**EASTERN MEDITERRANEAN UNIVERSITY**

****

**Software Engineering Department**

**CMSE491 – “Selected Topics in Software Engineering – I”**

**Fall 2020-2021 Semester**

**Famagusta, Northern Cyprus**

**Chatting System with NTRU Encryption**

**Team Members:**

**Ahmad Turjman**

**Ahmad Abdulbasit Abutahoon**

**Instructor:**

**Assoc. Prof. Dr. Alexander Chefranov**

**Lab Assistant:**

**Asst. Abdolrahman Farshgar**

**Outline**

1. Problem definition 3

2. Description of chat system (CS) 3

2.1 Structure of the CS 3

2.2 Implementation of CS without encryption 4

2.3 Testing of CS without encryption 5

3. Description of NTRU encryption in CS 7

3.1 Description of NTRU 7

3.2 Implementation of NTRU as standalone application 10

3.3 Testing of NTRU as standalone application 11

4. Integration of CS and NTRU 13

4.1 Design of interface CS-NTRU 14

4.2 Implementation of CS-NTRU system 15

4.3 Testing of CS-NTRU system 16

5. Conclusion 17

6. References 17

7. Appendices with the code developed and raw results obtained 18

1. **Problem Definition**

In this project, we will implement a Chatting System that allows multiple users to communicate via internet channels. Creating a chatting system is half of the way of getting it fit between the technologies around us. Encryption is one of the most important pillars in any communication system. To complete our project, we implemented NTRU for Integers encryption stand-alone application that allows the user to encrypt any kind of message, and decrypts its cipher text.

We implemented NTRU library that can be integrated with our chatting system, which allows users using the chatting system anonymous and safe over the internet.

We used python as a programming language since it is flexible and it allows us to get what we really wanted in our applications.

1. **Description of chat system (CS)**

Our chatting system consists of two main applications, server and client. Since any kind of communication program or web-based application connects clients with each other, we used socket programming to get that achieved. It helps us connect any end to end with a server in the middle.

* 1. **Structure of the CS**

Our server runs on a local network that allows not 2 clients, but 100s of clients, on the same network, connect and send messages to each other. We can call it a chatting room.

A server must be running and listening to any connections, in other words, it’s ready to receive any connection from any client and connects them with each other.

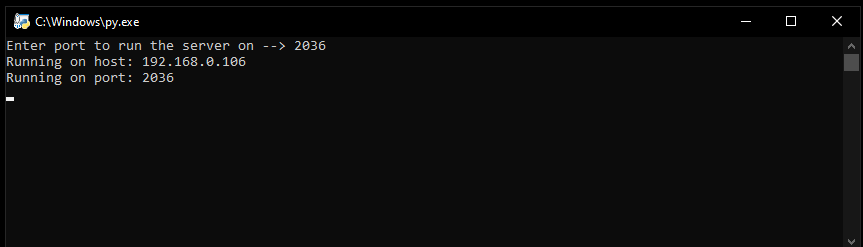
A client on the other hand can run on any computer on the network, and connect to the running server using an IP address / Host name and a specified port of the server.

In the process of sending a message from one client to another, the server receives any type of calls and broadcasts it to all the other clients. For example, let’s say *Client B* joined our chatting room, while *Client A* was already connected to it before. *Client B* sends a message to *Client A* asking “How big is this room?”. This message, which is a type of string, is encoded and converted to Bytes and sent to the server, then the server sends this message *Client A*, then its decoded back into its original type and previewed on the client screen.

* 1. **Implementation of CS without encryption**

Running our chatting system without any encryption is simple.

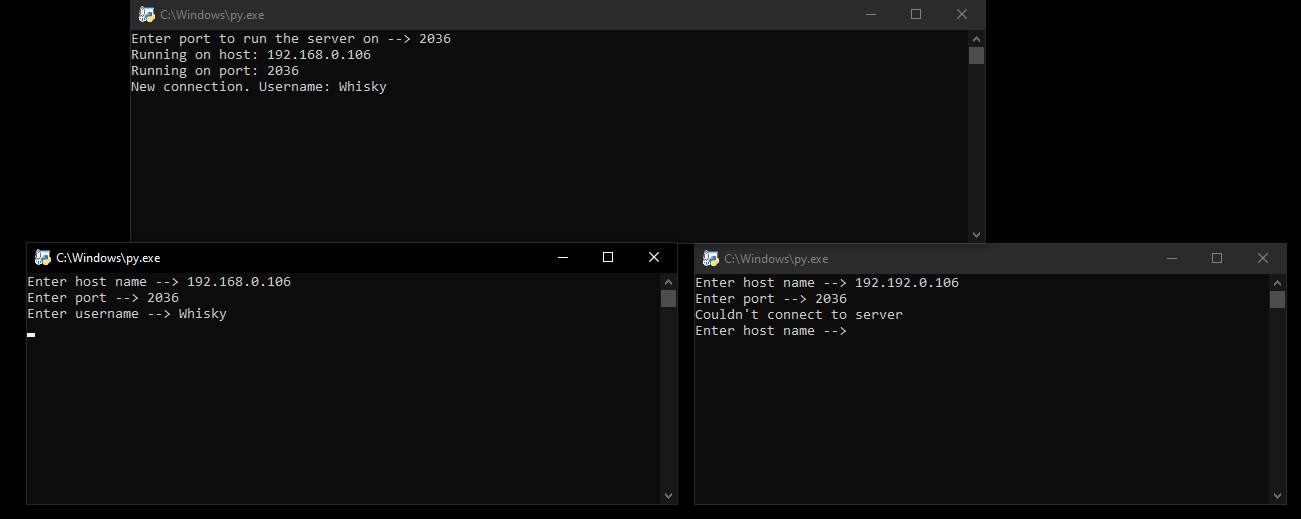
We first run our server and set a port to run on:



It then says that its running on *192.168.0.106*, which refers to the computer IPv4 address its running on.

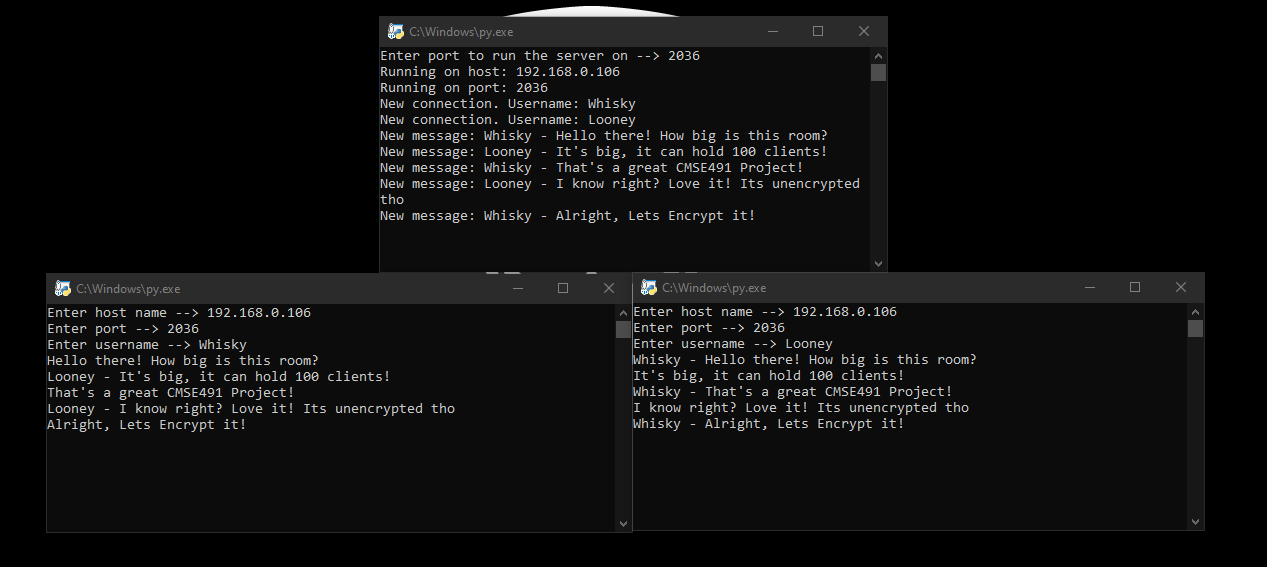
We then run the client application; we will have 2 instances in this report.

It asks to enter the host name and the port it’s going to connect to, if the host name and port were correct, it would take our input for joining the room with a username, if it was wrong it prints out an error. The picture below shows the illustration:



On the right-bottom corner, a wrong host name was inserted, it printed out *“Couldn’t connect to server”*. So, the user is asked to type in the host name again.

After connecting both of the clients successfully, they can communicate, and each time someone send a message to another client, the server prints it out to show us it’s been successfully sent.

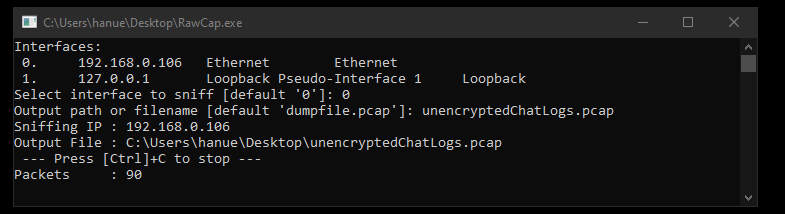


* 1. **Testing of CS without encryption**

For testing our chatting system, we used “*RawCap*” software that can sniff our traffic on the network.

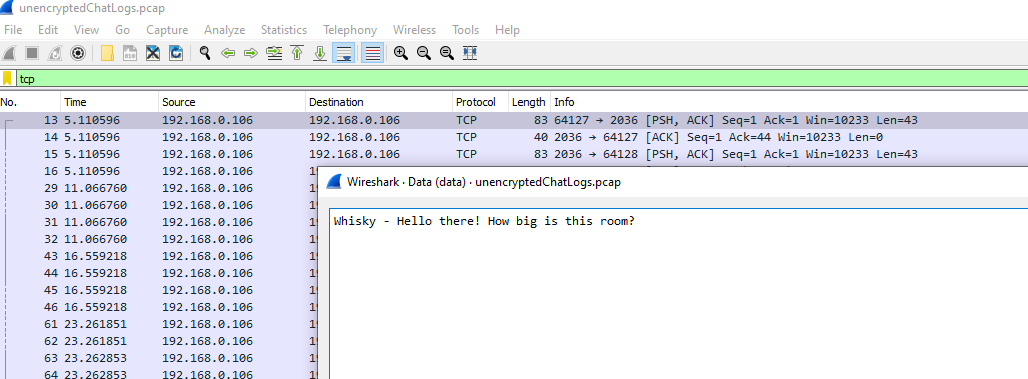
This software behaves as a *Man-In-The-Middle* attack, but works only for our system. It outputs a file, that allows us to obtain the data from the packets that were sent and received on our system, including the messages. This file can be opened with *“Wireshark”*.

After we ran *“RawCap”,* it sniffed our traffic and packets

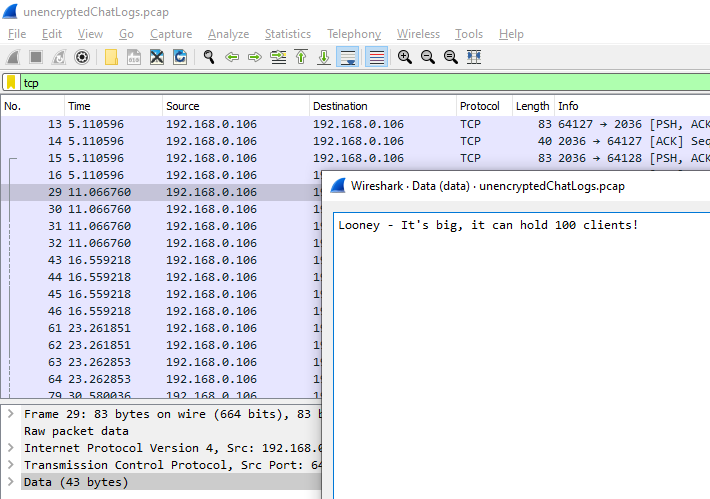


We named the output file, that was obtained using *“RawCap”,* *“encryptedChatLogs.pcap”.*

After looking into it using *“Wireshark”*, We can see that the messages were received as a plain text.



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



So, any intruder is able to intercept the messages that were sent and received without encryption.

1. **Description of NTRU encryption in CS**

NTRU encryption can be used to encrypt plain-text messages that are sent from one client to another.

In our chatting system, NTRU was made only for the client application, which means that when a client sends a message to another client, this message gets encrypted before it reaches the server, then it will send cipher text to the server, then the server will broadcast the cipher text, received from the first client, the other client receives the cipher text and decrypts it. This way, the hacker wouldn’t be able to get any information except cipher messages, and he won’t be able to decrypt the cipher text unless he has our private keys.

* 1. **Description of NTRU**

We created NTRU for Integers library in python, and it can be used by any type of application that is made by python. In this NTRU library we specified the given parameters, and are not going to change:

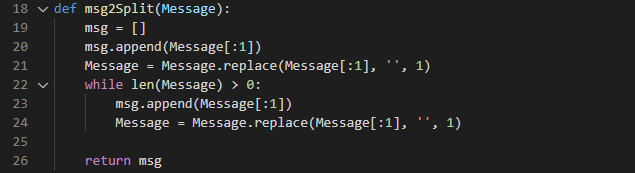


Those parameters aren’t changing since they will always cover any ASCII Value in plain-text message.

We came up with a plan that allows us to encrypt the plain-text message with some functions that are written inside the NTRU library

First, the message that the user inputs, goes through our *msg2Split()* method.

This method allows us to split the plain text into characters and save the characters into an array, for example: The string *“Hello”* becomes *[‘H’,’E’,’L’,’L’,’O’]*



Then the array obtained after splitting the string, goes through *char2Ascii()* method, which takes each character and convert it into ACSII Value and return an array containing the ASCII Value of each character converted before.



Then this new array of ASCII Values goes through *encryption()* method, which encrypts each element of the array that contains ASCII Values and return an array with encrypted ASCII elements.



After encryption, we will obtain a cipher text for each ASCII element in the array, we used this line of code to concatenate those elements inside the array into a whole string that can be shown as below

            print(''.join([str(n) for n in encryptedArr]))

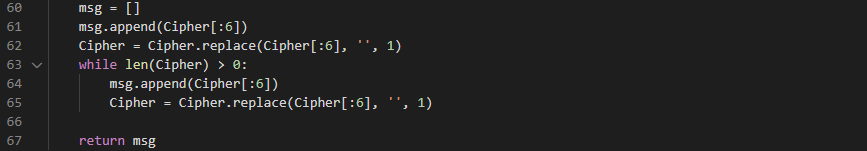
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

For the decryption part, we reversed the process. We first pass the cipher text to a method called *decryption()* that calls *cipherSplitter()* method that splits the cipher text into 6 digits, since our q is 6 digits, and then return an array containing elements that are 6 digits each, then the *decryption()* method decrypts each element back into the original ASCII Value. Then, it returns it as an array with ASCII elements.

*decryption()* method:



*cipherSplitter()* method:



Then we again used the line of code that concatenate elements into a single string.

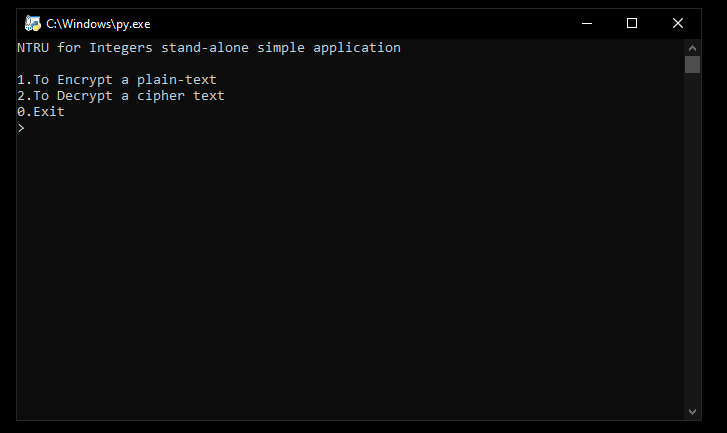
            print(''.join(chr(i) for i in decryptedArr))

Lastly, we obtain our decrypted message.

* 1. **Implementation of NTRU as standalone application**

We created a stand-alone application that demonstrates and proves that NTRU for Integers encryption works.

Running the application prints out a menu for the user to interact with.

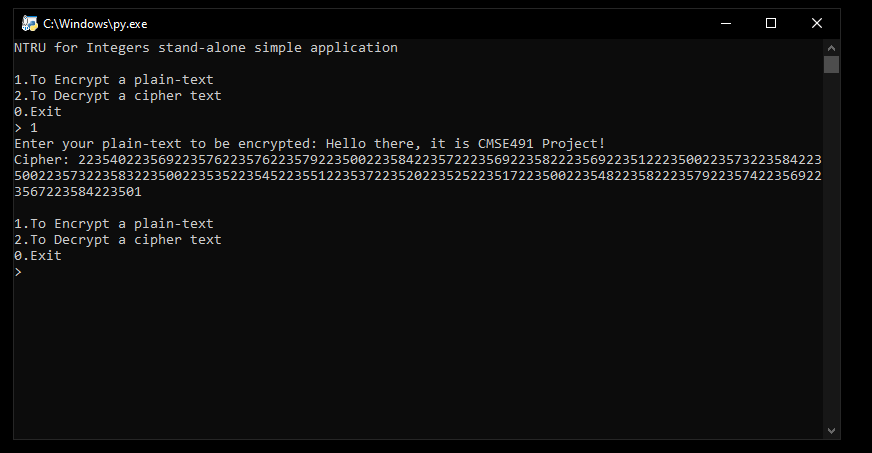


Entering *‘1’* allows the user to encrypt any message he inputs, and it returns the cipher text of it.

Entering *‘2’* allows the user to decrypt any cipher text he inputs, and it returns the plain-text form of it.

* 1. **Testing of NTRU as standalone application**

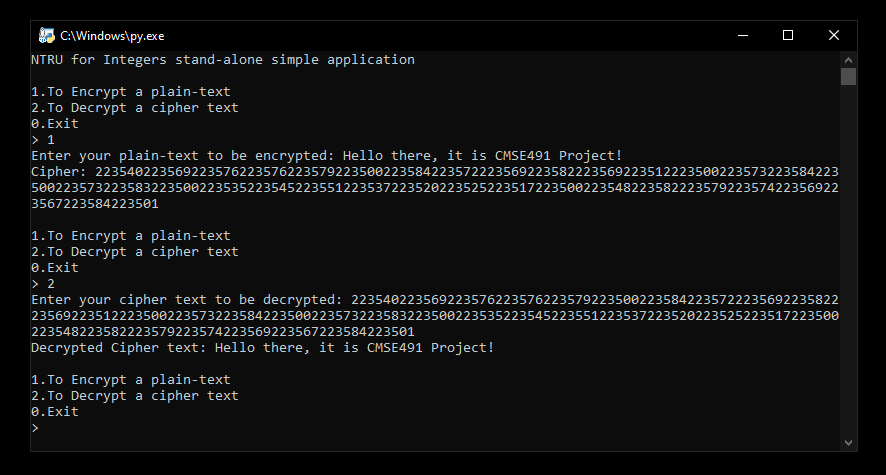
Entering *‘1’* it asks the user to enter a text to be encrypted



The plain-text message was: “*Hello there, it is CMSE491 Project!”* after the message was sent to our NTRU library and it applied the methods inside it, it returns the cipher text:

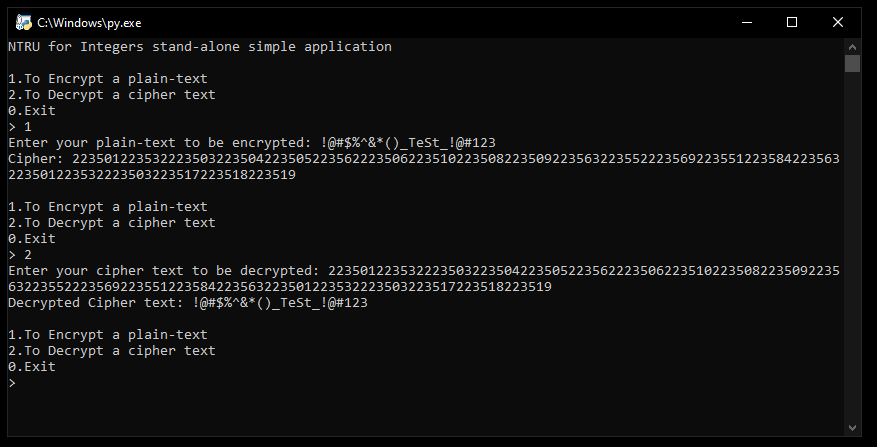
*223540223569223576223576223579223500223584223572223569223582223569223512223500223573223584223500223573223583223500223535223545223551223537223520223525223517223500223548223582223579223574223569223567223584223501*

Then the user enters *‘2’* to decrypt the cipher text obtained from the first step



The cipher text was successfully decrypted.

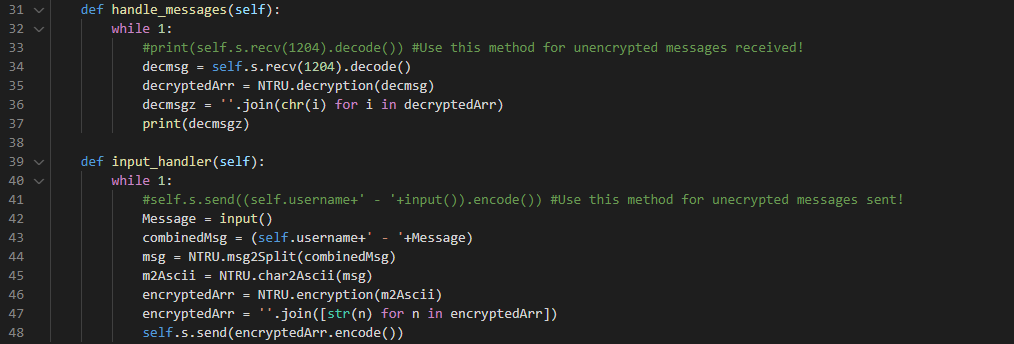
We tested other special characters and out stand-alone application successfully worked.



1. **Integration of CS and NTRU**

We imported the NTRU Library in our client code, and also to our server code to check if a hacker can intercept the decrypted messages broadcasted from the server. The hacker could. So, we used encryption and decryption on the client application only.

We added some method calls in client source code



*handle\_messages()* method is for the messages received from the server. Since sockets doesn’t work with data types we used in our programming languages, it first encodes them to Byte type then, it decodes them from Bytes to their original type. In this method it only decodes the received Bytes from there server.

Line 34, creates a variable that stores the decoded message received from the server.

Line 35, creates a variable that stores the returned array from *decryption()* method in NTRU.

Line 36, It concatenates the elements in the array obtained from the line before it.

Line 37, prints out the decrypted cipher text.

*input\_handler()* method, handles the messages that are going to be sent to the server, it encodes them and convert the data type to Byte, then it encrypted it and send it to the server.

Line 43, combines the username that is broadcasted to the server with the message, to make it unknown, which client is sending messages to the server, and accomplish anonymity.

Line 44, creates a variable that stores the returned array from *msg2Split()* method that splits the string into characters.

Line 45, creates a variable that stores the returned array of the method that converts each element into ASCII Value.

Line 46, creates a variable that stores the returned array after the encryption of each ASCII element.

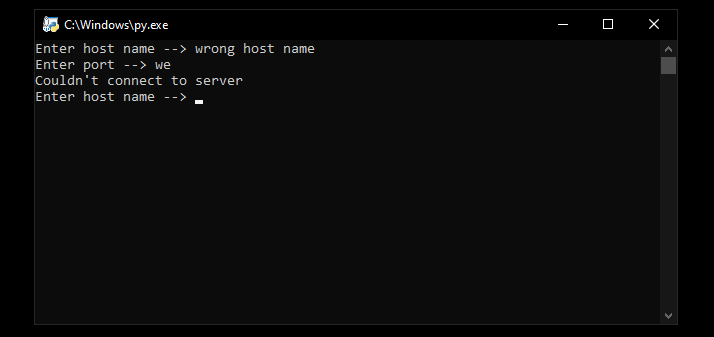
Line 47, concatenates the returned array from the *encryption()* method and makes it look like a whole string of numbers as a cipher text.

Line 48, encodes the ciphertext and converts it to Bytes, then sends it to the server.

* 1. **Design of Interface CS-NTRU**

Our CS-NTRU application runs on the command line without any UI, but it has some print outs that guides the user to connect the server using a host name and a port.

It returns error if there was connection issues and it also allow him to reenter the host name and port again, without closing the client application.

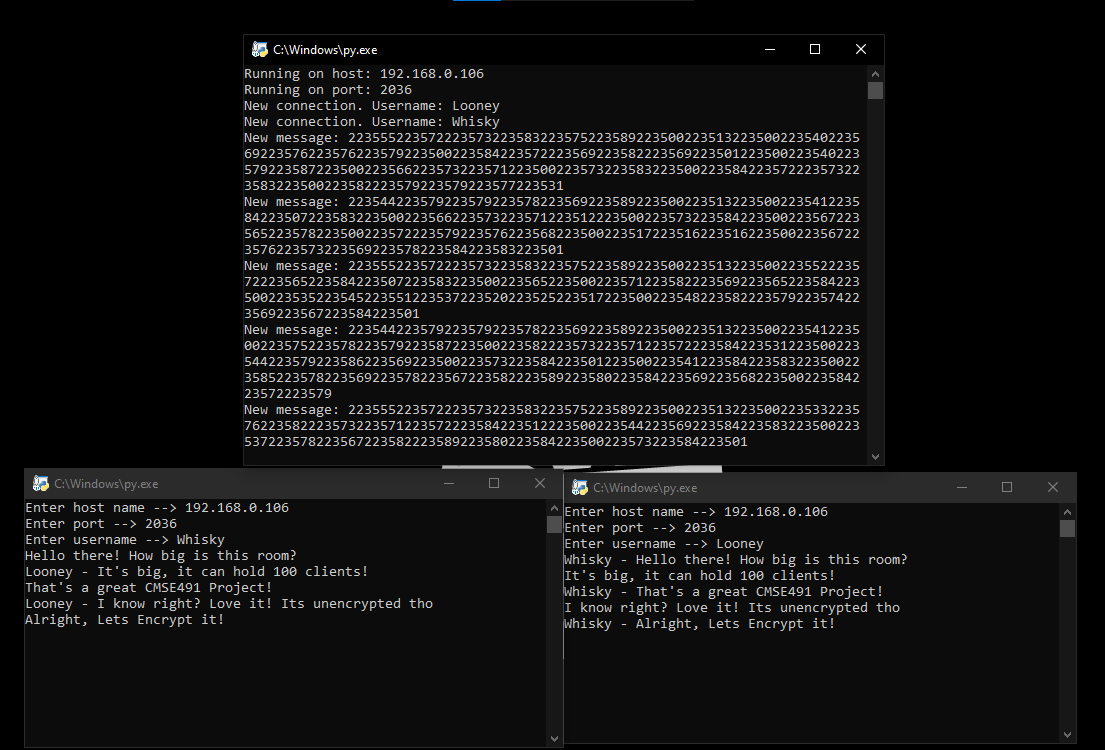


* 1. **Implementation of CS-NTRU system**

Implementing CS integrated with NTRU for Integers encryption, is basically the same thing the client going to see, the same message he sent to the other client is going to be as the same plain text.

But what’s happening behind the scenes, the message is getting encrypted and sent to the server, the server broadcasts the cipher text to the other clients and then the back end of the client application decrypts it and prints it as a normal message.

The server can’t know who are sending a message, and what is the content of the message.

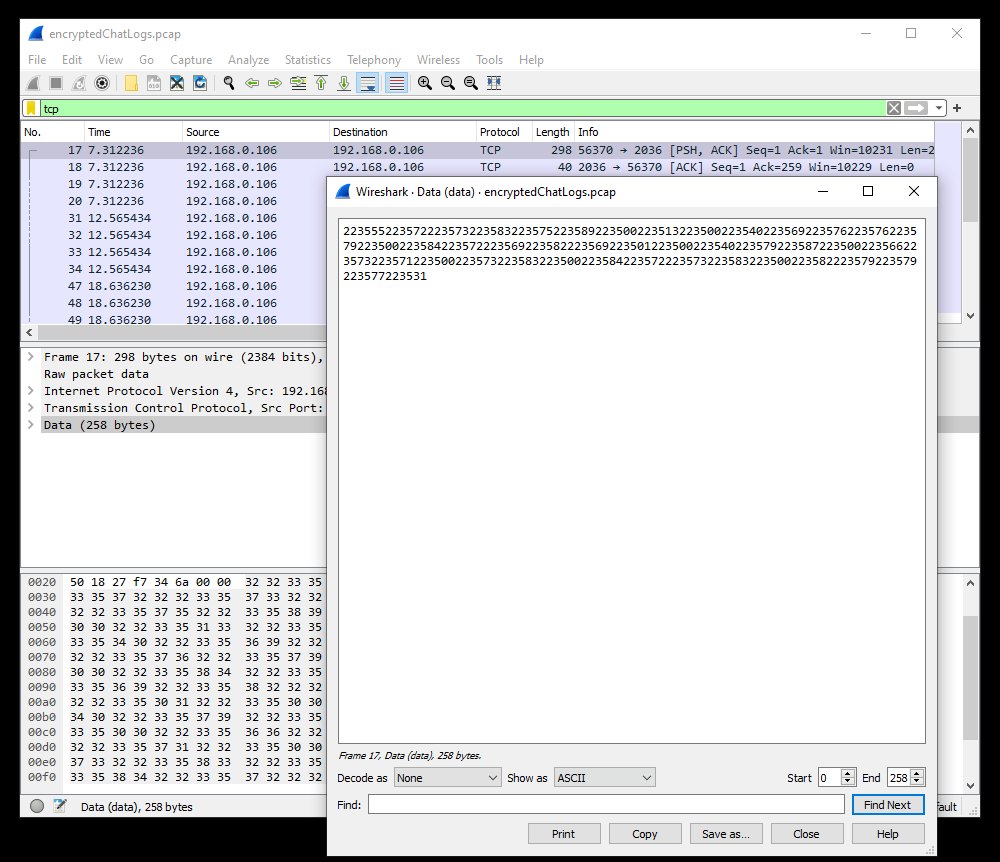


* 1. **Testing of CS-NTRU system**

We tested our encryption using *“RawCap”* and we sniffed the traffic and packets that is going through our clients and server, coming out with an output that shows us that the messages are successfully encrypted.

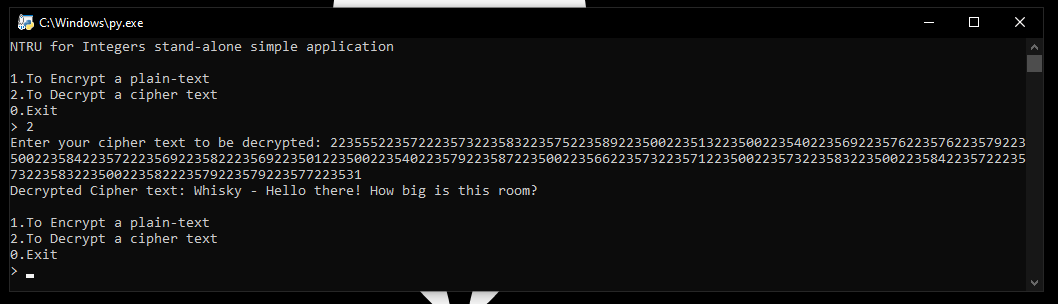
Hacker would fail to intercept the real messages that are going between the clients.

Here is an illustration of “Wireshark” file that we obtained from *“RawCap”*



As shown above, this is what the hacker can see when he uses a *MITM* attack.

Now we’ll decrypt this cipher text using our stand-alone NTRU application



The decrypted cipher text is the message client *“Whisky”* sent, *“Hello there! How big is this room?”*

1. **Conclusion**

We understood how NTRU encryption works and we fully implemented it on our computers using python language. We finally applied NTRU for Integers encryption and we achieved a good level of security for our private messages. We also tested our results using *“Wireshark”* and *“RawCap”.* NTRU is a very fast encryption compared to the other well-known ones.

1. **References**

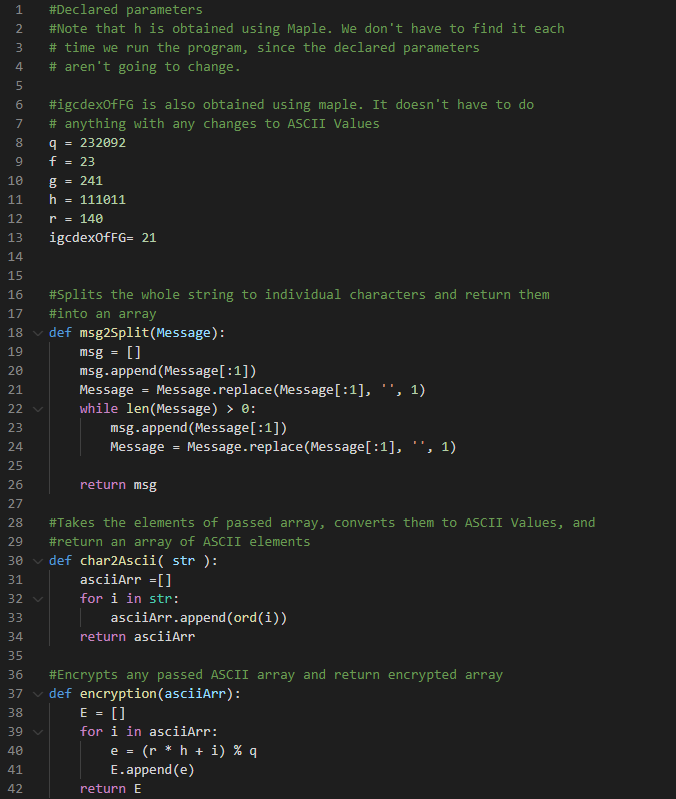
<https://github.com/ifarshgar/Encryption-Techniques>

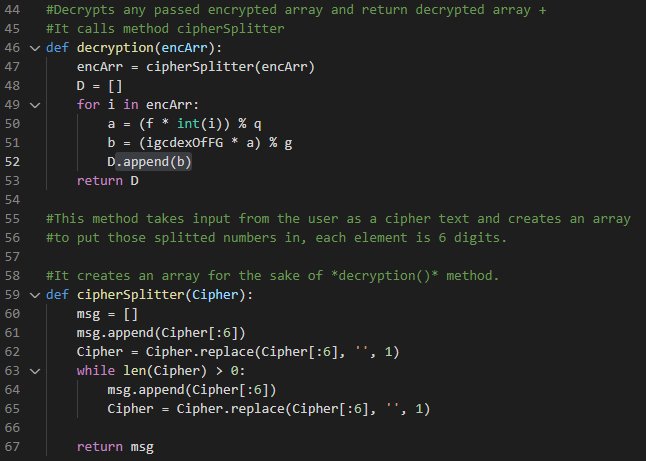
<https://github.com/TomPrograms/Python-Internet-Chat-Room>

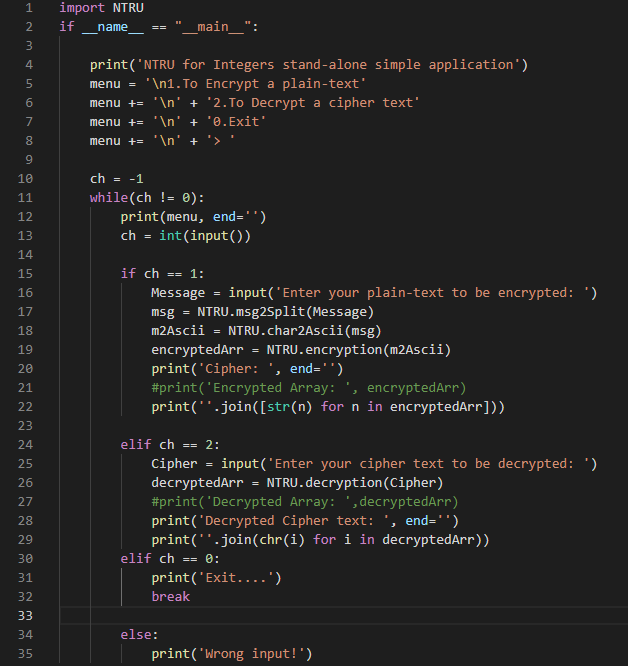
<https://staff.emu.edu.tr/alexanderchefranov/Documents/CMSE491/Fall2019/Hoffstein2015%20Introduction%20to%20Mathematical%20Cryptography373-376.pdf>

<https://staff.emu.edu.tr/alexanderchefranov/Documents/CMSE491/CMSE491%20Fall2020/CMSE491%20NTRU%20Int%2007102019.mw>

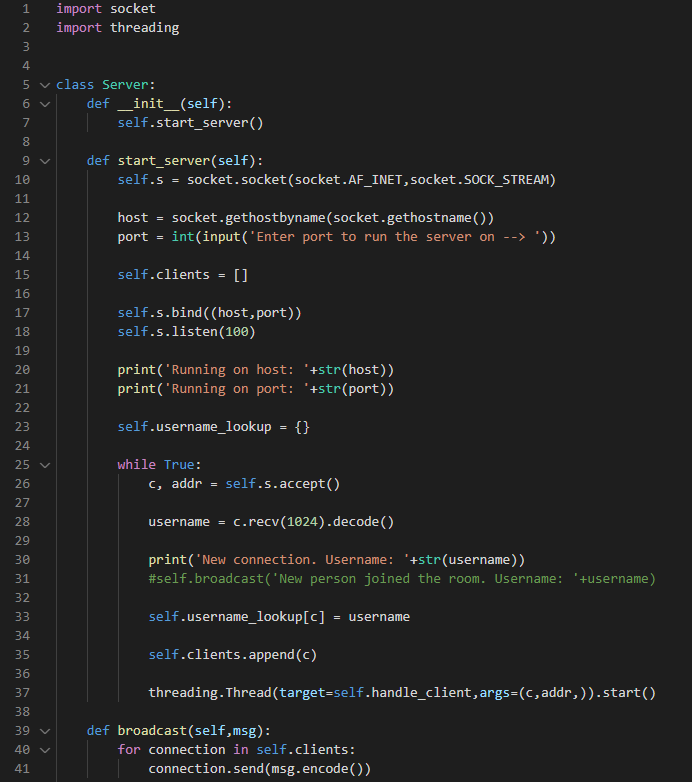
1. **Appendices with the code developed and raw results obtained**

**NTRU.py**

****

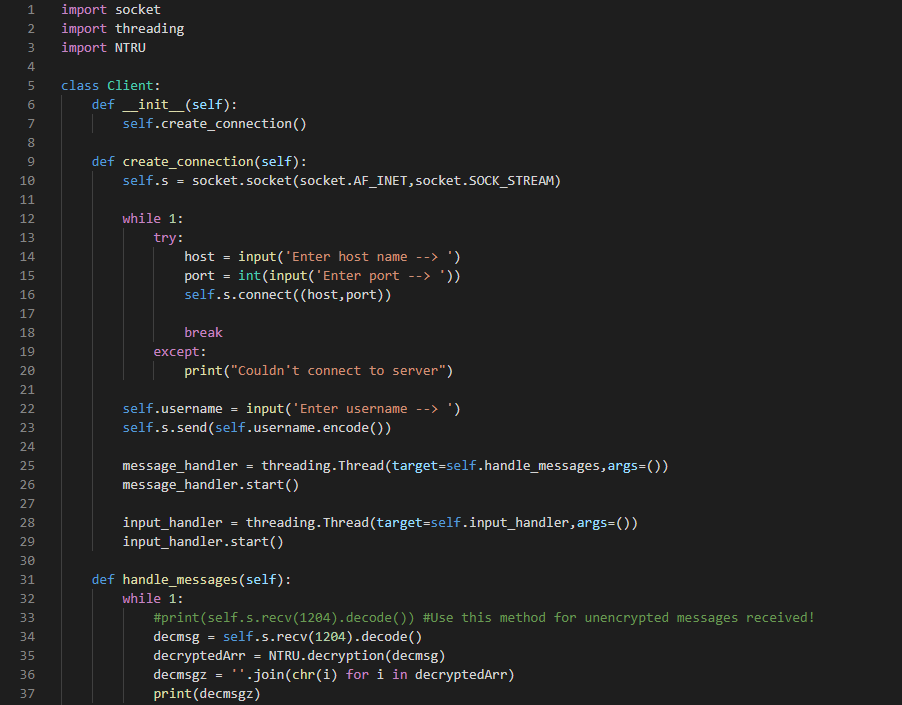
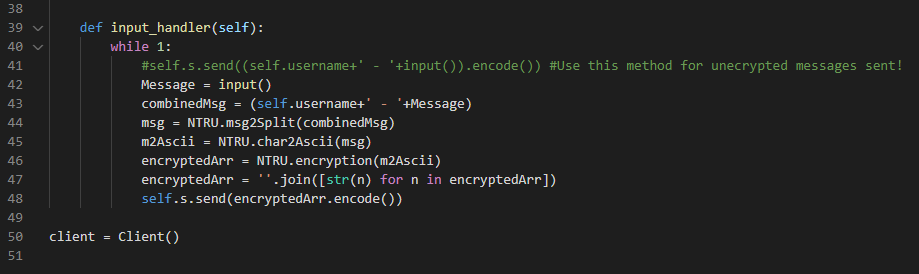
**stand-alone.py**

**server.py**

****

****

**client.py**

**** ****