

Δίκτυα Υπολογιστών II

Υποχρεωτική Εργασία Chat and VoIP App

Ομάδα AB



Figure 0.1: Το GUI του προγράμματος

Contents

1	Περίληψη	3
2	Περιγραφή κώδικα	4
2.1	UDP Chat	4
2.1.1	Η κλάση UDPChat	4
2.1.2	Η Chat λειτουργία με UDP στην κλάση App	5
2.2	VoIP	9
2.2.1	Οι κλάσεις AudioRecord και AudioPlayback	9
2.2.2	Η κλάση VoIP	11
2.2.3	Η VoIP λειτουργία στην κλάση App	13
2.3	TCP Chat	16
2.3.1	Η κλάση TCPChatClient	16
2.3.2	Η κλάση TCPChatSender	18
2.3.3	Η Chat λειτουργία με TCP στην κλάση App	19
2.4	Encryption	22
2.4.1	Ο αλγόριθμος "AES/GCM/NoPadding"	22
2.4.2	Ορισμός μεταβλητών και αρχικοποίηση	23
2.4.3	Ορισμός μεθόδων	24
2.5	Λοιπές λειτουργίες	31
2.6	Το κουμπί "Set Pass"	31
2.7	Το κουμπί "Clear Chat"	31
2.8	Ορισμός των κουμπιών του προγράμματος	31
3	Απεικόνιση πακέτων μέσω Wireshark	33
3.1	Πακέτα κειμένου	34
3.1.1	UDP πακέτα κειμένου	34
3.1.2	TCP πακέτα κειμένου	37
3.2	Stream πακέτων φωνής	40
3.3	Πακέτα φωνής	42
4	Limitations	44

1 Περίληψη

Η εργασία αφορά την ανάπτυξη μιας End-to-end Chat and VoIP εφαρμογής σε Java που επιτρέπει σε δύο διαφορετικούς χρήστες, local και remote, να ανταλλάσσουν κρυπτογραφημένα μηνύματα κειμένου αλλά και να επικοινωνούν φωνητικά σε επίπεδο τοπικού δικτύου (LAN) ή μέσω εικονικού ιδιωτικού δικτύου (VPN). Η End-to-end επικοινωνία, εξασφαλίζει πως τα δεδομένα αποστέλλονται απευθείας στον τελικό χρήστη, χωρίς να περνούν μέσω κάποιου κεντρικού server, όπως λειτουργούν οι περισσότερες σύγχρονες εφαρμογές επικοινωνίας μέσω διαδικτύου, όπως είναι το Skype, το Viber, κλπ. Η αποκεντροποιημένη αυτή μορφή επικοινωνίας ενισχύει την ιδιωτικότητα της επικοινωνίας αφού, κεντρικοί servers αυτών των συστημάτων αποτελούν στόχους συχνών και επικίνδυνων επιθέσεων, ακόμη και η προσωρινή αποθήκευση των μηνυμάτων μπορεί να οδηγήσει σε παραβίαση ιδιωτικότητας σε περίπτωση επιτυχημένης επίθεσης.

Η εργασία αποτελείται από δύο τμήματα τα οποία αφορούν την υλοποίηση των δύο βασικών λειτουργιών της εφαρμογής, οι οποίες είναι η αποστολή μηνυμάτων και η φωνητική επικοινωνία σε πραγματικό χρόνο. Για το πρώτο κομμάτι, γίνεται χρήση του πρωτοκόλλου UDP και του TCP για πιο ποιοτική και αξιόπιστη επικοινωνία καθώς και χρήση ενός αλγορίθμου κρυπτογράφησης, στην UDP επικοινωνία, για την επίτευξη πλήρους ανώνυμης και ιδιωτικής επικοινωνίας μεταξύ των δύο μερών. Στο δεύτερο κομμάτι γίνεται χρήση μόνο του UDP. Αρχικά, γίνεται σύντομη περιγραφή του κώδικα και μετά απεικονίζονται εικόνες από το Wireshark οι οποίες δείχνουν τη μορφή πακέτων κειμένου που στάλθηκαν μεταξύ των δύο users με χρήση UDP και TCP, το stream των πακέτων φωνής που ανταλλάσσονται με τη λειτουργία VoIP και τη μορφή πακέτων φωνής που ανταλλάχθηκαν μεταξύ των δύο users.

2 Περιγραφή κώδικα

2.1 UDP Chat

2.1.1 Η κλάση UDPChat

Για την υλοποίηση του chat με χρήση UDP χρησιμοποιείται η κλάση UDPChat. Με την UDPChat ο local στέλνει πακέτα κειμένου στον remote και λαμβάνει αντίστοιχα. Σε αυτήν την κλάση αρχικά, ορίζονται τα αντικείμενα remoteAddress, της κλάσης InetAddress που περιέχει την IP διεύθυνση του remote user, datagramSocket, της κλάσης DatagramSocket για τη δημιουργία UDP Socket σύνδεσης και sendBuffer και receiveBuffer, buffers μεγέθους 1024 bytes για αποθήκευση των μηνυμάτων κειμένου που θα σταλούν και θα ληφθούν αντίστοιχα. Έπειτα, αρχικοποιούνται τα remoteAddress και datagramSocket με τον constructor UDPChat.

```
1 package com.cn2.communication;
2
3 import java.io.IOException;
4 import java.net.DatagramPacket;
5 import java.net.DatagramSocket;
6 import java.net.InetAddress;
7 //import java.nio.charset.StandardCharsets; //for utf test[]
8
9
10 import javax.sound.sampled.LineUnavailableException;
11 import javax.swing.JTextArea;
12
13 public class UDPChat { // class for chat using UDP
14
15     private InetAddress remoteAddress; // define IP address remoteAddress, to set it as IP of remote
16     private DatagramSocket datagramSocket; // define DatagramSocket datagramSocket
17     private byte[] sendBuffer = new byte[1024]; // define buffer to store messages, size = 1024 byte
18     private byte[] receiveBuffer = new byte[1024];
19     public UDPChat(DatagramSocket datagramSocket, InetAddress remoteAddress) throws LineUnavailableException {
20         // constructor UDPChat, initialize datagramSocket, remoteAddress
21
22         this.remoteAddress = remoteAddress;
23         this.datagramSocket = datagramSocket;
24     }
25 }
```

Figure 2.1: Ορισμός και αρχικοποίηση των αντικειμένων της κλάσης UDPChat

Μετά, ορίζονται οι μέθοδοι send και receive της κλάσης UDPChat. Η μέθοδος send έχει ως όρισμα το μήνυμα κειμένου messageToRemote που στέλνει ο local στον remote. Αρχικά, το κείμενο messageToRemote μετατρέπεται σε bytes και αποθηκεύεται στον sendBuffer. Ορίζεται ένα αντικείμενο της κλάσης DatagramPacket, το datagramPacket που περιέχει το κείμενο messageToRemote και το στέλνεται στην IP διεύθυνση του remote user remoteAddress και πιο συγκεκριμένα στο port 1234.

```
26 public void send(String messageToRemote) throws LineUnavailableException { // method send, local sends text messageToRemote
27     try {
28         sendBuffer = messageToRemote.getBytes(); // convert messageToRemote to bytes and put to buffer
29         DatagramPacket datagramPacket = new DatagramPacket(sendBuffer, sendBuffer.length, remoteAddress, 1234); /* construct datagramPacket,
30         send packets of length of buffer, to IP inetAddress and port=1234 of remote */
31         datagramSocket.send(datagramPacket); // send datagramPacket
32
33         // debug, prints char length of messageToRemote on the console
34         System.err.println(messageToRemote.length());
35         // debug, prints byte length of messageToRemote on the console
36         System.err.println(messageToRemote.getBytes().length);
37         // debug, prints byte length of sendBuffer on the console
38         System.err.println(sendBuffer.length);
39         // debug, prints byte length of datagramPacket on the console
40         System.err.println(datagramPacket.getLength());
41     } catch (IOException e) { // in case of error
42         e.printStackTrace();
43     }
44 }
```

Figure 2.2: Η μέθοδος send της κλάσης UDPChat

Η μέθοδος receive έχει ως όρισμα την περιοχή κειμένου που εμφανίζονται τα μηνύματα που έχουν σταλεί ή ληφθεί, ένα αντικείμενο της κλάσης JTextArea, το textArea και ένα αντικείμενο της κλάσης κρυπτογράφησης AESci, το aesci. Σε ένα while loop που τρέχει για πάντα (while (true)), αφού ο local αναμένει μηνύματα κειμένου από τον remote, ορίζεται ένα αντικείμενο της κλάσης DatagramPacket, το datagrampacket με μέγεθος όσο ο receiveBuffer με το οποίο ο local λαμβάνει τα πακέτα που έστειλε ο remote. Στην συνέχεια, δημιουργείται ένα string, το messageFromRemote, από το datagrampacket, το οποίο αφότου αποκρυπτογραφηθεί από την aesci.decryptMessage, γίνεται append στο textArea (με πρόθεμα την φράση "remote: " για να φαίνεται πως το έστειλε ο remote). Η παραπάνω διαδικασία τοποθετείται σε Thread ώστε η λήψη μηνυμάτων να μην εμποδίζει τη λειτουργία του app. Το do...while(true) μεταφέρθηκε από τη main στη μέθοδο receive και μέσα στο Thread για να μην παράγεται μεγάλος αριθμός Thread όσο τρέχει το app (αλλιώς καταλαμβάνονται μεγάλα ποσά μνήμης και μειώνεται η αποδοτικότητα του app).

```

44 public void receive(JTextArea textArea, AESci aesci) throws InterruptedException { // method receive, local receives text messageFromRemote
45
46     new Thread() -> { // Thread the receive text process
47         while (true) { // local always waiting to receive data, infinite loop
48             try {
49                 DatagramPacket datagramPacket = new DatagramPacket(receiveBuffer, receiveBuffer.length); /* construct datagramPacket,
50                 receive packets of length of buffer */
51                 datagramSocket.receive(datagramPacket); // datagramPacket received from datagramSocket, blocking method
52                 String messageFromRemote = new String(datagramPacket.getData(), 0, datagramPacket.getLength());
53                 // creates string from datagramPacket byte array by remote, offset=0
54
55                 aesci.exportKeys(); // debug, prints key and IV used to decrypt message
56                 System.err.println(datagramPacket.getLength()); // debug, prints byte length of datagramPacket on the console
57                 messageFromRemote = aesci.decryptMessage(messageFromRemote);
58                 aesci.exportKeys(); // debug, prints key and IV used after decrypting message
59                 textArea.append("remote: " + messageFromRemote + "\n"); // append messageFromRemote to textArea and change line
60             }
61             catch (IOException e) { // in case of error
62                 e.printStackTrace();
63             }
64             catch (Exception e) {
65                 // TODO Auto-generated catch block
66                 e.printStackTrace();
67             } // maybe unnecessary
68         }
69     }.start(); // start Thread
70 }

```

Figure 2.3: Η μέθοδος receive της κλάσης UDPChat

2.1.2 Η Chat λειτουργία με UDP στην κλάση App

Τέλος, περιγράφεται η chat λειτουργία με UDP στην κύρια κλάση της εφαρμογής App, που περιέχει και τη main. Ξεκινάει ορίζοντας τα αντικείμενα remoteAddress της κλάσης InetAddress που περιέχει την IP διεύθυνση του remote user και chatUDP, της κλάσης UDPChat. Μετά, αρχικοποιείται το chatUDP με έναν non-static initialization block, αφού δεν είναι static, χρησιμοποιώντας τον constructor UDPChat με όρισμα ένα DatagramSocket από το port 1234 που χρησιμοποιήθηκε για τη chat επικοινωνία με UDP και την remoteAddress.

```

private InetAddress remoteAddress; // define IP address remoteAddress, to set it as IP of remote
private UDPChat chatUDP; // define UDPChat object for UDP Chat
private VoIP voip; // define VoIP object for VoIP
private boolean isCallActive = false; // VoIP call not happening

// A note for tcp
// If you plan to use it:
// 1) uncomment the appropriate object
// 2) uncomment the appropriate constructor
// 3) uncomment the TCP receive()
// 4) comment the UDP SEND area
// 5) uncomment the TCP SEND area

// private TCPChatClient chatTCP; // define TCPChat object for TCP Chat, if local is the "client"
private TCPChatServer chatTCP; // define TCPChat object for TCP Chat, if local is the "server"

// define AES variable (AES + *cipher -> AESci)
public AESci aescl;

• { // initialize network variables using non-static initialization block

try {
    remoteAddress = InetAddress.getByAddress("192.168.208.134"); // initialize remoteAddress, IP of remote
    chatUDP = new UDPChat(new DatagramSocket(1234), remoteAddress); /* initialize chatUDP,
    // pass DatagramSocket from port 1234 and remoteAddress to constructor UDPChat */
    voip = new VoIP(new DatagramSocket(1243), remoteAddress); /* initialize voip,
    // pass DatagramSocket from port 1243 and remoteAddress to constructor VoIP */

    // initialize chatTCP, pass Socket from port 2345 and IP of remote to constructor TCPChatSender
    chatTCP = new TCPChatClient(new Socket(remoteAddress, 2345));
    // initialize chatTCP, pass ServerSocket from port 2345 to constructor TCPChatReceiver
    chatTCP = new TCPChatServer(new ServerSocket(2345));
}

```

Figure 2.4: Ορισμός και αρχικοποίηση των αντικειμένων στην κλάση App για το chat με UDP

Τα μηνύματα λαμβάνονται με κλήση της receive στην main.

```

159 /*
160  * 2. Start receiving Chat messages
161  */
156 app.chatUDP.receive(textArea, app.aescl); // call method receive from chatUDP, receive text data
157
158 // TCP isn't encrypted
159 // Keep it commented if TCP isn't used
160 // app.chatTCP.receive(textArea); // call method receive from TCPChatSender or TCPChatReceiver, receive text data

```

Figure 2.5: Η μέθοδος receive της κλάσης UDPChat στην App για λήψη μηνυμάτων κειμένου

Για την αποστολή μηνυμάτων μέσω UDP έχουμε ορίσει στην μέθοδο actionPerformed την αντίστοιχη λειτουργία.

Αν πατηθεί το κουμπί "Send" τότε γράφουμε στην μεταβλητή messageToSend το κείμενο που θέλουμε να στείλουμε και εφόσον δεν είναι κενό και δεν ξεπερνά τους 500 χαρακτήρες, το κρυπτογραφούμε με την aescl.encryptMessage και το στέλνουμε διοχετευοντάς το ως όρισμα στην μέθοδο send. Επιπλέον κάνουμε append στο textArea το μήνυμα με πρόθεμα την φράση "local: " για να φαίνεται πως το στείλαμε εμείς και καθαρίζουμε το πεδίο αποστολής θέτοντας ως κενό string το όρισμα της inputTextField.setText().

```

180 if (e.getSource() == sendButton){ // The "Send" button was clicked
181
182     String messageToSend = inputTextField.getText(); // get string messageToSend from TextField inputTextField
183     // All of this if should be commented out if we plan to use TCP (TCP uses ~64Kb max buffer, we don't need chunks)
184     //UDP SEND [START]
185     if (!messageToSend.isEmpty()) { // if there is a messageToSend
186         try {
187             String plainMessage = messageToSend; //stores the message in plaintext temporarily
188
189             if (messageToSend.length() < 501) { // If message is under 500 chars it leaves as a single packet
190                 aescl.exportKeys(); // debug, prints the key and IV of the cipher we will use on the console
191
192                 messageToSend = aescl.encryptMessage(messageToSend,1); //encrypts the message to be send
193                 chatUDP.send(messageToSend); // call method send from chatUDP, send text data
194
195                 aescl.exportIV(); // debug, prints the new IV we created on the console
196             }

```

Figure 2.6: Η μέθοδος send της κλάσης UDPChat στην App για αποστολή μηνυμάτων κειμένου ≤ 500 chars

Αν το μήνυμα είναι πάνω από 500 χαρακτήρες (μέχρι 50k) πρέπει το

στείλουμε σε κομμάτια. Ορίζουμε ένα chunk να περιέχει 500 χαρακτήρες το μέγιστο από το τμήμα του μηνύματος. Εύκολα θα μπορούσαμε να αλλάξουμε το μέγεθος chunk, εφόσον σιγουρευτούμε πως τα sendBuffer και receiveBuffer της κλάσης αρκούν για το μήνυμα, καθώς και πως το μέγεθος του πακέτου UDP που δημιουργείται δεν ξεπερνά το μέγιστο ασφαλές μήκος. Ορίζουμε επίσης ως αρχή του μηνύματος το πρόθεμα "[Part]".

Με μία λούπα διαπερνάμε το μήνυμα ανά chunk, με έναν counter, το i, που κάθε φορά αυξάνει όσο το chunkSize και το endIndex που δείχνει στο τέλος του τμήματος του μηνύματος που θέλουμε να δημιουργήσουμε (στην πολύ πιθανή περίπτωση που το τελευταίο τμήμα του μηνύματος που περισσεύει είναι κάτω από 500 χαρακτήρες, θέτουμε το endIndex να δείχνει στον τελευταίο χαρακτήρα του συνολικού μηνύματος). Παράλληλα έχουμε έναν counter που αυξάνει +1 κάθε φορά στο τέλος της λούπας, που το χρησιμοποιούμε ως αύξοντα αριθμό για τα chunks μας. Αυτό μας δίνει ένα πολύ σημαντικό προτέρημα, σημαίνει πως αν τα chunks έρθουν σε διαφορετική σειρά θα ενωθούν με την σωστή σειρά με την χρήση του αύξοντος αριθμού.

Οπότε τα κρυπτογραφούμε βάζοντας ως πρόθεμα την ετικέτα "[Part]" και τον διψήφιο αύξοντα αριθμό (μέχρι 100 chunks θα σταθλούν, από 00 μέχρι 99). Επειδή υπάρχει μεγάλη περίπτωση να έρθουν με διαφορετική σειρά, ορίζουμε κατά την κρυπτογράφηση με το δεύτερο όρισμα ως 0, να μην δημιουργηθεί νέο ευφήμερο IV κλειδί για αυτά τα μηνύματα (περισσότερα για αυτό στην κλάση AESSci). Αν κάναμε νέο κλειδί και ερχόταν έστω και ένα με διαφορετική σειρά, όλο το σύστημα απο/κρυπτογράφησης θα αποτύγχανε μέχρι να ξανανοίξουμε την εφαρμογή. Μία άλλη λύση θα ήταν να περιμένουμε λίγο πριν στείλουμε το επόμενο chunk, αλλά επειδή αυτό δεν είναι σίγουρο πως θα λειτουργήσει δεν το εφαρμόσαμε.

```
197         else if (messageToSend.length() < 50001) { // texts over 500 chars will be split and sent in chunks
198             int chunkSize = 500; // Sets the maximum chunk size
199             // Debug, calculates the number of iterations needed to process the entire string:
200             int numIterations = messageToSend.length() / chunkSize;
201             // If (messageToSend.length() % chunkSize > 0) {
202             //     numIterations++;
203             // }
204             int j = 0; // Chunk counter
205             String part;
206             part = "[Part]"; // Identification tag for the decryption method
207             // Iterate over the string in chunks of a specified size
208             for (int i = 0; i < messageToSend.length(); i += chunkSize) {
209
210                 // Calculate the end index of the current chunk
211                 int endIndex = i + chunkSize; // 0, 500, 1000, 1500...
212                 // If the end index is greater than the length of the string, set it to the length of the string
213                 if (endIndex > messageToSend.length()) {
214                     endIndex = messageToSend.length();
215                 }
216
217                 // Extract and *encrypt* the string chunk from the original string
218                 // Note: for each encryption, it does NOT generate a new IV, hence the ", 0"
219                 if (j < 10) {
220                     part = "[Part]" + j + aesci.encryptMessage(messageToSend.substring(i, endIndex));
221                     part = (aesci.encryptMessage("[Part]" + j + messageToSend.substring(i, endIndex), 0));
222                     aesci.exportIV(); // debug, prints the new IV we created on the console
223                     Thread.sleep(150); // Waits 150ms for each packet to be sent before sending the next
224                 }
225                 else {
226                     part = (aesci.encryptMessage("[Part]" + j + messageToSend.substring(i, endIndex), 0));
227                     aesci.exportIV(); // debug, prints the new IV we created on the console
228                     Thread.sleep(150); // Waits 150ms for each packet to be sent before sending the next
229                 }
230             }
231         }
```

Figure 2.7: Η μέθοδος send της κλάσης UDPChat στην App για αποστολή μεγάλων μηνυμάτων χειμένου ≤ 50000 chars

```

230 // Result format is as such: <iv:16chars>[Part]0<encrypted-text>
231 // Alternative, needs locale library to set locale and be consistent:
232 // part = (aesci.encryptMessage("[Part]" + String.format("%02d", j) + messageToSend.substring(l, endIndex), 0) );
233
234 System.err.println("Chunk[" + j + "]: " + part + "\n"); // debug, prints each chunk on the console
235 chatUDP.send(part); // sends the chunk
236 j++; // Adds +1 to the chunk counter
237

```

Figure 2.8: Η μέθοδος send της κλάσης UDPChat στην App για αποστολή μεγάλων μηνυμάτων χειμένου ≥ 50000 chars

Το τελευταίο βήμα της διαδικασίας είναι, αφότου βγούμε από την λούπα, να στείλουμε κρυπτογραφημένο το string "[Part]FINISHED" (αυτήν την φορά με το δεύτερο όρισμα στο 1 παράγουμε νέο ευφήμερο IV κλειδί), το οποίο θα λειτουργήσει ως εντολή στον remote για να καταλάβει πως στείλαμε όλα τα chunks και πλέον μπορεί να τα συνενώσει σε ένα μεγάλο. Η καθυστέρηση 700ms που προηγείται της αποστολής αυτού του μηνύματος είναι για να είναι πιο πιθανό να έχουν φτάσει όλα τα chunks μέχρι να φτάσει το "[Part]FINISHED" μήνυμα. Φυσικά μετά κάνει print στον local όλο το μήνυμα που έστειλε στην περιοχή του chat.

```

238 Thread.sleep(700); // waits 700ms for all the packets to be sent before sending the next command
239
240 // Sends the command to the remote confirming that local finished sending chunks
241 // Forces the remote to combine the received chunks and print them on their screen
242 chatUDP.send(aesci.encryptMessage("[Part]FINISHED", 1) );
243

```

Figure 2.9: Η μέθοδος send της κλάσης UDPChat στην App για αποστολή μεγάλων μηνυμάτων χειμένου ≤ 50000 chars

Τέλος, αν το μήνυμα είναι υπερβολικά μεγάλο, πάνω από 50k χαρακτήρες, δεν το στέλνουμε και κάνουμε print στον local στην περιοχή του chat τα δύο strings "Message to big. Please send 50000 chars max." "Your message was not send." το ένα κάτω από το άλλο, μαζί και το μήνυμα που δεν στάλθηκε (για να μπορεί να το κάνει copy paste σε μικρότερα κομμάτια).

```

244 else if (messageToSend.length() > 50000) { // edge case where user sends >50k char message
245     textArea.append("Message to big. Please send 50000 chars max.\nYour message was not send.");
246 }
247
248 textArea.append("local: " + plainMessage + "\n"); // appear plainMessage to textArea and change line
249 inputTextField.setText(""); // erase messageToSend from inputTextField
250

```

Figure 2.10: Η μέθοδος send της κλάσης UDPChat στην App για ακύρωση αποστολής τεράστιων μηνυμάτων χειμένου > 50000 chars

2.2 VoIP

Για την υλοποίηση του VoIP χρησιμοποιούνται οι κλάσεις AudioRecord, AudioPlayback και VoIP.

2.2.1 Οι κλάσεις AudioRecord και AudioPlayback

Αρχικά, περιγράφεται η λειτουργία των AudioRecord και AudioPlayback. Με τις κλάσεις AudioRecord και AudioPlayback καταγράφεται ο ήχος με το μικρόφωνο και αναπαράγεται ήχος από τα ηχεία του υπολογιστή αντίστοιχα. Στην AudioRecord ορίζονται τα αντικείμενα targetLine, της κλάσης TargetDataLine για τη λήψη ήχου, audioFormat, της κλάσης AudioFormat για τη μορφή των δεδομένων ήχου, dataInfo, της κλάσης DataLine.Info που περιέχει τις πληροφορίες για τον ήχο και έναν buffer μεγέθους 1024 bytes για αποθήκευση του ήχου που ηχογραφείται. Μετά, τα audioFormat και dataInfo αρχικοποιούνται με τον constructor AudioRecord. Στο audioFormat υιοθετούμε διαμόρφωση PCM, με συχνότητα δειγματοληψίας 8000 samples/sec, μέγεθος δείγματος 8 bits, μονοφωνικό κανάλι (1 channel), signed δείγματα (true) και littleEndian (false) (για καλύτερη ποιότητα ήχου μπορούμε να χρησιμοποιήσουμε συχνότητα δειγματοληψίας 44100 samples/sec και μέγεθος δείγματος 16 bits). Το dataInfo περιέχει τις πληροφορίες του audioFormat για τη μορφή των δεδομένων ήχου του targetLine που ηχογραφείται, ενώ το targetLine δέχεται δεδομένα ήχου από το μικρόφωνο με μορφή που ορίζει το dataInfo και εξασφαλίζεται ότι το AudioSystem μπορεί να υποστηρίξει το συγκεκριμένο dataInfo με μία συνθήκη if.

```
1 package com.cn2.communication;
2
3 import javax.sound.sampled.AudioFormat;
4 import javax.sound.sampled.AudioSystem;
5 import javax.sound.sampled.DataLine;
6 import javax.sound.sampled.TargetDataLine;
7 import javax.sound.sampled.LineUnavailableException;
8
9 public class AudioRecord { // class for recording sound
10
11     private final TargetDataLine targetLine; // define targetLine for capturing audio
12     private final AudioFormat audioFormat; // define audio format
13     private final DataLine.Info dataInfo; // define info for the audio.
14     private byte[] buffer = new byte[1024]; // define buffer to store stream in
15
16     public AudioRecord() throws LineUnavailableException { // constructor AudioCapture, initialize variables
17
18         this.audioFormat = new AudioFormat(8000, 8, 1, true, false); /* audio format: sampleRate=8000 samples/sec,
19         sampleSize=8 bits, 1 channel, signed (true) PCM, littleEndian (false) */
20         this.dataInfo = new DataLine.Info(TargetDataLine.class, audioFormat); /* object dataInfo, contains information
21         on what type of audio format targetLine must have */
22
23         if (!AudioSystem.isLineSupported(dataInfo)) { // check if audio is supported
24             System.out.println("Not supported");
25         }
26
27         this.targetLine = (TargetDataLine) AudioSystem.getLine(dataInfo); // get targetLine
28     }
29 }
```

Figure 2.11: Ορισμός και αρχικοποίηση των αντικειμένων της κλάσης AudioRecord

Έπειτα, ορίζονται οι μέθοδοι της κλάσης AudioRecord. Με τη μέθοδο open ξεκινάει να λαμβάνεται ήχος από το μικρόφωνο, με μορφή audioFormat, η μέθοδος read διαβάζει τα δεδομένα ήχου που υπάρχουν στο targetLine,

τα βάζει στον buffer και επιστρέφει τα bytes που υπάρχουν σε αυτόν και η μέθοδος close σταματάει τη λήψη ήχου και κλείνει ό,τι μέσα χρησιμοποιούνται.

```
30 public void open() throws LineUnavailableException {
31     this.targetLine.open(audioFormat); // open targetLine
32     this.targetLine.start(); // microphone open, targetLine starts capturing data from microphone
33 }
34
35 public byte[] read() {
36     targetLine.read(buffer, 0, buffer.length); // read the recorded audio data from targetLine into buffer, offset=0 for real time usage
37     return buffer; // return bytes of data from buffer
38 }
39
40 public void close() {
41     targetLine.stop(); // stop the targetLine but retains its resources
42     targetLine.close(); // close the targetLine and releases resources
43 }
44
45 }
```

Figure 2.12: Οι μέθοδοι της κλάσης AudioRecord

Στην κλάση AudioPlayback ορίζονται αρχικά, τα αντικείμενα sourceLine, της κλάσης SourceDataLine για την αναπαραγωγή ήχου, audioFormat και dataInfo. Μετά, τα αντικείμενα αυτά αρχικοποιούνται με τον constructor AudioPlayback. Το audioFormat ορίζεται όπως στην AudioRecord, το dataInfo περιέχει τις πληροφορίες του audioFormat για τη μορφή των δεδομένων ήχου του sourceLine που αναπαράγεται και το sourceLine ορίζεται με τρόπο ώστε να αναπαράγει δεδομένα ήχου από το ηχείο με μορφή που ορίζει το dataInfo.

```
1 package com.cn2.communication;
2
3 import javax.sound.sampled.AudioFormat;
4 import javax.sound.sampled.AudioSystem;
5 import javax.sound.sampled.DataLine;
6 import javax.sound.sampled.SourceDataLine;
7 import javax.sound.sampled.LineUnavailableException;
8
9
10 public class AudioPlayback { // class for playing sound
11
12     private final SourceDataLine sourceLine; // define sourceLine for playing audio
13     private final AudioFormat audioFormat; // define audio format
14     private final DataLine.Info dataInfo; // define info for the audio
15
16     public AudioPlayback() throws LineUnavailableException { // constructor AudioPlay, initialize variables
17
18         this.audioFormat = new AudioFormat(8000, 8, 1, true, false); /* audio format: sampleRate=8000 samples/sec,
19         sampleSize=8 bits, 1 channel, signed (true) PCM, littleEndian (false) */
20         this.dataInfo = new DataLine.Info(SourceDataLine.class, audioFormat); /* object dataInfo, contains information
21         on what type of audio format sourceLine must have */
22         this.sourceLine = (SourceDataLine) AudioSystem.getLine(dataInfo); // get sourceLine
23     }
24 }
```

Figure 2.13: Ορισμός και αρχικοποίηση των αντικειμένων της κλάσης AudioPlayback

Έπειτα, ορίζονται οι μέθοδοι της κλάσης AudioPlayback. Η μέθοδος open αρχίζει να αναπαράγει ήχο με μορφή audioFormat από το ηχείο, η μέθοδος write έχοντας ως όρισμα έναν buffer, γράφει τα δεδομένα ήχου του buffer στο sourceLine και η μέθοδος close εξασφαλίζει ότι όλος ο ήχος έχει αναπαραχθεί, σταματάει την αναπαραγωγή ήχου και κλείνει και αποδεσμεύει ό,τι μέσα χρησιμοποιούνται.

```

25 public void open() throws LineUnavailableException {
26
27     sourceLine.open(audioFormat); // open sourceLine
28     sourceLine.start(); // speaker open, sourceLine starts playing audio from speaker
29 }
30
31 public void write(byte[] buffer) {
32     sourceLine.write(buffer, 0, buffer.length); // write the received audio data from buffer to sourceLine, offset=0 for real time usage
33 }
34
35
36 public void close() {
37     sourceLine.drain(); // ensure all data is played
38     sourceLine.stop(); // stop the sourceLine but retains its resources
39     sourceLine.close(); // close the sourceLine and releases resources
40 }
41
42
43 }

```

Figure 2.14: Οι μέθοδοι της κλάσης AudioPlayback

2.2.2 Η κλάση VoIP

Μετά, περιγράφεται η λειτουργία της VoIP. Ξεκινάει ορίζοντας τα αντικείμενα playback, της κλάσης AudioPlayback, record της κλάσης AudioRecord, remoteAddress της κλάσης InetAddress που περιέχει την IP διεύθυνση του remote user, datagramSocket της κλάσης DatagramSocket για τη δημιουργία UDP Socket σύνδεσης και isCallActive, μιας boolean μεταβλητής που δηλώνει αν γίνεται κλήση VoIP (true) ή όχι (false), με αρχική κατάσταση false. Έπειτα, αρχικοποιούνται τα playback, record, remoteAddress και datagramSocket με τον constructor VoIP.

```

1 package com.cn2.communication;
2
3 import java.net.DatagramPacket;
4 import java.net.DatagramSocket;
5 import java.net.InetAddress;
6 import javax.sound.sampled.LineUnavailableException;
7
8 public class VoIP { // class for VoIP
9
10     private AudioPlayback playback; // define object for playing sound
11     private AudioRecord record; // define object for recording sound
12     private InetAddress remoteAddress; // define IP address remoteAddress, to set it as IP of remote
13     private DatagramSocket datagramSocket; // define DatagramSocket datagramSocket
14     private volatile boolean isCallActive = false; // VoIP call state
15
16     public VoIP(DatagramSocket datagramSocket, InetAddress remoteAddress) throws LineUnavailableException {
17         // constructor VoIP, initialize playback, record, datagramSocket and remoteAddress
18
19         this.playback = new AudioPlayback();
20         this.record = new AudioRecord();
21         this.remoteAddress = remoteAddress;
22         this.datagramSocket = datagramSocket;
23     }
24 }

```

Figure 2.15: Ορισμός και αρχικοποίηση των αντικειμένων της κλάσης VoIP

Έπειτα, ορίζονται οι μέθοδοι της κλάσης VoIP. Με τη μέθοδο start ξεκινάει η επικοινωνία VoIP μεταξύ των δύο users. Αρχικά, η isCallActive γίνεται true ώστε ό,τι διεργασία γίνεται στη μέθοδο να εκτελείται όσο γίνεται κλήση VoIP. Καλούνται οι μέθοδοι open των κλάσεων AudioRecord και AudioPlayback ώστε το μικρόφωνο και το ηχείο να λαμβάνουν και να αναπαράγουν ήχο αντίστοιχα. Για την καταγραφή και αποστολή ήχου στον remote, ορίζεται ο buffer sendAudioBuffer μεγέθους 1024 bytes και μέσω ενός while loop, όσο η isCallActive είναι true, ο ήχος που στέλνει ο local αποθηκεύεται στον sendAudioBuffer, αφού μετατραπεί σε byte stream με τη μέθοδο read της κλάσης AudioRecord. Μετά, ορίζεται ένα αντικείμενο της κλάσης DatagramPacket,

το datagramPacket, που περιέχει τα δεδομένα του sendAudioBuffer και στέλνεται στην IP διεύθυνση του remote user remoteAddress και πιο συγκεκριμένα στο port 1243. Η παραπάνω διαδικασία τοποθετείται σε Thread ώστε η καταγραφή και αποστολή ήχου να μην εμποδίζει τη λειτουργία του app.

```
25 public void start() { // method start, start VoIP call
26
27     isCallActive = true; // set isCallActive to true, change state when "End Call" is pressed
28     try {
29         record.open(); // call method open from AudioRecord, open targetline-stream and start recording audio
30         playback.open(); // call method open from AudioPlayback, open source-line-stream and start playing audio
31
32         new Thread() -> { // Thread the capture and send audio process
33             try {
34                 byte[] sendAudioBuffer = new byte[1024]; // sendAudioBuffer, size=1024 bytes, to capture audio stream from microphone
35                 while (isCallActive) { // while VoIP call is happening
36                     sendAudioBuffer = record.read(); // sendAudioBuffer captures audio and returns byte stream
37                     DatagramPacket datagramPacket = new DatagramPacket(sendAudioBuffer, sendAudioBuffer.length, remoteAddress, 1243); /* construct datagram
38                     send packets of length of sendAudioBuffer, to IP remoteAddress and port=1243 of remote */
39                     datagramSocket.send(datagramPacket); // datagramPacket send
40                 }
41             }
42             catch (Exception e) { // in case of error
43                 e.printStackTrace();
44             }
45         }).start(); // start Thread
46 }
```

Figure 2.16: Η μέθοδος start της κλάσης VoIP για την καταγραφή και αποστολή ήχου

Ομοίως, για την λήψη και αναπαραγωγή ήχου από τον remote, ορίζεται ο buffer receiveAudioBuffer μεγέθους 1024 bytes και μέσω ενός while loop, όσο η isCallActive είναι true, ορίζεται ένα αντικείμενο της κλάσης DatagramPacket, το datagramPacket που περιέχει τα δεδομένα ήχου που έστειλε ο remote και καλούμε τη μέθοδο receive ώστε να το λάβουμε. Μετά, γράφουμε τον ήχο που λήφθηκε από το datagramPacket στον receiveAudioBuffer με τη μέθοδο write της κλάσης AudioPlayback. Βάζουμε την παρακάτω διαδικασία σε Thread ώστε η διαδικασία λήψης και αναπαραγωγής ήχου να μην εμποδίζει τη λειτουργία του app. Χρησιμοποιούνται δύο διαφορετικοί buffers και δύο διαφορετικά Threads για την αποστολή και λήψη φωνητικών μηνυμάτων για καλύτερη αποδοτικότητα.

```
48 new Thread() -> { // Thread the receive and play audio process
49     try {
50         byte[] receiveAudioBuffer = new byte[1024]; // receiveAudioBuffer, size=1024 bytes, to capture byte stream from remote
51         while (isCallActive) { // while VoIP call is happening
52             DatagramPacket datagramPacket = new DatagramPacket(receiveAudioBuffer, receiveAudioBuffer.length); /* construct datagramPacket,
53             receive packets of length of receiveAudioBuffer */
54             datagramSocket.receive(datagramPacket); // datagramPacket received from datagramSocket, blocking method
55             playback.write(datagramPacket.getData()); // call method write from AudioPlayback, write audio data from datagramPacket to source-line
56         }
57     }
58     catch (Exception e) { // in case of error
59         e.printStackTrace();
60     }
61 }).start(); // start Thread
62
63 }
64 catch (Exception e) { // in case of error
65     e.printStackTrace();
66 }
67 }
```

Figure 2.17: Η μέθοδος start της κλάσης VoIP για την λήψη και αναπαραγωγή ήχου

Με τη μέθοδο stop σταματάει η επικοινωνία VoIP μεταξύ των δύο users. Αρχικά, η isCallActive γίνεται false και μετά, καλούνται οι μέθοδοι close των κλάσεων AudioRecord και AudioPlayback ώστε να σταματήσουν και να κλείσουν τα voice streams.

```

69● public void stop() { // method stop, stop VoIP call
70
71     isCallActive = false; // set isCallActive to false, change state when "Call" is pressed
72     record.close(); // call method close from AudioRecord, close targetLine-stream
73     playback.close(); // call method close from AudioPlayback, close sourceLine-stream
74 }
75
76 }
77

```

Figure 2.18: Η μέθοδος stop της κλάσης VoIP

2.2.3 Η VoIP λειτουργία στην κλάση App

Τέλος, περιγράφεται η VoIP λειτουργία στην κύρια κλάση της εφαρμογής App. Ξεκινάει ορίζοντας τα αντικείμενα `remoteAddress` της κλάσης `InetAddress` που περιέχει την IP διεύθυνση του remote user, `voip`, της κλάσης `VoIP` και της boolean μεταβλητής `isCallActive` με αρχική κατάσταση `false`. Μετά, αρχικοποιείται το `voip` με έναν non-static initialization block, αφού δεν είναι static, χρησιμοποιώντας τον constructor `VoIP` με ορίσματα ένα `DatagramSocket` από το port 1243 που χρησιμοποιήθηκε για τη VoIP επικοινωνία και την `remoteAddress`.

```

private InetAddress remoteAddress; // define IP address remoteAddress, to set it as IP of remote
private UDPChat chatUDP; // define UDPChat object for UDP Chat
private VoIP voip; // define VoIP object for VoIP
private boolean isCallActive = false; // VoIP call not happening

// A note for tcp
// If you plan to use it:
// 1) uncomment the appropriate object
// 2) uncomment the appropriate constructor
// 3) uncomment the TCP receive()
// 4) comment the UDP SEND area
// 5) uncomment the TCP SEND area

// private TCPChatClient chatTCP; // define TCPChat object for TCP Chat, if local is the "client"
private TCPChatServer chatTCP; // define TCPChat object for TCP Chat, if local is the "server"

// define AES variable (AES + *ci*pher -> AESci)
public AESci aesci;

• { // initialize network variables using non-static initialization block
try {
    remoteAddress = InetAddress.getByName("192.168.208.134"); // initialize remoteAddress, IP of remote
    chatUDP = new UDPChat(new DatagramSocket(1234), remoteAddress); /* initialize chatUDP,
    // pass DatagramSocket from port 1234 and remoteAddress to constructor UDPChat */
    voip = new VoIP(new DatagramSocket(1243), remoteAddress); /* initialize voip,
    // pass DatagramSocket from port 1243 and remoteAddress to constructor VoIP */

    // initialize chatTCP, pass Socket from port 2345 and IP of remote to constructor TCPChatSender
    chatTCP = new TCPChatClient(new Socket(remoteAddress, 2345));
    // initialize chatTCP, pass ServerSocket from port 2345 to constructor TCPChatReceiver
    chatTCP = new TCPChatServer(new ServerSocket(2345));
}
}

```

Figure 2.19: Ορισμός και αρχικοποίηση των αντικειμένων στην κλάση App για τη VoIP λειτουργία

Η VoIP επικοινωνία ξεκινάει όταν κάποιος από τους δύο users πατήσει το κουμπί "Call", άρα υλοποιείται το δεύτερο if sentence της μεθόδου `actionPerformed`. Αρχικά, δεν είναι γνωστό ποιός από τους δύο user, ο remote ή ο local, έχει ξεκινήσει κλήση, γι' αυτό ορίζεται τη μεταβλητή `textAreaText` η οποία περιέχει το κείμενο του `textArea`. Όταν ο local πατάει το "Call" η `isCallActive` είναι `false` και όσο είναι `false`, με ένα if sentence, υλοποιούνται οι λειτουργίες του VoIP.

Αν στο `textArea` υπάρχει η έκφραση "remote: [Voice-Call] Calling...Pick up!", ο remote ξεκίνησε κλήση και μέσω της `UDPChat` ο local ενημερώνει τον remote ότι αποδέχθηκε την κλήση στέλνοντάς του "[Voice-Call] VoIP call

started." και το μήνυμα "remote: Calling...Pick up!" αντικαθίσταται με "VoIP call started."

Αν δεν υπάρχει, ο local ξεκινάει κλήση και μέσω της UDPChat ο local ενημερώνει τον remote ότι ξεκίνησε κλήση στέλνοντάς του "[Voice-Call] Calling...Pick up!" και εμφανίζεται στο textArea "[Voice-Call] Calling..." μέχρι ο remote να αποδεχτεί την κλήση και να στείλει "[Voice-Call] VoIP call started.". Είτε ξεκινήσει την κλήση ο local ή ο remote, αλλάζουμε το κουμπί "Call" σε "End Call", ξεκινάμε το VoIP call μέσω της μεθόδου start της VoIP και αλλάζει η isActive σε true ώστε όταν πατηθεί το κουμπί "End Call" να φεύγουμε από το if sentence που απαιτούσε το isActive να είναι false και να υλοποιούνται οι λειτουργίες του VoIP.

```
279 else if (e.getSource() == callButton) { // The "Call" button was clicked
280
281     // the [Voice-Call] tag is recognised by the UDPChat.receive method
282     String textAreaText = textArea.getText(); // get the text from textArea
283     if (!isActive) { // VoIP call happening
284         if (textAreaText.contains("remote: [Voice-Call] Calling...Pick up!")) { // if remote starts call
285             try {
286                 String message = ("[Voice-Call] VoIP call started."); // inform remote local has picked up, call started
287                 chatUDP.send(message); // by sending message
288             } catch (Exception ex) { // in case of error
289                 ex.printStackTrace();
290             }
291             String content = textArea.getText(); // get the text from textArea
292             content = content.replace("remote: [Voice-Call] Calling...Pick up!", "[Voice-Call] VoIP call started."); // replace the specific text
293             textArea.setText(content);
294         }
295     } else { // if local starts call
296         try {
297             String message = ("[Voice-Call] Calling...Pick up!"); // inform remote local is calling
298             chatUDP.send(message); // by sending message
299         } catch (Exception ex) { // in case of error
300             ex.printStackTrace();
301         }
302         textArea.append("[Voice-Call] Calling..." + newline); // appear "Calling..." to textArea and change line
303     }
304     callButton.setText("End Call"); // change button to End Call
305     voip.start(); // call method start from VoIP and start VoIP call
306     isActive = true; // change state when "End Call" is pressed
307 }
```

Figure 2.20: Η μέθοδος start της κλάσης VoIP στην App για εκκίνηση κλήσης

Όταν πατηθεί το κουμπί "End Call" σταματάει την κλήση. Αν στο textArea υπάρχει η έκφραση "remote: [Voice-Call] VoIP call ended.", ο remote έχει σταματήσει την κλήση και αφού ο local τη σταμάτησε και εκείνος, αντικαθίσταται το "remote: [Voice-Call] VoIP call ended." με "[Voice-Call] VoIP call ended."

Αν δεν υπάρχει, ο local σταματάει κλήση και ενημερώνει τον remote ότι τη σταμάτησε στέλνοντάς του "VoIP call ended." και και εμφανίζεται στο textArea "VoIP call ended.". Είτε σταματήσει την κλήση ο local ή ο remote, αλλάζουμε το κουμπί "End Call" σε "Call", σταματάει το VoIP call μέσω της μεθόδου stop της VoIP, αλλάζει η isActive σε false ώστε όταν πατηθεί το κουμπί "Call" να μπούμε στο if sentence που απαιτούσε το isActive να είναι false και να υλοποιούνται οι λειτουργίες του VoIP και σβήνεται το μήνυμα "[Voice-Call] Calling..."

```

311     else { // VoIP call not happening
312         if (textAreaText.contains("remote: [Voice-Call] VoIP call ended.")) { // if remote ended call
313             String content = textArea.getText(); // get the text from textArea
314             content = content.replace("remote: [Voice-Call] VoIP call ended.", "[Voice-Call] VoIP call ended."); // replace the specific text
315             textArea.setText(content);
316         }
317     }
318     else { // if local ended call
319         try {
320             String message = ("[Voice-Call] VoIP call ended."); // inform remote local has stopped the call
321             chatUDP.send(message); // by sending message
322         } catch (Exception ex) { // in case of error
323             ex.printStackTrace();
324         }
325         textArea.append("[Voice-Call] VoIP call ended."+ newline); // appear "VoIP call ended." to textArea and change line
326     }
327     callButton.setText("Call"); // change button to Call
328     voip.stop(); // call method stop from VoIP and stop VoIP call
329     isCallActive = false; // change state when "Call" is pressed
330
331     String twoContent = textArea.getText(); // get the text from textArea
332     twoContent = twoContent.replace("[Voice-Call] calling...", ""); // remove the specific text
333     textArea.setText(twoContent);
334 }
335 }

```

Figure 2.21: Η μέθοδος stop της κλάσης VoIP στην App για λήξη της κλήσης

Στο τέλος, αυτό που φαίνεται στο textArea είναι ποιος αποδέχτηκε την κλήση ώστε να ξεκινήσει και τη λήξη της.

2.3 TCP Chat

Για την υλοποίηση του chat με χρήση TCP δημιουργούμε τις κλάσεις TCPChatServer και TCPChatClient. Αρχικά περιγράφεται η TCPChatClient, την οποία χρησιμοποιεί ο χρήστης που ξεκινά την TCP σύνδεση για την ανταλλαγή μηνυμάτων κειμένου και στην συνέχεια περιγράφεται η TCPChatServer, την οποία χρησιμοποιεί ο χρήστης που αποδέχεται την TCP σύνδεση για την ανταλλαγή μηνυμάτων κειμένου (ο ένας χρήστης χρησιμοποιεί την κλάση TCPChatClient, ενώ ο άλλος την TCPChatServer).

2.3.1 Η κλάση TCPChatClient

Αρχικά περιγράφεται η TCPChatClient, την οποία χρησιμοποιεί ο χρήστης που ξεκινά την TCP σύνδεση για την ανταλλαγή μηνυμάτων κειμένου και στην συνέχεια περιγράφεται η TCPChatServer, την οποία χρησιμοποιεί ο χρήστης που αποδέχεται την TCP σύνδεση για την ανταλλαγή μηνυμάτων κειμένου (ο ένας χρήστης χρησιμοποιεί την κλάση TCPChatClient, ενώ ο άλλος την TCPChatServer). Σε αυτήν την κλάση ορίζονται τα αντικείμενα socket, της κλάσης Socket, bufferedReader, της κλάσης BufferedReader που περιέχει το μήνυμα που λαμβάνει ο local από τον remote σε μορφή κειμένου και bufferedWriter, της κλάσης BufferedWriter που περιέχει το μήνυμα που στέλνει ο local σε μορφή byte stream. Στην συνέχεια, τα socket, bufferedReader και bufferedWriter αρχικοποιούνται με τον constructor TCPChatClient. Ο bufferedReader έχει ως όρισμα το μήνυμα του remote σε μορφή κειμένου αφού, με την κλάση InputStreamReader μετατρέπεται από byte stream που λαμβάνεται από το socket σε character stream. Ο bufferedWriter έχει ως όρισμα το μήνυμα που στέλνει ο local στο socket, αφού το μετατραπεί από character stream σε byte stream με την κλάση OutputStreamWriter.

```
1 package com.cn2.communication;
2
3 import java.io.BufferedReader;
4 import java.io.BufferedWriter;
5 import java.io.IOException;
6 import java.io.InputStreamReader;
7 import java.io.OutputStreamWriter;
8 import java.net.Socket;
9 import javax.swing.JTextArea;
10
11 public class TCPChatClient { // class for the user "client" that sends the socket connection request
12
13     private Socket socket; // define socket
14     private BufferedReader bufferedReader; // define buffer bufferedReader, contains data sent from remote
15     private BufferedWriter bufferedWriter; // define buffer bufferedWriter, contains data local will send to remote
16
17     public TCPChatClient(Socket socket) { // constructor TCPChatSender, initialize Socket
18
19         try {
20             this.socket = socket;
21             this.bufferedReader = new BufferedReader(new InputStreamReader(socket.getInputStream())); /* put InputStreamReader to bufferedReader,
22             InputStreamReader converts the input byte stream coming from socket to character stream */
23             this.bufferedWriter = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream())); /* put OutputStreamWriter to bufferedWriter,
24             OutputStreamWriter converts the output character stream sent to socket to byte stream */
25         }
26         catch (IOException e) { // in case of error
27             e.printStackTrace();
28             System.out.println("Error initializing");
29             closeEverything(); // close streams
30         }
31     }
32 }
```

Figure 2.22: Ορισμός και αρχικοποίηση των αντικειμένων της κλάσης TCPChatClient

Έπειτα, ορίζονται οι μέθοδοι της κλάσης TCPChatClient. Με τη μέθοδο send αποστέλλεται το μήνυμα messageToRemote που εισάγεται ως όρισμα από τον local στον remote. Αρχικά, γράφεται στον bufferedWriter, με τη μέθοδο newline της κλάσης σηματοδοτείται το τέλος του character stream με έναν newline χαρακτήρα \n και μετά, ο bufferedWriter γίνεται flush στέλνοντας έτσι το περιεχόμενό του μόλις τελειώσει το μήνυμα και όχι όταν γεμίσει.

```
33 public void send(String messageToRemote) { // method send, local sends text messageToRemote
34
35     try {
36         bufferedWriter.write(messageToRemote); // messageToRemote to bufferedWriter
37         bufferedWriter.newLine(); // used to create line separator "\n" in buffer so we know when the messageToRemote is finished
38         bufferedWriter.flush(); // flush the stream when messageToRemote is finished
39     }
40     catch (IOException e) { // in case of error
41         e.printStackTrace();
42         closeEverything(); // close streams
43     }
44 }
45 }
```

Figure 2.23: Η μέθοδος send της κλάσης TCPChatClient

Η μέθοδος receive λαμβάνει τα μηνύματα από τον remote η οποία τρέχει μέσα σε ένα thread ώστε να μπορεί να τρέχει παράλληλα με άλλες μεθόδους του προγράμματος. Όσο η σύνδεση είναι ενεργή, δημιουργεί ένα string messageFromRemote στο οποίο περνάει το μήνυμα του remote και εφόσον δεν είναι κενό (κάνοντας τον έλεγχο με μία if) κάνει append στο textArea το μήνυμα με πρόθεμα την φράση "remote: ". Αν το μήνυμα είναι κενό, σημαίνει πως ο remote έκλεισε την εφαρμογή και συνεπώς γίνεται append η φράση "remote: Disconnected." για να ενημερωθεί ο local.

```
46 public void receive(JTextArea textArea) { // method receive, local receives text messageFromRemote
47
48     new Thread(() -> { // Thread the receive text process
49         while (socket.isConnected()) { // while socket connection is established
50             try {
51                 String messageFromRemote = bufferedReader.readLine(); // messageFromRemote the message remote sends to local
52
53                 if (messageFromRemote == null) { // check for null, remote closed the app
54                     textArea.append("remote: Disconnected." + "\n"); // inform local
55                     break; // break from loop
56                 }
57                 textArea.append("remote: " + messageFromRemote + "\n"); // appear messageFromRemote to textArea and change line
58             }
59             catch (IOException e) { // in case of error
60                 e.printStackTrace();
61                 closeEverything(); // close streams
62                 break; // break from loop
63             }
64         }
65     }).start(); // start Thread
66 }
67 }
```

Figure 2.24: Η μέθοδος receive της κλάσης TCPChatClient

Τέλος, η μέθοδος closeEverything κλείνει την σύνδεση μεταξύ local και remote. Πρώτα ελέγχει αν το socket υπάρχει και είναι ανοιχτό, οπότε το κλείνει. Μετά ελέγχει αν το bufferedReader και bufferedWriter δεν είναι κενά και τα κλείνει.

```

69 public void closeEverything() { // checking for null before closing streams to avoid a null pointer exception and closing streams
70
71     try {
72         if (socket != null && !socket.isClosed()) {
73             socket.close(); // close socket
74         }
75         if (bufferedReader != null) {
76             bufferedReader.close(); // close bufferedReader
77         }
78         if (bufferedWriter != null) {
79             bufferedWriter.close(); // close bufferedWriter
80         }
81     }
82     catch (IOException e) { // in case of error
83         e.printStackTrace();
84     }
85 }
86
87 }
88

```

Figure 2.25: Η μέθοδος closeEverything της κλάσης TCPChatClient

2.3.2 Η κλάση TCPChatSender

Η κλάση TCPChatSender είναι κατά κύριο λόγο ίδια με την TCPChatClient, με μικρές αλλαγές. Περιέχει το επιπλέον αντικείμενο serverSocket, της κλάσης ServerSocket που αναμένει αιτήματα σύνδεσης και αφού τα αποδεχτεί, δημιουργεί ένα socket. Τα αιτήματα σύνδεσης γίνονται αποδεκτά στον constructor, στον οποίο αρχικοποιούνται τα serverSocket, socket, bufferedReader και bufferedWriter.

```

1 package com.cn2.communication;
2
3 import java.io.BufferedReader;
4 import java.io.BufferedWriter;
5 import java.io.IOException;
6 import java.io.InputStreamReader;
7 import java.io.OutputStreamWriter;
8 import java.net.ServerSocket;
9 import java.net.Socket;
10 import javax.swing.JTextArea;
11
12 public class TCPChatServer { // class for the user "server" that accepts the socket connection request
13
14     private ServerSocket serverSocket; // define serverSocket, waits for requests for connection on a port and creates Socket object
15     private Socket socket; // define socket
16     private BufferedReader bufferedReader; // define buffer bufferedReader, contains data sent from remote
17     private BufferedWriter bufferedWriter; // define buffer bufferedWriter, contains data local will send to remote
18
19     public TCPChatServer(ServerSocket serverSocket) { // constructor TCPChatReceiver, initialize serverSocket and accept socket connection request
20
21         try {
22             this.serverSocket = serverSocket;
23             this.socket = serverSocket.accept(); // accept serverSocket
24             this.bufferedReader = new BufferedReader(new InputStreamReader(socket.getInputStream())); /* put InputStreamReader to bufferedReader,
25             InputStreamReader converts the input byte stream coming from socket to character stream */
26             this.bufferedWriter = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream())); /* put OutputStreamWriter to bufferedWriter,
27             OutputStreamWriter converts the output character stream sent to socket to byte stream */
28         }
29         catch (IOException e) { // in case of error
30             e.printStackTrace();
31             System.out.println("Error initializing");
32             closeEverything(); // close streams
33         }
34     }
35

```

Figure 2.26: Ορισμός και αρχικοποίηση των αντικειμένων της κλάσης TCPChatServer

```

36 public void send(String messageToRemote) { // method send, local sends text messageToRemote
37
38     try {
39         bufferedWriter.write(messageToRemote); // messageToRemote to bufferedWriter
40         bufferedWriter.newLine(); // used to create line separator "\n" in buffer so we know when the messageToRemote is finished
41         bufferedWriter.flush(); // flush the stream when messageToRemote is finished
42     }
43     catch (IOException e) { // in case of error
44         e.printStackTrace();
45         closeEverything(); // close streams
46     }
47 }
48

```

Figure 2.27: Η μέθοδος send της κλάσης TCPChatServer

```

60 public void receive(JTextArea textArea) { // method receive, local receives text messageFromRemote
61     new Thread() -> { // Thread the receive text process
62         while (socket.isConnected()) { // while socket connection is established
63             try {
64                 String messageFromRemote = bufferedReader.readLine(); // messageFromRemote the message remote sends to local
65
66                 if (messageFromRemote == null) { // check for null, remote closed the app
67                     textArea.append("remote: Disconnected." + "\n"); // inform local
68                     break; // break from loop
69                 }
70                 textArea.append("remote: " + messageFromRemote + "\n"); // appear messageFromRemote to textArea and change line
71             } catch (IOException e) { // in case of error
72                 e.printStackTrace();
73                 closeEverything(); // close streams
74                 break; // break from loop
75             }
76         }
77     }).start(); // start Thread
78 }

```

Figure 2.28: Η μέθοδος receive της κλάσης TCPChatServer

Η μέθοδος closeEverything της κλάσης TCPChatServer είναι παρόμοια με την closeEverything της TCPChatClient αλλά κλείνει και το αντικείμενο server-Socket.

```

72 public void closeEverything() { // checking for null before closing streams to avoid a null pointer exception and closing streams
73     try {
74         if (socket != null && !socket.isClosed()) {
75             socket.close(); // close socket
76         }
77         if (serverSocket != null && !serverSocket.isClosed()) {
78             serverSocket.close(); // close serverSocket
79         }
80         if (bufferedReader != null) {
81             bufferedReader.close(); // close bufferedReader
82         }
83         if (bufferedWriter != null) {
84             bufferedWriter.close(); // close bufferedWriter
85         }
86     } catch (IOException e) { // in case of error
87         e.printStackTrace();
88     }
89 }

```

Figure 2.29: Η μέθοδος closeEverything της κλάσης TCPChatServer

2.3.3 Η Chat λειτουργία με TCP στην κλάση App

Περιγράφεται η λειτουργία του TCPChat στην κλάση App. Ο χρήστης που χρησιμοποιεί την κλάση TCPChatClient έχει σε comment τα τμήματα που χρησιμοποιούν την TCPChatServer, ενώ αντίστροφα, ο χρήστης που χρησιμοποιεί την κλάση TCPChatServer έχει σε comment τα τμήματα που χρησιμοποιούν την TCPChatClient. Τα μηνύματα λαμβάνονται με κλήση της receive στην main. Πρώτα θα περιγράψουμε την κλάση App του χρήστη που χρησιμοποιεί την κλάση TCPChatClient. Αρχικά ορίζεται το chatTCP τύπου TCPChatClient και αρχικοποιείται χρησιμοποιώντας τον constructor TCPChatClient με ορίσματα ένα Socket που περιέχει την διεύθυνση IP του remote και το port 2345 που χρησιμοποιείται για την chat επικοινωνία με TCP.

```

40 // define network variables
41 private InetAddress remoteAddress; // define IP address remoteAddress, to set it as IP of remote
42 private UDPChat chatUDP; // define UDPChat object for UDP Chat
43 private VoIP voip; // define VoIP object for VoIP
44 private boolean isCallActive = false; // VoIP call not happening
45
46 // A note for tcp
47 // If you plan to use it:
48 // 1) uncomment the appropriate object
49 // 2) uncomment the appropriate constructor
50 // 3) uncomment the TCP receive()
51 // 4) comment the UDP SEND area
52 // 5) uncomment the TCP SEND area
53
54 // private TCPChatClient chatTCP; // define TCPChat object for TCP Chat, if local is the "client"
55 private TCPChatServer chatTCP; // define TCPChat object for TCP Chat, if local is the "server"
56
57 // define AES variable (AES + *cipher -> AESci)
58 public AESci aesci;
59
60 { // initialize network variables using non-static initialization block
61
62     try {
63         remoteAddress = InetAddress.getByName("192.168.208.134"); // initialize remoteAddress, IP of remote
64         chatUDP = new UDPChat(new DatagramSocket(1234), remoteAddress); /* initialize chatUDP,
65 // pass DatagramSocket from port 1234 and remoteAddress to constructor UDPChat */
66         voip = new VoIP(new DatagramSocket(1243), remoteAddress); /* initialize voip,
67 // pass DatagramSocket from port 1243 and remoteAddress to constructor VoIP */
68
69         // initialize chatTCP, pass Socket from port 2345 and IP of remote to constructor TCPChatSender
70 // chatTCP = new TCPChatClient(new Socket(remoteAddress, 2345));
71 // initialize chatTCP, pass ServerSocket from port 2345 to constructor TCPChatReceiver
72 // chatTCP = new TCPChatServer(new ServerSocket(2345));

```

Figure 2.30: Ορισμός και αρχικοποίηση των αντικειμένων στην κλάση App για το chat με TCP

Τα μηνύματα λαμβάνονται με κλήση της receive στην main.

```

159 /*
160 * 2. Start receiving Chat messages
161 */
162 app.chatUDP.receive(textArea, app.aesci); // call method receive from chatUDP, receive text data
163
164 // TCP isn't encrypted
165 // Keep it commented if TCP isn't used
166 app.chatTCP.receive(textArea); // call method receive from TCPChatSender or TCPChatReceiver, receive text data
167 }

```

Figure 2.31: Η μέθοδος receive της κλάσης TCPChatClient στην App για λήψη μηνυμάτων κειμένου

Για την αποστολή μηνυμάτων μέσω TCP χρησιμοποιούμε την αντίστοιχη λειτουργία στην μέθοδο actionPerformed. Η διαφορά με το UDPchat είναι πως όταν πατηθεί το κουμπί "Send", τότε γράφουμε στην μεταβλητή messageToSend το κείμενο που θέλουμε να στείλουμε και αφού ελέγξουμε πως δεν είναι κενό ή πάνω από 50k χαρακτήρες, το διοχετεύουμε απευθείας ως όρισμα στην μέθοδο chatTCP.send και το στέλνουμε χωρίς κρυπτογράφηση. Επιπλέον κάνουμε append στο textArea το μήνυμα με πρόθεμα την φράση "local: " για να φαίνεται πως το στείλαμε εμείς και καθαρίζουμε το πεδίο αποστολής θέτοντας ως κενό string το όρισμα της inputTextField.setText(). Θεωρητικά θα μπορούσαμε να περάσουμε το μήνυμα από την aesci.encryptMessage προτού το στείλουμε και η receive να το αποκρυπτογραφεί με την aesci.decryptMessage, αλλά για λόγους απλότητας δεν το υλοποιήσαμε (επίσης χωρίς αυτήν την υλοποίηση μπορούν να διαβαστούν τα μηνύματα σε plaintext για καλύτερη κατανόηση της λειτουργίας του προγράμματος στην ανάλυση με το wireshark).

```

257 // //TCP SEND [START]
258     try {
259         String plainMessage = messageToSend;
260         // Checks needed for TCP, this whole if should be commented if we plan to use UDP
261         // TCP can send ~64Kb packets easily, a lot bigger than what we normally need for this app
262         if (!messageToSend.isEmpty() || (messageToSend.length() > 50000) ) { // if there is a messageToSend
263             chatTCP.send(messageToSend); // call method send from chatTCP, send text data
264         }
265         else {
266             textArea.append("Message to big. Please send 50000 chars max.\nYour message was not send.");
267         }
268         textArea.append("local: " + plainMessage + newline); // appear plainMessage to textArea and change line
269         inputTextField.setText(""); // erase messageToSend from inputTextField
270     }
271     catch (Exception ex) { // in case of error
272         ex.printStackTrace();
273     }
274 // //TCP SEND [END]

```

Figure 2.32: Η μέθοδος send της κλάσης TCPChatClient στην App για αποστολή μηνυμάτων κειμένου

Ακόμη, όταν η εφαρμογή κλείνει, έχει προστεθεί ένα κομμάτι κώδικα στην μέθοδο windowClosing της κλάσης App το οποίο με μια if ελέγχει αν υπάρχει το αντικείμενο chatTCP και αν ναι, τότε καλεί την μέθοδο chatTCP, closeEverything για να κλείσει την TCP σύνδεση προτού τερματιστεί η εφαρμογή.

```

435 ● @Override
436     public void windowClosing(WindowEvent e) { // close the app
437         try {
438             if (chatTCP != null) {
439                 chatTCP.closeEverything(); // close streams
440             }
441         }
442         catch (Exception ex) { // in case of error
443             ex.printStackTrace();
444         }
445         finally { // always executed
446             dispose();
447             System.exit(0);
448         }
449     }

```

Figure 2.33: Η μέθοδος WindowClosing της κλάσης App για κλείσιμο της σύνδεσης με το κλείσιμο του app

Τέλος, η υλοποίηση της κλάσης App του χρήστη που χρησιμοποιεί την κλάση TCPChatServer είναι ίδια, με μόνη διαφορά ότι στην αρχή ορίζεται το chatTCP τύπου TCPChatServer και αρχικοποιείται χρησιμοποιώντας τον constructor TCPChatServer με όρισμα ένα ServerSocket με port 2345 που χρησιμοποιείται για την chat επικοινωνία με TCP.

2.4 Encryption

2.4.1 Ο αλγόριθμος "AES/GCM/NoPadding"

Το πρόγραμμα περιέχει την κλάση AESci (AES+ci-pher) για την συμμετρική κρυπτογράφηση των μηνυμάτων κειμένου. Θα πούμε πρώτα λίγα λόγια για τον συγκεκριμένο είδος κρυπτογράφησης.

Επιλέξαμε τον αλγόριθμο AES (Advanced Encryption Standard) και συγκεκριμένα τον συνδυασμό "AES/GCM/NoPadding". Η κρυπτογράφηση AES έχει 6 modes και επιλέξαμε το mode GCM (Galois/Counter Mode).

- Το GCM είναι η μέθοδος που προτείνει το INST (National Institute of Standards and Technology) των ΗΠΑ. Το GCM χρησιμοποιεί μια παραλλαγή του Counter Mode (CTR) για την κρυπτογράφηση. Σε αυτό το mode, ένας "counter" (μετρητής) αυξάνεται για κάθε μπλοκ που κρυπτογραφείται, και αυτός ο μετρητής συνδυάζεται με το κλειδί κρυπτογράφησης για να παραχθεί το κρυπτογραφημένο κείμενο. Επίσης χρησιμοποιεί τις μαθηματικές ιδιότητες των Galois Fields για να υπολογίσει την ακεραιότητα των δεδομένων. Αυτό γίνεται μέσω μιας διαδικασίας που ονομάζεται "authentication tag", η οποία επιβεβαιώνει ότι τα δεδομένα δεν έχουν τροποποιηθεί. Τέλος παρέχει μια μέθοδο για επαλήθευση της ακεραιότητας των δεδομένων μέσω δημιουργίας ενός "authentication tag". Αυτό το tag υπολογίζεται κατά τη διάρκεια της κρυπτογράφησης και μπορεί να χρησιμοποιηθεί για να ελέγξει αν τα δεδομένα έχουν αλλαχθεί.

- Padding χρησιμοποιείται όταν το block που κρυπτογραφείται (το aes λειτουργεί με blocks) δεν είναι πολλαπλάσιο των 8bytes. Στην λειτουργία GCM όμως, αυτό υλοποιείται μόνο του, οπότε δεν χρησιμοποιούμε Padding (NoPadding).

- Τέλος, η λειτουργία αυτή χρησιμοποιεί ένα τυχαίο byte array 12 bytes, το IV (Initialization Vector). Αυτό το κλειδί πρέπει να αλλάζει μετά από κάθε κρυπτογράφηση μηνύματος. Σκοπός του είναι να συνδυάζεται με το κλειδί που έχουμε, ώστε ακόμα και αν σταλθεί το ίδιο μήνυμα, να μην είναι ίδιο στην κρυπτογραφημένη του μορφή.

Πρακτικά, η υλοποίησή μας χρησιμοποιεί ένα κλειδί που το ορίζουμε με ένα password (το οποίο είναι seed μαζί με ένα σταθερό salt για να γίνει randomised το κλειδί) μαζί με ένα ευφήμερο κλειδί (IV), ώστε να παράγει μηνύματα που είναι δύσκολο να αποκρυπτογραφηθούν.

2.4.2 Ορισμός μεταβλητών και αρχικοποίηση

Συνεχίζοντας περιγράφεται η κλάση AESci. Στην AESci ορίζονται τα αντικείμενα key, της κλάσης SecretKey που αποτελεί το κλειδί της κρυπτογράφησης, τα int μεγέθη 'KEY SIZE' και 'T LEN' του κλειδιού και του authentication tag αντίστοιχα, IV, το byte array που αποθηκεύεται το "ευφήμερο" κλειδί, salt για να γίνει πιο τυχαία η παραγωγή κλειδιού με το password του χρήστη, encodeIV για την αποθήκευση του IV σε string μορφή κωδικοποιημένη σε base64, ivstr για την προσωρινή αποθήκευση του IV σε string μορφή κωδικοποιημένη σε base64 και τέλος το bufferedMes, ένα string array 100 στοιχείων για την αποθήκευση μηνυμάτων μεγάλου μήκους (μέχρι 50k χαρακτήρες).

```
1 package com.cn2.communication;
2
3 import javax.crypto.Cipher;
4 import javax.crypto.KeyGenerator;
5 import javax.crypto.SecretKey;
6 import javax.crypto.SecretKeyFactory; //for generating key with password
7
8 import javax.crypto.spec.GCMParameterSpec;
9 import javax.crypto.spec.SecretKeySpec;
10 import javax.crypto.spec.PBEKeySpec;
11
12 //import java.nio.charset.StandardCharsets;
13
14 import java.security.NoSuchAlgorithmException; //for generating key with password
15 import java.security.SecureRandom;
16 import java.security.spec.InvalidKeySpecException; //for generating key with password
17 import java.util.Arrays;
18 import java.util.Base64;
19
20 public class AESci {
21     private SecretKey key; // the AES key
22     // private final variable: its value is constant throughout the execution of the program and can only be accessed within the class where it is defined
23     private final int KEY_SIZE = 128; // the size of the AES key
24     private final int T_LEN = 128; // the length of the tag used in the Galois/Counter Mode (GCM) of operation in the AES encryption algorithm
25     // in GCM, a tag is appended to the ciphertext during encryption
26     // it is used during decryption to verify the integrity of the ciphertext
27     private byte[] IV = new byte[12]; // Initialization Vector: This is the IV (the real IV) our cipher uses, we will call it IV
28     // it is essentially the ephemeral key that changes after every encryption
29     private String salt = "potato"; // the salt used to generate the key from our password
30     private String encodedIV = ""; // the base64 encoded string version of the IV
31     private String ivstr = ""; // the base64 encoded string containing the new IV, before it is assigned to IV
32     public String[] bufferedMes = new String[100]; // the buffer to contain messages over 500chars in plaintext
33     // Should hold 100 strings max of 500 chars max (50k chars max total)
```

Figure 2.34: Ορισμός των μεταβλητών της κλάσης Aesci

Μετά αρχικοποιούνται τα key και IV με τον constructor AESci. Το key αρχικοποιείται με βάση το password "VerySecurePassword" καλώντας την initFromPassword, ενώ το IV γεμίζει τα στοιχεία του με την byte μορφή του int 1. Είναι απαραίτητο να ορίζονται με τον ίδιο τρόπο σε όλες τις συσκευές, αλλιώς δεν θα μπορέσει να υπάρξει επικοινωνία, θα αποτυγχάνει συνεχώς η αποκρυπτογράφηση.

```
36 // constructor
37 public AESci() throws Exception {
38     initFromPassword("VerySecurePassword"); // Sets key
39     // IV = new byte[12]; // Create a byte array of length 10
40     Arrays.fill(IV, (byte) 1);
41     ivstr = encode(IV);
42     // IVgen(); //Generates IV string and assigns it to ivstr variable
43     // setIV(ivstr); //Assigns IV stored in ivstr to the real IV
44 }
```

Figure 2.35: Αρχικοποίηση των αντικειμένων της κλάσης Aesci

2.4.3 Ορισμός μεθόδων

Με τη μέθοδο `initFromPassword` δημιουργείται το κλειδί μας, με βάση το `password` που του δίνουμε και το `salt` που έχουμε ορίσει.

```
48 // Generates key from password
49 public void initFromPassword(String password) throws NoSuchAlgorithmException, InvalidKeySpecException {
50 // Creates a SecretKeyFactory instance for the PBKDF2WithHmacSHA256 algorithm
51 SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
52 // Create a PBEKeySpec object with the password, salt, iteration count, and key size
53 // Password is converted to char array, while salt to byte array
54 // Salt and iteration counts help prevent dictionary attacks and brute force attacks
55 PBEKeySpec spec = new PBEKeySpec(password.toCharArray(), salt.getBytes(), 65536, 128);
56 // Creates a SecretKeySpec object (our key) with the generated PBEKeySpec object and the AES algorithm
57 key = new SecretKeySpec(factory.generateSecret(spec).getEncoded(), "AES");
58 }
```

Figure 2.36: Η μέθοδος `initFromPassword` της κλάσης `Aesci` για δημιουργία και ορισμό νέο κλειδιού

Η μέθοδος `IVgen` παράγει ένα τυχαίο byte array, το οποίο εισάγεται ως string στην `ivstr` κωδικοποιημένο με base64. Ο λόγος που δεν το εισάγουμε απευθείας στο IV, είναι επειδή θέλουμε να κρυπτογραφήσουμε το μήνυμα με το προϋπάρχον IV. Ο `remote` δεν γνωρίζει το καινούργιο που μόλις δημιουργήσαμε ώστε να αποκρυπτογραφήσει το μήνυμα.

```
60 // Generates new and random IV (should be called after every encryption, it should be ephemeral)
61 // The new IV should be supplied in the beginning of the sent encrypted message, *before* be put in use
62 // The receiver should decrypt the message with the old IV, extract the new IV from the message and use that for the next message
63 // The base64 IV we generate is 16 chars (should be the first 16 chars of a normal encrypted message)
64 public void IVgen() {
65 SecureRandom random = new SecureRandom(); // Creates the random seed
66 byte[] iv2 = new byte[12]; // Creates a temporary 12-byte array (iv2)
67 random.nextBytes(iv2); // Fills the temporary iv2 array with 12 bytes (nextBytes() is a method of SecureRandom)
68 ivstr = encode(iv2); // Encodes the iv as a base64 string and assigns it to ivstr variable
69 // System.err.println(ivstr); // Debug, prints ivstr on the console
70 // this.IV = decode(ivstr); // Unnecessary, used to directly assign the byte array to the real IV upon generating a new
71 }
```

Figure 2.37: Η μέθοδος `IVgen` της κλάσης `Aesci` για δημιουργία νέου IV

Με την `setIV` εισάγουμε ένα base64 string (την `ivstr` έχουμε ως όρισμα συνήθως) στην IV, αφού το αποκωδικοποιήσουμε και το μετατρέψουμε σε απλό byte array.

```
73 // Sets the IV from a base64 string
74 public void setIV(String IVnew) {
75 if (!IVnew.isEmpty()) { // Checks if the string is empty
76 this.IV = decode(IVnew); // Decodes the base64 string to a byte array and assigns it to IV
77 }
78 }
```

Figure 2.38: Η μέθοδος `setIV` της κλάσης `Aesci` για ορισμό νέου IV

Η `encrypt` κρυπτογραφεί το string που εισάγουμε επιστρέφοντας ένα κρυπτογραφημένο μήνυμα κωδικοποιημένο σε base64 string. Μετατρέπει αρχικά το μήνυμα σε byte array, δημιουργεί το object του cipher με τον αλγόριθμο που θέλουμε σε ENCRYPT mode, εισάγει τις παραμέτρους της κρυπτογράφησης (τα IV και το μέγεθος του authentication tag) μέσω του `GCMParameterSpec` object και με την `doFinal` δημιουργεί το κρυπτογραφημένο μήνυμα σε μορφή byte array, το οποίο το επιστρέφει ως base64 encoded string.


```

81 // Encrypts text
82 public String encrypt(String message) throws Exception {
83     byte[] messageInBytes = message.getBytes(); // Encodes and assigns string message to a byte array to be encrypted
84     // byte[] messageInBytes = message.getBytes(StandardCharsets.UTF_8); // This didn't work somehow
85     Cipher encryptionCipher = Cipher.getInstance("AES/GCM/NoPadding"); // Creates a new AES/GCM/NoPadding cipher object
86     GCMParameterSpec spec = new GCMParameterSpec(T_LEN, IV); // Creates a GCMParameterSpec object with the given tag length (T_LEN) and IV
87     encryptionCipher.init(Cipher.ENCRYPT_MODE, key, spec); // Initialize the cipher in "encryption" mode with the AES key and the GCMParameterSpec
88     byte[] encryptedBytes = encryptionCipher.doFinal(messageInBytes); // Encrypts the message using the cipher
89     return encode(encryptedBytes); // Returns a base64 encoded string of the message
90 }

```

Figure 2.39: Η μέθοδος encrypt της κλάσης Aesci για κρυπτογράφηση String

Αντίστοιχα, η decrypt αποκρυπτογραφεί το string που εισάγουμε επιστρέφοντας ένα κρυπτογραφημένο μήνυμα κωδικοποιημένο σε απλό string. Είναι περίπου η ίδια διαδικασία με κύρια διαφορά το ότι το cipher είναι σε DE-CRYPT mode.

```

92 // Decrypts text
93 public String decrypt(String encryptedMessage) throws Exception {
94     byte[] messageInBytes = decode(encryptedMessage); // Decodes and assigns base64 string message to a byte array to be decrypted
95     Cipher decryptionCipher = Cipher.getInstance("AES/GCM/NoPadding"); // Creates a new AES/GCM/NoPadding cipher object for decryption
96     GCMParameterSpec spec = new GCMParameterSpec(T_LEN, IV); // Creates a GCMParameterSpec object with the same tag length and IV used for encryption
97     decryptionCipher.init(Cipher.DECRYPT_MODE, key, spec); // Initializes the cipher in decryption mode with the AES key and the GCMParameterSpec object
98     byte[] decryptedBytes = decryptionCipher.doFinal(messageInBytes); // Decrypts the message using the cipher
99     // return new String(decryptedBytes, StandardCharsets.UTF_8); // This didn't work somehow
100     return new String(decryptedBytes); // Returns decrypted string of the message
101 }

```

Figure 2.40: Η μέθοδος decrypt της κλάσης Aesci για κρυπτογράφηση String

Δύο απλές, αλλά σημαντικές μέθοδοι είναι οι encode και decode. Η πρώτη δέχεται ένα byte array και το επιστρέφει ως base64 string (έτσι μπορεί να αναπαρασταθεί ως κείμενο σωστά), ενώ η δεύτερη δέχεται ένα base64 string (έχει νόημα η διαδικασία μόνο αν είναι κωδικοποιημένο σε base64) και επιστρέφει την αποκωδικοποιημένη byte array μορφή του (έτσι μπορεί να επεξεργασθεί από τις διάφορες μεθόδους, όπως να αποκρυπτογραφηθεί από την decrypt).

```

103 // Converts byte array to Base64 string (to be readable and printed)
104 private String encode(byte[] data) {
105     return Base64.getEncoder().encodeToString(data);
106 }
107
108
109 // Converts Base64 string to byte array (to be used in cipher mainly)
110 private byte[] decode(String data) {
111     return Base64.getDecoder().decode(data);
112 }

```

Figure 2.41: Οι μέθοδοι encode και decode της κλάσης Aesci για κωδικοποίηση byte array σε base64 encoded String και αποκωδικοποίηση απλό String αντίστοιχα

Συνεχίζοντας έχουμε τρεις debug μεθόδους, τις exportKeys, exportIV και exportKey, που κάνουν print στο console το κλειδί με το IV, το IV μόνο και το κλειδί μόνο αντίστοιχα.

```

114 // Prints Key and IV encoded in base64 on the console
115 public void exportKeys() {
116     System.err.println("Key (base64 encoded): " + encode(key.getEncoded()));
117     System.err.println("IV (base64 encoded): " + encode(IV));
118     // System.err.println(key.getEncoded()); // bare byte array
119     // System.err.println(IV); // bare byte array
120 }
121
122 // Prints IV encoded in base64 on the console
123 public void exportIV() {
124     System.err.println("IV (base64 encoded): " + encode(IV));
125 }
126
127 // Prints Key encoded in base64 on the console
128 public void exportKey() {
129     System.err.println("Key (base64 encoded): " + encode(key.getEncoded()));
130 }

```

Figure 2.42: Οι μέθοδοι `exportKeys`, `exportIV` και `exportKey` της κλάσης `Aesci` για print στο console το κλειδί με το IV, το IV και το κλειδί αντίστοιχα

Η `encryptMessage` δέχεται ένα string, το μήνυμά μας, μαζί με έναν δείκτη και επιστρέφει το κρυπτογραφημένο μήνυμα ως string (για να σταλεί με την κατάλληλη μέθοδο). Ανεξαρτήτως την τιμή του δείκτη, κρυπτογραφεί το μήνυμα βάζοντας ως πρόθεμα το `ivstr`, δηλαδή το προσωρινό IV.

Αν δεν έχει προηγηθεί η `IVgen`, τότε το `ivstr` είναι ίδιο με το IV, αλλιώς το `ivstr` είναι το νέο που μόλις παράξαμε. Κρυπτογραφούμε πάντα το μήνυμα με το "παλιότερο" IV που έχουμε, μεταξύ των IV και `ivstr`, ώστε ο remote να μπορεί να αποκρυπτογραφήσει το μήνυμα, καθώς δεν γνωρίζει το νέο IV.

Σε κάθε περίπτωση, στέλνουμε το `ivstr` μαζί με το μήνυμα έτσι ώστε όταν το αποκρυπτογραφήσει ο remote, να γνωρίζει τι IV πρόκειται να χρησιμοποιήσουμε αφότου στείλουμε το μήνυμα. Έτσι ο remote θα χρησιμοποιήσει αυτό το ενημερωμένο IV που του στείλαμε, για να κρυπτογραφήσει το μήνυμα που πρόκειται να στείλει ή να αποκρυπτογραφήσει το επόμενο μήνυμα που πρόκειται να λάβει. Με αυτό τον τρόπο local και remote ενημερώνονται δι-αρκώς για το IV εν χρήση.

Αν ο δείκτης είναι 0, τότε η `encryptMessage` κρυπτογραφεί και στέλνει το μήνυμα χωρίς να δημιουργεί νέο IV. Αυτό ενώ πάει αντίθετα με όσα λέγαμε, ο λόγος που το χρησιμοποιούμε είναι για να στείλουμε ένα μήνυμα σε chunks. Οπότε πρακτικά χρησιμοποιούμε ένα IV ένα μήνυμα. Αν ο δείκτης είναι 1, τότε με το `IVgen` παράγει νέο IV, κρυπτογραφεί το μήνυμα (μαζί με το νέο IV) και μετά με την `setIV` ορίζει ως νέο IV το `ivstr` που μόλις παράξαμε.

```

132 // Encrypts the message when we press the send button
133 public String encryptMessage(String plainMessage, int i) throws Exception {
134     System.err.println("Plain message to be sent: " + plainMessage); // Debug, prints plainMessage on the console
135     String encryptedMessage=""; // Generates the string where the encrypted message will be stored
136     // exportIV(); // Debug, prints current (old) IV on the console
137
138     if(i==0) { // Does not create new IV, uses the old (until all chunks are sent)
139         encryptedMessage = encrypt(ivstr + plainMessage); // Generates encrypted message **without** prepending new IV
140     }
141     else if(i==1) { // i>=1 It **does** generate a new IV
142         IVgen(); // generates new IV to add in the message before sending it (saved in ivstr)
143         encryptedMessage = encrypt(ivstr + plainMessage); // Generates encrypted message after prepending the new IV
144         setIV(ivstr); //sets the new IV which was previously generated //ONLY WORKS FOR SEPARATE DEVICES
145     }
146
147     // exportIV(); // Debug, prints current (new or old, depending on the if actions) IV on the console
148     System.err.println("Message encrypted and sent: " + encryptedMessage); // Debug, prints encryptedMessage on the console
149     return (encryptedMessage); // Returns encrypted message to the caller (to be sent)
150 }

```

Figure 2.43: Η μέθοδος encryptMessage της κλάσης Aesci για κρυπτογράφηση μηνύματος προς αποστολή

Η decryptMessage είναι μάλλον η μεγαλύτερη μέθοδος της κλάσης. Δέχεται ένα string, το κρυπτογραφημένο μήνυμα που λάβαμε και επιστρέφει το αποκρυπτογραφημένο μήνυμα ως string (για να το διαβάσει ο χρήστης με την κατάλληλη μέθοδο), με μερικές εξαιρέσεις. Τα τυπικά μηνύματα είναι κρυπτογραφημένα και όταν αποκρυπτογραφηθούν, οι πρώτοι 16 χαρακτήρες τους είναι το νέο IV κωδικοποιημένο σε base64 και το υπόλοιπο είναι το μήνυμα του χρήστη.

```

132 // Encrypts the message when we receive one (it is called by the UDP/TCPchat receive methods)
133 public String decryptMessage(String encryptedMessage) throws Exception {
134     // This if checks if it's related to voice call or it is part of a big message
135     // We send some commands in plain text (some times prepended to encrypted text)
136     // "[Voice-Call]" is for Voice-call related commands and comes **un**encrypted
137     // "[Part]" is for messages that come in multiple chunks (useful with UDP) and comes **en**crypted
138     if ((encryptedMessage.substring(0, 12).equals("[Voice-call]"))) { // No need for out of bounds check, the IV is 16 (>12) chars already
139         // exportIV(); // Debug, prints current (old) IV on the console
140         System.err.println("Message received: " + encryptedMessage); // Debug, prints (received) encrypted message on the console
141
142         encryptedMessage = decrypt(encryptedMessage); // Decrypts encryptedMessage and assigns it to encryptedMessage
143         ivstr = encryptedMessage.substring(0, 16); // Assigns the first 16 chars of the message to IVnew
144         setIV(ivstr); //Sets the new IV from IVnew //ONLY WORKS FOR SEPARATE DEVICES
145         encryptedMessage = encryptedMessage.substring(16, encryptedMessage.length()); //Removes IV string from message
146
147         // First we check if the "[Part]FINISHED" command was sent
148         // If true, this means remote has finished sending messages in chunks and we are ready to return the assembled message
149         if (encryptedMessage.length() > 13) { // At least 14 chars, out of bounds check needed
150             if (encryptedMessage.substring(0, 14).equals("[Part]FINISHED")) {
151                 return mesAssembler();
152             }
153             else if (encryptedMessage.substring(0, 6).equals("[Part]")) { // In case we receive a chunk with over 14 chars
154                 return chunkStorer(encryptedMessage);
155             }
156         }
157         // It checks if the chunks of a composite message are being received
158         // They should start with "[Part]" tag, followed by their increment number with a single leading zero
159         // The should be 100 max (though we could configure the system to hold more than that)
160         else if (encryptedMessage.length() > 6) { // At least 14 chars, out of bounds check needed
161             if (encryptedMessage.substring(0, 6).equals("[Part]")) {
162                 return chunkStorer(encryptedMessage);
163             }
164         }
165     }
166     System.err.println("Received message decrypted: " + encryptedMessage); // Debug, prints (received) decrypted message on the console
167     return encryptedMessage; // Returns the decrypted message (this is called only in the first if statement)
168 }

```

Figure 2.44: Η μέθοδος decryptMessage για την αποκρυπτογράφηση μηνυμάτων που λαμβάνονται

- Κάτι σημαντικό πριν εξηγήσουμε τα υπόλοιπα, είναι να πούμε πως το πρόγραμμα στέλνει και μερικές "εντολές" πέρα από τα μηνύματα του χρήστη. Συγκεκριμένα, στέλνει χωρίς κρυπτογράφηση μερικά μηνύματα που ενημερώνουν τον remote πως εκτελείται ή τελείωσε μία κλήση (η κλήση δεν είναι κρυπτογραφημένη, οπότε δεν είναι τόσο σοβαρό το ότι αυτά τα μηνύματα δεν κρυπτογραφούνται). Αυτά τα μηνύματα αρχίζουν με το πρόθεμα "[Voice-Call]".

- Επιπλέον, όταν στέλνεται ένα μεγάλο μήνυμα, έρχεται σε chunks. Αυτά τα είναι κρυπτογραφημένα και όταν αποκρυπτογραφηθούν, οι πρώτοι 16

χαρακτήρες τους είναι το νέο IV, οι αμέσως 6 επόμενοι είναι το string "[Part]" που ακολουθείται από δύο ψηφία που δηλώνουν τον αύξοντα αριθμό του chunk (πχ. "00") και το υπόλοιπο είναι το τμήμα του μηνύματος που μεταφέρει το chunk.

· Τέλος, μόλις σταλθούν όλα τα chunks, στέλνεται ένα επιπλέον κρυπτογραφημένο μήνυμα, όταν αποκρυπτογραφηθεί οι πρώτοι 16 χαρακτήρες είναι το νέο IV κωδικοποιημένο σε base64 και το υπόλοιπο είναι το string "[Part]FINISHED". Αυτά τα τρία είδη μηνυμάτων αντιμετωπίζονται διαφορετικά από την decryptMessage.

Αρχικά η decryptMessage βλέπει αν το μήνυμα είναι τύπου "[Voice-Call]" και αν ναι, το επιστρέφει στην καλούσα μέθοδο χωρίς κάποια τροποποίηση.

```
152 // Encrypts the message when we receive one (it is called by the UDP/ICPchat receive methods)
153 public String decryptMessage(String encryptedMessage) throws Exception {
154     // This if checks if it's related to voice call or it is part of a big message
155     // We send some commands in plain text (some times prepended to encrypted text)
156     // "[Voice-Call]" is for Voice-call related commands and comes **un**encrypted
157     // "[Part]" is for messages that come in multiple chunks (useful with UDP) and comes **en**crypted
158     if ((encryptedMessage.substring(0, 12).equals("[Voice-Call]"))) { // No need for out of bounds check, the IV is 16 (>12) chars already
159
160         System.err.println("Received message decrypted: " + encryptedMessage); // Debug, prints (received) decrypted message on the console
161     }
162     return encryptedMessage; // Returns the decrypted message (this is called only in the first if statement)
163 }
```

Figure 2.45: Το απόσπασμα της μεθόδου decryptMessage της κλάσης Aesci για διαχείριση μηνύματος τύπου "[Voice-Call]" που λήφθηκε

Αν δεν είναι τέτοιο μήνυμα, τότε το αποκρυπτογραφεί, εξάγει το νέο IV (τους πρώτους 16 χαρακτήρες) από το μήνυμα, το ορίζει ως το νέο IV και το αφαιρεί από το μήνυμα.

```
152 if ((encryptedMessage.substring(0, 12).equals("[Voice-Call]"))) { // No need for out of bounds check, the IV is 16 (>12) chars already
153     exportIV(); // Debug, prints current (old) IV on the console
154     System.err.println("Message received: " + encryptedMessage); // Debug, prints (received) encrypted message on the console
155
156     encryptedMessage = decrypt(encryptedMessage); // Decrypts encryptedMessage and assigns it to encryptedMessage
157     Ivstr = encryptedMessage.substring(0, 16); // Assigns the first 16 chars of the message to IVnew
158     // (It should be the new IV which remote uses by now)
159     setIV(Ivstr); // Sets the new IV from IVnew // ONLY WORKS FOR SEPARATE DEVICES
160     encryptedMessage = encryptedMessage.substring(16, encryptedMessage.length()); // Removes IV string from message
161 }
```

Figure 2.46: Το απόσπασμα της μεθόδου decryptMessage της κλάσης Aesci για αποκρυπτογράφηση του μηνύματος που λήφθηκε μαζί με εξαγωγή, αφαίρεση και αποθήκευση νέου IV από το μήνυμα

Αν το υπόλοιπο μήνυμα δεν αρχίζει από "[Part]", τότε το επιστρέφει στην καλούσα συνάρτηση.

```
157 System.err.println("Received message decrypted: " + encryptedMessage); // Debug, prints (received) decrypted message on the console
158 }
159 return encryptedMessage; // Returns the decrypted message (this is called only in the first if statement)
160 }
```

Figure 2.47: Το απόσπασμα της μεθόδου decryptMessage της κλάσης Aesci για επιστροφή αποκρυπτογραφημένου "απλού" μηνύματος που λήφθηκε

Αν όμως είναι ειδική περίπτωση και αρχίζει από "[Part]", τότε πρώτα ελέγχει (μετά από έναν έλεγχο για να αποφύγει out of bounds error) αν είναι μήνυμα τύπου "[Part]FINISHED" (δηλαδή ο remote έστειλε όλα τα chunks του μεγάλου μηνύματος) και αν ναι, επιστρέφει στην καλούσα συνάρτηση την mesAssembler, η οποία επιστρέφει το μεγάλο μήνυμα.

```

158 // First we check if the "[Part]FINISHED" command was sent
159 // If true, this means remote has finished sending messages in chunks and we are ready to return the assembled message
160 if (encryptedMessage.length() > 13) { // At least 14 chars, out of bounds check needed
161     if(encryptedMessage.substring(0, 14).equals("[Part]FINISHED")) {
162         return mesAssembler();
163     }
164 }

```

Figure 2.48: Το απόσπασμα της μεθόδου decryptMessage της κλάσης Aesci για διαχείριση μηνύματος τύπου "[Part]FINISHED" που λήφθηκε

Τέλος, υπάρχουν 2 περιπτώσεις που απομένουν να ελέγξει. Πρώτα την περίπτωση να στάλθηκε chunk μεγέθους χαρακτήρων μεγαλύτερο από 13 χαρακτήρες (περιλαμβάνοντας την λέξη "[Part]", τον διψήφιο αύξοντα αριθμό και το τμήμα του μεγάλου μηνύματος), το οποίο γίνεται χωρίς νέο out of bounds έλεγχο (έχει προηγηθεί) και αν αποτύχει εξετάζει την δεύτερη περίπτωση να στάλθηκε chunk μεγέθους κάτω από 14 χαρακτήρες με τον κατάλληλο out of bounds έλεγχο. Στις 2 αυτές τελευταίες περιπτώσεις, καλείται η chunkStorer που αποθηκεύει το chunk στον buffer της κλάσης και επιστρέφει το string "Recieving long message, wait...", για να ενημερώσει τον χρήστη πως λαμβάνει ένα μεγάλο μήνυμα.

```

168 // First we check if the "[Part]FINISHED" command was sent
169 // If true, this means remote has finished sending messages in chunks and we are ready to return the assembled message
170 if (encryptedMessage.length() > 13) { // At least 14 chars, out of bounds check needed
171     if(encryptedMessage.substring(0, 14).equals("[Part]FINISHED")) {
172         return mesAssembler();
173     }
174     else if(encryptedMessage.substring(0, 6).equals("[Part]")){ // In case we receive a chunk with over 14 chars
175         return chunkStorer(encryptedMessage);
176     }
177 }
178 // It checks if the chunks of a composite message are being received
179 // They should start with "[Part]" tag, followed by their increment number with a single leading zero
180 // The should be 100 max (though we could configure the system to hold more than that)
181 else if(encryptedMessage.length() > 6) { // At least 14 chars, out of bounds check needed
182     if(encryptedMessage.substring(0, 6).equals("[Part]")){
183         return chunkStorer(encryptedMessage);
184     }
185 }

```

Figure 2.49: Το απόσπασμα της μεθόδου decryptMessage της κλάσης Aesci για διαχείριση μηνύματος τύπου chunk (τμήμα μεγαλύτερου μηνύματος) που λήφθηκε

Ακολουθεί η mesAssembler η οποία συνενώνει όλα τα μη-null στοιχεία του bufferMes που περιέχει τα chunks ενός δεδομένου μεγάλου μηνύματος και επιστρέφει το συνολικό μήνυμα. Αρχικά δημιουργεί το finalMessage, της κλάσης StringBuilder. Αυτό το object μπορεί να αποθηκεύει μια σειρά χαρακτήρων. Η μέθοδος διαπερνά όλο το buffer κατά αύξοντα αριθμό των στοιχείων και προσθέτει στο τέλος του finalMessage το κάθε μη-null στοιχείο. Μόλις προσθέσει ένα στοιχείο στο finalMessage, κάνει το στοιχείο null για να μην μείνουν κατάλοιπα στον buffer. Επειδή τα chunks δημιουργούνται σειριακά και αριθμούνται και αποθηκεύονται με αύξοντα αριθμό, αυτό σημαίνει πως μόλις η for λούπα βρει ένα null στοιχείο, όλα τα υπόλοιπα στοιχεία θα είναι null και βγαίνει απευθείας από την λούπα. Τελικά μετατρέπει το finalMessage σε string και το επιστρέφει.

```

192 // This is called upon receiving the command "[Part]FINISHED" to assemble and return the composite message
193 public String mesAssembler() {
194     StringBuilder finalMessage = new StringBuilder(); // We use StringBuilder class to append all parts to a single variable
195
196     // It iterates through the whole buffer that contains the **decrypted** chunks
197     for (int j = 0; j < bufferedMes.length; j++) {
198         if (bufferedMes[j] == null) { // If it find a null object, it means all parts have been appended, the rest should be empty
199             break;
200         }
201         // Debug, prints each (non-null) chunk from the buffer on the console
202         System.err.println("Buffered Message Part " + j + " : \n" + bufferedMes[j]);
203
204         finalMessage.append(bufferedMes[j]); // Appends each chunk to the variable
205         bufferedMes[j] = null; // Gradually flushes the array per object to be used again in the future
206     }
207     String finalString = finalMessage.toString(); // Converts variable to string to be printed
208     System.err.println("PART FINISHED: \n" + finalString); // Debug, prints the composite message
209     return finalString; // Returns the composite message to be printed on the chat
210 }

```

Figure 2.50: Η μέθοδος mesAssembler για την συνένωση chunks μηνυμάτων που έχουν ληφθεί

Η τελευταία μέθοδος είναι η chunkStorer που δέχεται ως όρισμα ένα chunk, δηλαδή ένα αποκρυπτογραφημένο μήνυμα, το οποίο αποτελείται από το πρόθεμα string "[Part]", τον διψήφιο αύξοντα αριθμό (οι αμέσως δύο επόμενοι χαρακτήρες) και τελειώνει με το τμήμα του μεγάλου μηνύματος. Εξάγει τον αύξοντα αριθμό από το chunk και το χρησιμοποιεί για να αποθηκεύσει μόνο το τμήμα του μεγάλου μηνύματος στο στοιχείο του buffer που ορίζει ο αύξοντας αριθμός. Επιστρέφει το string "Receiving long message, wait..." για να ενημερώσει τον χρήστη πως λαμβάνει ένα μεγάλο μήνυμα από τον remote.

```

212 // This is called upon receiving a chunk of a composite message and stores the message in the local buffer
213 public String chunkStorer(String encryptedMessage) throws Exception {
214     int i = Integer.parseInt(encryptedMessage.substring(6, 8)); // Extract the numbering of the part (format: [Part]00)
215
216     // Removes the "[Part]00" tag from the message and **decrypts** the message
217     bufferedMes[i] = decrypt(encryptedMessage.substring(8, encryptedMessage.length()));
218     // It comes decrypted to this method
219     bufferedMes[i] = encryptedMessage.substring(8, encryptedMessage.length());
220     //NOPE String ivnew = bufferedMes[i].substring(0, 16); // Assigns the first 16 chars of the message to IVnew
221     // (It should be the new IV which remote uses by now)
222     //setIV(ivnew); //Sets the new IV //ONLY WORKS FOR SEPARATE DEVICES
223     //NOPE bufferedMes[i] = bufferedMes[i].substring(16, bufferedMes[i].length()); //Removes IV string from message and saves it to buffer in plain text
224     // Debug, prints message which is now stored in the buffer on the console
225     System.err.println("Buffered Message Part " + i + " : \n" + bufferedMes[i]);
226     return ("Receiving long message, wait..."); // We ask the user to wait as the system collects each chunk
227 }

```

Figure 2.51: Η μέθοδος chunkStorer για την αποθήκευση chunks μηνυμάτων που λαμβάνονται

2.5 Λοιπές λειτουργίες

Στο πρόγραμμα υπάρχουν άλλες δύο λειτουργίες υλοποιημένες στην κλάση `app` που ενεργοποιούνται μέσω δύο νέων κουμπιών.

2.6 Το κουμπί "Set Pass"

Μόλις πατηθεί το κουμπί "Set Pass" μόλις πατηθεί, χρησιμοποιείται ως όρισμα το κείμενο που έχει γράψει ο χρήστης στο πεδίο νέου μηνύματος και εφόσον είναι πάνω από 5 χαρακτήρες, χρησιμοποιείται ως password για την δημιουργία νέου κλειδιού κρυπτογράφησης καλώντας την `aesci.initFromPassword`. Είναι χρήσιμο, καθώς το default password είναι γνωστό σε όποιο άτομο έχει πρόσβαση στον κώδικα της εφαρμογής. Επιπλέον, μπορεί να χρησιμοποιηθεί σε περίπτωση που για κάποιο λόγο φαίνεται να έχει διαρρεύσει το υπάρχον password. Για να λειτουργήσει η απο/κρυπτογράφηση, πρέπει και `local` και `remote` να θέσουν το ίδιο password.

```
337 else if (e.getSource() == passButton){ // if user wants to change password-AES key (both should have the same)
338     String pass = inputTextField.getText();
339     if (!pass.isEmpty() && (pass.length() < 6)) { // checks if password is empty or under 6 chars
340         aesci.exportKeys(); // debug, prints key and IV on the console
341     }
342     try {
343         aesci.initFromPassword(pass);
344     } catch (NoSuchAlgorithmException | InvalidKeySpecException e1) {
345         // TODO Auto-generated catch block
346         e1.printStackTrace();
347     }
348     aesci.exportKeys(); // debug, prints key and IV on the console
349     textArea.append("Password changed to : " + pass.substring(0, 1) + "*****" + pass.substring((pass.length()-1), pass.length()) + newline);
350     inputTextField.setText(""); //for extra protection we can remove the password from the textfield
351 //
352 }
353 }
354 }
355 }
356 }
```

Figure 2.52: Η μέθοδος `chunkStorer` για την αποθήκευση chunks μηνυμάτων που λαμβάνονται

2.7 Το κουμπί "Clear Chat"

Το κουμπί "Clear Chat" ως σκοπό τον καθαρισμό της περιοχής μηνυμάτων, μόλις πατηθεί.

```
358 else if (e.getSource() == clearButton){ // Clears the chat area
359     textArea.setText("Text Cleared :) " + newline);
360     inputTextField.setText("");
361 }
362 }
```

Figure 2.53: Η μέθοδος `chunkStorer` για την αποθήκευση chunks μηνυμάτων που λαμβάνονται

2.8 Ορισμός των κουμπιών του προγράμματος

Για αυτές τις δύο λειτουργίες έχουν δημιουργηθεί δύο κουμπιά στην κλάση `App`.

```

36     static JButton passButton; //button to change AES key
37 // static JButton ipButton; //button to change remote IP
38     static JButton clearButton; //button to clear chat

118     //Setting up the buttons
119     sendButton = new JButton("Send");
120     callButton = new JButton("Call");
121     passButton = new JButton("Set Pass");
122 // ipButton = new JButton("Set Remote IP");
123     clearButton = new JButton("Clear Chat");
124     /*
125      * 2. Adding the components to the GUI
126      */
127     add(scrollPane);
128     add(inputTextField);
129     add(sendButton);
130     add(callButton);
131     add(passButton);
132 // add(ipButton);
133     add(clearButton);
134
135     /*
136      * 3. Linking the buttons to the ActionListener
137      */
138     sendButton.addActionListener(this);
139     callButton.addActionListener(this);
140     passButton.addActionListener(this);
141 // ipButton.addActionListener(this);
142     clearButton.addActionListener(this);
143 }

```

Figure 2.54: Ορισμός των κουμπιών στην κλάση App

3 Απεικόνιση πακέτων μέσω Wireshark

Σε αυτό το θέμα παρουσιάζονται εικόνες από το Wireshark οι οποίες δείχνουν τη μορφή πακέτων κειμένου που στάλθηκαν μεταξύ των δύο users με χρήση UDP και TCP, το stream των πακέτων φωνής που ανταλλάσσονται με τη λειτουργία VoIP και τη μορφή πακέτων φωνής που ανταλλάχθηκαν μεταξύ των δύο users. Οι IP διευθύνσεις των δύο user προκύπτουν με την εντολή `ipconfig` στο `command line` (ή `ip address` στα `linux`) του καθενός. Για διευκόλυνση, χρησιμοποιείται η εντολή `ip.addr == "IPaddress" and udp` ή `tcp` ανάλογα το πρωτόκολλο που χρησιμοποιείται, όπου "IPaddress" η IP του remote για να εντοπιστούν μέσα από το Wireshark τα πακέτα που ανταλλάσσει μόνο ο local με τον remote.

3.1 Πακέτα κειμένου

3.1.1 UDP πακέτα κειμένου

Παρακάτω φαίνονται οι εικόνες από το textArea και από το Wireshark για μια σύντομη κρυπτογραφημένη chat συνομιλία με UDP μεταξύ του local (192.168.208.148) και του remote (192.168.208.134).

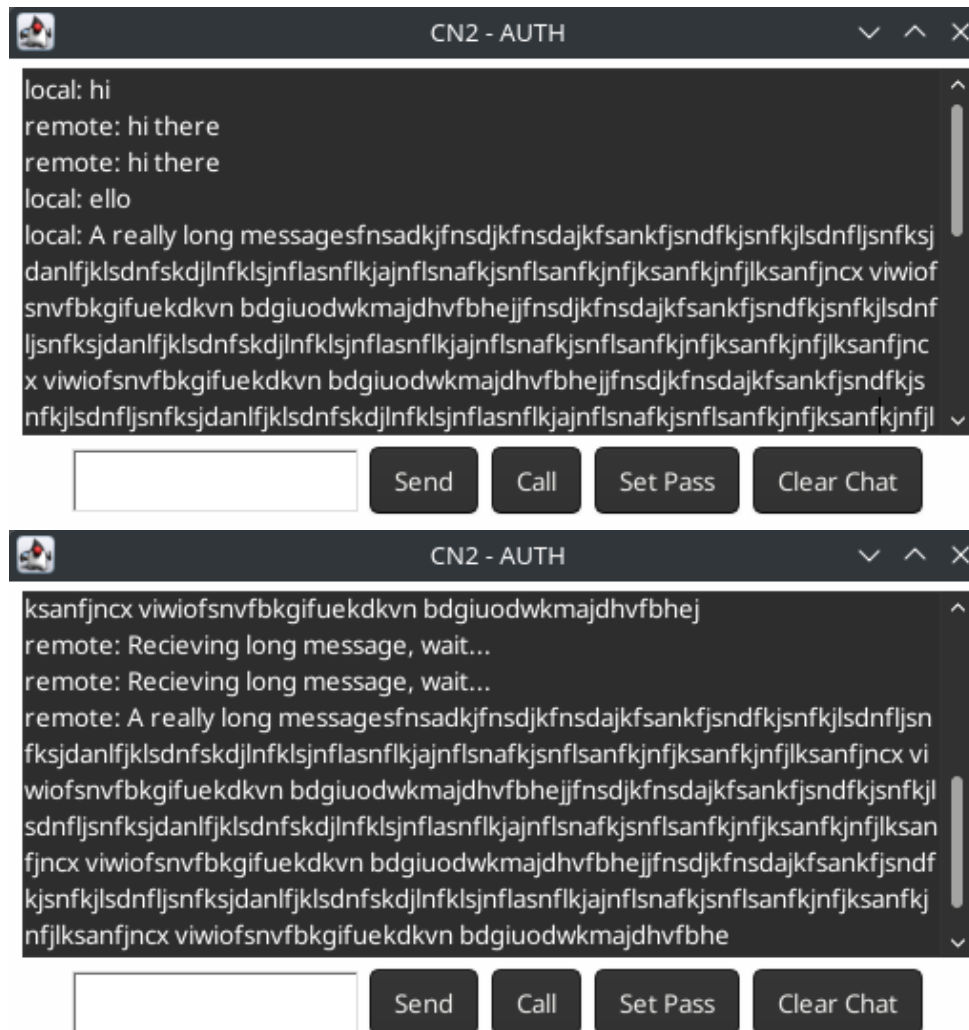


Figure 3.1: Συνομιλία στο textArea

No.	Time	Source	Destination	Protocol	Length	Info
7	14.720840090	192.168.208.148	192.168.208.134	UDP	60	1234 → 1234 Len=48
32	23.140882819	192.168.208.134	192.168.208.148	UDP	98	1234 → 1234 Len=56
53	30.51571601	192.168.208.134	192.168.208.148	UDP	98	1234 → 1234 Len=56
59	39.469761963	192.168.208.148	192.168.208.134	UDP	90	1234 → 1234 Len=48
74	50.852232287	192.168.208.148	192.168.208.134	UDP	762	1234 → 1234 Len=720
75	50.852748115	192.168.208.148	192.168.208.134	UDP	130	1234 → 1234 Len=88
81	51.553636242	192.168.208.148	192.168.208.134	UDP	106	1234 → 1234 Len=64
109	64.100389489	192.168.208.134	192.168.208.148	UDP	762	1234 → 1234 Len=720
110	64.100389529	192.168.208.134	192.168.208.148	UDP	130	1234 → 1234 Len=88
111	64.714551481	192.168.208.134	192.168.208.148	UDP	106	1234 → 1234 Len=64

Figure 3.2: Ανταλλαγή 12 πακέτων κειμένου

Υπογραμμίζεται το payload σε κάθε πακέτο και το κομμάτι από πάνω του είναι το Network/Internet header. Τα δύο αυτά μέρη του πακέτου παρουσιάζ-

ζονται τόσο σε εξαδική όσο και σε μορφή κειμένου.

> Frame 7: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface wlp2s0, id 0	0000	60 a5 e2 a0 c8 37 58 fb 84 cc 4d a4 08 00 45 007X.....M...E
> Ethernet II, Src: Intel_cc:4d:a4 (58:fb:84:cc:4d:a4), Dst: Intel_a0:c8:37 (60:a5:e2:a0:c8:37)	0010	00 4c 00 b9 40 00 00 11 ac 7b c0 a8 00 94 c0 a8	..Lk.0.0.8.
> Internet Protocol Version 4, Src: 192.168.208.148, Dst: 192.168.208.134	0030	4c 41 58 5c 58 54 74 64 2f 64 43 87 71 75 31 39DPLIX1f4..Dcoqu19
> User Datagram Protocol, Src Port: 1234, Dst Port: 1234	0040	85 45 39 59 35 76 65 59 62 6b 46 73 2b 49 34 36	..eSY5veY bHMs+I45
> Data (48 bytes)	0050	74 63 6d 24 45 51 77 67 30 30	..m4EQun
> Data (48 bytes)			
> Data: 7458483134384c44506c585474642f62436f7717531396545395935766559626b48732b49343574636d43445			
> [Length: 48]			

Figure 3.3: Μορφή του 1ου πακέτου, μήνυμα "hi" του local στον remote

> Frame 32: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface wlp2s0, id 0	0000	58 fb 84 cc 4d a4 60 a5 e2 a0 c8 37 08 00 45 00	X...M...7...E
> Ethernet II, Src: Intel_a0:c8:37 (60:a5:e2:a0:c8:37), Dst: Intel_cc:4d:a4 (58:fb:84:cc:4d:a4)	0010	00 54 0d a2 00 00 80 11 0a 8a c0 a8 00 86 c0 a8Lk.0.0.8.
> Internet Protocol Version 4, Src: 192.168.208.134, Dst: 192.168.208.148	0030	76 67 37 58 5b 4b 57 44 9a 5a 76 51 58 75 35 70v07HYKb0..5v9ku0p
> User Datagram Protocol, Src Port: 1234, Dst Port: 1234	0040	8a 44 32 69 77 77 69 61 48 65 59 6a 68 75 31 64	..J02IwIda..hCYJhuId
> Data (56 bytes)	0050	33 4b 45 79 6c 7a 32 35 70 34 54 30 34 2f 79 77	..hYey1z25..y4Tt47w
> Data (56 bytes)	0060		
> Data: 356d454e474c766f3750594b57446a53765158753765a44326977769614863596a68753164336845796c			
> [Length: 56]			

Figure 3.4: Μορφή του 2ου πακέτου, μήνυμα "hi there" του remote στον local

> Frame 53: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface wlp2s0, id 0	0000	58 fb 84 cc 4d a4 60 a5 e2 a0 c8 37 08 00 45 00	X...M...7...E
> Ethernet II, Src: Intel_a0:c8:37 (60:a5:e2:a0:c8:37), Dst: Intel_cc:4d:a4 (58:fb:84:cc:4d:a4)	0010	00 54 0d a3 00 00 80 11 0a 8a c0 a8 00 86 c0 a8Lk.0.0.8.
> Internet Protocol Version 4, Src: 192.168.208.134, Dst: 192.168.208.148	0030	63 6b 76 58 38 35 6c 4c 6c 5a 59 35 36 35 6e 590kvpB5L1..1P565vN
> User Datagram Protocol, Src Port: 1234, Dst Port: 1234	0040	7a 47 4c 32 58 72 43 4c 39 31 38 55 63 6e 4b 43	..ZGL2XrCL..918ucnKC
> Data (56 bytes)	0050	64 35 45 4a 61 6f 6d 65 4e 35 56 35 69 2f 58 21	..HUI2aome..F3V5L1X0
> Data (56 bytes)	0060		
> Data: 2f6f57434d77636b765838356c4c6c49583536356e597a474c325872434c39313855636e4b436454554a61			
> [Length: 56]			

Figure 3.5: Μορφή του 3ου πακέτου, μήνυμα "hi there" του remote στον local. Έχει διαφορετική μορφή από το προηγούμενο, παρότι είναι το ίδιο μήνυμα σε plain text.

> Frame 59: 90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface wlp2s0, id 0	0000	60 a5 e2 a0 c8 37 58 fb 84 cc 4d a4 08 00 45 007X.....M...E
> Ethernet II, Src: Intel_cc:4d:a4 (58:fb:84:cc:4d:a4), Dst: Intel_a0:c8:37 (60:a5:e2:a0:c8:37)	0010	00 4c 05 6e 49 00 00 11 62 c6 c0 a8 00 94 c0 a8	..Ln0.0.0.Lk.0.0.
> Internet Protocol Version 4, Src: 192.168.208.148, Dst: 192.168.208.134	0030	47 33 46 37 5e 37 52 7a 44 52 67 4e 6d 76 4e 5a03F7h782..UqHk02
> User Datagram Protocol, Src Port: 1234, Dst Port: 1234	0040	8c 39 37 39 6e 68 44 30 46 43 7a 70 69 33 72 62	..l97nhd00..KZCp13rb
> Data (48 bytes)	0050	2b 32 66 44 44 77 2f 58 36 4e	..zrD6w0..E
> Data (48 bytes)			
> Data: 4c2b746d5143473346376e37527a4452674e6d784e5a6c3937396e68443846437a70693372622b32664444			
> [Length: 48]			

Figure 3.6: Μορφή του 4ου πακέτου, μήνυμα "ello" του local στον remote.

Για μεγάλα μεγέθη μηνυμάτων, πάνω από 500 χαρακτήρες, δηλαδή μεγαλύτερα μεγέθη πακέτων, παρατηρούμε πως τα μηνύματα στέλνονται σε chunks όπως έχουμε ορίσει και τελευταίο έρχεται ένα μικρό μήνυμα, το οποίο περιέχει την εντολή "[Part]FINISHED" ώστε να συνενώσει τα κομμάτια σε ένα και να το τυπώσει στο textArea του δέκτη.

> Frame 74: 762 bytes on wire (6096 bits), 762 bytes captured (6096 bits) on interface wlp2s0, id 0	0000	60 a5 e2 a0 c8 37 58 fb 84 cc 4d a4 08 00 45 007X.....M...E
> Ethernet II, Src: Intel_cc:4d:a4 (58:fb:84:cc:4d:a4), Dst: Intel_a0:c8:37 (60:a5:e2:a0:c8:37)	0010	62 ec cb 74 49 00 00 11 4a 20 c0 a8 00 94 c0 a8	...Lk.0.0.8.
> Internet Protocol Version 4, Src: 192.168.208.148, Dst: 192.168.208.134	0030	8c 63 56 56 50 78 47 65 43 4a 79 59 66 4a 63 41lLvCPKoe..0lyrf3JA
> User Datagram Protocol, Src Port: 1234, Dst Port: 1234	0040	78 59 44 6a 45 58 33 55 66 47 50 79 39 78 38 4aK1DjEKU0..p08ba000
> Data (720 bytes)	0050	48 72 7a 48 48 8c 74 47 56 32 36 5a 60 30 2b 89W72H1T0..U25Za9vY
> Data (720 bytes)	0060	56 50 50 71 6f 41 66 39 69 63 4a 66 39 78 71 36VPRqoA79..kcJ79xqs
> Data (720 bytes)	0070	51 75 30 70 35 7a 36 69 42 69 61 68 24 4f 27 71U9p9S261..81a80U7
> Data (720 bytes)	0080	00 49 2f 2b 62 5a 45 49 56 94 57 67 67 48 71 590774ZB51..x47g0Yv
> Data (720 bytes)	0090	77 42 2f 79 4c 71 64 59 61 46 34 6a 75 56 4c 39wBvYlqdY..aE1jvUv9
> Data (720 bytes)	00a0	57 7a 6d 58 37 57 74 52 58 41 6d 5a 4d 54 82p0m7W8R..AAa0M7
> Data (720 bytes)	00b0	73 66 71 76 78 59 61 45 33 56 67 7a 7a 59 6a 08HfaypPaE..3l0zz2vK
> Data (720 bytes)	00c0	63 6b 4b 74 58 68 53 34 77 4b 4b 6f 41 41 50 47cKk1Xh54..wKoaA0P6
> Data (720 bytes)	00d0	6d 35 4c 4c 49 00 35 62 73 6b 63 32 56 36 51 6ap0H1Hsb..hKcZ206
> Data (720 bytes)	00e0	6f 6a 50 67 63 41 41 42 6e 77 56 31 54 49 35 09HjPqCzAB..hwV1T5P
> Data (720 bytes)	00f0	6b 79 36 54 50 68 41 2f 69 73 62 70 54 55 4e 39ky0TPHA7..1shpTUN9
> Data (720 bytes)	0100	84 41 49 45 4e 31 4f 33 43 55 71 67 45 69 79 34HATE1K00..0Uw9Ew7
> Data (720 bytes)	0110	32 79 6c 52 4d 63 63 6e 63 38 78 7a 74 51 59 4a2y1Rksgn..0lyr10Yv
> Data (720 bytes)	0120	5a 6e 45 51 72 4d 55 6b 53 36 6a 4d 54 4c 34 35ZneQrMuk..59jMTL45
> Data (720 bytes)	0130	47 78 56 30 4c 45 32 63 42 5a 6c 39 37 4a 470xv0IE2c..8210J1a6
> Data (720 bytes)	0140	73 63 62 4e 3e 5a 4e 41 46 47 76 62 33 33 52 08h0m0ZNA..Nov033Kv
> Data (720 bytes)	0150	4b 76 58 55 57 69 74 47 75 4d 38 30 2b 36 71 4ckvXUv110..w89+6QL
> Data (720 bytes)	0160	55 6a 4a 35 4a 32 68 53 7a 37 39 37 64 51 77 78UJ9J2n5..27970u0v
> Data (720 bytes)	0170	6a 4a 35 63 53 72 73 78 49 70 4c 45 5a 7a 63 891J5CSr5x..1pLIZ2v9
> Data (720 bytes)	0180	76 63 61 6c 6c 57 72 70 4c 39 44 36 66 39 77 48veal1wPp..1906F9w
> Data (720 bytes)	0190	73 46 5a 45 49 0f 40 45 6e 50 2b 6e 58 4d 4c 74KZEEN0Hh..p0+0MkL1
> Data (720 bytes)	01a0	4e 47 36 36 42 4e 42 52 57 42 77 69 52 50 6d 74M66EBNER..Wb0uR0et
> Data (720 bytes)	01b0	66 44 51 38 45 73 4b 76 47 69 6d 51 34 57 31 4bfQ08EavN..0a10dWkK
> Data (720 bytes)	01c0	6f 72 71 73 55 73 32 61 39 59 36 31 30 2f 49 55p0x0u52a..0V610JZ1
> Data (720 bytes)	01d0	8c 40 68 40 78 72 71 45 33 56 49 36 41 79 7a 6c1MkXrgE..3v16Ay21
> Data (720 bytes)	01e0	44 65 6e 34 43 51 71 51 48 66 38 36 6a 47 73 6c0e0dCQ0Q..1f8610a1
> Data (720 bytes)	01f0	46 36 37 52 70 34 71 6c 54 35 45 49 39 67 45p070RYq1..1f8610a2
> Data (720 bytes)	0200	75 59 45 32 59 46 32 71 48 30 4c 6a 53 43 6c 61UyE2PF2Q..h0L15C5
> Data (720 bytes)	0210	85 49 36 30 46 44 47 72 64 49 46 43 46 6f 62 45E166FD0r..o1FC+0eB
> Data (720 bytes)	0220		

Figure 3.7: Μορφή του 5ου πακέτου, το 1ο τμήμα του μεγάλου μηνύματος του local στον remote.

```

0218 85 48 36 30 46 44 47 72 64 40 46 43 46 67 62 45 0100FD00 01F5F001
0219 76 77 02 74 79 70 37 58 52 50 2F 41 66 38 4B 65 00RtYyIX 27/AnRk1
0220 6F 62 52 58 69 31 72 76 56 65 65 53 31 30 4B 48 0hRX11rv VeeS10Rk
0240 73 63 2F 40 49 20 32 79 76 59 4A 65 65 74 6C 66 6C/F1c2y vJneSt1a
0250 48 49 46 52 37 59 55 48 36 44 4C 46 32 43 61 58 0HFRVWH 60Lk2C01
0260 47 46 47 66 39 2F 4d 37 72 65 38 46 46 2F 69 5A 6EGn9/M7 re8H/1Z
0270 56 58 49 70 48 66 75 72 64 7A 34 4d 59 79 34 59 0XIVHfUr 0z4My4Y
0280 36 7A 55 4E 74 5A 63 72 78 43 33 44 64 48 4D 65 6cU1ZcZr yC3D0Rk6
0290 59 48 49 62 32 33 33 66 66 69 4b 32 6C 6F 53 67 0YH1233n f1k21o8e
02a0 4A 2F 40 5A 4b 78 53 61 35 56 67 35 20 38 62 46 0/MDKx5n 0Vg5+80k
02b0 37 42 35 55 54 79 64 61 44 56 63 78 66 60 77 78Uu5n 0V05+T1a
02c0 4C 6C 54 37 4F 49 41 50 38 57 39 38 31 33 64 64 0L1701AP 0W0813d6
02d0 08 46 31 59 49 68 71 58 35 75 71 49 6A 53 63 4C 0P1X1h0K 0uq1JSc1
02e0 74 47 36 67 40 72 54 67 58 78 55 78 37 69 44 48 0G5pR10 0u07J102
02f0 75 61 72 68 68 33 53 78 68 21 0arh35X 17

```

Figure 3.8: Μορφή του 6ου πακέτου, το 2ο τμήμα του μεγάλου μήνυματος του local στον remote.

```

> Frame 75: 136 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on interface wlp2s0, 1c
> Ethernet II, Src: Intel_cc:4d:a4 (58:fb:84:cc:4d:a4), Dst: Intel_a0:c8:37 (60:a5:e2:a0:c8:37)
> Internet Protocol Version 4, Src: 192.168.208.148, Dst: 192.168.208.134
> User Datagram Protocol, Src Port: 1234, Dst Port: 1234
  Data (88 bytes)
    60 a5 e2 a0 c8 37 58 fb 84 cc 4d a4 08 00 45 00 0000 60 a5 e2 a0 c8 37 58 fb 84 cc 4d a4 08 00 45 00
    0010 00 74 cb 75 40 00 40 11 4c 97 c0 a8 09 94 c9 a8 0010 00 74 cb 75 40 00 40 11 4c 97 c0 a8 09 94 c9 a8
    0020 00 86 04 d2 04 d2 00 00 00 43 64 73 73 20 57 45 0020 00 86 04 d2 04 d2 00 00 00 43 64 73 73 20 57 45
    0030 6c 63 56 63 59 78 4f 65 43 4a 79 59 66 4a 63 41 0030 6c 63 56 63 59 78 4f 65 43 4a 79 59 66 4a 63 41
    0040 78 69 4a 6a 45 58 33 55 6a 48 46 4e 73 6f 64 7a 0040 78 69 4a 6a 45 58 33 55 6a 48 46 4e 73 6f 64 7a
    0050 36 6e 77 6d 79 69 73 66 38 33 30 73 54 79 38 5a 0050 36 6e 77 6d 79 69 73 66 38 33 30 73 54 79 38 5a
    0060 64 57 4f 36 55 54 61 74 36 61 4d 2f 54 43 4f 4b 0060 64 57 4f 36 55 54 61 74 36 61 4d 2f 54 43 4f 4b
    0070 24 6e 46 79 47 72 45 4c 60 75 21 63 34 73 45 40 0070 24 6e 46 79 47 72 45 4c 60 75 21 63 34 73 45 40
    0080 20 58 0080 20 58
  [Length: 88]

```

Figure 3.9: Μορφή του 7ου πακέτου, η (κωδικοποιημένη) εντολή "[Part]FINISHED" του local στον remote.

Τα υπόλοιπα μηνύματα δεν έχει κάποιος εκπαιδευτικό ενδιαφέρον για να τα αναλύσουμε, ήταν κυρίως για λόγους επιβεβαίωσης λειτουργίας του προγράμματος. Το μόνο που θα μπορούσε να αναφερθεί είναι πως όταν λήφθηκαν τα chunks από τον remote στον local, η εφαρμογή έγραψε στο chat "remote: Recieving long message, wait...", όπως την έχουμε ορίσει να κάνει.

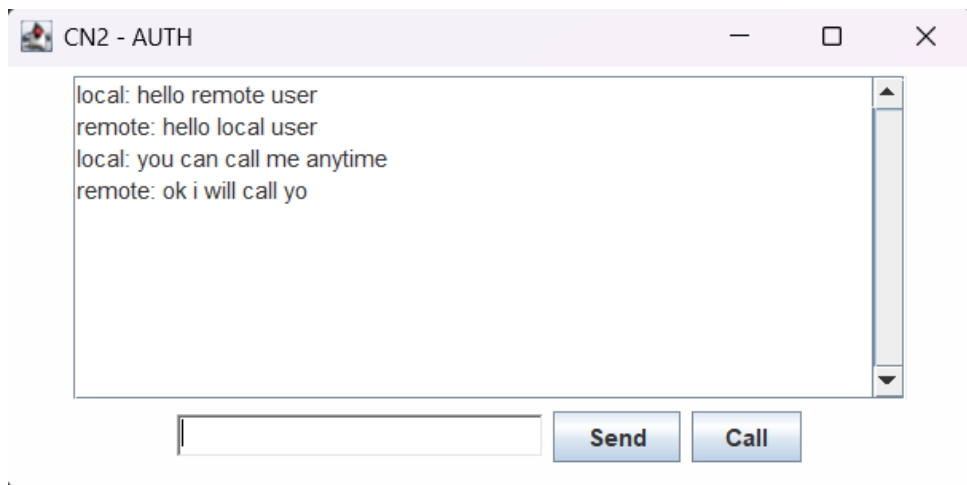


Figure 3.10: Συνομιλία στο textArea

No.	Time	Source	Destination	Protocol	Length	Info
1006	206.646984	192.168.1.18	192.168.1.14	UDP	59	1234 → 1234 Len=17
1082	232.758754	192.168.1.14	192.168.1.18	UDP	60	1234 → 1234 Len=16
1520	316.471032	192.168.1.18	192.168.1.14	UDP	59	1234 → 1234 Len=17
1539	324.812928	192.168.1.14	192.168.1.18	UDP	60	1234 → 1234 Len=16
1743	368.637633	192.168.1.18	192.168.1.14	UDP	65	1234 → 1234 Len=23
1779	381.469834	192.168.1.14	192.168.1.18	UDP	65	1234 → 1234 Len=23

Figure 3.11: Ανταλλαγή πακέτων χειμένου

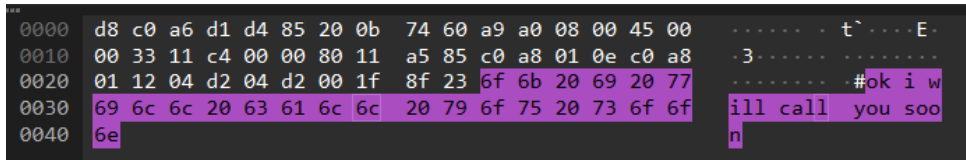


Figure 3.12: Μορφή του τελευταίου πακέτου, μήνυμα του remote στον local

3.1.2 TCP πακέτα κλειμένου

Παρακάτω φαίνονται οι εικόνες από το textArea και από το Wireshark για μια σύντομη chat συνομιλία με TCP μεταξύ του local (192.168.1.18) και του remote (192.168.1.14).

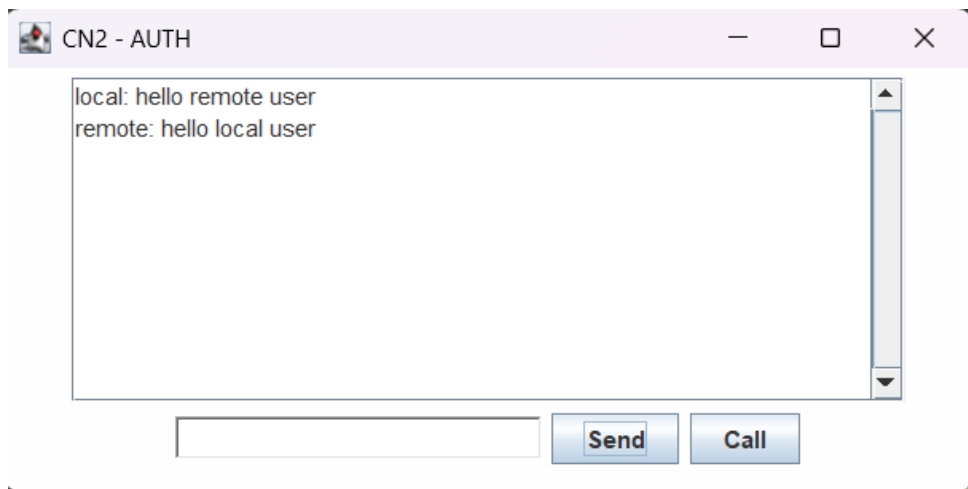


Figure 3.13: Συνομιλία στο textArea

Καθώς χρησιμοποιείται TCP, τα segments που ανταλλάσσονται δεν είναι μόνο segments κειμένου, αλλά περιλαμβάνουν κι άλλες λειτουργίες όπως το Three-Way-Handshake για εγκαθίδρυση της σύνδεσης μεταξύ των δύο users, τα ACKs για επιβεβαίωση λήψης μηνυμάτων και το Four-Way-Handshake για λήξη της σύνδεσης μεταξύ των δύο users. Η ανταλλαγή τέτοιων segments σε συνδυασμό με τα segments κειμένου φαίνεται παρακάτω.

No.	Time	Source	Destination	Protocol	Length	Info
684	111.027775	192.168.1.18	192.168.1.14	TCP	66	58221 → 2345 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
687	111.718631	192.168.1.14	192.168.1.18	TCP	66	2345 → 58221 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
689	111.718951	192.168.1.18	192.168.1.14	TCP	54	58221 → 2345 [ACK] Seq=1 Ack=1 Win=65280 Len=0
716	123.039277	192.168.1.18	192.168.1.14	TCP	72	58221 → 2345 [PSH, ACK] Seq=1 Ack=1 Win=65280 Len=19
718	123.291861	192.168.1.14	192.168.1.18	TCP	60	2345 → 58221 [ACK] Seq=1 Ack=20 Win=65280 Len=0
764	135.579031	192.168.1.14	192.168.1.18	TCP	72	2345 → 58221 [PSH, ACK] Seq=1 Ack=20 Win=65280 Len=18
765	135.626112	192.168.1.18	192.168.1.14	TCP	54	58221 → 2345 [ACK] Seq=20 Ack=19 Win=65280 Len=0
785	140.894892	192.168.1.18	192.168.1.14	TCP	54	58221 → 2345 [FIN, ACK] Seq=20 Ack=19 Win=65280 Len=0
789	141.108826	192.168.1.14	192.168.1.18	TCP	60	2345 → 58221 [ACK] Seq=19 Ack=21 Win=65280 Len=0
807	145.549983	192.168.1.14	192.168.1.18	TCP	60	2345 → 58221 [FIN, ACK] Seq=19 Ack=21 Win=65280 Len=0
808	145.550509	192.168.1.18	192.168.1.14	TCP	54	58221 → 2345 [ACK] Seq=21 Ack=20 Win=65280 Len=0

Figure 3.14: Ανταλλαγή segments με Four-Way-Handshake connection termination

Τα πρώτα τρία segments αφορούν το Three-Way-Handshake. Αφού ο local έχει τον ρόλο του client, στέλνει segment με SYN ενημερώνοντας τον remote ότι έχει ISN = 0. Αυτός αφού έλαβε το SYN του local στέλνει ACK με τον seq του επόμενου segment που πρέπει να σταλεί, ACK = 1 και segment με

SYN ενημερώνοντας τον local ότι έχει $ISN = 0$. Ο local λαμβάνει το SYN του remote και στέλνει ACK με τον seq του επόμενου segment που πρέπει να σταλεί, $ACK = 1$. Στο συγκεκριμένο παράδειγμα ο remote τρέχει πρώτος το app και δεν χρειάζεται να κάνει retransmit ο local το SYN flag μέχρι να το λάβει ο remote.

Τα τελευταία τέσσερα segments αφορούν το Four-Way-Handshake. Ο local δηλώνει ότι δεν θα στείλει άλλα δεδομένα, στέλνει segment με FIN ενημερώνοντας τον remote και μεταβαίνει από την κατάσταση ESTABLISHED στην κατάσταση FIN_WAIT_1 . Ο remote λαμβάνει το FIN flag, στέλνει ACK ως επιβεβαίωση το οποίο λαμβάνει ο local και μεταβαίνει στην κατάσταση FIN_WAIT_2 . Όταν ο remote θέλει και αυτός να διακοπεί η σύνδεση, στέλνει segment με FIN στον local και όταν ο local το λάβει μεταβαίνει στην κατάσταση $TIME_WAIT$. Ενώ βρίσκεται σε αυτήν την κατάσταση στέλνει ACK ως επιβεβαίωση του FIN flag που έλαβε και μετά από λίγο τερματίζεται η TCP σύνδεση.

Τα υπογραμμισμένα με μαύρο segments είναι τα μηνύματα κειμένου που ανταλλάσσονται (PUSH) μεταξύ του local και του remote. Μετά από κάθε μήνυμα κειμένου στέλνεται ACK από τον δέκτη, με τον seq του επόμενου segment που πρέπει να σταλεί. Παρακάτω φαίνεται το payload και το Network/Internet header του καθενός, το οποίο προφανώς είναι μεγαλύτερο σε σχέση με αυτό του UDP.

0000	20 0b 74 60 a9 a0 d8 c0	a6 d1 d4 85 08 00 45 00	t`.....E.
0010	00 3b f8 ae 40 00 80 06	7e 9d c0 a8 01 12 c0 a8	;..@...~.....
0020	01 0e e3 2a 09 29 ee 1e	17 9c ce 07 d8 b9 50 18	...*).....P.
0030	00 ff ea 64 00 00 68 65	6c 6c 6f 20 72 65 6d 6f	...d...he llo remo
0040	74 65 20 75 73 65 72 0d	0a	te user..

Figure 3.15: Μορφή του πρώτου segment κειμένου, μήνυμα του local στον remote

0000	d8 c0 a6 d1 d4 85 20 0b	74 60 a9 a0 08 00 45 00t`....E.
0010	00 3a 11 ca 40 00 80 06	65 83 c0 a8 01 0e c0 a8	..:..@...e.....
0020	01 12 09 29 e3 2a ce 07	d8 b9 ee 1e 17 af 50 18	...*).....P.
0030	08 05 23 8e 00 00 68 65	6c 6c 6f 20 6c 6f 63 61	..#...he llo loca
0040	6c 20 75 73 65 72 0d 0a		l user..

Figure 3.16: Μορφή του δεύτερου segment κειμένου, μήνυμα του remote στον local

Σε περίπτωση που δεν οριστεί να τερματιστεί η σύνδεση με το κλείσιμο του app, στη μέθοδο `WindowClosing` της `App`, η TCP σύνδεση τερματίζεται απότομα, δηλαδή, στέλνονται RST flags με το κλείσιμο του app. Παρακάτω φαίνεται η αντίστοιχη εικόνα από το Wireshark για μια σύντομη chat συνομιλία με TCP μεταξύ του local και του remote όταν ο remote κλείσει το app χωρίς υλοποίηση της μεθόδου `WindowClosing`.

No.	Time	Source	Destination	Protocol	Length	Info
25	7.579469	192.168.1.18	192.168.1.14	TCP	66	58154 → 2345 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
27	7.868405	192.168.1.14	192.168.1.18	TCP	66	2345 → 58154 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
28	7.868567	192.168.1.18	192.168.1.14	TCP	54	58154 → 2345 [ACK] Seq=1 Ack=1 Win=65280 Len=0
81	19.578999	192.168.1.18	192.168.1.14	TCP	73	58154 → 2345 [PSH, ACK] Seq=1 Ack=1 Win=65280 Len=19
55	20.162931	192.168.1.14	192.168.1.18	TCP	60	2345 → 58154 [ACK] Seq=1 Ack=20 Win=525568 Len=0
251	7.702503	192.168.1.14	192.168.1.18	TCP	72	2345 → 58154 [PSH, ACK] Seq=1 Ack=20 Win=525568 Len=18
252	32.788627	192.168.1.18	192.168.1.14	TCP	54	58154 → 2345 [ACK] Seq=20 Ack=19 Win=65280 Len=0
275	43.995768	192.168.1.14	192.168.1.18	TCP	60	2345 → 58154 [RST, ACK] Seq=19 Ack=20 Win=0 Len=0

Figure 3.17: Ανταλλαγή segments με Abrupt connection termination

3.2 Stream πακέτων φωνής

Παρακάτω φαίνονται οι εικόνες από το textArea και από το Wireshark για μία κλήση VoIP μεταξύ του local και του remote.

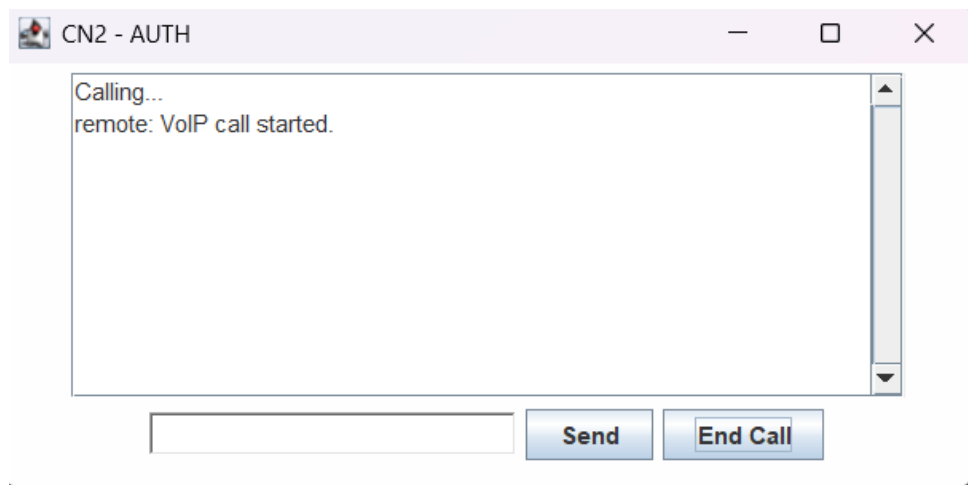


Figure 3.18: Συνομιλία στο textArea

Ο local πατάει το κουμπί "Call", ενημερώνει τον remote ότι καλεί και στέλνει πακέτα φωνής. Η κλήση ξεκινάει όταν ο remote αποδεχτεί την κλήση και στείλει πίσω "VoIP call started.". Τότε οι δύο user ανταλλάζουν πακέτα φωνής. Τα streams τέτοιων πακέτων φαίνονται παρακάτω.

No.	Time	Source	Destination	Protocol	Length	Info
44	10.434702	192.168.1.18	192.168.1.14	UDP	60	1234 → 1234 Len=18
45	10.894408	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
46	11.019287	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
47	11.019520	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
48	11.144258	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
49	11.144484	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
50	11.269708	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
51	11.270137	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
52	11.394682	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
53	11.395043	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
54	11.520450	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
55	11.520821	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
56	11.645421	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
57	11.645644	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
58	11.771058	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
59	11.771391	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
60	11.896700	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
61	11.896945	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
62	12.021951	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
63	12.022465	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024

Figure 3.19: Αρχή του stream πακέτων φωνής

No.	Time	Source	Destination	Protocol	Length	Info
59	11.771391	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
60	11.896700	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
61	11.896945	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
62	12.021951	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
63	12.022465	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
64	12.148072	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
65	12.148522	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
66	12.274729	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
67	12.275195	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
68	12.399811	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
69	12.400152	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
70	12.465030	192.168.1.14	192.168.1.18	UDP	60	1234 → 1234 Len=18
71	12.524870	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
72	12.525109	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
73	12.649919	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
74	12.650188	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
75	12.775096	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
76	12.775538	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
79	12.900620	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
80	12.900850	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
81	13.025675	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
82	13.026048	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
83	13.151642	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
84	13.151896	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
85	13.277138	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
86	13.277649	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024

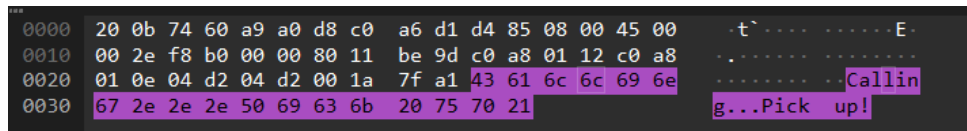
Figure 3.20: Έναρξη VoIP κλήσης

85	13.277138	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
86	13.277649	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
87	13.403601	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
88	13.404041	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
89	13.530065	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
90	13.655285	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
91	13.655651	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
92	13.780293	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
93	13.780693	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
94	13.906162	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
95	13.906400	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
96	13.953095	192.168.1.14	192.168.1.18	UDP	1066	1243 → 1243 Len=1024
97	13.953497	192.168.1.14	192.168.1.18	UDP	1066	1243 → 1243 Len=1024
98	14.032236	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
99	14.032485	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
100	14.074569	192.168.1.14	192.168.1.18	UDP	1066	1243 → 1243 Len=1024
101	14.157310	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
102	14.157529	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024
103	14.199967	192.168.1.14	192.168.1.18	UDP	1066	1243 → 1243 Len=1024
104	14.282707	192.168.1.18	192.168.1.14	UDP	1066	1243 → 1243 Len=1024

Figure 3.21: Stream πακέτων φωνής που ανταλλάσσονται

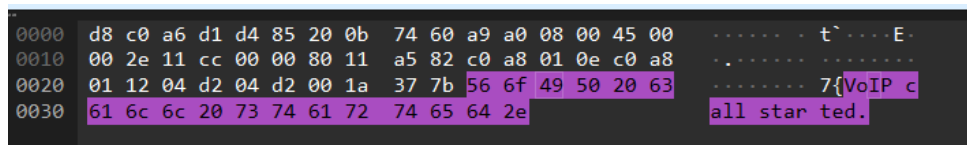
3.3 Πακέτα φωνής

Παρακάτω φαίνονται οι εικόνες της μορφής των πακέτων κειμένου για εγκαθίδρυση της κλήσης και των πακέτων φωνής από το Wireshark για μία κλήση VoIP μεταξύ του local (192.168.1.18) και του remote (192.168.1.14).



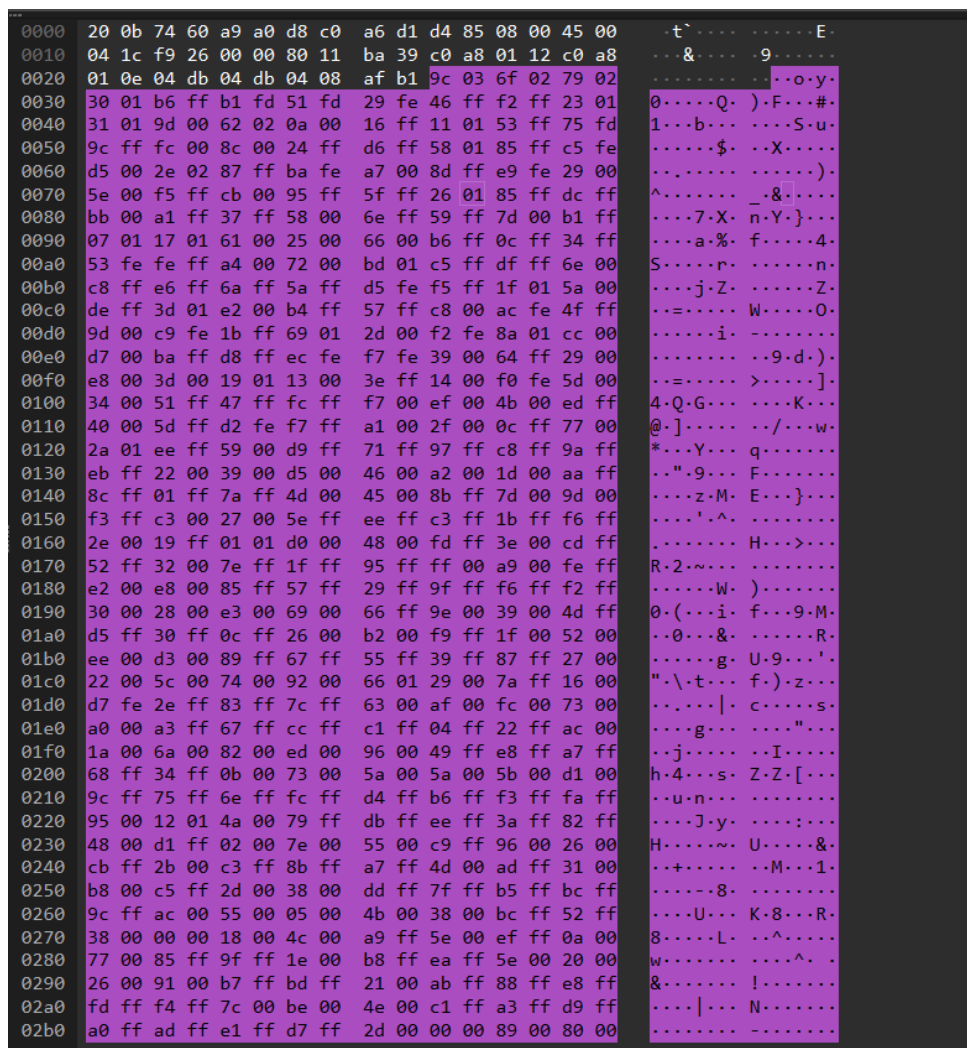
0000	20 0b 74 60 a9 a0 d8 c0 a6 d1 d4 85 08 00 45 00	.t`....E.
0010	00 2e f8 b0 00 00 80 11 be 9d c0 a8 01 12 c0 a8
0020	01 0e 04 d2 04 d2 00 1a 7f a1 43 61 6c 6c 69 6eCallin
0030	67 2e 2e 2e 50 69 63 6b 20 75 70 21	g...Pick up!

Figure 3.22: Μορφή του πακέτου που στέλνει ο local στον remote για αίτημα κλήσης



0000	d8 c0 a6 d1 d4 85 20 0b 74 60 a9 a0 08 00 45 00t`....E.
0010	01 0e 11 cc 00 00 80 11 a5 82 c0 a8 01 0e c0 a8
0020	01 12 04 d2 04 d2 00 1a 37 7b 56 6f 49 50 20 637{VoIP c
0030	61 6c 6c 20 73 74 61 72 74 65 64 2e	all star ted.

Figure 3.23: Μορφή του πακέτου που δηλώνει εκκίνησης της κλήσης



0000	20 0b 74 60 a9 a0 d8 c0 a6 d1 d4 85 08 00 45 00	.t`....E.
0010	04 1c f9 26 00 00 80 11 ba 39 c0 a8 01 12 c0 a8	...&....9.....
0020	01 0e 04 db 04 db 04 08 af b1 9c 03 6f 02 79 02-o-y.
0030	30 01 b6 ff b1 fd 51 fd 29 fe 46 ff f2 ff 23 01	0....Q.).F...#.
0040	31 01 9d 00 62 02 0a 00 16 ff 11 01 53 ff 75 fd	1...b...S.u.
0050	9c ff fc 00 8c 00 24 ff d6 ff 58 01 85 ff c5 fe\$. .X....
0060	d5 00 2e 02 87 ff ba fe a7 00 8d ff e9 fe 29 00).&....
0070	5e 00 f5 ff cb 00 95 ff 5f ff 26 01 85 ff dc ff	^....._&....
0080	bb 00 a1 ff 37 ff 58 00 6e ff 59 ff 7d 00 b1 ff7.X. n.Y.}...
0090	07 01 17 01 61 00 25 00 66 00 b6 ff 0c ff 34 ff	...a.% f....4.
00a0	53 fe fe ff a4 00 72 00 bd 01 c5 ff df ff 6e 00	S....r.n.
00b0	c8 ff e6 ff 6a ff 5a ff d5 fe f5 ff 1f 01 5a 00j.Z.Z.
00c0	de ff 3d 01 e2 00 b4 ff 57 ff c8 00 ac fe 4f ff	==.... W....0.
00d0	9d 00 c9 fe 1b ff 69 01 2d 00 f2 fe 8a 01 cc 00i.
00e0	d7 00 ba ff d8 ff ec fe f7 fe 39 00 64 ff 29 009.d.)
00f0	e8 00 3d 00 19 01 13 00 3e ff 14 00 f0 fe 5d 00	==....>....]
0100	34 00 51 ff 47 ff fc ff f7 00 ef 00 4b 00 ed ff	4.Q.G...K...
0110	40 00 5d ff d2 fe f7 ff a1 00 2f 00 0c ff 77 00	@.]....../.w.
0120	2a 01 ee ff 59 00 d9 ff 71 ff 97 ff c8 ff 9a ff	*...Y... q.....
0130	eb ff 22 00 39 00 d5 00 46 00 a2 00 1d 00 aa ff	..".9... F.....
0140	8c ff 01 ff 7a ff 4d 00 45 00 8b ff 7d 00 9d 00Z.M. E....}
0150	f3 ff c3 00 27 00 5e ff ee ff c3 ff 1b ff f6 ff'.^.....
0160	2e 00 19 ff 01 01 d0 00 48 00 fd ff 3e 00 cd ffH...>...
0170	52 ff 32 00 7e ff 1f ff 95 ff ff 00 a9 00 fe ff	R.2~.....
0180	e2 00 e8 00 85 ff 57 ff 29 ff 9f ff f6 ff f2 ffW.).....
0190	30 00 28 00 e3 00 69 00 66 ff 9e 00 39 00 4d ff	0.(...i. f...9.M.
01a0	d5 ff 30 ff 0c ff 26 00 b2 00 f9 ff 1f 00 52 00	..0...&.....R.
01b0	ee 00 d3 00 89 ff 67 ff 55 ff 39 ff 87 ff 27 00g. U.9...'
01c0	22 00 5c 00 74 00 92 00 66 01 29 00 7a ff 16 00	".\t... f.)z...
01d0	d7 fe 2e ff 83 ff 7c ff 63 00 af 00 fc 00 73 00 c.....s
01e0	a0 00 a3 ff 67 ff cc ff c1 ff 04 ff 22 ff ac 00	...g... .."
01f0	1a 00 6a 00 82 00 ed 00 96 00 49 ff e8 ff a7 ff	..j.....I.....
0200	68 ff 34 ff 0b 00 73 00 5a 00 5a 00 5b 00 d1 00	h.4...s. Z.Z.[...
0210	9c ff 75 ff 6e ff fc ff d4 ff b6 ff f3 ff fa ff	...u.n.....
0220	95 00 12 01 4a 00 79 ff db ff ee ff 3a ff 82 ff	...J-y.;
0230	48 00 d1 ff 02 00 7e 00 55 00 c9 ff 96 00 26 00	H.....~ U....&
0240	cb ff 2b 00 c3 ff 8b ff a7 ff 4d 00 ad ff 31 00	..+.....M...1.
0250	b8 00 c5 ff 2d 00 38 00 dd ff 7f ff b5 ff bc ff8.....
0260	9c ff ac 00 55 00 05 00 4b 00 38 00 bc ff 52 ff	...U... K.8...R.
0270	38 00 00 00 18 00 4c 00 a9 ff 5e 00 ef ff 0a 00	8....L. ..^.....
0280	77 00 85 ff 9f ff 1e 00 b8 ff ea ff 5e 00 20 00	w..... ..^.....
0290	26 00 91 00 b7 ff bd ff 21 00 ab ff 88 ff e8 ff	&.....!.....
02a0	fd ff f4 ff 7c 00 be 00 4e 00 c1 ff a3 ff d9 ff N.....
02b0	a0 ff ad ff e1 ff d7 ff 2d 00 00 00 89 00 80 00

Figure 3.24: Παράδειγμα πακέτου φωνής

0000	d8 c0 a6 d1 d4 85 20 0b	74 60 a9 a0 08 00 45 00 t`....E.
0010	04 1c 11 d3 00 00 80 11	a1 8d c0 a8 01 0e c0 a8
0020	01 12 04 db 04 db 04 08	05 6d 4e 1e ee df fb 1fmN.....
0030	f6 df 10 39 06 e7 06 06	fb d3 c7 04 f8 dc f6 18	...9....
0040	25 0e 10 20 34 21 fd 19	34 24 fa 0f 30 13 e6 fc	%.. 4!.. 4\$.~0..
0050	0c ec d8 e4 f6 e3 d5 e3	e7 d2 cb cc d1 d0 c6 cb
0060	ef 34 0a e9 45 7e 35 d8	37 4c f3 da e6 1e 15 db	~4..E~5. 7L.....
0070	fd 3d 27 ef f9 11 01 de	bc ea 07 d2 dc 15 29 0d	~='.....)
0080	06 22 36 28 fe 0c 35 2f	04 0d 2d 26 01 f7 0d f7	~"6(..5/ ...~8....
0090	da d9 e7 e2 ce d8 e7 d5	ca d3 d4 d1 c3 c3 f3 2d
00a0	fc e4 49 7e 20 dd 3f 50	f0 ca fa 2e 00 d1 09 42	..I~ .?PB
00b0	17 e0 fa 1a 00 cc c5 ff	03 cf e1 20 25 02 05 2c %.,
00c0	3b 13 f7 22 3c 11 f5 21	36 0e f5 13 17 ed db e6	;~"<..! 6.....
00d0	ec cb c3 dc d8 c2 c7 dd	df cf c5 ef 34 f9 df 3a4...
00e0	76 2f d5 3e 52 fc df ed	2f 15 db 02 3e 1f e4 02	v/.>R... /...>...
00f0	0c f5 dc bf f0 02 d2 e2	14 22 07 0a 28 2e 15 01"~(...
0100	1e 28 0e 01 23 28 0c 09	1c 0e f2 ec f3 ea ca ca	~(.~#(.. ..
0110	d5 cd b9 bf d6 c9 be e2	29 fa e4 3d 69 26 db 3a)=i&..
0120	46 e8 e4 02 27 13 ef 1b	44 1d e9 09 16 eb d5 d2	F...'. D.....
0130	03 f4 ce f9 20 0a f4 1d	2c 15 f9 0f 28 10 ea 04 ,...~(...
0140	35 0e f6 11 29 0e e2 f5	05 e3 c3 cb e0 cf ba cf	5....).....
0150	e1 c7 e9 28 f1 e9 3c 5b	09 de 40 2b e0 e0 03 1e	...~<[..@+....
0160	00 f4 24 41 14 f9 17 10	eb d8 e0 fb fb e7 0a 2b	..\$A.... ..+...
0170	20 16 1d 25 11 f9 07 08	f5 f4 00 10 04 f7 07 0b	..%.... ..
0180	f5 ea eb e9 d2 c6 d0 d5	cb cb d7 f1 30 01 e6 4a~0..J
0190	69 0b c9 41 3a c5 d6 09	26 fc e3 24 4c 0c de 17	i..A:.. &..\$L...
01a0	15 e7 cc d7 11 02 dc 07	45 29 06 25 33 1c f9 00 E).%3...
01b0	19 02 ea 08 11 fb f2 fc	06 e9 d6 e6 e0 c3 bc cf
01c0	ce c8 b4 fd 42 dd f3 66	72 f5 da 62 1e c2 db 08B..f r..b....
01d0	2a f5 ec 3a 50 02 ee 24	0b d9 c6 e7 15 e9 d6 29	*...:P..\$
01e0	40 0b 03 29 2f fd e8 18	19 ed fb 1e 15 f7 f5 12	@..)/... ..
01f0	03 d3 d3 e4 d2 ba bf d3	cd c4 0d 16 da 16 5e 38^8
0200	d5 19 52 f0 de f4 22 1b	e5 13 47 2a f3 0e 29 f8	..R...". ..G*..).
0210	e1 e0 05 0a da ff 2f 14	fd 10 24 10 e5 fa 17 f5/. ..\$.
0220	eb 03 12 08 fa 04 0e f4	db e2 d6 cc c2 c0 c8 fc
0230	1c e0 01 56 59 ea dc 49	04 bf d2 fe 1c f0 fe 41	...VY..IA
0240	4a 0e 0e 31 0b f0 e3 f5	12 f7 fd 2a 2d 18 16 1b	J..1.... ..*....
0250	0f eb e5 f3 ee de e4 02	02 f3 f4 02 ed d1 d6 da
0260	cc bf c8 ee 36 03 e5 57	7c 13 ce 46 3f b8 c9 f9	...~6..~W ..F?...
0270	11 ed d5 1a 46 0b e0 27	21 e1 e6 f1 0c 06 e7 18	...~F... ' !.....
0280	41 1c 0d 2a 34 09 ec 0e	0c e5 e2 01 02 e4 e7 f6	A...*4... ..
0290	e9 ca c5 d1 c8 bc ba d7	20 08 db 3c 7f 31 d6 3c ~<.1.<
02a0	64 d7 cd 0d 25 ff e2 21	49 14 ee 27 1b e6 e7 ec	d...%..! I...'.
02b0	fd f7 e3 04 24 0c 02 1c	25 09 e9 0b 14 e5 e6 07	...~\$.~ %.....

Figure 3.25: Παράδειγμα πακέτου φωνής

4 Limitations

Το παρόν πρόγραμμα έχει εκπαιδευτικό σκοπό και δεν στοχεύει να αντικαταστήσει άλλα υπάρχοντα messengers. Φτιάχτηκε για να κατανοήσουμε καλύτερα την p2p επικοινωνία με την χρήση UDP και TCP πρωτοκόλλων για την αποστολή και λήψη κρυπτογραφημένων και μη μηνυμάτων κειμένου και ήχου.

Παρότι αντιμετωπίσαμε μερικά edge cases, παραμένουν διάφορα ακόμα. Τα tags-εντολές που χρησιμοποιούνται από το UDP, μπορεί να μπερδέψουν την εφαρμογή (πχ. αν ένα άτομο στέλνει το μήνυμα "[Part]01: Αρχή της ταινίας", η εφαρμογή θα νομίζει λανθασμένα πως στάλθηκε chunk. Επίσης το tcp chat δεν μπορεί να τρέξει ως server στα NixOS (unstable, 12/2024), καθώς μάλλον μπλοκάρονται κάποιες θύρες που χρησιμοποιούνται. Ακόμη, λόγω της χρήσης UDP που κανονικά δεν θα έπρεπε να χρησιμοποιείται για αποστολή μηνυμάτων κειμένου, άμα ένα κρυπτογραφημένο πακέτο έρθει σε λάθος σειρά (εξαιρέση τα μηνύματα που περιέχουν chunks), όλο το σύστημα απο/κρυπτογράφησης θα αποτύχει μέχρι να ξανανοίξει η εφαρμογή. Ένα άλλο θέμα είναι πως η εφαρμογή λειτουργεί μόνο για αγγλικούς χαρακτήρες. Επίσης, καθώς χρησιμοποιούνται οι default χαρακτήρες, το μέγιστο πιθανό UDP πακέτο μηνύματος κειμένου υπολογίζεται να είναι 1048 bytes, μεγαλύτερο και από τα buffers στην UDPchat και από το μέγεθος που συνήθως επιτρέπουν οι dns (500bytes). Συγκεκριμένα, θα είναι τύπος chunk, αποτελούμενο από 16 χαρακτήρες το IV, 8 το tag "[Part]00" και 500 το υπόλοιπο μήνυμα. Αν πιάνουν 2 bytes ο κάθε χαρακτήρας, τότε συνολικά είναι 1048 (συν το μικρό header) Όμως πολλοί χαρακτήρες πιάνουν 1 byte το οποίο κάνει εφικτή την επικοινωνία έτσι πως έχουμε ορίσει την εφαρμογή. Αν υπάρχει θέμα, θα μπορούσε να μειωθεί και το μέγιστο μέγεθος του μηνύματος που στέλνεται ολόκληρο και το chunkSize στους 200 χαρακτήρες.

Πέρα από τα edge cases, λείπουν και διάφορα σημαντικά features, όπως η ασύμμετρη απο/κρυπτογράφηση για την αυθεντικοποίηση και αρχικοποίηση επικοινωνίας μεταξύ των peers, δεν υποστηρίζονται πάνω από 2 peers, δεν διατηρούνται τα μηνύματα μόλις κλείσει η εφαρμογή κλπ.

Τέλος περιέχει διάφορα κενά ασφαλείας (και αξιοπιστίας), όπως το ότι τα μεταδεδομένα δεν έχουν κάποια ιδιαίτερη αντιμετώπιση για την προστασία των χρηστών, το tcp δεν χρησιμοποιεί κρυπτογράφηση, πιθανόν δεν γίνεται η καλύτεροι έλεγχοι για προστασία από χακάρισμα και δεν γίνεται χρήση του double ratchet αλγορίθμου που είναι το standard κρυπτογράφησης μηνυμάτων σε εφαρμογές messenger (το οποίο παρέχει features όπως confidentiality, integrity, authentication, participant consistency, destination validation, forward

secrecy, backward secrecy (future secrecy), causality preservation, message unlinkability, message repudiation, participation repudiation, and asynchronicity).

Συνοψίζοντας, όποιο άτομο θέλει να χαλάσει την λειτουργία της εφαρμογής ή να την χακάρει, μπορεί σχετικά εύκολα. Στόχος είναι μέσω της εφαρμογής να μάθουμε καλύτερα την encrypted p2p ανταλλαγή κειμένου με UDP και TCP και ήχου με UDP.