

Lab 4

Gunnar Yonker

(1)

MeetingRoomClient.java – uses private key to encrypt session key

MeetingRoomServer.java – uses public key to decrypt session key

LoginCreds.txt – Server checks this file for authorized logins

MeetingTimes.txt – Server checks this file for timeslots

PublicKey.txt – generated from Lab 3 program, 2048 bit key

PrivateKey.txt – generated from Lab 3 program, 2048 bit key

(2)

a. Protocol Message Format

Client generates a 128 bit session key

Client uses private key to encrypt the session key

Client sends “You are ready to reserve the room” special acknowledgement message to server

Client sends OAEP encrypted session key to server

Server receives special acknowledgement message

Server decrypts the encrypted session key using the public key

Server encrypts special acknowledgment message using session key

Server sends encrypted special acknowledgement key to client

Client receives encrypted special acknowledgement

Client decrypts special acknowledgement message using session key

Client matches the created special acknowledgement message to the message received from server

If match, then the key exchange is successful, and client is prompted to login

Client is able to log in and communicate with the server to reserve timeslots

If there is no match, acknowledgement fails and the program exits

Lab 4

Gunnar Yonker

b.

Successful key exchange screenshots:

Client View:

```
14 import java.awt.event.*;
15 import java.io.*;
16 import java.net.*;
17 import java.security.*;
18 import java.util.*;
19
20 public class MeetingRoomClient {
21     //Port 755 to connect to localhost server
22     private static final String HOST = "localhost";
23     private static final int PORT = 755;
24
25     public static void main(String[] args) throws IOException, ClassNotFoundException, NoSuchAlgorithmException,
26     Scanner sc = new Scanner(System.in);
27     System.out.println("Meeting Room Scheduler v1");
28     //Sets program to use GPG padding when private and public key are used for encryption or decryption
29     String paddingType = "GPG";
30     //Generate session key
31     KeyGenerator keygen = KeyGenerator.getInstance("AES");
32     Key key = keygen.generateKey();
33     SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("AES");
34
35     //Open connection to the MeetingRoomServer
36     try (Socket socket = new Socket(HOST, PORT)) {
37         DataInputStream in = new DataInputStream(socket.getInputStream());
38         DataOutputStream out = new DataOutputStream(socket.getOutputStream());
39
40         //Send the PrivateKey.txt for the private key for encryption
41         String privateKey = " ";
42         try (BufferedReader br = new BufferedReader(new FileReader("PrivateKey.txt"))) {
43             privateKey = br.readLine();
44         } catch (IOException e) {
45             System.out.println("Error reading private key file: " + e.getMessage());
46         }
47         byte[] privateKeyBytes = Base64.getDecoder().decode(privateKey);
48
49         //Private key to be used for encryption
50         X509EncodedKeySpec pubKeySpec = new X509EncodedKeySpec(privateKeyBytes);
51         KeyFactory keyFactory = KeyFactory.getInstance("RSA");
52         PublicKey publicKey = keyFactory.generatePublic(pubKeySpec);
53
54         //Set cipher to use GPG padding with the public key encryption
55         Cipher cipher = Cipher.getInstance("RSA/CryptoCipherPadding");
56         Cipher.init(cipher, new SecretKeySpec(key, "AES"));
57         byte[] encryptedSessionKey = cipher.doFinal(sessionKey.getEncoded());
58
59         //Send the encrypted session key to the server
60         String encryptedSessionKeyStr = Base64.getEncoder().encodeToString(encryptedSessionKey);
61
62         //Send Special Acknowledgment message
63         String specialMessage = "You are ready to reserve the room";
64         out.writeUTF(specialMessage);
65         System.out.println("Special Acknowledgment Sent");
66
67         //Send encrypted session key to the server
68         out.writeUTF(encryptedSessionKeyStr);
69         System.out.println("Encrypted Session Key Sent to Server");
70
71         //Receive the encrypted message encrypted using the session key
72         let encryptedMessageLength = in.readInt();
73         byte[] encryptedMessage = new byte[encryptedMessageLength];
74         in.readFully(encryptedMessage);
75
76         //Initialize a cipher with the session key for decryption
77         Cipher cipher2 = Cipher.getInstance("AES");
78         cipher2.init(cipher2, new SecretKeySpec(key, "AES"));
79
80         //Decrypt the encrypted message
81         String decryptedMessage = new String(cipher2.doFinal(encryptedMessage));
82         System.out.println("Decrypted Message: " + decryptedMessage);
83
84         //Print out that server is up and waiting for response from client
85         System.out.println("Server is up and waiting for response from client");
86
87         //Start the client loop
88         try (ServerSocket serverSocket = new ServerSocket(PORT)) {
89             Socket socket = serverSocket.accept();
90             DataInputStream in = new DataInputStream(socket.getInputStream());
91             DataOutputStream out = new DataOutputStream(socket.getOutputStream());
92
93             //Receive Special Acknowledgment message
94             String specialMessage = in.readUTF();
95             System.out.println("Special Acknowledgment Received");
96
97         }
98     }
99 }
```

Server View:

```
14 import java.awt.event.*;
15 import java.io.*;
16 import java.net.*;
17 import java.security.*;
18 import java.util.*;
19
20 public class MeetingRoomServer {
21     //Port 755 to connect to localhost server
22     private static final String HOST = "localhost";
23     private static final int PORT = 755;
24
25     public static void main(String[] args) throws IOException, ClassNotFoundException, NoSuchAlgorithmException,
26     Scanner sc = new Scanner(System.in);
27     System.out.println("Meeting Room Scheduler v1");
28     //Sets program to use GPG padding when private and public key are used for encryption or decryption
29     String paddingType = "GPG";
30     //Generate session key
31     KeyGenerator keygen = KeyGenerator.getInstance("AES");
32     Key key = keygen.generateKey();
33     SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("AES");
34
35     //Open connection to the MeetingRoomClient
36     try (Socket socket = new Socket(HOST, PORT)) {
37         DataInputStream in = new DataInputStream(socket.getInputStream());
38         DataOutputStream out = new DataOutputStream(socket.getOutputStream());
39
40         //Send the PrivateKey.txt for the private key for encryption
41         String privateKey = " ";
42         try (BufferedReader br = new BufferedReader(new FileReader("PrivateKey.txt"))) {
43             privateKey = br.readLine();
44         } catch (IOException e) {
45             System.out.println("Error reading private key file: " + e.getMessage());
46         }
47         byte[] privateKeyBytes = Base64.getDecoder().decode(privateKey);
48
49         //Private key to be used for encryption
50         X509EncodedKeySpec pubKeySpec = new X509EncodedKeySpec(privateKeyBytes);
51         KeyFactory keyFactory = KeyFactory.getInstance("RSA");
52         PublicKey publicKey = keyFactory.generatePublic(pubKeySpec);
53
54         //Set cipher to use GPG padding with the public key encryption
55         Cipher cipher = Cipher.getInstance("RSA/CryptoCipherPadding");
56         Cipher.init(cipher, new SecretKeySpec(key, "AES"));
57         byte[] encryptedSessionKey = cipher.doFinal(sessionKey.getEncoded());
58
59         //Send the encrypted session key to the server
60         String encryptedSessionKeyStr = Base64.getEncoder().encodeToString(encryptedSessionKey);
61
62         //Send Special Acknowledgment message
63         String specialMessage = "You are ready to reserve the room";
64         out.writeUTF(specialMessage);
65         System.out.println("Special Acknowledgment Sent");
66
67         //Send encrypted session key to the server
68         out.writeUTF(encryptedSessionKeyStr);
69         System.out.println("Encrypted Session Key Sent to Server");
70
71         //Receive the encrypted message encrypted using the session key
72         let encryptedMessageLength = in.readInt();
73         byte[] encryptedMessage = new byte[encryptedMessageLength];
74         in.readFully(encryptedMessage);
75
76         //Initialize a cipher with the session key for decryption
77         Cipher cipher2 = Cipher.getInstance("AES");
78         cipher2.init(cipher2, new SecretKeySpec(key, "AES"));
79
80         //Decrypt the encrypted message
81         String decryptedMessage = new String(cipher2.doFinal(encryptedMessage));
82         System.out.println("Decrypted Message: " + decryptedMessage);
83
84         //Print out that server is up and waiting for response from client
85         System.out.println("Server is up and waiting for response from client");
86
87         //Start the client loop
88         try (ServerSocket serverSocket = new ServerSocket(PORT)) {
89             Socket socket = serverSocket.accept();
90             DataInputStream in = new DataInputStream(socket.getInputStream());
91             DataOutputStream out = new DataOutputStream(socket.getOutputStream());
92
93             //Receive Special Acknowledgment message
94             String specialMessage = in.readUTF();
95             System.out.println("Special Acknowledgment Received");
96
97         }
98     }
99 }
```

The key exchange was successful and the client program is able to login.

Lab 4
Gunnar Yonker

Successful login and reserve Client View:

[illegible]

Successful login and reserve Server View:

The image shows a screenshot of an IDE (likely IntelliJ IDEA) with two Java files open. The left file is `KeyPairGenerator.java` and the right file is `MeetingRoomServer.java`.

KeyPairGenerator.java
This file contains a `KeyPairGenerator` class. It has a `main` method that generates an RSA key pair and prints it. It also has a `generateKeyPair` method that takes a modulus `n` and a public exponent `e` as input and returns a `KeyPair` object.

MeetingRoomServer.java
This file contains a `MeetingRoomServer` class. It has a `main` method that starts the server on port 7555. It has several methods for handling client requests:
- `handleClientRequest`: This method receives a `Socket` from the client and processes the request. It can handle key exchange, session establishment, and meeting scheduling.
- `exchangeKeys`: This method handles the key exchange process between the client and the server.
- `establishSession`: This method handles the session establishment process between the client and the server.
- `scheduleMeeting`: This method handles the meeting scheduling process between the client and the server.
The server uses a `ByteBuffer` to read and write data from the client. It also uses a `HashMap` to store the keys and sessions for each client.

The IDE interface includes a top toolbar with various icons for editing, running, and debugging. The left sidebar shows the project structure with the two Java files listed. The bottom status bar shows the current file and line number.

Lab 4

Gunnar Yonker

The program now works with the client using their private key to encrypt the session key, the server decrypts the session key using the public key, then the key exchange using the special acknowledgment and session key as encryption/decryption is verified before the client and server continue on with login and reservation functions.

The only part that I could not get to work was using the OAEP padding type, I was running to errors trying to encrypt the session key using OAEP padding. The program uses PKCS1 padding to encrypt and decrypt the session key as I was able to get that padding type to function properly.