

Internet security.

PROJECT SUMMARY.

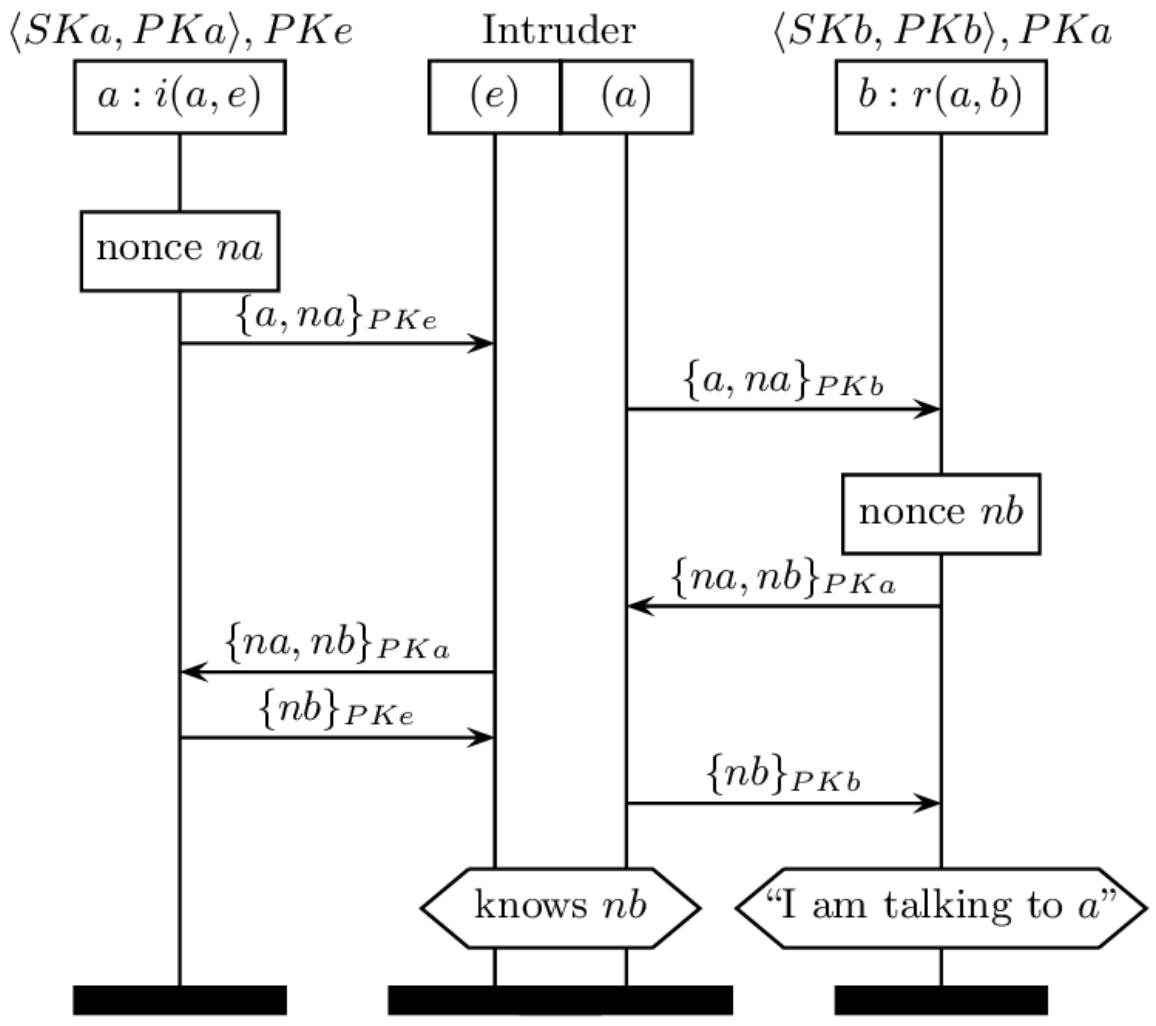
By: Bate Ye.

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.

msc Man-in-the-middle 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 Eve.

Alice and Bob are normal agents while Eve 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 try to connect to servers and then change messages with the servers.

Now to complete the protocol, I made 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 used just once.

And for the attack, I created one third agent with name Eve. This agent at the beginning will be running like Bob, just waiting a connection, once have a connected client, he will ask you if you want to start an attack to another agent, the connected client will be Alice, and Eve 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 stored public keys and private keys of each agent.

For Alice I have: alice.pub.pem and alice.pem, alice.pub.pem is where Alice public key is, and alice.pem is where Alice private key is.

For Bob I have: bob.pub.pem and bob.pem, bob.pub.pem is where Bob public key is, bob.pem is where Bob private key is.

For Eve I have: eve.pub.pem and eve.pem, eve.pub.pem is where Eve public key is, eve.pem is where Eve private key is.

For make keys files I have used:

Openssl genrsa -out name.pem 2048 (this one is for generate private and public keys).

Openssl rsa -in name.pem -outform PEM -pubout -out namepub.pem (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 Encryption.

struct Encryption 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 `Encryption`).

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 Eve Bob's nonce, and as now Eve 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, Eve 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 Eve). 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 Eve) 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.

```

bateye@bateye-VirtualBox:~/Escritorio$ cd Escritorio/
bateye@bateye-VirtualBox:~/Escritorio$ gcc -o bob BobDef\(\1\).c -lssl -lcrypto
BobDef\(\1\).c:1:9: Fatal error: mkaddr.c: No existe el archivo o el directorio
#include "mkaddr.c"
^
compilation terminated.
bateye@bateye-VirtualBox:~/Escritorio$ gcc -o bob BobDef\(\1\).c -lssl -lcrypto
bateye@bateye-VirtualBox:~/Escritorio$ ./bob
-----Bob ready-----
>Waiting for some connection...

```

Then, I compile and run Alice:

```

bateye@bateye-VirtualBox:~/Escritorio$ cd Escritorio/
bateye@bateye-VirtualBox:~/Escritorio$ gcc -o alice AliceDef\(\1\).c -lssl -lcrypto
bateye@bateye-VirtualBox:~/Escritorio$ ./alice
-----Starting ALICE-----
-->
>Please press 1 if you want to connect with BOB
>And press 2 if you want to connect with IVE

```

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.

```

bateye@bateye-VirtualBox:~/Escritorio$ ./alice
-----Starting ALICE-----
-->
>Please press 1 if you want to connect with BOB
>And press 2 if you want to connect with IVE
1
>You have chosen: BOB
>Autenticating...
>Encrypting the authentication message...
>Sending authentication message...
>Receiving authentication message...
bateye@bateye-VirtualBox:~/Escritorio$ ./bob
-----Bob ready-----
>Waiting for some connection...
>Autenticating...
>Decrypting message...
>Decrypted message: alice,1952846632
>Who has written the message?
>Press 1 if Alice
>Press 2 if you don't know

```

```

bateye@bateye-VirtualBox:~/Escritorio$ ./alice
-----Starting ALICE-----
...
>Please press 1 if you want to connect with BOB
>And press 2 if you want to connect with IVE
1
>You have chosen: BOB
>Autenticating...
>Encrypting the autentication message...
>Sending autentication message...
>Receiving autentication message...
>Decrypting the received autentication message...
>Decrypted message: 1778268752,883255269
>Taking out the nonces...
>SUCCESS!
>Comparing nonces to autenticate the partner...
>Authenticated, is right partner
>Returning partner nonce...
>Starting the chat operation, if you want finish it, do CTRL+C
[]

bateye@bateye-VirtualBox:~/Escritorio$ cd Escritorio/
bateye@bateye-VirtualBox:~/Escritorio$ gcc -o bob BobDef\(\1\).c -lssl -lcrypto
bateye@bateye-VirtualBox:~/Escritorio$ ./bob
-----Bob ready-----
>Waiting for some connection...
>Autenticating...
>Encrypting message...
>Decrypted message: alice,883255269
>Who has written the message?
>Press 1 if Alice
>Press 2 if you don't know
1
>Message: alice
Nonce: 883255269
>Sending my nonce, and returning the other nonce...
>Sended
>Receiving authentication message....
Decrypted 2: 1778268752
Authentication success
>For finish it, do CTRL + C
>What do you want to say to Alice?: []

```

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.

```

bateye@bateye-VirtualBox:~/Escritorio$ ./alice
-----Starting ALICE-----
...
>Please press 1 if you want to connect with BOB
>And press 2 if you want to connect with IVE
1
>You have chosen: BOB
>Autenticating...
>Encrypting the autentication message...
>Sending autentication message...
>Receiving autentication message...
>Decrypting the received autentication message...
>Decrypted message: 1778268752,883255269
>Taking out the nonces...
>SUCCESS!
>Comparing nonces to autenticate the partner...
>Authenticated, is right partner
>Returning partner nonce...
>Starting the chat operation, if you want finish it, do CTRL+C
>Server message: Hello alice how are you?>Write your message: Hello Bob, fine th
ank you
>Sending the message...
[]

bateye@bateye-VirtualBox:~/Escritorio$ gcc -o bob BobDef\(\1\).c -lssl -lcrypto
bateye@bateye-VirtualBox:~/Escritorio$ ./bob
-----Bob ready-----
>Waiting for some connection...
>Autenticating...
>Encrypting message...
>Decrypted message: alice,883255269
>Who has written the message?
>Press 1 if Alice
>Press 2 if you don't know
1
>Message: alice
Nonce: 883255269
>Sending my nonce, and returning the other nonce...
>Sended
>Receiving authentication message....
Decrypted 2: 1778268752
Authentication success
>For finish it, do CTRL + C
>What do you want to say to Alice?: Hello alice how are you?
>Sended
>Waiting the answer...
>Answer: Hello Bob, fine thank you
>What do you want to say to Alice?: []

```

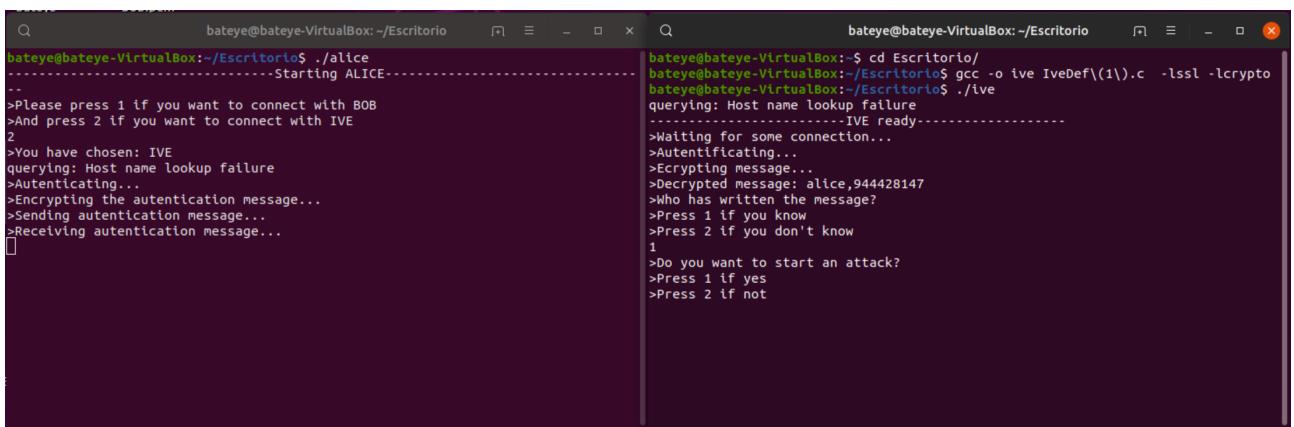
This encryption procedure will be made by using Alice's public key.

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.

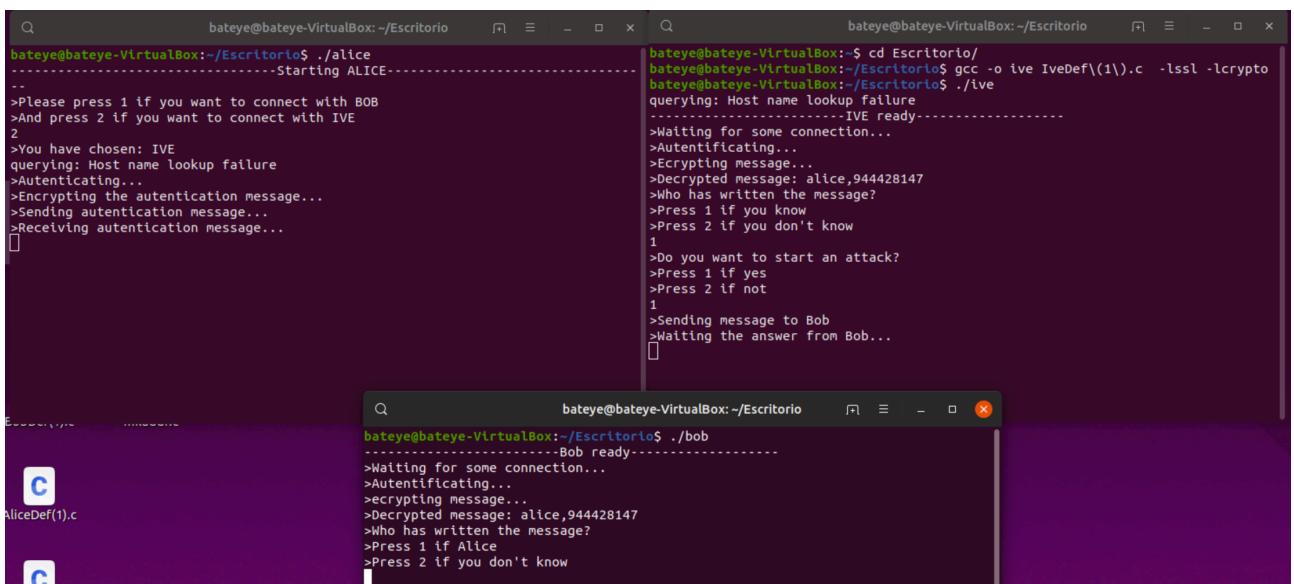
For the attack protocol, the only thing changed is that Alice will connect to IVE instead of Bob.



```
bateye@bateye-VirtualBox:~/Escritorio$ ./alice
-----Starting ALICE-----
--Please press 1 if you want to connect with BOB
>And press 2 if you want to connect with IVE
2
>You have chosen: IVE
querying: Host name lookup failure
>Autenticating...
>Encrypting the autentication message...
>Sending autentication message...
>Receiving autentication message...
1
bateye@bateye-VirtualBox:~/Escritorio$ cd Escritorio/
bateye@bateye-VirtualBox:~/Escritorio$ gcc -o ive IveDef\(\1\).c -lssl -lcrypto
bateye@bateye-VirtualBox:~/Escritorio$ ./ive
querying: Host name lookup failure
-----IVE ready-----
>Waiting for some connection...
>Autenticating...
>Encrypting message...
>Decrypted message: alice,944428147
>Who has written the message?
>Press 1 if you know
>Press 2 if you don't know
1
>Do you want to start an attack?
>Press 1 if yes
>Press 2 if not
```

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.



```
bateye@bateye-VirtualBox:~/Escritorio$ ./alice
-----Starting ALICE-----
--Please press 1 if you want to connect with BOB
>And press 2 if you want to connect with IVE
2
>You have chosen: IVE
querying: Host name lookup failure
>Autenticating...
>Encrypting the autentication message...
>Sending autentication message...
>Receiving autentication message...
1
bateye@bateye-VirtualBox:~/Escritorio$ cd Escritorio/
bateye@bateye-VirtualBox:~/Escritorio$ gcc -o ive IveDef\(\1\).c -lssl -lcrypto
bateye@bateye-VirtualBox:~/Escritorio$ ./ive
querying: Host name lookup failure
-----IVE ready-----
>Waiting for some connection...
>Autenticating...
>Encrypting message...
>Decrypted message: alice,944428147
>Who has written the message?
>Press 1 if you know
>Press 2 if you don't know
1
>Do you want to start an attack?
>Press 1 if yes
>Press 2 if not
1
>Sending message to Bob
>Waiting the answer from Bob...
1
bateye@bateye-VirtualBox:~/Escritorio$ ./bob
-----Bob ready-----
>Waiting for some connection...
>Autenticating...
>Encrypting message...
>Decrypted message: alice,944428147
>Who has written the message?
>Press 1 if Alice
>Press 2 if you don't know
```

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).

The screenshot shows three terminal windows on a Linux desktop environment. The desktop background is purple, and there are icons for 'bobDef(1).c', 'mkaddr.c', 'aliceDef(1).c', 'eveDef(1).c', and 'bobp.pem' on the left.

- Terminal 1 (Left):** Shows the execution of the Alice program: `bateye@bateye-VirtualBox:~/Escritorio$./alice`. It prints a welcome message and asks for a connection choice (1 for Bob, 2 for IVE). The user inputs '2'. It then prints the chosen partner (IVE) and begins the authentication process, including encrypting messages and receiving responses from IVE.
- Terminal 2 (Top Right):** Shows the execution of the IVE program: `bateye@bateye-VirtualBox:~/Escritorio$./ive`. It prints a host name lookup failure, indicating it is ready. It waits for a connection from Bob. It then performs its own authentication steps, including sending a message to Bob and receiving a response.
- Terminal 3 (Bottom Right):** Shows the execution of the Bob program: `bateye@bateye-VirtualBox:~/Escritorio$./bob`. It prints a welcome message and asks for a connection choice (1 for Alice, 2 for IVE). The user inputs '1'. It then performs its own authentication steps, including sending a message to Alice and receiving a response.

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.

The image shows two terminal windows side-by-side. The left terminal window, titled 'bateye@bateye-VirtualBox: ~/Escritorio', runs the command './alice'. It displays a series of messages from the program, including prompts for connection selection (press 1 for BOB, 2 for IVE), the selection of IVE, and the process of encrypting, decrypting, and verifying an authentication message. The right terminal window, also titled 'bateye@bateye-VirtualBox: ~/Escritorio', runs the command './ive'. It shows a similar sequence of messages, including a host name lookup failure, waiting for a connection, and a successful decryption of the message 'alice,154015343'.

```

bateye@bateye-VirtualBox:~/Escritorio$ ./alice
-----
----- Starting ALICE -----
-----
>Please press 1 if you want to connect with BOB
>And press 2 if you want to connect with IVE
2
>You have chosen: IVE
querying: Host name lookup failure
>Autentificating...
>Encrypting the autentication message...
>Sending autentication message...
>Receiving autentication message...
>Decrypting the received autentication message...
>Decrypted message: 265181836,154015343
>Taking out the nonces...
>SUCCESS!
>Comparing nonces to autenticate the partner...
>Authenticated, is right partner
>Returning partner nonce...
>Starting the chat operation, if you want finish it, do CTRL+C
[]

bateye@bateye-VirtualBox:~/Escritorio$ ./ive
-----
----- IVE ready -----
-----
querying: Host name lookup failure
>Waiting for some connection...
>Autentificating...
>Encrypting message...
>Decrypted message: alice,154015343
>Who has written the message?
>Press 1 if you know
>Press 2 if you don't know
1
>Do you want to start an attack?
>Press 1 if yes
>Press 2 if not
2
>Sending my nonce, and returning the other nonce...
>Sended
>Receiving authentication message....
>Decrypted 2: 265181836
Authentication success
>For finish do CTRL + C
>What do you want to say to Alice?: []

```

That is how I've will make the Lowe-attack.

If I've 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/manmaster/man3/RSA_public_encrypt.html)

[https://www.openssl.org/docs/man1.0.2/man3/
RSA_private_decrypt.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](https://www.openssl.org/docs/man1.0.2/man3/PEM_read_RSA_PUBKEY.html)

