University of Stavanger

Assignment 2
Security and Vulnerability of the Networks

Abolfazl Taleb zadeh

October 2021

Abstract

The second mandatory assignment is mainly about practicing key exchange capability using Diffie Hellman algorithm. Utilizing key exchange algorithms play a vital role in the world of cryptography particularly under the circumstances in which using a symmetric algorithm is urged. Therefore, a unique single key is needed to be exchanges among concerned parties. As far as transferring the keys without following the confidentiality principles will put a big question mark on the primary purpose of keeping the secrecy of the message, a well-trusted method is needed for sharing the credentials among all the parties within a communication. For implementing the Diffie Hellman algorithm, a class structure is preferred. In the class the arguments have been defined as the properties and different stages of the instruction is defined as methods which can be called in the proper order by the user. In addition, a pseudo-random number generator algorithm (CSPRNG) is used to fortify the enumerated key, which in this specific case the choice is B.B.S (Blum Blum Shub) algorithm. It can be observed that using B.B.S the calculated key gets significantly more secure and also the intended size of the key for being usable by the user-defined encryption algorithm which would be SDES, is delivered thoroughly. The beneficiary and efficiency of application of both Diffie Hellman and B.B.S is well verified and experimented in the second part of the assignment which aims to establish a safe mutual communication tool using the assumptions implemented in the class.

I. Introduction

Using symmetric encryption algorithms has always been known for having the burden of sharing the keys between the two sides of the communication. Diffie-Hellman, also known as DH, was the first algorithm ever used for sharing the keys of a symmetric encryption method which was published in 1976 by Whitfield Diffie and Martin Hellman. DH makes it possible for two parties of a mutual communication which use a symmetric encryption method to share a single key without any need of special equipment or predefined secret information. The most important downfall of this method is that none of the sides of the communication cannot be sure about the authenticity of the public key that is obtained from the other side. In the other word there always is a scale of man in the middle attack risk [1]. A solution which is given for this problem is Public Key Infrastructure (PKI) which as stated in www.wikipedia.com, "have been proposed as a workaround for the problem of identity authentication. In their most usual implementation, each user applies to a "certificate authority" (CA), trusted by all parties, for a digital certificate which serves for other users as a non-tamperable authentication of identity". ("Key Exchange" November 2014 section 2.3 public key infrastructure). There are also other methods like Password-Authentication Key Exchange (PAKE), Quantum Key Distribution (QKD) and etc. which we are not going to go any further into details. In this assignment since the generated password using DH is pretty simple, a pseudo-random number generator algorithm (PRNG) is being used to fortify the enumerate key using DH algorithm, which in this case B.B.S (Blum Blum Shub) algorithms is chosen. Pseudo-random number generators implement a value called seed to generate a random number which will be repeated in a specific range. There are multiple types of PRNGs which use different methods (e.g.RC4, NIST CTR_DRBG, ANSI X9.17). in the first part of the assignment a secure key exchange instruction is practiced using DH with having the benefit of key extension capability using B.B.S algorithm. And in the second part this method has been implemented in real a mutual communication using two servers through different ports.

II. Design and Implementation

In this part the implementation method and the technique which is used in different part of each section will be discussed and analyzed in detail. there are eight steps in the first part of the program which will be argued in turns. The first part of the program is constructed in the form of a Python class because it will be used in other parts of the program. In the class, there are several public variables which would be known as the properties of the class after the instances are made. These variables store the necessary values concerning different stages of the class. These variables include, a list which is called “primes” which holds a list of prime numbers that will be used for multiple purposes, a list called “z” which stored the values of a cyclic group of “p”, an integer variable called “f_k_dec” which keeps the decimal final key which is calculated by B.B.S algorithm, an integer variable named “p” which holds the prime value which is used for DH algorithm, integer variables “ya” and “yb” which store the public keys for the first and the second communication side also known as “alice” and “bob”, an integer variable called “k” which holds the value of the key enumerated by DH algorithm, integer variables “a” and “b” which store the private keys for both parties, an integer variable named “q_times_p” which holds the value of multiplication of two prime numbers which is used by B.B.S algorithm. The class also has an initiation method which receives a number “n” in the defining stage. This number assigns how big the chosen prime number and as the result the calculated key can be. In the initiative method of the class several methods also get executed which will be discussed individually. The very first method that is called is “prime_gen()” which has the task of generating a set of prime number smaller that the number stored in “n”. within this method, a flag which is named “is_prime” is used for specifying whether the generated number is a real prime number. There are to nested loops in this method, the outer one iterated from two, which is to be the first prime number, to the number “n”. and the inner loop runs with value of two which is the smallest value for the outer loop, to the current value of the outer loop which is stored in variable “i”. This is to check every single number smaller than the current value of the outer loop as a factor, and test if its correspondent mod is zero and as the result the number is a prime number or not. So, it checks if the mod of the outer loop current value (i) has the remainder of zero on the division by the value of the inner loop (j). If so, it changes the value of the flag “is_prime” and break the inner loop and so on. But if no factor was found for regarding a value for variable “i”, the “is_prime” flag remains “True” and going through the second conditional phrase the value “i” is appended to the primes list which we mentioned before. This way, all the prime numbers smaller that “n” value is calculated and stored. The next method which is called in initiative of the class is “q_assigner” which has the task of choosing one of the prime numbers in the “prime” list. In this method, a random number is chosen in the range of the length of the list and is used as the index of an element of the prime list. This element is chosen and assigned in “p” variable of the class. The next and of course one the most important methods of the program will be “g_calculator()”. This method is responsible for producing the list of generators of the cyclic group and selecting a single generator from the list randomly. At the beginning of the method a local list named “generators” is defined (the reason the list of generators is not stored as a property of the class is that, as far as a single generator is randomly chosen from the generated list and stored in the public variable “g” and also the list of generators does not include in the program mandatory requested output, it is not necessary to keep the list in the class). After defining the generator list, the cyclic group is produced and kept in the public list “z” through a simple loop which runs for the carriable “i” from one to p-1, in which p is the prime number selected previously. As far as a cyclic group of Z_p^* is a set of numbers from one to p-1, the numbers just simply get appended to the “z” list. Then considering the definition of a generator of a cyclic group which is stated in www.wikipedia.com, “When $(Z/nZ)^\times$ is cyclic, its generators are called primitive roots modulo n” (cyclic groups, section 2.2, p.1 l.4). A loop tests every member of “primes” list if they are a generator of Z_p^*. It is done by calculating the result of X to the power of Y mod p, which X would be a member of “primes” list (every prime number smaller than p) and Y is every member of the cyclic group Z_p^*. The result then is stored in a list, gets converted to a set to remove repeated results and gets compared to the Z_p^*, if the set was identical to the cyclic group, then "g" is known as a generator appends to "generator" list and process continues for the next number in "primes" list. Eventually after all the generators found, in a process similar to the method "q_assigner()" one of the generators get selected and assigned to public "g" variable and generator is returned for the purpose of printing to the output. The example below shows how the generators of a cyclic group is calculated:

$$p = 7 \qquad Z_7^* = \{1,2,3,4,5,6\} \text{ and for each } x \mid x \in P_7 = \{2,3,5\}$$

$$for\ x = 2$$

$$outputs = \{1,2,4\} \text{ which is not equal to } Z_7^*$$

$$2^1 mod\ 7 = 2 \qquad 2^4 mod\ 7 = 2$$

$$2^2 mod\ 7 = 4 \qquad 2^5 mod\ 7 = 4$$

$$2^3 mod\ 7 = 1 \qquad 2^6 mod\ 7 = 1$$

$$for\ x = 3$$

$$outputs = \{1,2,3,4,5,6\} \text{ which is equal to } Z_7^* \text{ so 3 is a generator for cyclic group } Z_7^*$$

$$3^1 mod\ 7 = 3 \qquad 3^4 mod\ 7 = 4$$

$$3^2 mod\ 7 = 2 \qquad 3^5 mod\ 7 = 5$$

$$3^3 mod\ 7 = 6 \qquad 3^6 mod\ 7 = 1$$

$$for\ x = 5$$

$$outputs = \{1,2,3,4,5,6\} \text{ which is equal to } Z_7^* \text{ so 5 is a generator for cyclic group } Z_7^*$$

$$5^1 mod\ 7 = 5 \qquad 5^4 mod\ 7 = 2$$

$$5^2 mod\ 7 = 4 \qquad 5^5 mod\ 7 = 3$$

$$5^3 mod\ 7 = 6 \qquad 5^6 mod\ 7 = 1$$

$$so\ the\ generators\ of\ Z_7^* = \{3,5\}$$

The next method in the initiative of the class is "bbs_random_selector()" which chooses a prime number suitable for B.B.S algorithm. Before going through the method in detail, let's take a closer look on how B.B.S method exactly works. As was told earlier, B.B.S is a PRNG (pseudo-random number generator) algorithm. The reason behind using this algorithm is to generate a bigger and more reliable pseudo-random number based on a smaller random number (k) which is calculated using DH algorithm. The instruction of generating the more complex and secure random number using B.B.S is to use choose two prime numbers in a defined range also having the remainder of three when divided by four. For this purpose, a similar method to "primes_gen()" is used which also checks the required condition for B.B.S prime numbers. Two numbers "p" and "q" are selected and their multiplication result is stored in the public variable "q_times_p". The class also includes some other methods which is discussed in details now. The first method to talk about is "secret_number_generator()" which assigns two random number to variables "a" and "b" as the private key of two individuals (let's say Bob and Alice). And then calculates their corresponding public keys using the formula $Pub_a = g^a mod(p)$. And assigns it to the public class variables "ya" and "yb". The next method "is k_generator()" which does the task of generating a key for the DH algorithm. The formulation for calculating the key in DH algorithm is $k_{ab} = (Pub_a/Pub_b)^{b/a} mod(p)$. The value for the key is calculated and assigned to the class public variable "k" as the key. There is still one more method which does the task of generating the final extended key using B.B.S. the name of that method is "bbs()". Before discussing the stages in the method let's see how B.B.S algorithm creates an extended key using two prime numbers smaller than "p" and a seed value which is equal to the key calculated by the DH algorithm. The process is done by updating the value of seed by raising it to the power of tow and calculating its remainder of the division to "p" times "q". and then finding the remainder of the calculated seed from dividing it to the number two which either will be 1 or 0. Then that will be assigned to the index "i" of a binary number which "i" stands for the number of the rounds which this process is repeated. So, this process goes on until the number of bits gets equal to the interested amount (which in this case is ten, as far as the required key in SDES algorithm that is going to be used in our program is 10 bits). The example below clarifies the procedure in a better way:

For the values p = 883 and q = 523 we have $q \times p = 461809$ and also, we have $seed = 4$ then:

$$i = 0 \rightarrow 4^2 mod\ 461809 => X = 16\ mod\ 2 = 0$$

$$i = 1 \rightarrow 16^2 mod\ 461809 => X = 256\ mod\ 2 = 0$$

$$i = 2 \rightarrow 256^2 mod\ 461809 => X = 65536\ mod\ 2 = 0$$

$$i = 3 \rightarrow 65536^2 mod\ 461809 => X = 143596\ mod\ 2 = 0$$

$$i = 4 \rightarrow 143596^2 mod\ 461809 => X = 39366\ mod\ 2 = 0$$

$$i = 5 \rightarrow 39366^2 mod\ 461809 => X = 312761\ mod\ 2 = 1$$

$$i = 6 \rightarrow 312761^2 mod\ 461809 => X = 446168\ mod\ 2 = 0$$

$$i = 7 \rightarrow 446168^2 mod\ 461809 => X = 343920\ mod\ 2 = 0$$

$$i = 8 \rightarrow 343920^2 mod\ 461809 => X = 136275\ mod\ 2 = 1$$

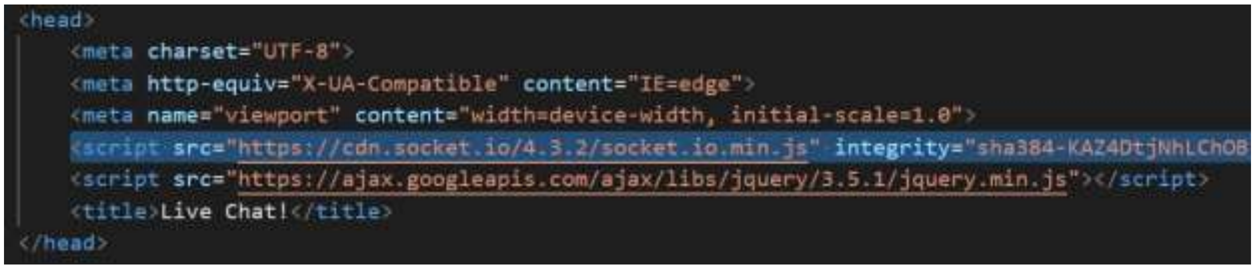
$$i = 9 \rightarrow 136275^2 mod\ 461809 => X = 150308\ mod\ 2 = 0$$

X	0	0	0	0	0	1	0	0	1	0
i	0	1	2	3	4	5	6	7	8	9

Binary key = 0000010010

Decimal key = 18

So, in the method, a local variable named “seed” is defined and takes the value of the public variable “k” which is the key calculated by DH method. A variable named “f_k” and a list named “key” is also specified which keep the final key in decimal and binary respectively. Then there is a loop which iterates from zero to 10 (for a key with the length of ten bits). Within the loop, the value of the variable “seed” gets updated in each repetition with the value of $seed^2 \ mod \ (p \times q)$ and the remainder of “seed” from the division to two will be assigned to “key[i]” and simultaneously, at the next line the value of decimal key is updated. At the end of the for loop, the list “key” is converted to a string via a for loop and is assigned to the class public variable “f_k_bin” and the value of “f_k” is assigned to “f_k_dec”. To fulfill what is required in the first part of the assignment. An instance of the class has been defined, the methods “g_calculator()”, “secret_number_generator()”, “k_generator()” and “bbs()” are called in turns. In the initiative method of the class and also by calling the first three methods all variables p, z, g, Alice’s public key, Bob’s public key, Alice’s private key, Bob’s private key, the key calculated by DH and p times q are assigned and printed out and also by calling “bbs()” the variables for final key in decimal format and also final key in binary format are assigned. Now, it’s possible to use the values for encryptions and decryptions. Before with start the cryptographic part, it should be stated that, because the aim of the first part is to implement the algorithms, the class is designed in the way that the variables for both “Alice” and “Bob” are stored in the same instance. Nevertheless, in the more realistic examples (as will be seen in part II) each instance holds the variables for just one party of the communication. And it’s obvious that the value of the major variables like “p” and “g” which needs to be assigned by just one side and exchanged with the other, can be assigned to the properties of the class later on, manually by the user who has agreed to admit the other sides calculated values. In the other words, the instances of both sides of the communication, in the essence, play the role of a complement along with each other. In the second part of the assignment, the assumption is such that, there are two servers running on two different ports (port 5000 and 80). The clients on each server should be able to communicate to each other in a secure way. The messages are supposed to be encrypted using a symmetric encryption algorithm of choice, and for the start of the communication over, two servers should come to an agreement on the major DH parameters including the prime number (p), the generator (g), the multiplication of two prime numbers of B.B.S algorithm (q_times_p) and they also should exchange their public keys. After they have all the stages above done, the clients can send messages to the server, the messages get encrypted using the key which is calculated using DH and fortified by B.B.S algorithms, and sent over to the other server where gets decrypted using the same key and transmitted to the correspondent client. The servers communicate with clients using flask and socketIO library. By opening http://127.0.0.1:portnumber/ (which port number can be either 5000 or 80) the index page loads and requests the user to enter his/her name. the name then gets registered on the server and on the next page the user should enter the port number to look for the user to chat to. When the user enters a port number, the server looks for the server on the specific port and tries to connect to the user on that server by sending a request message containing the DH and B.B.S algorithm parameters. Then the agreement is made and to parties get ready to start the talk. Throughout the process the users do not understand if they are the one who has sent the agreement message or in the opposite way because it is done by the server through checking a flag. If the value of the flag is True it means that the agreement is already done by the other side but if the flag is false, it means that no agreement has been made and the server is the starter of the communication process and the initiator of the agreement though. So, now different parts of the server file will be discussed in detail and in the next step the client side which is written in Jquery will be analyzed and described. On the server file at the beginning of the program several libraries including Flask, redirect, url_for, render_template, request, jsonify and make_response form flask, socketIO and send from flask_sockeIO and sys, request, DH and SDES individually are imported. There are also quite a few variables which are define publicly including, plain which holds the value for the plain text, n which stores the range of the key in DH algorithm, enc_session which is a dictionary containing status which is the flag for the status of the agreement, “p” or the prime number, “z” for the cyclic group, “g” for holding the value of the selected generator, “y” which keeps the public key of the client, “a” for the private key assigned to the client, “k” for the key that is enumerated by DH algorithm, “key_dec” for keeping the decimal value of the key expanded by the B.B.S, “key_bin” for storing the binary value of the extended key using B.B.S and finally “p_q” which keeps the multiplication of two prime number for the algorithm B.B.S. It should be stated that “enc_session” dictionary plays an important role on the program while it stores pretty important data for encryption of the messages transmitting to the other party. There are also two dictionaries named “home” and “target” containing name and the port of the home and target servers. There are also two other variables called “sendingMessages” and “receivingMessages” which stores the messages that are being sent and received by the server. The flask app is defined by the syntax app=Flask(__name__) and then right below that and instance of the socketio is defined and the flask app is entered as the server parameter. The rest of the server program is consisted of the views of both socketIO and also flask app which whill be discussed according to the appearance order. The first view is the index which calls the index.html as soon as the clients gets connected to the server. It’s also necessary to mention that the html files for the server in flask, are stored in the template folder in the main project directory. The second view is called upon entering the name of the client and hitting send bottom. The name and the port are sent to the server using “POST” method and assigned to the “name” and “port” keys on “home” dictionary and the page “apply.html” is rendered and “name” and “port” are sent to it. The next view is “search” which is called after the user types the target port and hits search. At the beginning of the view variables “home”, “target” and “enc_session” are called globally. The port number is received from the client using “POST” method and gets stored in the “port” key of the “target” dictionary. Following that it checks if the “status” flag is True or False. If the flag is false, the server attains the role of the communication starter and starts generating the keys and parameters. An instance of DH is made using the initial value “n” and the methods “g_calculator()” and “secret_number_generattor()” get executed and the values “p”, “g”, “y”, “a” and “q_p” is assigned to enc_session dictionary from the DH instance. Then a dictionary named “payload” is formed using “type”, “name”, “p”, “g”, “y” and “p_q” and is sent to the target server using http request get method. And the answer is stored in a variable named r. The data is sent to the “connection” socketio view of the receiver, where the “home”, “target” and “enc_session” are called globally and in a conditional phrase if the “type” value is equal to “auth” it is understood that this message is sent for initiating a communication. Thus, the value “name” stores in “name” key of the “target” dictionary and “p”, “g”, “y” and “p_q” is saved manually in the DH instance. In the next step the methods “k_generator()” is called and calculates the key value using DH algorithms and then “bbs()” is called and the key gets extended. The last step of this view is to assign the DH parameters into “enc_session” and store them globally in the server. The value of the “y” variable which is the public key of the server also gets returned to the communication initiator so that it can make up the keys and get ready to encrypt and transmit the messages. The communication initiator server then stores the other party’s public key in the DH instance and calculates the keys and assign them to “enc_session” dictionary. To get a clear sense of the logic procedure of the app, at first the instruction of how messages are sent from the chatbox.html page is discussed and then the corresponding views will be covered and described. In the chatbox.html file right at the top of the page, there are three variables passing into the page by the server, first one is the name of the user, the second is the target’s name and the third is the target port. Using these parameters, a welcoming message is made and also the user gets the information about the person which is on the other side. In addition to these there are also two text area elements which the first one is used to show the sent and received messages and the second which is located right below the first one has the task of receiving the messages from the user. And finally, there is a button below the second text area for sending the messages which is entered. The process of sending and receiving the messages to the server is being done using jguery language and specifically, using the socketIO library which is included in form of CDN at head part of the html page in a script tag (figure 1).



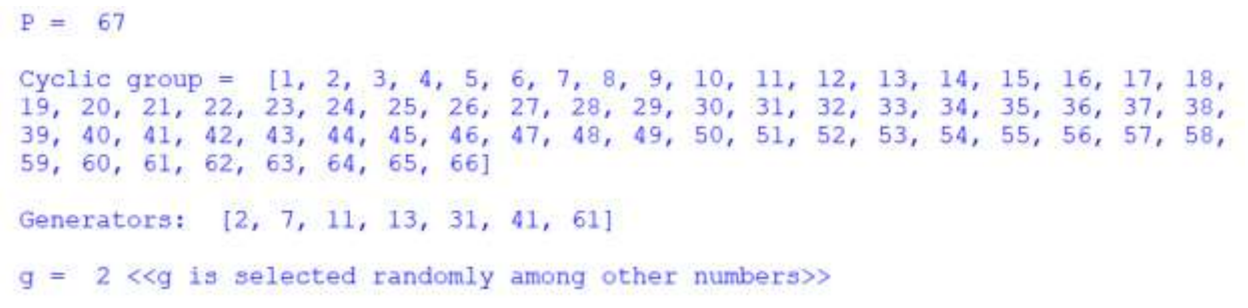
```
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script src="https://cdn.socket.io/4.3.2/socket.io.min.js" integrity="sha384-KAZ4DtjNhLChOB
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
    <title>Live Chat!</title>
</head>
```

Figure 1

In the script below the page, everything happens on the “$(document).ready()” part because we want it to be alerted and keep repeating when the page is running. Within the event, there is a function which is has the responsibility for all the things happening. The first two variables are defined, first one keeps the server’s name and the port number and second is an instance defined from socketio which connects to the server which has been entered in the first variable. Then we have the “on connect” event which sends a Json message containing keys type, body and port. The next step there is “on message” event, which receives the message from the server and appends it to the first text area and the last event which is “key pressed”, and in this event if the user presses the “Enter key” the “send button click” will be called. Now, the view in the server which handle the task of sending and receiving the messages will be discussed. When the chatbox.html page loads it automatically tries to connect to the server and send a message containing the connection request message along with the port number and other stuff. The server receives the message in “message” view of the socketIO. In that event, at first a local variable called “enc_mess” is defined, and “home”, “target” and “sendingMessages” are called globally. There there’s if conditional state which check the type of received message, if the type of the message is “connection” then home port is set in the “port” key of the “home” dictionary and an informing message “You can now chat!” is sent to the client. But, if the type of the message is equal to “msg”, it means that a message is received from the client and it needs to be encrypted and sent to the other party’s server so called the target. Thus, the next step is to change the convert the characters in the message to ascii codes and then to binary format to be ready for encryption, all of which is done within a pre-implemented function named “str_bin()”. The result of the function which is a string of zeros and ones is stored in a variable name “bin_msg”. As far as the encryption is being done for just one binary byte at a time, the string needs to be divided into eight-bit segments. This is what “bin-spliter()” function does. It receives a string of the type binary and an integer value, then returns a list containing binary segments of the message with length of entered integer value. The result the is kept in a list named “bin_split_msg”. the next step is to encrypt all the segments (using the credentials calculated before via the agreements made between two servers) in a for loop and adding them to “enc_mess” variable which is defined earlier. Following that, a dictionary named “payload” is formed and sent to target server containing the type of the message, message body and the sender’s port. It’s also necessary to mention that, the encrypted message is printed in the server log for supervision purposes. The message also is transferred to the client to get appended into the received text area. The receiver sever then receives the message from the transmitter server on the “receive” view. Within this view the first step is to build an instance of the “SDES” class for decrypting the message received from the server. Following that, the type of the received message is check, if the type is a equal to “msg” it means a new message is received from the other party and it requires to undergo the process of decryption. In this stage, the same thing happened in the encryption process by the sender happens, but actually in the opposite way, meaning, first the message gets broken into segments of eight bits, then each segment gets decrypted by the credential which has agreed earlier, and then they are converted into the asci code and characters. In the next step, the message is transferred to the client using “socketIO send method” and also gets printed to the server log for the purpose of supervision and debugging.

III. The Results

In the first part of the assignment, for DH algorithm, all the parameters like the first prime number, the cyclic group, the generator set and the selected generator can be calculated just with entering a number at in the stage of creating the instance of the class. The result of the first step of DH algorithm can be found in figure 2. Then after calculating the mentioned parameters the private keys and the public keys and are enumerated which the results can be in figure 3. The next step is to calculate the key by using each side's private key and the other side's public key. The result of which can be observed in figure 4. Following this step, the B.B.S algorithm can be started and the extension of the key happens by calling the corresponding method. The results related to this process can be seen in figure 5. Now having an extended key file can be read from a file and get decrypted and the result gets written to a file. The process can be seen in figure 6.



```
P =  67

Cyclic group =  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38,
39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58,
59, 60, 61, 62, 63, 64, 65, 66]

Generators:  [2, 7, 11, 13, 31, 41, 61]

g =  2 <<g is selected randomly among other numbers>>
```

Figure 2

```
Alice's private Key=  28
Alice's Pub Key=  23

Bob's private Key=  42
Bob's Pub Key=  24
```

Figure 3

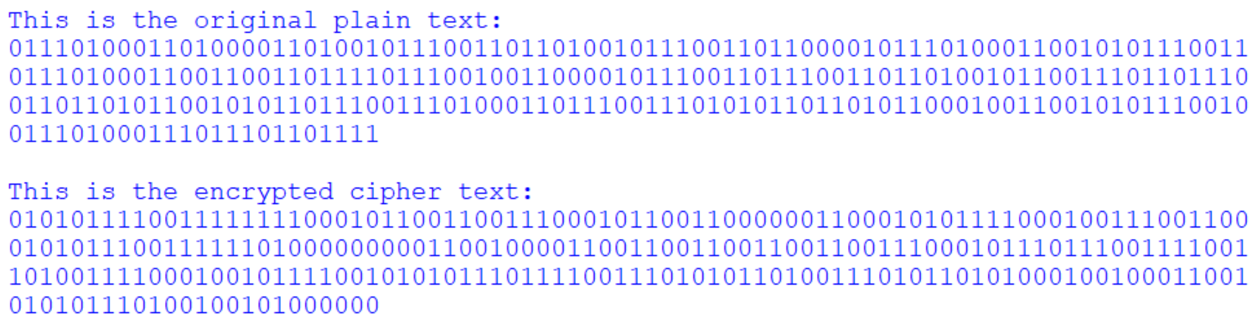
```
k(a,b)=(Alice's Pub key:23 ^ Bob's Priv key:42) mod (p:67) which equals:15
```

Figure 4

```
Alice and Bob agreed on the values 'q' and 'p' which (n = q * p)=221633

final key in binary: 1100101110
final key in decimal:814
```

Figure 5

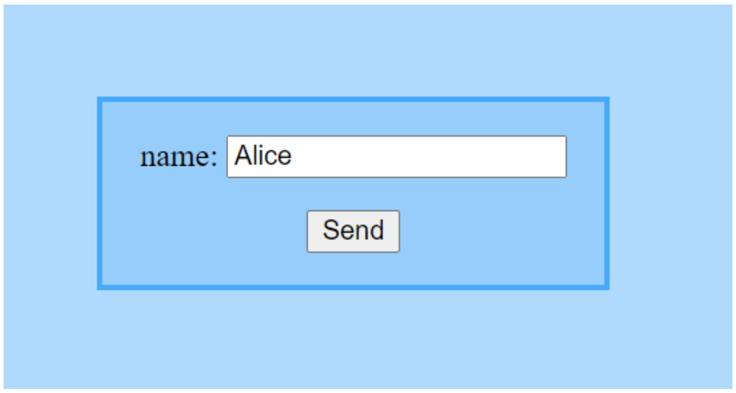


```
This is the original plain text:
01110100011010000110100101110011011010010111001101100001011101000110010101110011
01110100011001100110111101110010011000010111001101110011011010010110011101101110
01101101011001010110111001110100011011100111010101101101011000100110010101110010
011101000111011101101111

This is the encrypted cipher text:
01010111100111111110001011001100111000101100110000000110001010111100010011100 1100
01010111001111110100000000011001000011001100110011001100111000101110111001111001
10100111100010010111100101010111011110011101010110100111010110101000100100011001
010101110100100101000000
```

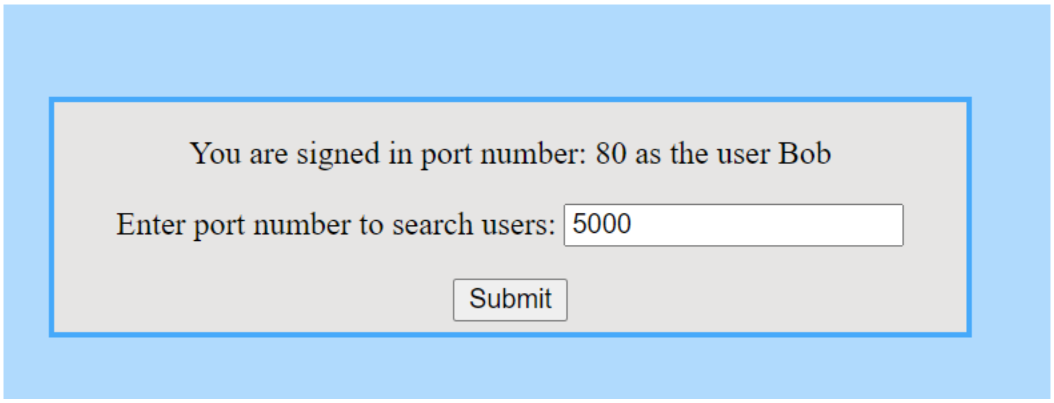
Figure 6: the content of files: 1.txt and 2.txt

In the second part of the assignment, the results are shown in the client side which would be the internet browser page and the server running page where some outputs are printed out for the observation purposes. After two servers start running, and clients are connected to the server, the client in the first page enters the name, then in the next page by entering the target port number server looks for a client which is ready to talk on that specific server and opens the chat box page. On that page, the users can type their messages and hit send button, the message then goes through the server, encrypted, and transmitted to the other client which is connected through another server. On figure 7, figure 8 and figure 9 the out put windows of the results for the second part of the assignment is shown.



name: Alice

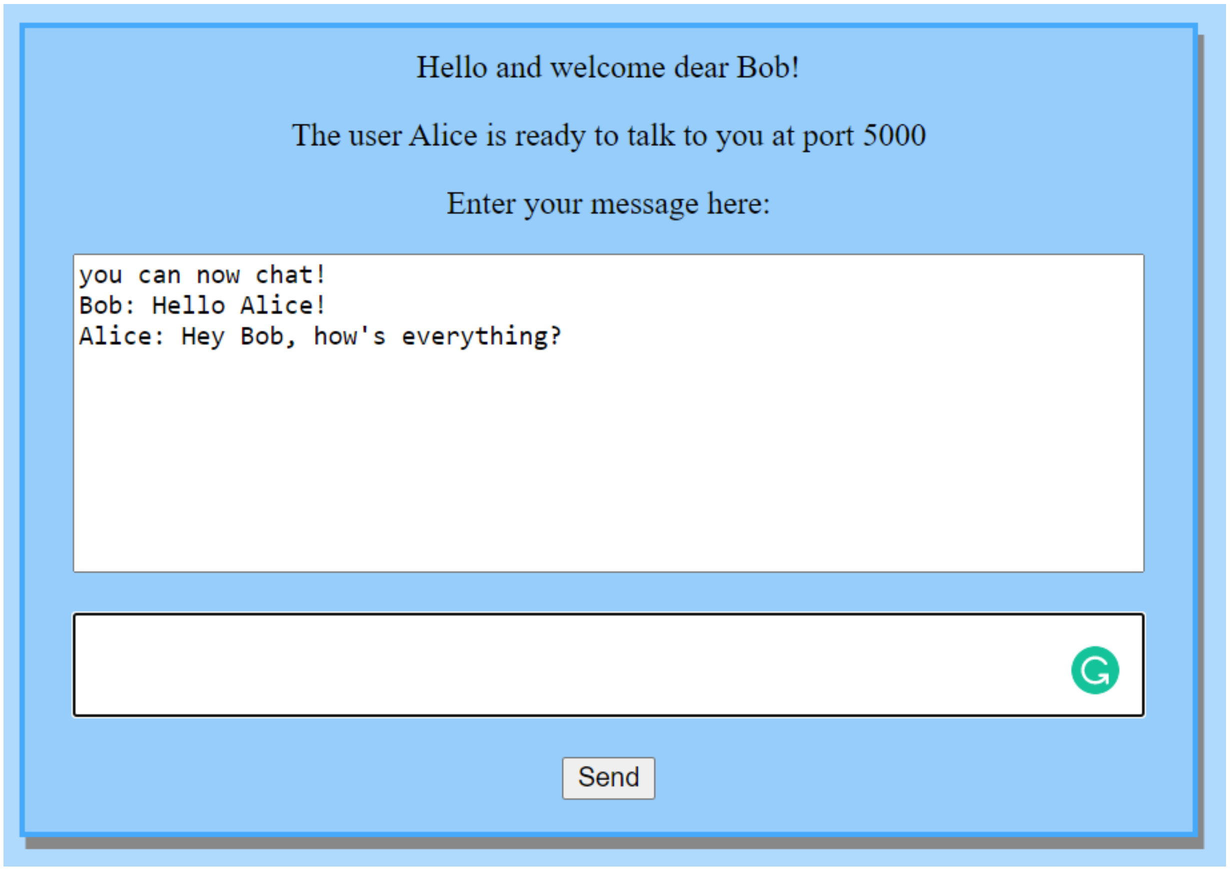
Figure 7



You are signed in port number: 80 as the user Bob

Enter port number to search users: 5000

Figure 8


Hello and welcome dear Bob!

The user Alice is ready to talk to you at port 5000

Enter your message here:

```
you can now chat!
Bob: Hello Alice!
Alice: Hey Bob, how's everything?
```

Figure 9

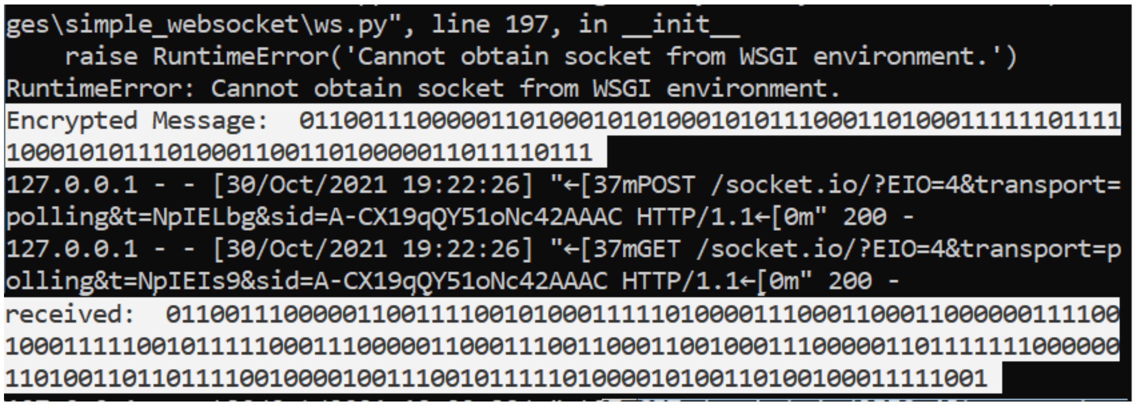

```
ges\simple_websocket\ws.py", line 197, in __init__
    raise RuntimeError('Cannot obtain socket from WSGI environment.')
RuntimeError: Cannot obtain socket from WSGI environment.
Encrypted Message:  01100111000001101000101010001010111000110100011111101111
10001010111010001100110100000110111101111
127.0.0.1 - - [30/Oct/2021 19:22:26] "←[37mPOST /socket.io/?EIO=4&transport=
polling&t=NpIELbg&sid=A-CX19qQY51oNc42AAAC HTTP/1.1←[0m" 200 -
127.0.0.1 - - [30/Oct/2021 19:22:26] "←[37mGET /socket.io/?EIO=4&transport=p
olling&t=NpIEIs9&sid=A-CX19qQY51oNc42AAAC HTTP/1.1←[0m" 200 -
received:  0110011100000110011110010100011111010000111000110001100000011110
1000111110010111110001110000011000111001100011001000111000001101111111000000
110100110110111100100001001110010111110100001010011010010001111001
```

Figure 10: Encrypted Outputs on the servers

IV. Discussion

According to what has been done within two parts of this assignment, using an algorithm like DH as a solution for the problem of exchanging the key while applying a symmetric encryption algorithm and subsequently a single encryption key along with a PRNG algorithm for strengthening that key derived from

DH algorithm, can be considered an ultimately effective solution. It's also essential to mention that although the whole theory of the sort of the tools applied for this reason still remains the same, the kind of specific algorithms which have been utilized can be chosen deliberately amongst lots of varieties of kinds which existed. There is just this principal issue that there should be a way which two parties be able to have the same key without the necessity of transmitting any confidential information between each other. And also, the importance of "pseudo-random number generator" algorithm is understood in an efficient way. As it can be seen in figure 4 the key which was calculated by DH algorithm was equal to "15" how turned out to be "814" using B.B.S and how incredibly it can be effective on increasing the security of the quality and rigidness of an encryption. In part two of the assignment also after using the program, it can be understood how safe the communication can get after encrypting it without the need of exchanging the key. And as far as the messages are transmitted as a cipher text not the plain text, the risk of eavesdropping certainly decreases or at least getting to know the content of the messages would be far more difficult or time consuming.

V. Conclusion

The aim of both parts of this assignment as has been told multiple time during the report was to get a better understanding of the concept of applying symmetric algorithm in a more efficient way. Precisely in this special case implementing a method to exchange the keys without jeopardizing the secrecy of the communication. And also strengthening the keys which has been chosen under a restricted circumstance and forcefully week using "pseudo-random number generators". During this assignment pretty conventional methods were used, but it's obvious that there are lots of kinds of algorithm which can be used for this reason. The future challenge can be finding the best combination of the algorithms and building a framework for having the confidential communication in best possible way, because as the cyber-communication grows daily, there is always a new need in keeping the process of transmitting data secure.

VI. Works Cited

[1] https://en.wikipedia.org/wiki/Key_exchange

[2] https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

[3] https://www.youtube.com/watch?v=Yjrfm_oRO0w

[4] https://www.youtube.com/watch?v=hp7bpkNL790

[5] https://dev.to/techparida/how-to-deploy-a-flask-app-on-heroku-heb

[6] https://www.youtube.com/watch?v=M-0qt6tdHzk&t=7s

[7] https://www.youtube.com/watch?v=9MHYHgh4jYc

[8] Cryptography and Network Security: Principles and Practice, Stallings, W.

[9] http://cs.wellesley.edu/~webdb/lectures/flask/flask.html

[10] https://www.youtube.com/watch?v=Yjrfm_oRO0w&t=317s

[11] CS255 Stanford course textbook "Cryptography and Computer Security. Very basic number theory fact sheet"