatClient("localhost", 8818);

        UserListPane userListPane = new UserListPane(client);
        JFrame frame = new JFrame("User List");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(600, 1000);

        frame.getContentPane().add(userListPane, BorderLayout.CENTER);
        frame.setVisible(true);
    }

    @Override
    public void online(String login) {
        userListModel.addElement(login);
    }

    @Override
    public void offline(String login) {
        userListModel.removeElement(login);
    }
}
```

CONCLUSION

We have created a Multi-User chat that allows for one-on-one communication between online users. All chat messages are encrypted using asymmetric cryptography (RSA). Each user can converse with multiple different users simultaneously. A User-list pops up once the user logs in, which allows them to choose who to communicate with. The privacy aspect of this chat (Preventing communication from unwanted parties) is implemented using a "consent" system. Two users can only converse if each user has double clicked on the other user's name on the list (and as a result, opened the respective chat box). If User A decides to send a message to User B, and B hasn't double clicked on A from the user list, all the messages A sends does not get received by B. This simulates "blocking". The chat works on a "blocking" until consent (in the form of double clicking the name) system.

REFERENCES

- Christensson, P. (2016, June 17). Client-Server Model Definition. Retrieved 2018, Aug 31, from https://techterms.com
- Christensson, P. (2006). TCP/IP Definition. Retrieved 2018, Aug 31, from https://techterms.com
- RFC 1122, Requirements for Internet Hosts – Communication Layers, R. Braden (ed.), October 1989.
- Cerf, Vinton G. & Kahn, Robert E. (1974), A Protocol for Packet Network Intercommunication (PDF), 5
- Qusay H. Mahmoud (Dec 11, 1996) Java tutorial: Sockets programming
- "TCP/IP Internet Protocol". Retrieved 2017-12-31.
- Computer Hope (May 21, 2018) https://www.computerhope.com/jargon/t/tcpip.htm
- Gorry Fairhurst (Jan 11, 2009) EG3567
- Michael Lamont (Nov 11, 2015) Introduction to TCP/IP
- Mantra Malhotra (July 8, 2018) https://hackernoon.com/is-java-good-for-your-web-application-development-83c5b2d0344a
- Pankaj (April 2, 2018) Java Socket Programming
- Tutorialspoint Java - Networking
- Knowledge Base (May 14, 2018) What is Telnet?
- Codrut Neagu (May 8, 2014) Uses of Telnet
- Fall, K. R., & Stevens, W. R. (2011). *TCP/IP illustrated, volume 1: The protocols*. addison-Wesley. https://books.google.co.in/books?hl=en&lr=&id=a23OAn5i8R0C&oi=fnd&pg=PR9&dq=TCP/IP&ots=R8govUll74&sig=ZCM70q28A5KrQ7UuFYvvnJKoKWo#v=onepage&q=TCP%2FIP&f=false
- Jonsson, J., & Kaliski, B. (2003). *Public-key cryptography standards (PKCS)# 1: RSA cryptography specifications version 2.1* (No. RFC 3447). http://www.rfc-editor.org/info/rfc3447
- Saint-Andre, P. (2018). Multi-User Chat. https://xmpp.org/extensions/xep-0045.html
- Rivest, R. L., & Kaliski, B. (2011). RSA problem. In *Encyclopedia of cryptography and security* (pp. 1065-1069). Springer, Boston, MA. https://link.springer.com/referenceworkentry/10.1007%2F978-1-4419-5906-5_475
- Boneh, D. (1999). Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS*, *46*(2), 203-213. https://www.ams.org/notices/199902/boneh.pdf
- Loy, M., Eckstein, R., Wood, D., Elliott, J., & Cole, B. (2002). *Java swing*. " O'Reilly Media, Inc.". https://books.google.co.in/books?hl=en&lr=&id=fU1K9MxaWp0C&oi=fnd&pg=PT4&dq=java+swing+&ots=9QkQhbYTjC&sig=Lf5XV70SbYG4pgbj9gwLy8HgiQk#v=onepage&q=java%20swing&f=false