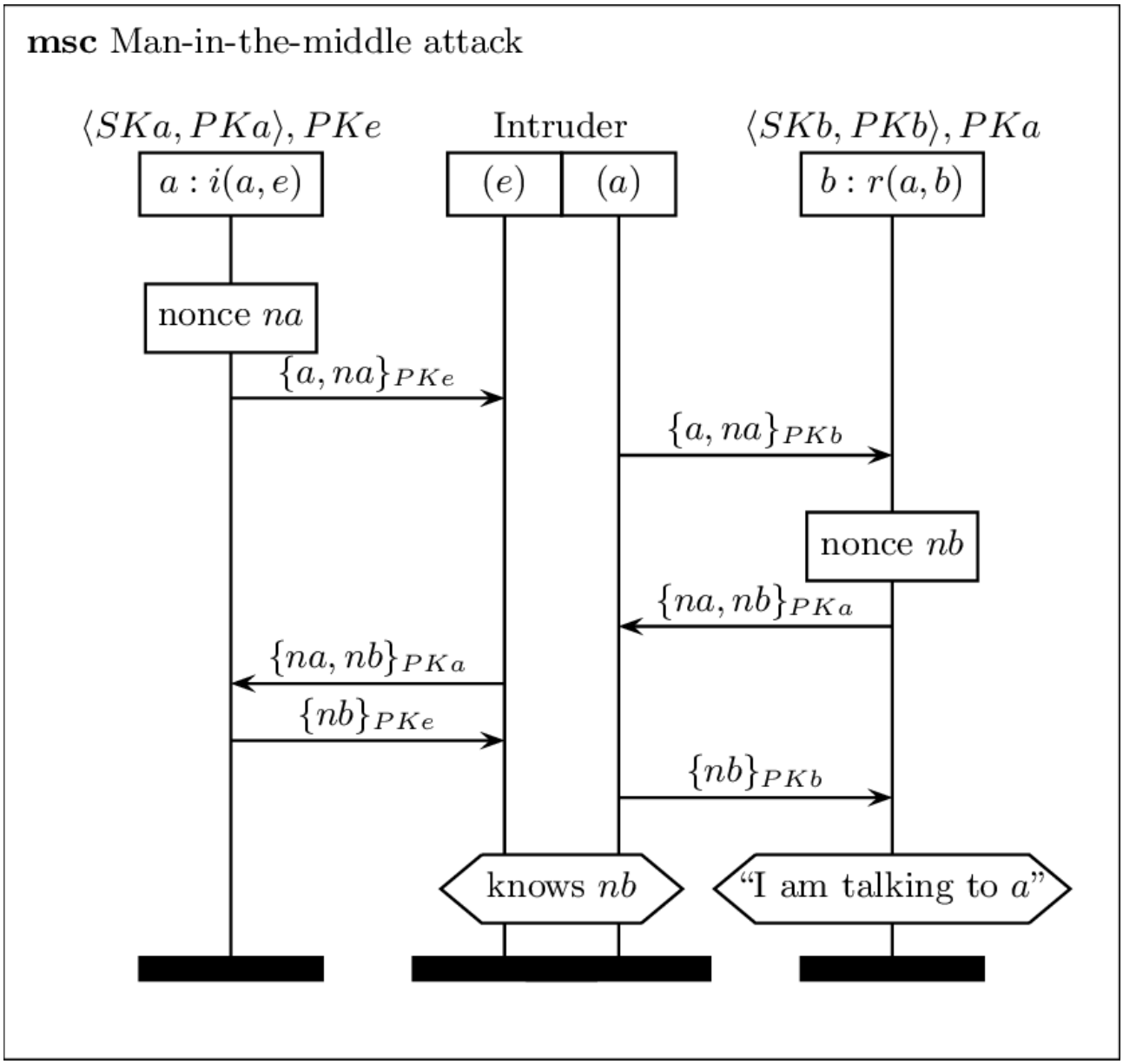
Internet security.

PROJECT SUMMARY.

By: Bate Ye.

1. The idea of my project:

My project is about a simple chat interface making with Sockets and doing the authentication with asymmetric cryptography and all that on C language.
And also a simulation of the Lowe attack.



Lowe-Attack, graphic.

2. What do I use for make this project:

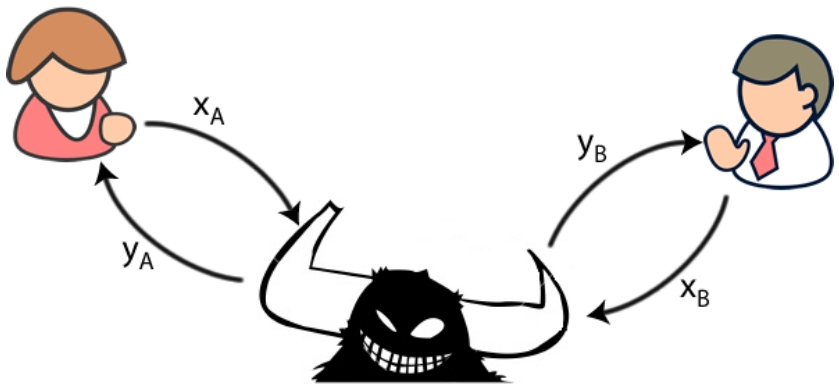
Well, the most important part of my project, is about how to encrypt a message m with a public key k, so for encrypt, I do use the library of OpenSSL for Ubuntu.
To be more exact, I use a lot of rsa functions of openssl/rsa.h library.
If you are using ubuntu (linux) you can get openssl with this set of commands (write it of the Terminal or console and them pulse ENTER):

1. For download do:
    - 1.1: cd /tmp
    - 1.2: wget https://www.openssl.org/source/openssl-1.1.1.tar.gz
    - 1.3: tar xvf openssl-1.1.1.tar.gz
2. After download, do:
    - 2.1: cd openssl-1.1.1
    - 2.2: sudo ./config -Wl,—enable-new-dtags,-rpath,'$(LIBRPATH)'
    - 2.3: sudo make
    - 2.4: sudo make install

You can also use: ‘sudo apt-get install’ openssl and them write your password and it download it and install it.

3. How does the protocol work.

The protocol has 3 agent, Alice, Bob and Ive.
Alice and Bob are normal agents while Ive is an attacker.
Alice will be the one who will be trying to connect to each another agent.



4. The agents:

First of all, I have an agent (client in socket) who will send messages to servers. I called this agent Alice.
Alice will just trying to connect to servers and then change messages with the servers.

Now for complete the protocol, I made an another agent whose call Bob and it is a server.
Bob will be running all the time and will be waiting for a connection of a client, and then he will authenticate with the client using nonce which are random numbers that will be use just once.

And for the attack, I created one third agent with name Ive. This agent at the beginning will be running like Bob, just waiting a connection, once have a connect client, he will ask you if you want to start an attack to another agent, the connected client will be Alice, and Ive will try to pose like Alice and cheat Bob, by resending Alice authentication messages to Bob and vice versa.

5. How did I make each agent?

Before making the agents, I have made a lot of .pem files.
.pem files are where I have store public keys and private keys of each agent.
For Alice I have: alicepub.pem and alice.pem, alicepub.pem is where Alice public key is, and alice.pem is where Alice private key is.
For Bob I have: bobp.pem and bob.pem, bobp.pem s where Bob public key is, bob.pem is where Bob private key is.
For Ive I have: ivep.pem and ive.pem, ivep.pem is where Ive public key is, ive.pem is where Ive private key is.

For make keys files I have use:
Openssl genrsa -out name.file 2048 (this one is for generate private and public keys.
Openssl rsa -in name.file -outform PEM -pubout -out namep.file (this one is for extract the public key to another file)

-Alice:

Alice is the client socket, for the socket I have use some variables:

int client_socket, s_addr_len;

struct sockaddr_in server_addr;

client_socket is an int variable that will be the connection, using this variable Alice can send and receive messages from the server.

s_addr_len is the length of server_addr.

server_addr, is the variable where I store the address of the server.

For initialize the client_socket, I use the function socket:

```
client_socket = socket(AF_INET, SOCK_STREAM, 0);
```

SOCK_STREAM indicate that we will use a Stream sockets.

For make the address I use a function mkaddr from mkaddr.c, this function receive server_addr, the name of the server and the port of the server like parameters, the function use the servername and the server port for make the address and store it inside the server_addr.
Example: `mkaddr(&server_addr, NAME, htons(PORT));`

The htons function is for transform the port number in net short form.

After making the server address, we will try to connect to the server with the function connect:

```
connect(client_socket, (struct sockaddr *) &server_addr, s_addr_len = sizeof(server_addr) );
```

Connect will fail if you don't have the server running before the client.

Now, Alice make two struct Encription.
struct Encription is a struct that have two variable, one string encrip where will store the encrypted message and one integer tamEn where will store the size of the encrypted message.

For encrypt the message, I use the function public_encrypt. public_encrypt receive as parameters the plain text, the length of the plain text, the server public key file name, and the encrypted (encrip of struct Encription).
Inside the public_encrypt() function, I have a RSA * variable that, using the function createRSAWithFilename(), this function receive the key file name and a integer, and return a rsa like result. And then I use RSA_public_encrypt() function for encrypt the message. Here the encrypted message will be the name of the first agent (Alice) and a nonce.
Nonce is generate with rand() function, for make it more random, I use srand(getpid()) at the beginning of the main function, this function make the rand() function giving different random numbers for each execution of the project.

After sending the first encrypt message, Alice will be waiting the answer of the server, which will be her nonce and one more nonce, is the server nonce. Alice first will decrypt the message with the function: private_decrypt(), this function receive as parameters a string which is the encrypted text, the size of the encrypted text, the private key file name, and a string which store the decrypted text. The function works like public_encrypt but changing the RSA_public_encrypt function by RSA_private_decrypt() for decrypt messages.
Later Alice prove that the server has send back her nonce, if yes, Alice can authenticate the server, and if not Alice will cancel the operation.
Alice at the end send back the server nonce to the server and if the server prove that the back nonce is his nonce, he will authenticate Alice and them, they start the conversation.

For the conversation, I make an infinite loop, that you can stop it with CTRL + C.
Inside the conversation loop, you can text your messages and them will be sending by the socket to the server.

-Bob:

For bob, I do almost the same as in Alice.

I declare this variables different than Alice:

int server_socket, connect_socket;
socklen_t client_addr_len;
struct sockaddr_in client_addr, server_addr;

Server_socket work like client_socket in Alice, initializing with the same function (socket(…)).
Now we will make the address server_addr with mkaddr function like in Alice.
After making the address, I bind it with the server_socket integer with bind function.
And now, I put the server listening connections with listen function.
Now when someone is trying to connect to the server, Bob can accept a connection with the function accept().
After accept the connection Bob receive the first encrypted message for authentication, and if he know whose message is it, Bob send back the receive nonce and his own nonce and wait to receive again, his nonce.
After the authentication, Bob do the same as Alice.

-Ive:

Ive at the beginning, works like Bob, until he receive the first authentication message, and will ask you if you want to start the attack, if you press yes, Ive will make another socket for connect it to Bob, once connect, Ive send the message received before to Bob (decrypting it before, and encrypting it with the public key of Bob).
Bob will decrypt the message, and will see the message, but that message is not really from Alice (in this case, Alice is connected with Ive), but Bob does not know it, so he will send back Alice's nonce with his nonce encrypted with Alice public key, this message will be received by Ive and he will resend it to Alice (because the message is encrypted with Alice public key and Ive can not decrypt it). Alice receive the message believing that is from Ive, she decrypt

the message and then she send to Ive Bob's nonce, and as now Ive have Bob's nonce, he will send it to Bob and make Bob think that he is Alice.

If you choose no doing the attack, Ive will comport with exactly the same as Bob with Alice.

6. Running the project.

For run this project, I compile all 3 agents (Alice, Bob and Ive).
For compile its, I use gcc compiler with options -lssl for use openssl libraries and -lcrypto for use crypto libraries.

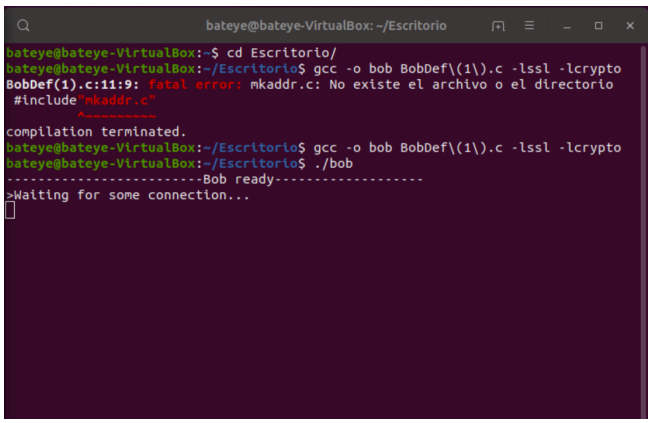
The entire compile line is:
gcc -o agent agent.c -lssl -lcrypto.

-o is for give a name to the executable file.
You have to compile and execute the project in the same directory than then .pem files because the programs will try to find the .pem files on their own directory.

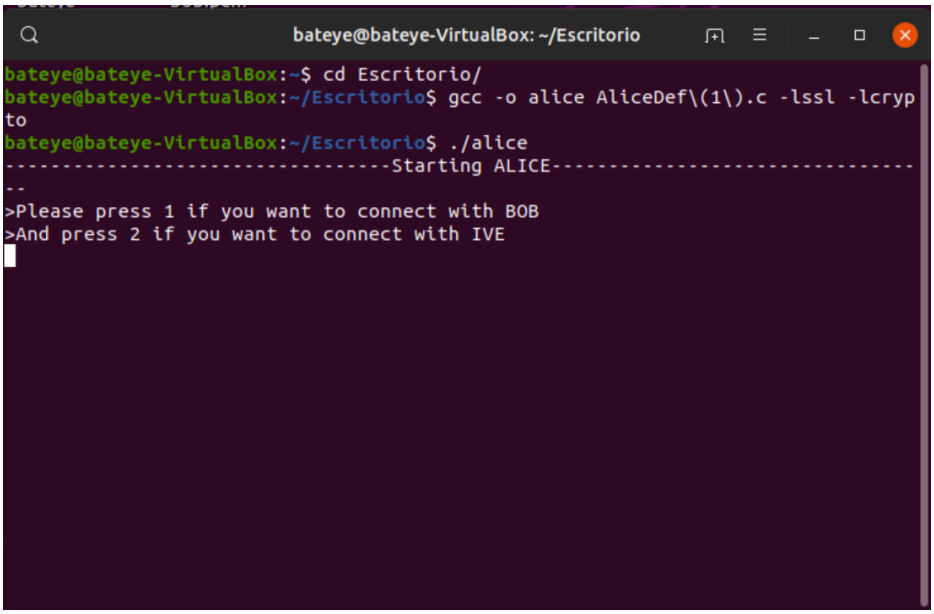
In this project, as I have a client/server socket, of course I have to run first the servers (Bob and Ive) before the client.

7. Results.

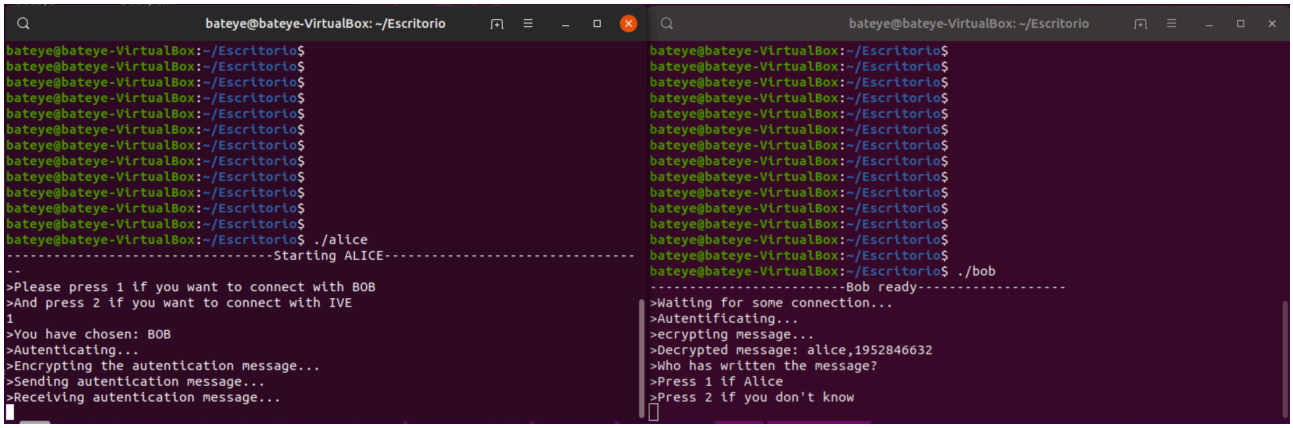
Here is some proves that I made.
For the normal protocol (without attack) I compile and run first Bob server.

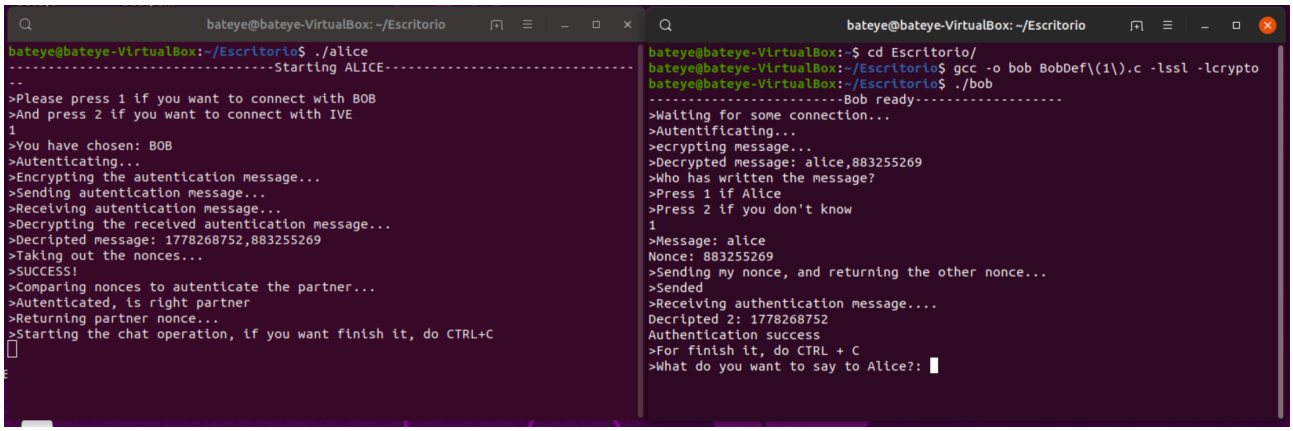


Then, I compile and run Alice:



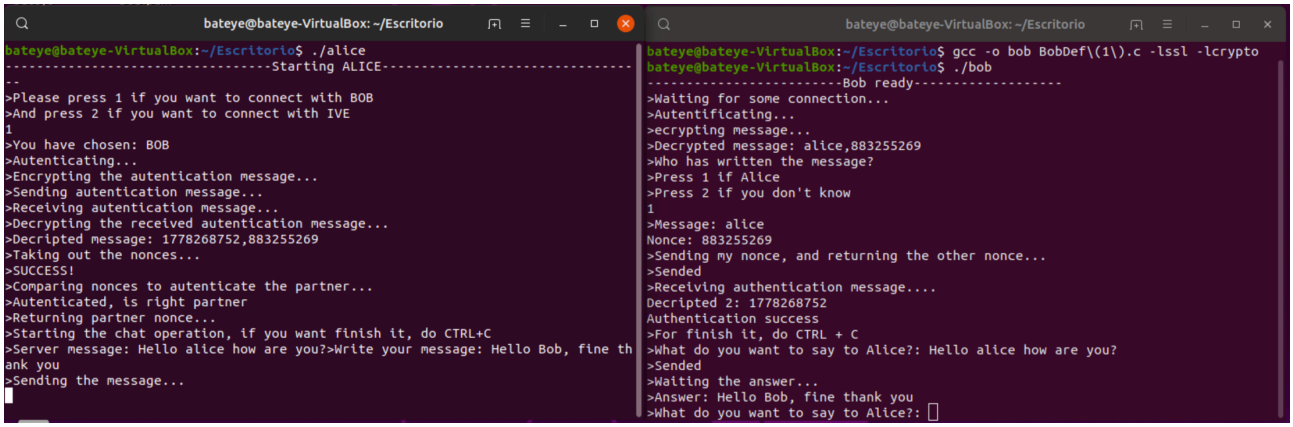
As you can see, when Alice is running, the first thing she do, is asking you with which server do you want to connect.
At this moment, I just press 1 and connect to Bob.





Alice after choosing the server, will start the authentication procedure.
In this procedure, Alice will generate a nonce and will send the nonce and her name to the server in an encrypted message.
Bob, once received the message, will decrypt it and will ask you if you know who has sent the message, if you know it, Bob will continue the process and if not, Bob will close the connection.

Once you press 1 in Bob to tell him that you know who is connecting, Bob will extract the nonce from the message, generate his own nonce and he will send back to the connected socket an encrypted message with his nonce and the before client nonce.

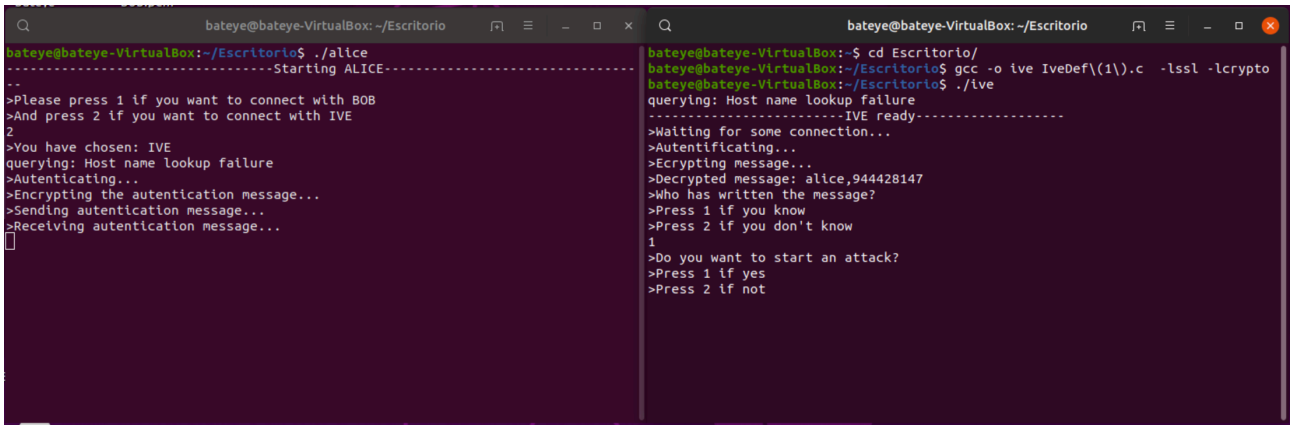


This encryption procedure will be made by using Alice's public key.

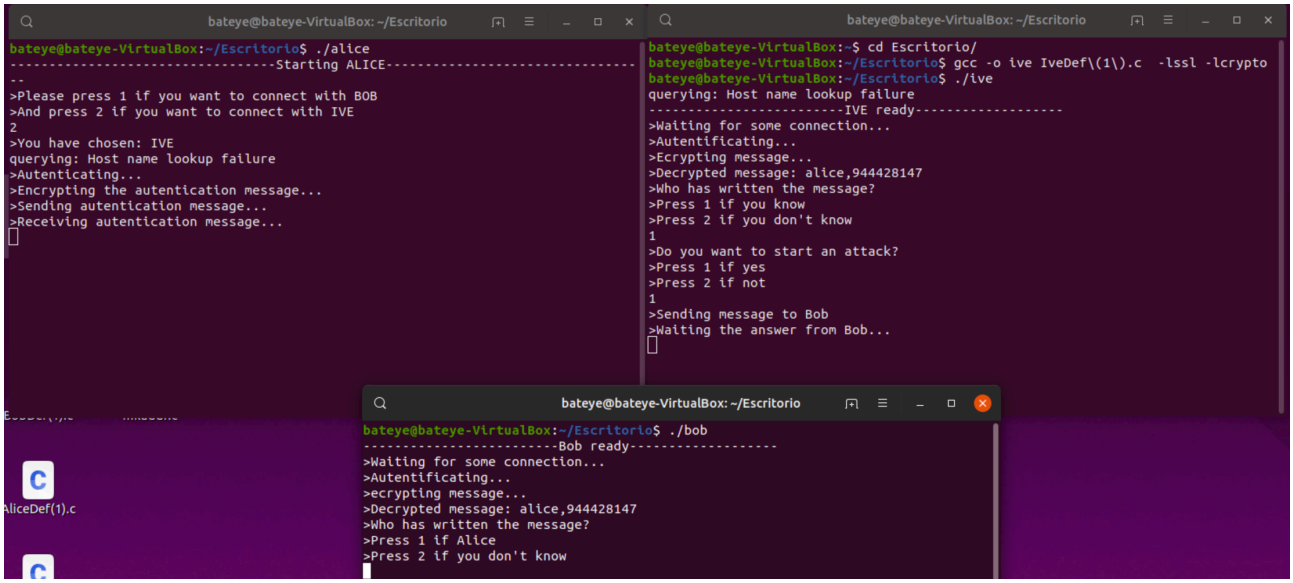
And Alice when she receive the message, she will decrypted that with her own private key, and then prove that Bob has sent back

her nonce, if yes, Alice can authenticate Bob and she will send back Bob's nonce in an encrypted message.
Bob after receive the last message, he will check that the message contain his nonce, if yes, Bob can authenticate Alice and then they can start a conversation.

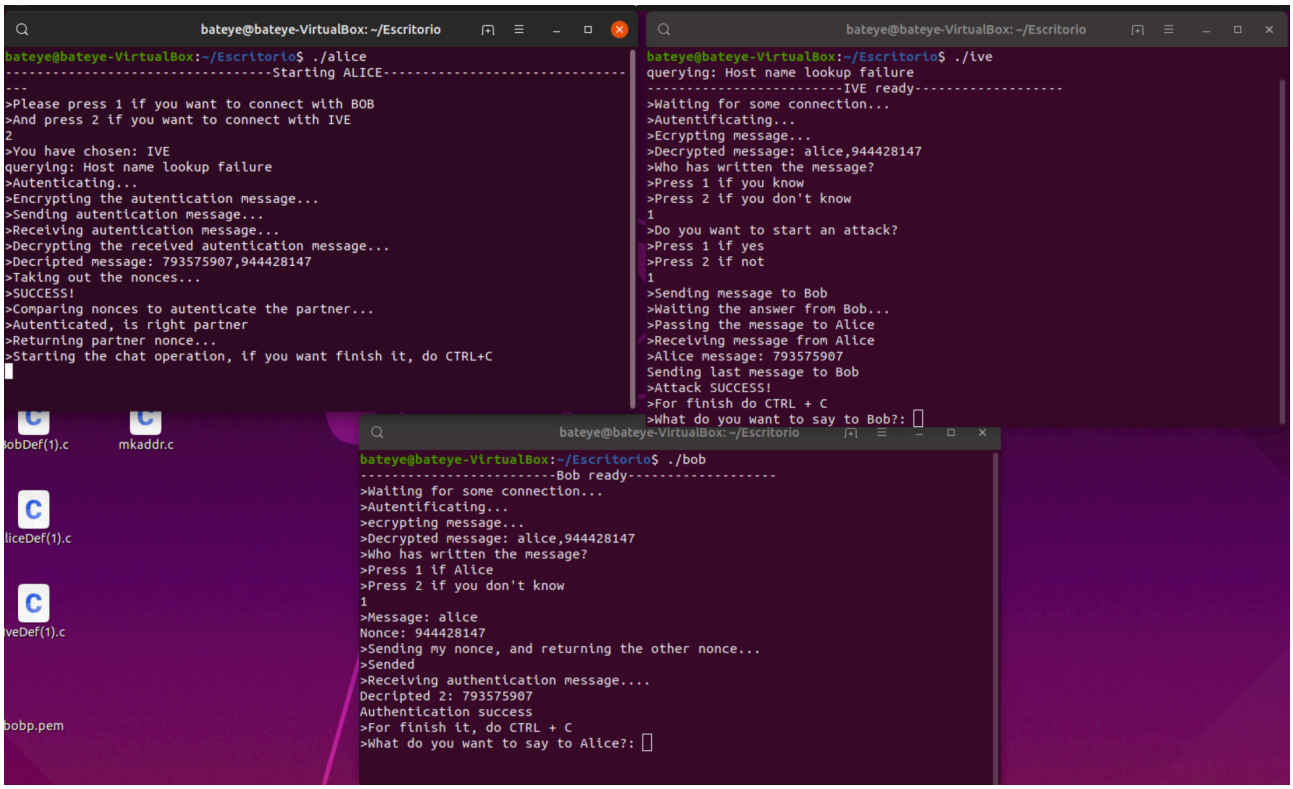
For the attack protocol, the only thing changed is that Alice will connect to Ive instead of Bob.



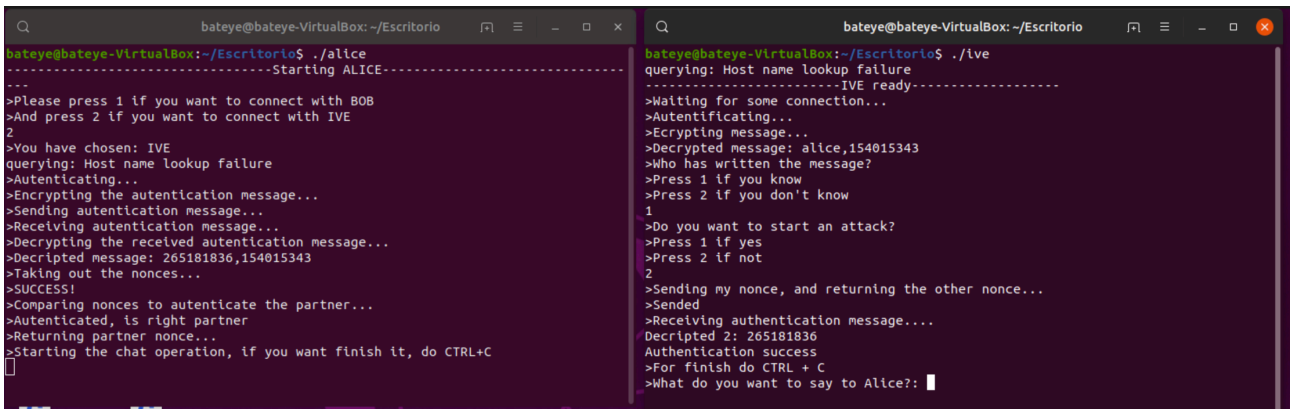
Ive has the same beginning than Bob.
But after asking if you know who is connecting to him (and you say yes you know), he will ask you if you want to start an attack.



If you press 1 (yes), Ive will resend the same message that Alice has sent him, to Bob (encrypted with Bob's public key of course). And Bob will do the same procedure as before (asking if you know who is trying to connect, send nonces and authenticating).



As you can see here, Ive has made the attack, because Bob does not know that the message was sending by Ive instead of Alice, so Bob just do the same as normal protocol, he send back the Alice nonce and his nonce to Ive, encrypting the message with Alice public key, Ive take the message and resend it directly to Alice, and Alice after decrypt the message receive from Ive, she will send Bob's nonce to Ive believing that the nonce received is from Ive and no one else.
Now Ive has Bob's nonce and he will send it to Bob.
Bob will check the nonce but he will not realize about any fail so he will think that he is talking with Alice.



That is how Ive will make the Lowe-attack.

If Ive does not want to start an attack and just want to have a conversation with Alice, that will work exactly like the normal protocol.

8. Websites that I consult.

https://www.openssl.org/docs/manmaster/man3/RSA_public_encrypt.html

https://www.openssl.org/docs/man1.0.2/man3/RSA_private_decrypt.html

https://stackoverflow.com

https://www.openssl.org/docs/man1.0.2/man3/PEM_read_RSA_PUBKEY.html