DAT510-1 20H Network security and vulnerability. Assignment 2. Implement Secure Communications

Asahi Cantu Moreno (student id: 253964)

October 7, 2020

Abstract

Having knowledge on the way symmetric algorithms are implemented and work, this report focuses now on the theoretical explanation and results for the implementation of a secure framework for message transmission by applying several encryption and secure mechanisms which although simple, clearly describe the way real application work. The work and developed tasks are described in 2 parts:

1. **Part I. Secure Communication Algorithm implementation.** Implementing algorithms for:
    - Secure Key Exchange algorithm. Diffie-Hellman Key exchange method.
    - CSPRNG[1] Implementing BBS[2] generator
2. **Part II. Server and client implementation for secure communication.** A simple but real life implementation for secure messaging simulating the key exchange mechanisms and sending encrypted messages through the network.

For each part a detailed explanation and emulation is provided, the workflow for key exchange and message encryption as well as demonstration of the application. The developed code and implementations for each part of the assignment can be found together in the contents of this document repository.

[1]CSPRNG stands for Cryptography secure pseudo random number generator
[2]BBS is the Blum Blum Shub CSPRNG

1 Design and implementation

1.1 Part I. Secure Communication Algorithm implementation.

1.1.1 Diffie-Hellman Key Exchange algorithm

The first published public-key algorithm appeared in the seminal paper by Diffie and Hellman that defined public-key cryptography [DIFF76b] and is generally referred to as Diffie–Hellman key exchange. A number of commercial products employ this key exchange technique [Stallings(2017)]. This algorithm allows two users to exchange their public key and send encrypted information without the need of sharing nor sending he private key.

The core of such algorithm relies on its ability to find the primitive root[3] of a given prime number. Its effectiveness then relies on the difficulty to compute the discrete logarithm. Figure 1 illustrates with more detail the key exchange method..

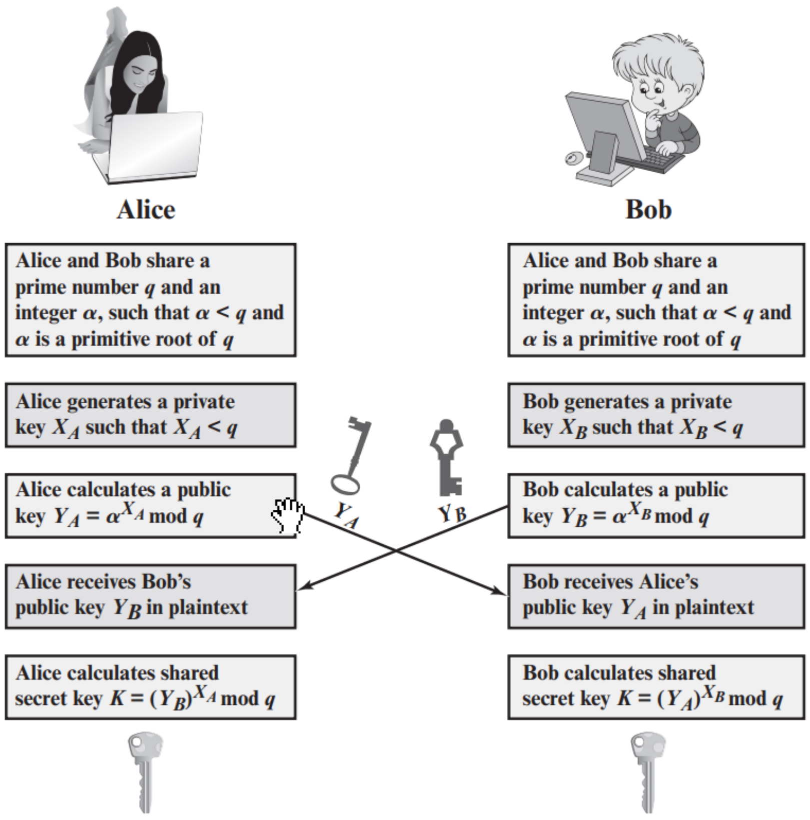

Alice
Bob
Alice and Bob share a prime number q and an integer α, such that $\alpha < q$ and α is a primitive root of q
Alice and Bob share a prime number q and an integer α, such that $\alpha < q$ and α is a primitive root of q
Alice generates a private key X_A such that $X_A < q$
Bob generates a private key X_B such that $X_B < q$
Alice calculates a public key $Y_A = \alpha^{X_A} \bmod q$
Y_A
Y_B
Bob calculates a public key $Y_B = \alpha^{X_B} \bmod q$
Alice receives Bob's public key Y_B in plaintext
Bob receives Alice's public key Y_A in plaintext
Alice calculates shared secret key $K = (Y_B)^{X_A} \bmod q$
Bob calculates shared secret key $K = (Y_A)^{X_B} \bmod q$

Figure 1: Diffie-Hellman algorithm explained. [Stallings(2017)]

Once the public keys are shared and shared keys submitted to each one of the users, the message can be generated, encrypted and send one to another. The decryption mechanism will then work without having to share the private key. This method was strengthened and

[3]A primitive root mod n is an integer g such that every integer relatively prime to n is congruent to a power of g mod n. The integer g is a primitive root (mod n) if for every number a relatively prime to n there is an integer z such that $a \equiv g^z \mod n$

enriched by randomly generating a private key by using a CSPRNG-BBS implementation. Once the message is sent, a new seed is created and passed to the BBS generator so a new random prime number can be emitted and later on used as a new private key. Once this is done the shared keys have to be generated and sent to the corresponding users.

The Blum Blum Shub random number generator was chosen for its simplicity on its implementation and reliability towards the generation of random numbers.

1.1.2 Code implementation

To perform all the requirements for this part I the algorithm and methodologies were developed using python language and some additional packages which can be found under the source code of this section. Python files created:

1. DH.py - Contains the implementation of Diffie-Hellman key exchange methodology
2. BBS.py - Contains the implementation of Blum Blum Shug CSPRNG to regenerate the private keys

   Finally a simulation for key sharing and message exchange through encryption was written. Log information can be shown in figure 3.

Key Exchange code simulation

```
import importlib
from DH import DH
from BBS import BBS
import sympy
import random
from Cryptodome.Cipher import AES
key_1 = sympy.randprime(2**17,2**18)
key_2 = DH.primRoots(key_1)[0]
dh_a = DH(key_2, key_1)
dh_b = DH(key_2, key_1)
messages = ['Hello World','This is a secret message','come here']
for message in messages:
    dh_a.create_shared_key()
    dh_b.create_shared_key()
    dh_a.create_full_key(dh_b.shared_key)
    dh_b.create_full_key(dh_a.shared_key)
    enc = dh_a.encrypt(message)
    dec = dh_b.decrypt(enc)
    dh_a.next_key()
    dh_b.next_key()
```

1.2 Part II. Implementation of Secure Chat messenger.

By confirming the developed algorithm works it was now required to implement a simple chat application that operates with the key exchange and encryption mechanisms previously developed. For this implementation the same methodology as in Part I applies, with the creation of a server and client applications.

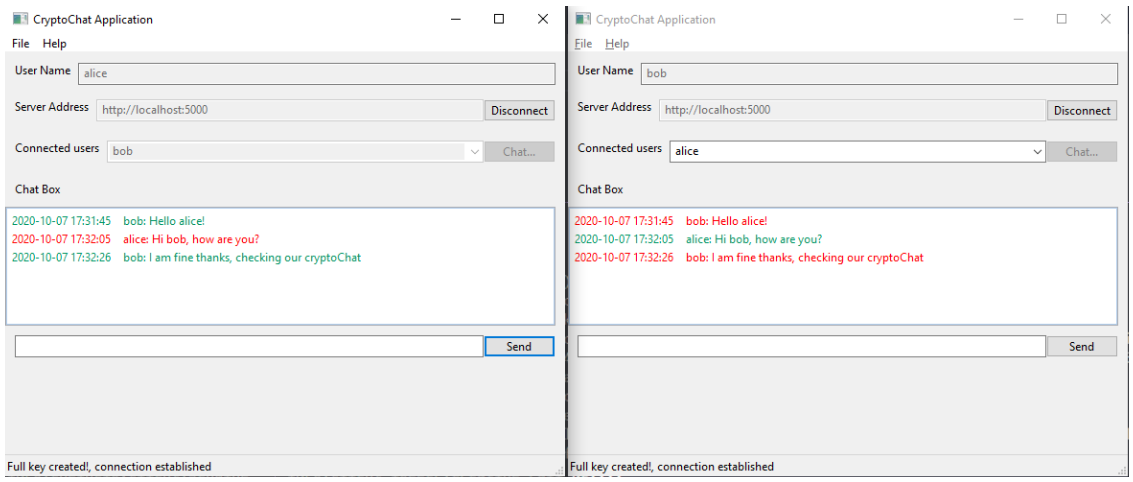
1.2.1 Code Implementation

Python files created

1. Server.py: Is the server application. Runs in python with flask package and uses socketio communication protocol. The server just allows client-to-client communication by redirecting to the right session the messages and key exchanges. During the log session in the server it is possible to visualize that in any time the messaging information always is received encrypted and the only visible data are the public and shared keys.

2. Client.py. Is the client application: Runs in python with Graphical User Interface and a socketio communication. It was thought in the beginning to develop the client side on a web page, but the main difficulty on doing that relied on the technology from which the algorithms were implemented, in this case, python. Having chosen web development would have meant to re-code in a different language such as JavaScript or make the application more complex otherwise. The graphical code was created using wxpython and employing the same encryption as in Part I.

In figure 2 a simulation of the program shows the results of the messaging exchange system.



User Name: alice
Server Address: http://localhost:5000
Connected users: bob

```
2020-10-07 17:31:45    bob: Hello alice!
2020-10-07 17:32:05    alice: Hi bob, how are you?
2020-10-07 17:32:26    bob: I am fine thanks, checking our cryptoChat
```

Full key created!, connection established

User Name: bob
Server Address: http://localhost:5000
Connected users: alice

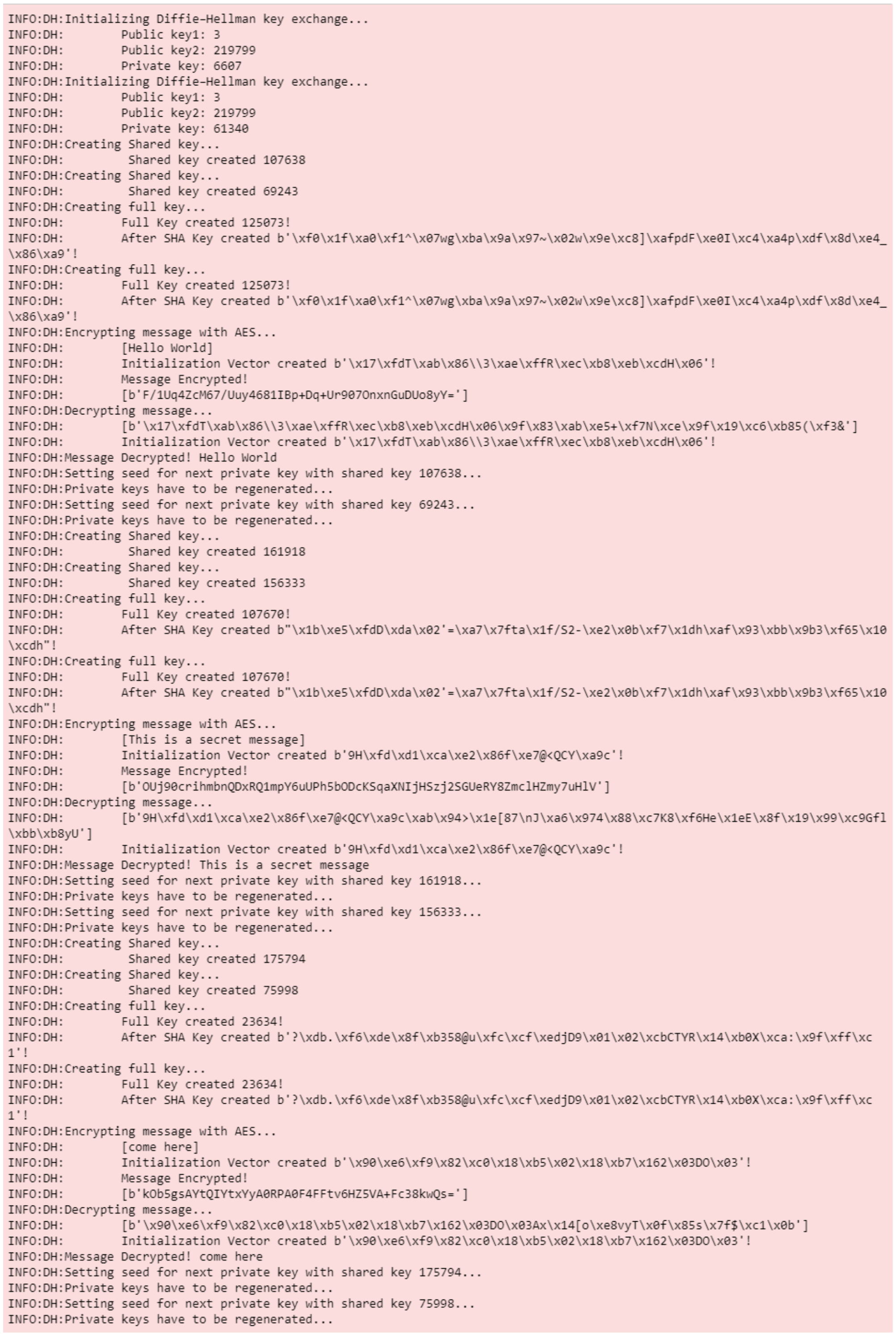
```
2020-10-07 17:31:45    bob: Hello alice!
2020-10-07 17:32:05    alice: Hi bob, how are you?
2020-10-07 17:32:26    bob: I am fine thanks, checking our cryptoChat
```

Full key created!, connection established

Figure 2: The log generated through running the emulated key exchange code.

2 Test Results

2.1 Part I Log generation



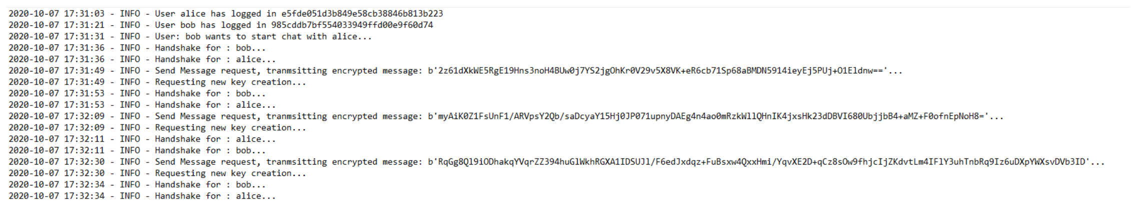
```
INFO:DH:Initializing Diffie-Hellman key exchange...
INFO:DH:        Public key1: 3
INFO:DH:        Public key2: 219799
INFO:DH:        Private key: 6607
INFO:DH:Initializing Diffie-Hellman key exchange...
INFO:DH:        Public key1: 3
INFO:DH:        Public key2: 219799
INFO:DH:        Private key: 61340
INFO:DH:Creating Shared key...
INFO:DH:         Shared key created 107638
INFO:DH:Creating Shared key...
INFO:DH:         Shared key created 69243
INFO:DH:Creating full key...
INFO:DH:        Full Key created 125073!
INFO:DH:        After SHA Key created b'\xf0\x1f\xa0\xf1^\x07wg\xba\x9a\x97~\x02w\x9e\xc8]\xafpdF\xe0I\xc4\xa4p\xdf\x8d\xe4_
\x86\xa9'!
INFO:DH:Creating full key...
INFO:DH:        Full Key created 125073!
INFO:DH:        After SHA Key created b'\xf0\x1f\xa0\xf1^\x07wg\xba\x9a\x97~\x02w\x9e\xc8]\xafpdF\xe0I\xc4\xa4p\xdf\x8d\xe4_
\x86\xa9'!
INFO:DH:Encrypting message with AES...
INFO:DH:        [Hello World]
INFO:DH:        Initialization Vector created b'\x17\xfdT\xab\x86\\3\xae\xffR\xec\xb8\xeb\xcdH\x06'!
INFO:DH:        Message Encrypted!
INFO:DH:        [b'F/1Uq4ZcM67/Uuy4681IBp+Dq+Ur907OnxnGuDUo8yY=']
INFO:DH:Decrypting message...
INFO:DH:        [b'\x17\xfdT\xab\x86\\3\xae\xffR\xec\xb8\xeb\xcdH\x06\x9f\x83\xab\xe5+\xf7N\xce\x9f\x19\xc6\xb85(\xf3&']
INFO:DH:        Initialization Vector created b'\x17\xfdT\xab\x86\\3\xae\xffR\xec\xb8\xeb\xcdH\x06'!
INFO:DH:Message Decrypted! Hello World
INFO:DH:Setting seed for next private key with shared key 107638...
INFO:DH:Private keys have to be regenerated...
INFO:DH:Setting seed for next private key with shared key 69243...
INFO:DH:Private keys have to be regenerated...
INFO:DH:Creating Shared key...
INFO:DH:         Shared key created 161918
INFO:DH:Creating Shared key...
INFO:DH:         Shared key created 156333
INFO:DH:Creating full key...
INFO:DH:        Full Key created 107670!
INFO:DH:        After SHA Key created b"\x1b\xe5\xfdD\xda\x02'=\xa7\x7fta\x1f/S2-\xe2\x0b\xf7\x1dh\xaf\x93\xbb\x9b3\xf65\x10
\xcdh"!
INFO:DH:Creating full key...
INFO:DH:        Full Key created 107670!
INFO:DH:        After SHA Key created b"\x1b\xe5\xfdD\xda\x02'=\xa7\x7fta\x1f/S2-\xe2\x0b\xf7\x1dh\xaf\x93\xbb\x9b3\xf65\x10
\xcdh"!
INFO:DH:Encrypting message with AES...
INFO:DH:        [This is a secret message]
INFO:DH:        Initialization Vector created b'9H\xfd\xd1\xca\xe2\x86f\xe7@<QCY\xa9c'!
INFO:DH:        Message Encrypted!
INFO:DH:        [b'OUj90crihmbnQDxRQ1mpY6uUPh5bODcKSqaXNIjHSzj2SGUeRY8ZmclHZmy7uHlV']
INFO:DH:Decrypting message...
INFO:DH:        [b'9H\xfd\xd1\xca\xe2\x86f\xe7@<QCY\xa9c\xab\x94>\x1e[87\nJ\xa6\x974\x88\xc7K8\xf6He\x1eE\x8f\x19\x99\xc9Gfl
\xbb\xb8yU']
INFO:DH:        Initialization Vector created b'9H\xfd\xd1\xca\xe2\x86f\xe7@<QCY\xa9c'!
INFO:DH:Message Decrypted! This is a secret message
INFO:DH:Setting seed for next private key with shared key 161918...
INFO:DH:Private keys have to be regenerated...
INFO:DH:Setting seed for next private key with shared key 156333...
INFO:DH:Private keys have to be regenerated...
INFO:DH:Creating Shared key...
INFO:DH:         Shared key created 175794
INFO:DH:Creating Shared key...
INFO:DH:         Shared key created 75998
INFO:DH:Creating full key...
INFO:DH:        Full Key created 23634!
INFO:DH:        After SHA Key created b'?\xdb.\xf6\xde\x8f\xb358@u\xfc\xcf\xedjD9\x01\x02\xcbCTYR\x14\xb0X\xca:\x9f\xff\xc
1'!
INFO:DH:Creating full key...
INFO:DH:        Full Key created 23634!
INFO:DH:        After SHA Key created b'?\xdb.\xf6\xde\x8f\xb358@u\xfc\xcf\xedjD9\x01\x02\xcbCTYR\x14\xb0X\xca:\x9f\xff\xc
1'!
INFO:DH:Encrypting message with AES...
INFO:DH:        [come here]
INFO:DH:        Initialization Vector created b'\x90\xe6\xf9\x82\xc0\x18\xb5\x02\x18\xb7\x162\x03DO\x03'!
INFO:DH:        Message Encrypted!
INFO:DH:        [b'kOb5gsAYtQIYtxYyA0RPA0F4FFtv6HZ5VA+Fc38kwQs=']
INFO:DH:Decrypting message...
INFO:DH:        [b'\x90\xe6\xf9\x82\xc0\x18\xb5\x02\x18\xb7\x162\x03DO\x03Ax\x14[o\xe8vyT\x0f\x85s\x7f$\xc1\x0b']
INFO:DH:        Initialization Vector created b'\x90\xe6\xf9\x82\xc0\x18\xb5\x02\x18\xb7\x162\x03DO\x03'!
INFO:DH:Message Decrypted! come here
INFO:DH:Setting seed for next private key with shared key 175794...
INFO:DH:Private keys have to be regenerated...
INFO:DH:Setting seed for next private key with shared key 75998...
INFO:DH:Private keys have to be regenerated...
```

Figure 3: The log generated through running the emulated key exchange code.

It is finally possible to confirm through this code emulation that the algorithms and key exchange are properly working. 16-bit prime key numbers were chosen to avoid slow runs on the code.

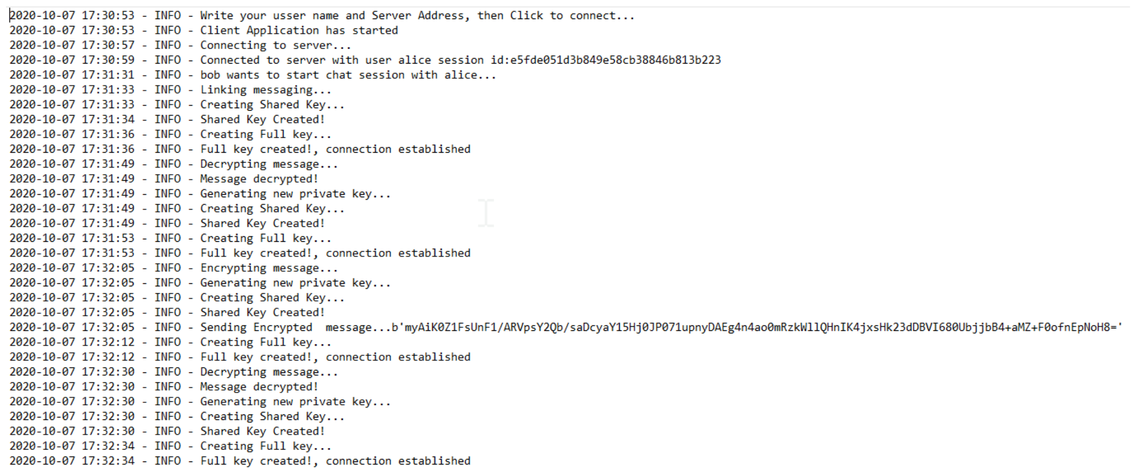
2.1.1 Log files

In the following images the log for each entity is presented.



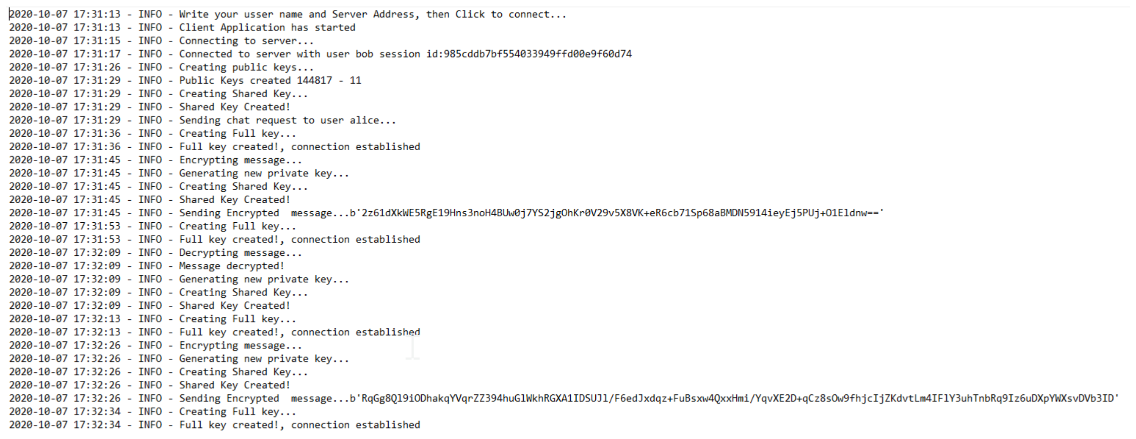
```
2020-10-07 17:31:03 - INFO - User alice has logged in e5fde051d3b849e58cb38846b813b223
2020-10-07 17:31:21 - INFO - User bob has logged in 985cddb7bf554033949ffd00e9f60d74
2020-10-07 17:31:31 - INFO - User: bob wants to start chat with alice...
2020-10-07 17:31:36 - INFO - Handshake for : bob...
2020-10-07 17:31:36 - INFO - Handshake for : alice...
2020-10-07 17:31:49 - INFO - Send Message request, tranmsitting encrypted message: b'2z61dXkWE5RgE19Hns3noH4BUw0j7YS2jgOhKr0V29v5X8VK+eR6cb71Sp68aBMDN5914ieyEj5PUj+O1Eldnw=='...
2020-10-07 17:31:49 - INFO - Requesting new key creation...
2020-10-07 17:31:53 - INFO - Handshake for : bob...
2020-10-07 17:31:53 - INFO - Handshake for : alice...
2020-10-07 17:32:09 - INFO - Send Message request, tranmsitting encrypted message: b'myAiK0Z1FsUnF1/ARVpsY2Qb/saDcyaY15Hj0JP071upnyDAEg4n4ao0mRzkWllQHnIK4jxsHk23dDBVI680UbjjbB4+aMZ+F0ofnEpNoH8='...
2020-10-07 17:32:09 - INFO - Requesting new key creation...
2020-10-07 17:32:11 - INFO - Handshake for : alice...
2020-10-07 17:32:11 - INFO - Handshake for : bob...
2020-10-07 17:32:30 - INFO - Send Message request, tranmsitting encrypted message: b'RqGg8Ql9iODhakqYVqrZZ394huGlWkhRGXA1IDSUJl/F6edJxdqz+FuBsxw4QxxHmi/YqvXE2D+qCz8sOw9fhjcIjZKdvtLm4IFlY3uhTnbRq9Iz6uDXpYWXsvDVb3ID'...
2020-10-07 17:32:30 - INFO - Requesting new key creation...
2020-10-07 17:32:34 - INFO - Handshake for : bob...
2020-10-07 17:32:34 - INFO - Handshake for : alice...
```

Figure 4: The log generated for the server entity.



```
2020-10-07 17:30:53 - INFO - Write your usser name and Server Address, then Click to connect...
2020-10-07 17:30:53 - INFO - Client Application has started
2020-10-07 17:30:57 - INFO - Connecting to server...
2020-10-07 17:30:59 - INFO - Connected to server with user alice session id:e5fde051d3b849e58cb38846b813b223
2020-10-07 17:31:31 - INFO - bob wants to start chat session with alice...
2020-10-07 17:31:33 - INFO - Linking messaging...
2020-10-07 17:31:33 - INFO - Creating Shared Key...
2020-10-07 17:31:34 - INFO - Shared Key Created!
2020-10-07 17:31:36 - INFO - Creating Full key...
2020-10-07 17:31:36 - INFO - Full key created!, connection established
2020-10-07 17:31:49 - INFO - Decrypting message...
2020-10-07 17:31:49 - INFO - Message decrypted!
2020-10-07 17:31:49 - INFO - Generating new private key...
2020-10-07 17:31:49 - INFO - Creating Shared Key...
2020-10-07 17:31:49 - INFO - Shared Key Created!
2020-10-07 17:31:53 - INFO - Creating Full key...
2020-10-07 17:31:53 - INFO - Full key created!, connection established
2020-10-07 17:32:05 - INFO - Encrypting message...
2020-10-07 17:32:05 - INFO - Generating new private key...
2020-10-07 17:32:05 - INFO - Creating Shared Key...
2020-10-07 17:32:05 - INFO - Shared Key Created!
2020-10-07 17:32:05 - INFO - Sending Encrypted  message...b'myAiK0Z1FsUnF1/ARVpsY2Qb/saDcyaY15Hj0JP071upnyDAEg4n4ao0mRzkWllQHnIK4jxsHk23dDBVI680UbjjbB4+aMZ+F0ofnEpNoH8='
2020-10-07 17:32:12 - INFO - Creating Full key...
2020-10-07 17:32:12 - INFO - Full key created!, connection established
2020-10-07 17:32:30 - INFO - Decrypting message...
2020-10-07 17:32:30 - INFO - Message decrypted!
2020-10-07 17:32:30 - INFO - Generating new private key...
2020-10-07 17:32:30 - INFO - Creating Shared Key...
2020-10-07 17:32:30 - INFO - Shared Key Created!
2020-10-07 17:32:34 - INFO - Creating Full key...
2020-10-07 17:32:34 - INFO - Full key created!, connection established
```

Figure 5: The log generated for 'alice' client entity.



```
2020-10-07 17:31:13 - INFO - Write your usser name and Server Address, then Click to connect...
2020-10-07 17:31:13 - INFO - Client Application has started
2020-10-07 17:31:15 - INFO - Connecting to server...
2020-10-07 17:31:17 - INFO - Connected to server with user bob session id:985cddb7bf554033949ffd00e9f60d74
2020-10-07 17:31:26 - INFO - Creating public keys...
2020-10-07 17:31:29 - INFO - Public Keys created 144817 - 11
2020-10-07 17:31:29 - INFO - Creating Shared Key...
2020-10-07 17:31:29 - INFO - Shared Key Created!
2020-10-07 17:31:29 - INFO - Sending chat request to user alice...
2020-10-07 17:31:36 - INFO - Creating Full key...
2020-10-07 17:31:36 - INFO - Full key created!, connection established
2020-10-07 17:31:45 - INFO - Encrypting message...
2020-10-07 17:31:45 - INFO - Generating new private key...
2020-10-07 17:31:45 - INFO - Creating Shared Key...
2020-10-07 17:31:45 - INFO - Shared Key Created!
2020-10-07 17:31:45 - INFO - Sending Encrypted  message...b'2z61dXkWE5RgE19Hns3noH4BUw0j7YS2jgOhKr0V29v5X8VK+eR6cb71Sp68aBMDN5914ieyEj5PUj+O1Eldnw=='
2020-10-07 17:31:53 - INFO - Creating Full key...
2020-10-07 17:31:53 - INFO - Full key created!, connection established
2020-10-07 17:32:09 - INFO - Decrypting message...
2020-10-07 17:32:09 - INFO - Message decrypted!
2020-10-07 17:32:09 - INFO - Generating new private key...
2020-10-07 17:32:09 - INFO - Creating Shared Key...
2020-10-07 17:32:09 - INFO - Shared Key Created!
2020-10-07 17:32:13 - INFO - Creating Full key...
2020-10-07 17:32:13 - INFO - Full key created!, connection established
2020-10-07 17:32:26 - INFO - Encrypting message...
2020-10-07 17:32:26 - INFO - Generating new private key...
2020-10-07 17:32:26 - INFO - Creating Shared Key...
2020-10-07 17:32:26 - INFO - Shared Key Created!
2020-10-07 17:32:26 - INFO - Sending Encrypted  message...b'RqGg8Ql9iODhakqYVqrZZ394huGlWkhRGXA1IDSUJl/F6edJxdqz+FuBsxw4QxxHmi/YqvXE2D+qCz8sOw9fhjcIjZKdvtLm4IFlY3uhTnbRq9Iz6uDXpYWXsvDVb3ID'
2020-10-07 17:32:34 - INFO - Creating Full key...
2020-10-07 17:32:34 - INFO - Full key created!, connection established
```

Figure 6: The log generated for 'bob' client entity.

3 Discussion

3.1 Part I

The log files present in the Jupyter notebook allow to follow and see the different key exchange processes simulating two users sending messages each other. It is possible to confirm that all the algorithms work together by confirming that without sharing their private keys both end up having the same full key and later on manage to encrypt and decipher the information arriving to the same result.

3.2 Part II

It is possible to see that under no circumstance the messages are read by the server in raw text mode, it is agnostic of the key exchange mechanism and only acts as a communication coordinator. Each client has its own implementation to generate their private keys and they do not share it, they just send the shared keys. The server is completely agnostic and works as a coordinator for both instances. Under the server log it is possible to visualize that it transmits the message encrypted. The log files for the client show how they generate their own private keys based on pseudo random generation and later on send their shared key. Once any message is sent a new private key is generated.

4 Conclusion

During the realization of this assignment it was possible to emulate and build a simple encyption comunication application by building from scratch the algorithms for: Key Exchange (Diffie-Hellman), CSPRNG (BBS) and the use of other AES symmetric encryption algorithm to encrypt the messages. Later on and through the use of Python language as well as socketio communication protocol it was possible to learn and confirm the importance of such encryption system for reliable and accurate communication as well as the considerable latency that strong encryption systems can generate. These tests were made on a normal computer without end-to-end communication, however the latency is perceptible for each communication mechanism.

A very important lesson learned from this analysis is the realization that encryption algorithms are complex and they add an extra layer of complexity for building applications. That said, it is very important to have a quality control rule for communication processes and to think how far it is possible to get into these encryption systems without affecting the latency of the communication applications.

A video presentation has been elaborated and can be visualized in this link.

References

[Stallings(2017)] William Stallings. *Cryptography And Network Security Principles And Practice*, volume 1, chapter Apendix G. Simplified DES. Pearson, Edinburgh Gate Harlow Essex CM20 2JE England, 7 2017.