# CSE 433S - Project Report

XIN ZHAO

December 13, 2022

**Here is my code**

## 1 Introduction

*In this project, I completed a end-to-end encryption chat room with Java.*

*The motivation to do this as my final project is that when I was doing my homework, an encrypted connection channel with C /C++, I found myself interested in Cryptography, and also want to learn more about network programming in Java language through this project.*

*So I went through things about Java Web, learned how to use NIO, Netty, Socket to build a channel for communicating. The challege for me is to build a window like wechat to show the communication.*

*In this project, I designed a window like wechat to record the communication, a sockt channel to communicate, and use javax.crypto to encrypt the end-to-end communication.*

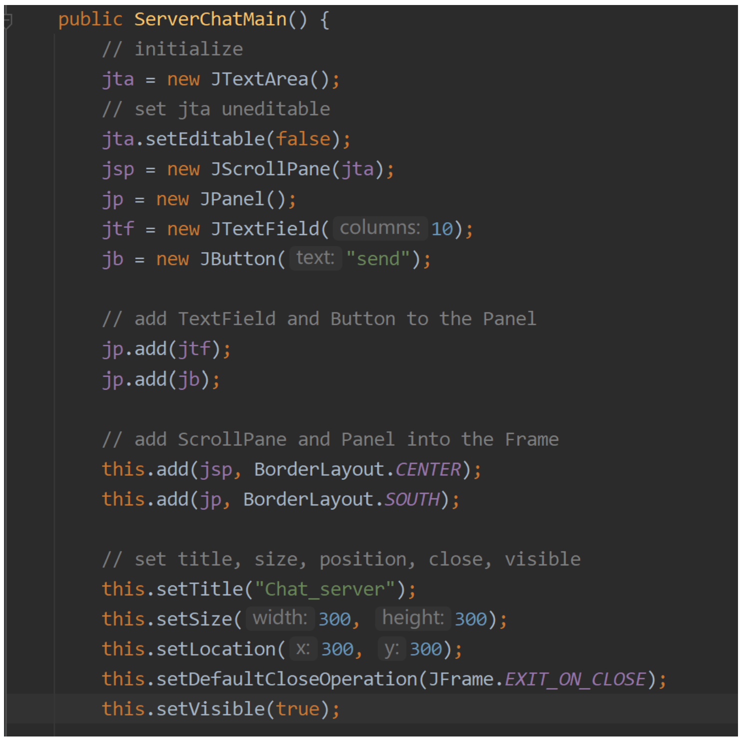
*Finally, use the wireshark to launch an Eavesdropping attack to verify this chat room's Confidentiality. Judging from the results, this chat room is very safe.*

## 2 Method

*The code of this project can be roughly divided into three parts: the front-end UI design; the socket part and the encryption part, so this chapter is divided into three subsections to introduce the implementation methods of these three parts.*

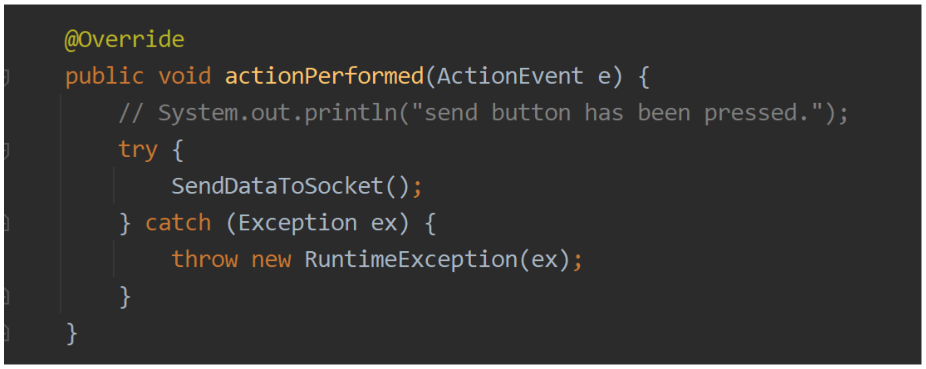
### 2.1 front-end UI

*To build a front-end UI, I use JFrame in Java to get different components needed(JTextArea, JScrollPane, JPanel, JTextFiled, JButton) to build the window, ActionListener to get the I/O stream, use KeyListener to get the signal of the keyboard. And the code is as below:*



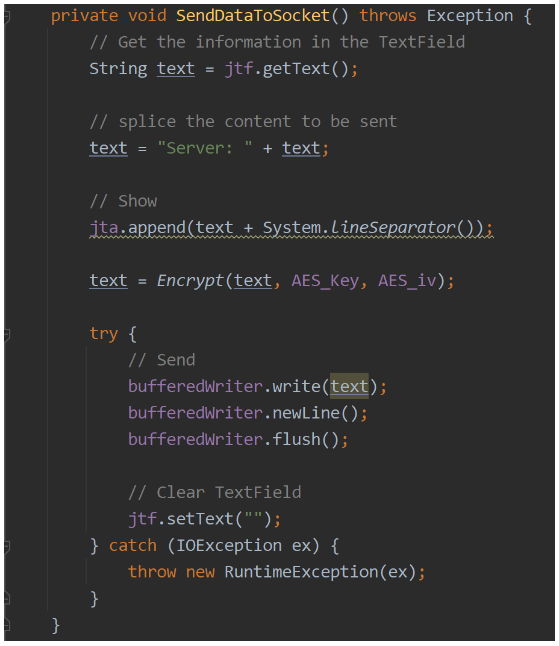
```java
public ServerChatMain() {
    // initialize
    jta = new JTextArea();
    // set jta uneditable
    jta.setEditable(false);
    jsp = new JScrollPane(jta);
    jp = new JPanel();
    jtf = new JTextField( columns: 10);
    jb = new JButton( text: "send");

    // add TextField and Button to the Panel
    jp.add(jtf);
    jp.add(jb);

    // add ScrollPane and Panel into the Frame
    this.add(jsp, BorderLayout.CENTER);
    this.add(jp, BorderLayout.SOUTH);

    // set title, size, position, close, visible
    this.setTitle("Chat_server");
    this.setSize( width: 300,  height: 300);
    this.setLocation( x: 300,  y: 300);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.setVisible(true);
```

Figure 1: JFrame



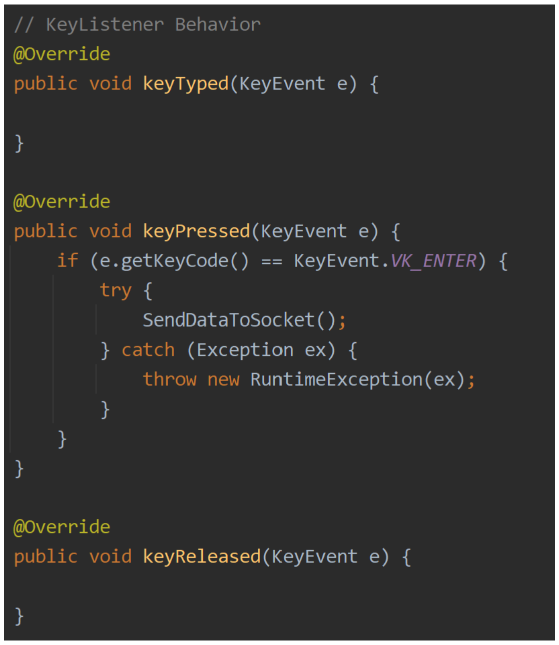
```java
@Override
public void actionPerformed(ActionEvent e) {
    // System.out.println("send button has been pressed.");
    try {
        SendDataToSocket();
    } catch (Exception ex) {
        throw new RuntimeException(ex);
    }
}
```

Figure 2: ActionListener



```java
private void SendDataToSocket() throws Exception {
    // Get the information in the TextField
    String text = jtf.getText();

    // splice the content to be sent
    text = "Server: " + text;

    // Show
    jta.append(text + System.lineSeparator());

    text = Encrypt(text, AES_Key, AES_iv);

    try {
        // Send
        bufferedWriter.write(text);
        bufferedWriter.newLine();
        bufferedWriter.flush();

        // Clear TextField
        jtf.setText("");
    } catch (IOException ex) {
        throw new RuntimeException(ex);
    }
}
```

Figure 3: SendDataToSocket

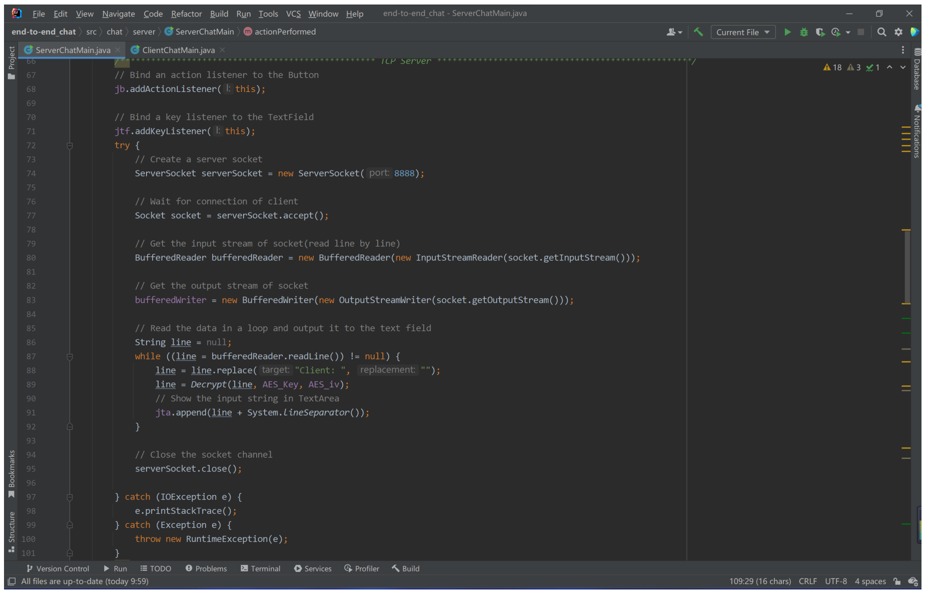


```java
// KeyListener Behavior
@Override
public void keyTyped(KeyEvent e) {

}

@Override
public void keyPressed(KeyEvent e) {
    if (e.getKeyCode() == KeyEvent.VK_ENTER) {
        try {
            SendDataToSocket();
        } catch (Exception ex) {
            throw new RuntimeException(ex);
        }
    }
}

@Override
public void keyReleased(KeyEvent e) {

}
```

Figure 4: KeyListener

## 2.2 Socket

*In the server, we create a server socket first, and wait for client to connect, then we use the socket to get the I/O stream and output them to text field. If the window was closed, we close the socket, too.*

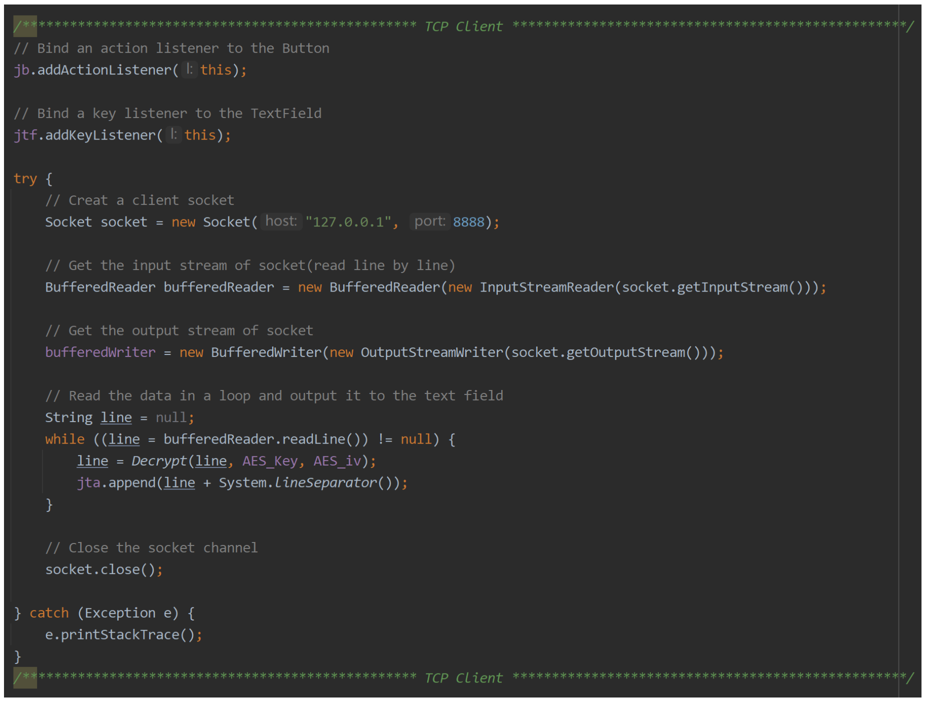


```java
        // Bind an action listener to the Button
        jb.addActionListener(this);

        // Bind a key listener to the TextField
        jtf.addKeyListener(this);
        try {
            // Create a server socket
            ServerSocket serverSocket = new ServerSocket(8888);

            // Wait for connection of client
            Socket socket = serverSocket.accept();

            // Get the input stream of socket(read line by line)
            BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));

            // Get the output stream of socket
            bufferedWriter = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));

            // Read the data in a loop and output it to the text field
            String line = null;
            while ((line = bufferedReader.readLine()) != null) {
                line = line.replace("Client: ", "");
                line = Decrypt(line, AES_Key, AES_iv);
                // Show the input string in TextArea
                jta.append(line + System.lineSeparator());
            }

            // Close the socket channel
            serverSocket.close();

        } catch (IOException e) {
            e.printStackTrace();
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
```

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
end-to-end_chat - ServerChatMain.java
end-to-end_chat › src › chat › server › ServerChatMain › actionPerformed
ServerChatMain.java
ClientChatMain.java
Current File
Version Control
Run
TODO
Problems
Terminal
Services
Profiler
Build
All files are up-to-date (today 9:59)
109:29 (16 chars) CRLF UTF-8 4 spaces

Figure 5: Server Socket

*In the client, we do not need to wait the connection:*

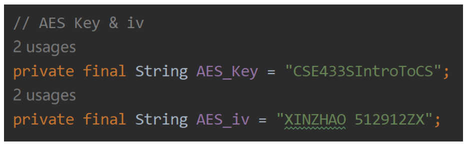


```
/*************************************************** TCP Client ***************************************************/
// Bind an action listener to the Button
jb.addActionListener( l: this);

// Bind a key listener to the TextField
jtf.addKeyListener( l: this);

try {
    // Creat a client socket
    Socket socket = new Socket( host: "127.0.0.1",  port: 8888);

    // Get the input stream of socket(read line by line)
    BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));

    // Get the output stream of socket
    bufferedWriter = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));

    // Read the data in a loop and output it to the text field
    String line = null;
    while ((line = bufferedReader.readLine()) != null) {
        line = Decrypt(line, AES_Key, AES_iv);
        jta.append(line + System.lineSeparator());
    }

    // Close the socket channel
    socket.close();

} catch (Exception e) {
    e.printStackTrace();
}
/*************************************************** TCP Client ***************************************************/
```

Figure 6: Client Socket

## 2.3 Encrypt & Decrypt

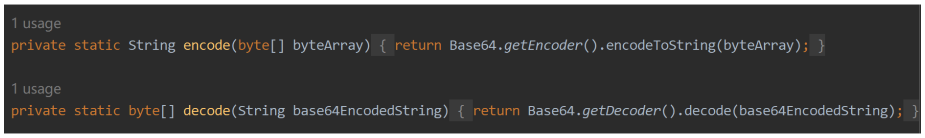
*In this project, we use the group symmetric encryption algorithm AES128CBC to encrypt the session. Firstly, we set AES_Key and AES_iv as following:*



```
// AES Key & iv
2 usages
private final String AES_Key = "CSE433SIntroToCS";
2 usages
private final String AES_iv = "XINZHAO 512912ZX";
```

Figure 7: AES Key & AES iv

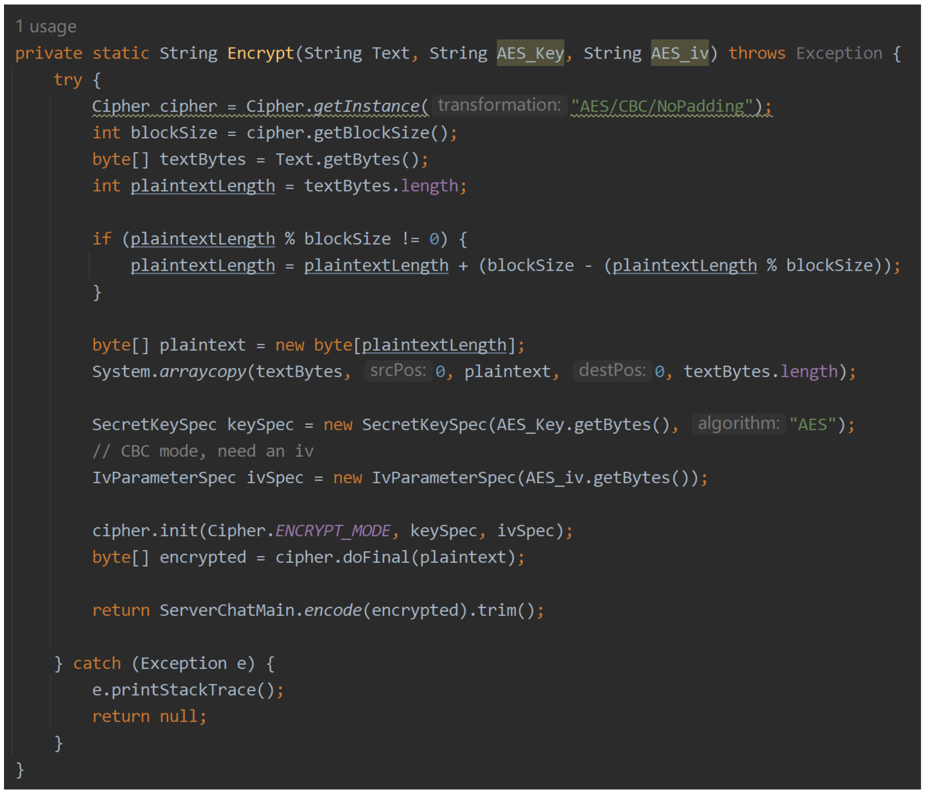
*We also use java.util.Base64 to convert our data between String and byte[]:*



```
1 usage
private static String encode(byte[] byteArray) { return Base64.getEncoder().encodeToString(byteArray); }

1 usage
private static byte[] decode(String base64EncodedString) { return Base64.getDecoder().decode(base64EncodedString); }
```

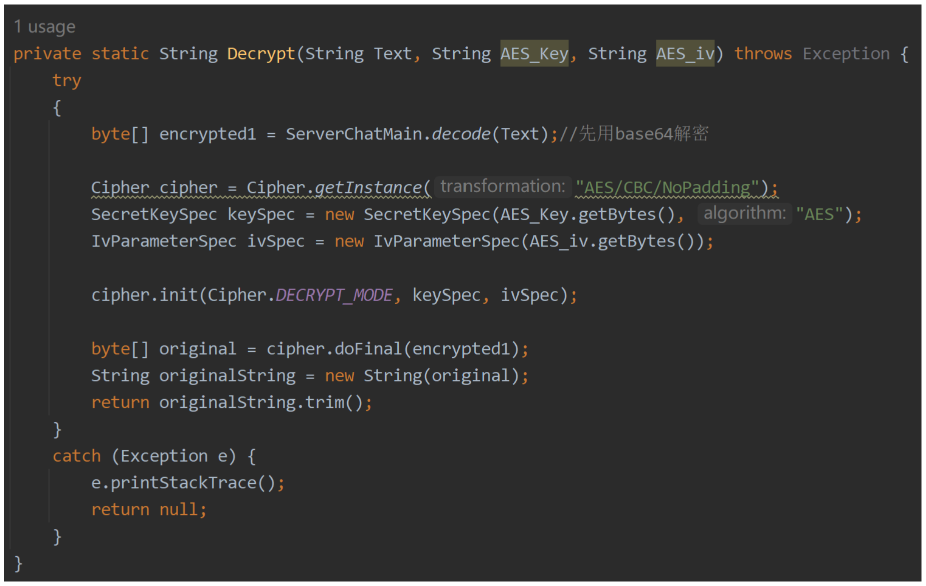
Figure 8: Base64

*And we use javax.crypto to encrypt and decrypt the message:*



```
1 usage
private static String Encrypt(String Text, String AES_Key, String AES_iv) throws Exception {
    try {
        Cipher cipher = Cipher.getInstance( transformation: "AES/CBC/NoPadding");
        int blockSize = cipher.getBlockSize();
        byte[] textBytes = Text.getBytes();
        int plaintextLength = textBytes.length;

        if (plaintextLength % blockSize != 0) {
            plaintextLength = plaintextLength + (blockSize - (plaintextLength % blockSize));
        }

        byte[] plaintext = new byte[plaintextLength];
        System.arraycopy(textBytes,  srcPos: 0, plaintext,  destPos: 0, textBytes.length);

        SecretKeySpec keySpec = new SecretKeySpec(AES_Key.getBytes(),  algorithm: "AES");
        // CBC mode, need an iv
        IvParameterSpec ivSpec = new IvParameterSpec(AES_iv.getBytes());

        cipher.init(Cipher.ENCRYPT_MODE, keySpec, ivSpec);
        byte[] encrypted = cipher.doFinal(plaintext);

        return ServerChatMain.encode(encrypted).trim();

    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}
```

Figure 9: Encrypt



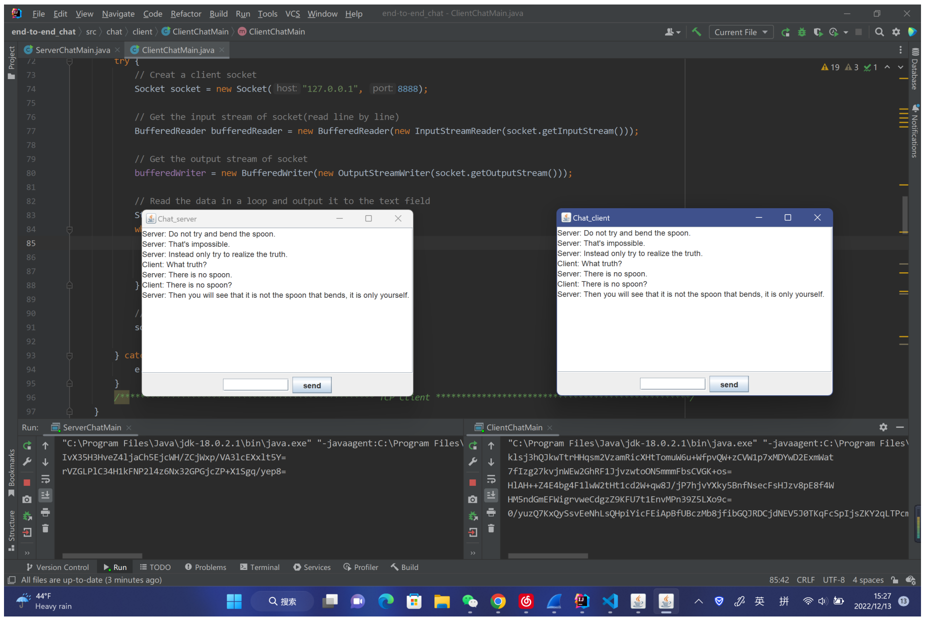
```
1 usage
private static String Decrypt(String Text, String AES_Key, String AES_iv) throws Exception {
    try
    {
        byte[] encrypted1 = ServerChatMain.decode(Text);//先用base64解密

        Cipher cipher = Cipher.getInstance( transformation: "AES/CBC/NoPadding");
        SecretKeySpec keySpec = new SecretKeySpec(AES_Key.getBytes(), algorithm: "AES");
        IvParameterSpec ivSpec = new IvParameterSpec(AES_iv.getBytes());

        cipher.init(Cipher.DECRYPT_MODE, keySpec, ivSpec);

        byte[] original = cipher.doFinal(encrypted1);
        String originalString = new String(original);
        return originalString.trim();
    }
    catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}
```

Figure 10: Decrypt

## 3 Experiment and Result

*Then we run the ServerChatMain.java file first, and then run the ClientChat-Main.java, we will have two dialog boxes shown in the screen, we can chat in them, and print the message received in server and client program.*

*Let's use the Classic Dialogue about spoon in The Matrix to test this chat room:*


```
        try {
            // Creat a client socket
            Socket socket = new Socket( host: "127.0.0.1",  port: 8888);

            // Get the input stream of socket(read line by line)
            BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));

            // Get the output stream of socket
            bufferedWriter = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));

            // Read the data in a loop and output it to the text field
```

```
Server: Do not try and bend the spoon.
Server: That's impossible.
Server: Instead only try to realize the truth.
Client: What truth?
Server: There is no spoon.
Client: There is no spoon?
Server: Then you will see that it is not the spoon that bends, it is only yourself.
```

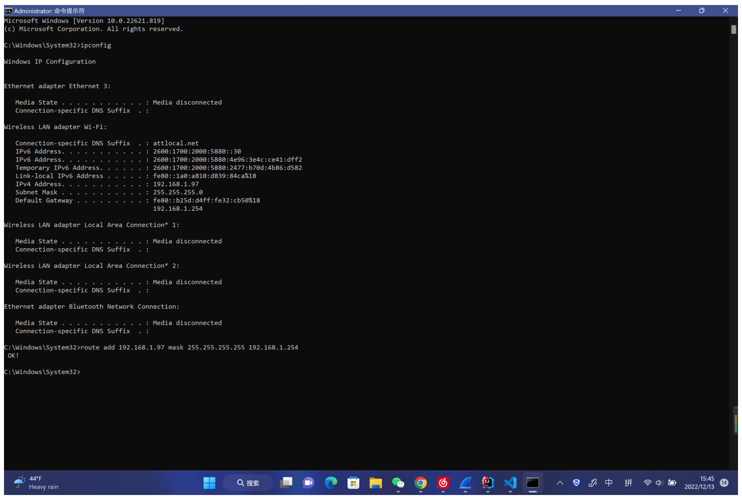
```
"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" "-javaagent:C:\Program Files\
IvX35H3HveZ4ljaCh5EjcWH/ZCjWxp/VA3lcEXxlt5Y=
rVZGLPlC34H1kFNP2l4z6Nx32GPGjcZP+X1Sgq/yep8=
```

```
"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" "-javaagent:C:\Program Files\
klsj3hQJkwTtrHHqsm2VzamRicXHtTomuW6u+WfpvQW+zCVW1p7xMDYwD2ExmWat
7fIzg27kvjnWEw2GhRF1JjvzwtoON5mmmFbsCVGK+os=
HlAH++Z4E4bg4F1lwW2tHt1cd2W+qw8J/jP7hjvYXky5BnfNsecFsHJzv8pE8f4W
HM5ndGmEFWigrvweCdgzZ9KFU7t1EnvMPn39Z5LXo9c=
0/yuzQ7KxQySsvEeNhLsQHpiYicFEiApBfUBczMb8jfibGQJRDCjdNEV5J0TKqFcSpIjsZKY2qLTPcn
```

Figure 11: Chatting

*As we can see above, the message we want to send can be shown correctly in the Text Area, However, the message passed in the socket channel is encrypt, a Hacker who are able to catch the package cannot figure out what we are talking about with out AES_Key and AES_iv.*

*To make sure that a hacker cannot get to know the real message , we use wireshark to capture the packages, Since WireShark can only capture packets passing through the computer Default Gateway, we use the following operations to allow local traffic to pass through the Default Gateway:*

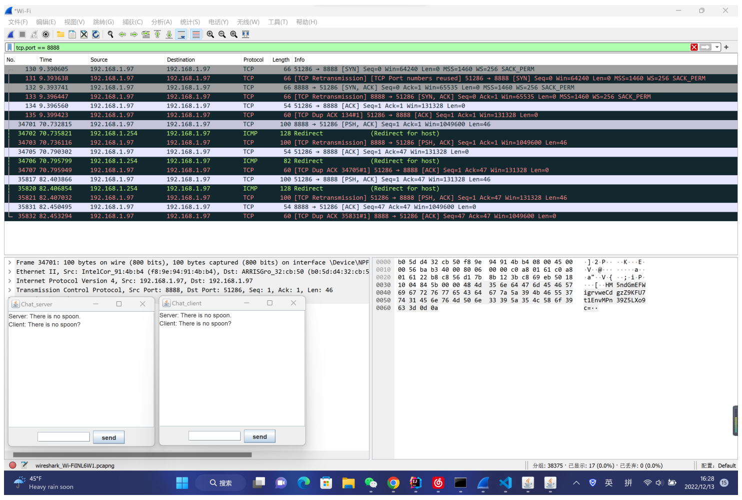


```
Microsoft Windows [Version 10.0.22621.819]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>ipconfig

Windows IP Configuration


Ethernet adapter Ethernet 3:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . : attlocal.net
   IPv6 Address. . . . . . . . . . . : 2600:1700:2000:5880::30
   IPv6 Address. . . . . . . . . . . : 2600:1700:2000:5880:4e96:3e4c:ce41:dff2
   Temporary IPv6 Address. . . . . . : 2600:1700:2000:5880:2477:b70d:4b86:d582
   Link-local IPv6 Address . . . . . : fe80::1a0:a810:d839:84ca%18
   IPv4 Address. . . . . . . . . . . : 192.168.1.97
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : fe80::b25d:d4ff:fe32:cb50%18
                                       192.168.1.254

Wireless LAN adapter Local Area Connection* 1:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 2:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Ethernet adapter Bluetooth Network Connection:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

C:\Windows\System32>route add 192.168.1.97 mask 255.255.255.255 192.168.1.254
 OK!

C:\Windows\System32>
```

Figure 12: Add route of local IP

*Then we use wireshark to capture the package, whose port is 8888, and run the program:*

*Firstly, we send message: There is no spoon. From server to client, and find out the message in package is as below:*

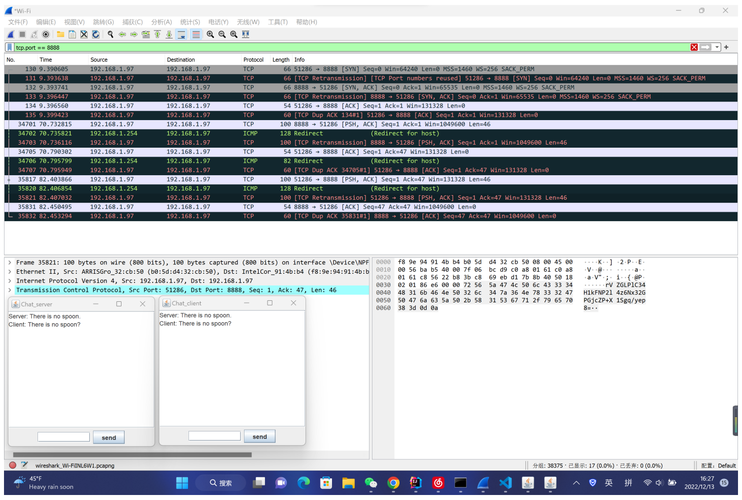


```
tcp.port == 8888

No.   Time      Source         Destination   Protocol Length Info
  130 9.390605  192.168.1.97   192.168.1.97  TCP       66 51286 → 8888 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
  131 9.393638  192.168.1.97   192.168.1.97  TCP       66 [TCP Retransmission] [TCP Port numbers reused] 51286 → 8888 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
  132 9.393741  192.168.1.97   192.168.1.97  TCP       66 8888 → 51286 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
  133 9.396447  192.168.1.97   192.168.1.97  TCP       66 [TCP Retransmission] 8888 → 51286 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
  134 9.396560  192.168.1.97   192.168.1.97  TCP       54 51286 → 8888 [ACK] Seq=1 Ack=1 Win=131328 Len=0
  135 9.399423  192.168.1.97   192.168.1.97  TCP       60 [TCP Dup ACK 134#1] 51286 → 8888 [ACK] Seq=1 Ack=1 Win=131328 Len=0
34701 70.732815 192.168.1.97   192.168.1.97  TCP      100 8888 → 51286 [PSH, ACK] Seq=1 Ack=1 Win=1049600 Len=46
34702 70.735821 192.168.1.254  192.168.1.97  ICMP     128 Redirect             (Redirect for host)
34703 70.736116 192.168.1.97   192.168.1.97  TCP      100 [TCP Retransmission] 8888 → 51286 [PSH, ACK] Seq=1 Ack=1 Win=1049600 Len=46
34705 70.790302 192.168.1.97   192.168.1.97  TCP       54 51286 → 8888 [ACK] Seq=1 Ack=47 Win=131328 Len=0
34706 70.795799 192.168.1.254  192.168.1.97  ICMP      82 Redirect             (Redirect for host)
34707 70.795949 192.168.1.97   192.168.1.97  TCP       60 [TCP Dup ACK 34705#1] 51286 → 8888 [ACK] Seq=1 Ack=47 Win=131328 Len=0
35817 82.403866 192.168.1.97   192.168.1.97  TCP      100 51286 → 8888 [PSH, ACK] Seq=1 Ack=47 Win=131328 Len=46
35820 82.406854 192.168.1.254  192.168.1.97  ICMP     128 Redirect             (Redirect for host)
35821 82.407032 192.168.1.97   192.168.1.97  TCP      100 [TCP Retransmission] 51286 → 8888 [PSH, ACK] Seq=1 Ack=47 Win=131328 Len=46
35831 82.450495 192.168.1.97   192.168.1.97  TCP       54 8888 → 51286 [ACK] Seq=47 Ack=47 Win=1049600 Len=0
35832 82.453294 192.168.1.97   192.168.1.97  TCP       60 [TCP Dup ACK 35831#1] 8888 → 51286 [ACK] Seq=47 Ack=47 Win=1049600 Len=0

> Frame 34701: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface \Device\NPF
> Ethernet II, Src: IntelCor_91:4b:b4 (f8:9e:94:91:4b:b4), Dst: ARRISGro_32:cb:50 (b0:5d:d4:32:cb:5
> Internet Protocol Version 4, Src: 192.168.1.97, Dst: 192.168.1.97
> Transmission Control Protocol, Src Port: 8888, Dst Port: 51286, Seq: 1, Ack: 1, Len: 46

0000  b0 5d d4 32 cb 50 f8 9e  94 91 4b b4 08 00 45 00   ·]·2·P·· ··K···E·
0010  00 56 ba b3 40 00 80 06  00 00 c0 a8 01 61 c0 a8   ·V··@··· ·····a··
0020  01 61 22 b8 c8 56 d1 7b  8b 12 3b c8 69 eb 50 18   ·a"··V·{ ··;·i·P·
0030  10 04 84 5b 00 00 48 4d  35 6e 64 47 6d 45 46 57   ···[··HM 5ndGmEFW
0040  69 67 72 76 77 65 43 64  67 7a 5a 39 4b 46 55 37   igrvweCd gzZ9KFU7
0050  74 31 45 6e 76 4d 50 6e  33 39 5a 35 4c 58 6f 39   t1EnvMPn 39Z5LXo9
0060  63 3d 0d 0a                                        c=··
```

```
Chat_server
Server: There is no spoon.
Client: There is no spoon?

Chat_client
Server: There is no spoon.
Client: There is no spoon?
```

Figure 13: WireShark server to client

*Secondly, we send message: There is no spoon? From client to server, and find out the message in package is as below:*



```
tcp.port == 8888
No.    Time        Source         Destination    Protocol Length Info
  130  9.390605    192.168.1.97   192.168.1.97   TCP        66 51286 → 8888 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
  131  9.393638    192.168.1.97   192.168.1.97   TCP        66 [TCP Retransmission] [TCP Port numbers reused] 51286 → 8888 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
  132  9.393741    192.168.1.97   192.168.1.97   TCP        66 8888 → 51286 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
  133  9.396447    192.168.1.97   192.168.1.97   TCP        66 [TCP Retransmission] 8888 → 51286 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
  134  9.396560    192.168.1.97   192.168.1.97   TCP        54 51286 → 8888 [ACK] Seq=1 Ack=1 Win=131328 Len=0
  135  9.399423    192.168.1.97   192.168.1.97   TCP        60 [TCP Dup ACK 134#1] 51286 → 8888 [ACK] Seq=1 Ack=1 Win=131328 Len=0
34701  70.732815   192.168.1.97   192.168.1.97   TCP       100 8888 → 51286 [PSH, ACK] Seq=1 Ack=1 Win=1049600 Len=46
34702  70.735821   192.168.1.254  192.168.1.97   ICMP      128 Redirect              (Redirect for host)
34703  70.736116   192.168.1.97   192.168.1.97   TCP       100 [TCP Retransmission] 8888 → 51286 [PSH, ACK] Seq=1 Ack=1 Win=1049600 Len=46
34705  70.790302   192.168.1.97   192.168.1.97   TCP        54 51286 → 8888 [ACK] Seq=1 Ack=47 Win=131328 Len=0
34706  70.795799   192.168.1.254  192.168.1.97   ICMP       82 Redirect              (Redirect for host)
34707  70.795949   192.168.1.97   192.168.1.97   TCP        60 [TCP Dup ACK 34705#1] 51286 → 8888 [ACK] Seq=1 Ack=47 Win=131328 Len=0
35817  82.403866   192.168.1.97   192.168.1.97   TCP       100 51286 → 8888 [PSH, ACK] Seq=1 Ack=47 Win=131328 Len=46
35820  82.406854   192.168.1.254  192.168.1.97   ICMP      128 Redirect              (Redirect for host)
35821  82.407032   192.168.1.97   192.168.1.97   TCP       100 [TCP Retransmission] 51286 → 8888 [PSH, ACK] Seq=1 Ack=47 Win=131328 Len=46
35831  82.450495   192.168.1.97   192.168.1.97   TCP        54 8888 → 51286 [ACK] Seq=47 Ack=47 Win=1049600 Len=0
35832  82.453294   192.168.1.97   192.168.1.97   TCP        60 [TCP Dup ACK 35831#1] 8888 → 51286 [ACK] Seq=47 Ack=47 Win=1049600 Len=0

> Frame 35821: 100 bytes on wire (800 bits), 100 bytes captured (800 bits) on interface \Device\NPF
> Ethernet II, Src: ARRISGro_32:cb:50 (b0:5d:d4:32:cb:50), Dst: IntelCor_91:4b:b4 (f8:9e:94:91:4b:b
> Internet Protocol Version 4, Src: 192.168.1.97, Dst: 192.168.1.97
> Transmission Control Protocol, Src Port: 51286, Dst Port: 8888, Seq: 1, Ack: 47, Len: 46

0000  f8 9e 94 91 4b b4 b0 5d  d4 32 cb 50 08 00 45 00   ····K··] ·2·P··E·
0010  00 56 ba b5 40 00 7f 06  bc d9 c0 a8 01 61 c0 a8   ·V··@··· ·····a··
0020  01 61 c8 56 22 b8 3b c8  69 eb d1 7b 8b 40 50 18   ·a·V"·;· i··{·@P·
0030  02 01 86 e6 00 00 72 56  5a 47 4c 50 6c 43 33 34   ······rV ZGLPlC34
0040  48 31 6b 46 4e 50 32 6c  34 7a 36 4e 78 33 32 47   H1kFNP2l 4z6Nx32G
0050  50 47 6a 63 5a 50 2b 58  31 53 67 71 2f 79 65 70   PGjcZP+X 1Sgq/yep
0060  38 3d 0d 0a                                        8=··
```

Chat_server

Server: There is no spoon.
Client: There is no spoon?

Chat_client

Server: There is no spoon.
Client: There is no spoon?

*Wi-Fi
文件(F) 编辑(E) 视图(V) 跳转(G) 捕获(C) 分析(A) 统计(S) 电话(Y) 无线(W) 工具(T) 帮助(H)
send
send
wireshark_Wi-FiIINL6W1.pcapng
配置: Default
45°F
Heavy rain soon
16:27
2022/12/13

Figure 14: WireShark client to server

*In this way, we can gurantee that the attacker cannot get to know the talk without the key and iv, even if he can monitor the channel.*

## 4 Discussion and Conclusion

*In the project, we finished a end-to-end encrypt chat room. Then use wireshark to capture the message and find out that it is safe. However, this project still has some improvement directions: for example, the front-end UI design needs to be improved to become more beautiful, and more encryption methods can be provided to meet the needs, etc.*