CSCI 3403: Project 3

Emulating SSL and Password Verification

Matt Niemiec and Abigail Fernandes

Due: 12/xx/18

Background

While no knowledge of networking is required for this project, you are encouraged to know the basics of sockets. Feel free to refer to the following link and other similar articles to learn about the details that are already implemented here. You are also given code related to Python file I/O, though you should have experience with this from your introductory coursework. Once again, these portions are completely implemented for you.

Goal

The purpose of this assignment is to give you a deeper insight into how everything secure on the internet works, including HTTPS and SSH. So, the purpose is to create a completely encrypted interaction between a client and server that is completely confidential and has complete integrity.

To begin with you're provided a basic client/server program, as well as some placeholder functions to give you an idea of what you'll need to do to implement your program. Please note that the complete solution uses exactly these functions, but you should feel free to add or remove functions if you feel it is necessary for your implementation. Parts of the main function are also provided, but, particularly on the client side, you may need to implement some things yourself. You will not need to change any of the networking functionality, unless your
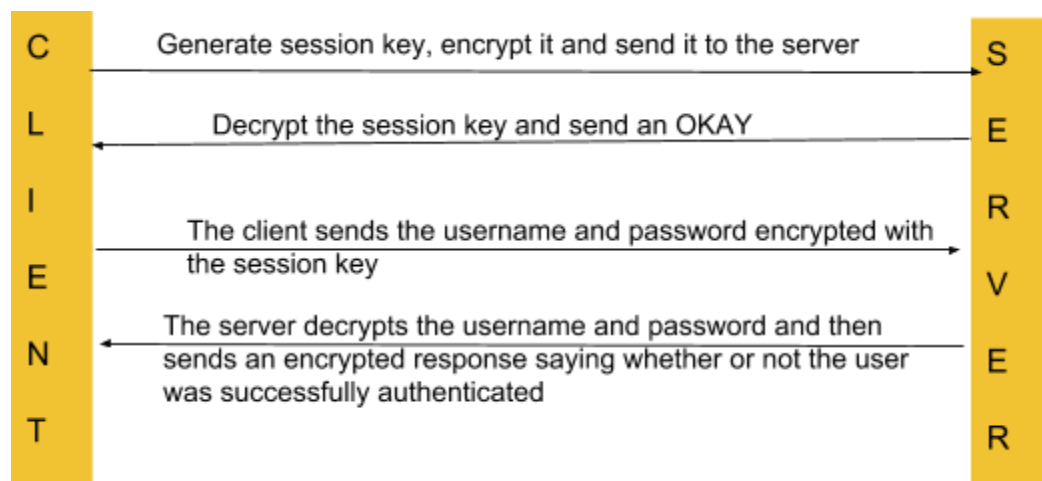

Figure 1

implementation specifically requires you to do so. You are also provided an **add_user.py**. This does not need to be changed in any way. All it does is take in a username and password and store the username, salt, and hash.
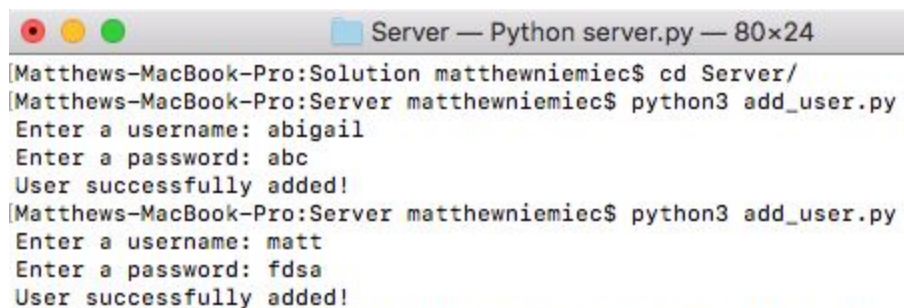
You may refer to Figure 1 for a visual representation of what your code should be doing. First, the server will listen for incoming connections. Next, the client will generate a random AES key, encrypt it with the server's public key, and send that to the server. From this point onward all communication will be done over AES with this key. Next the server will respond with an "okay" - this may be encrypted, but it isn't necessary, since SSL doesn't protect against availability attacks, and this isn't sensitive information. Next the client will send the username and password to the server. The server will take this and compare it to the stored hash file to check if the username and password are correct. The server will send back an encrypted response accordingly.

Of course, in a real-life scenario, this would just be the start of the communication. However, for the purposes of this assignment, after this both the client and the server will tear down the connection. The client process will close and the server process will resume looking for connections. Note that this server, unless you choose to change it, is not multi-threaded, so it can only handle one connection at a time. This is intended to be a rudimentary example.

Setup

You will be expected to use Python3 for this project. You may also have to download appropriate related libraries. To get started on the project, please refer to the following steps:

1. Create a separate folder for the Client and Server and place the files given to you as well as the files you generate accordingly.
2. You now need to generate SSH keys. You may look into **ssh-keygen** as one way of doing it. You will need the public and private keys to encrypt/decrypt the session key.
3. Create users to be added to the passfile.txt file. You will need this to get your client-server running. You will do this by first running the **add_user.py**. You can see below what you should expect to see.


```
[Matthews-MacBook-Pro:Solution matthewniemiec$ cd Server/
[Matthews-MacBook-Pro:Server matthewniemiec$ python3 add_user.py
 Enter a username: abigail
 Enter a password: abc
 User successfully added!
[Matthews-MacBook-Pro:Server matthewniemiec$ python3 add_user.py
 Enter a username: matt
 Enter a password: fdsa
 User successfully added!
```

4. Now open the server.py file and client.py. You will find a list of TODOs that you are required to implement. The client - server interaction is depicted in Figure 1.

Python Libraries that could be useful to implement the TODO functions:
- pycrypto
- hashlib
- uuid

Results

The screenshots below demonstrate the end functionality you should expect to see when running your program. **Note that they do not explicitly demonstrate every step.** Also note that, while printed in plaintext, everything that is sent over the network is ciphertext (except for the "okay"). When printed in Python, you can expect your ciphertext to look something like this: b'\xe4n\x91\x06\xa4(\x89\x99\xbc\xf8e\xd3\xd7\xf7uA', though the encrypted session key will be much longer.

```
[Matthews-MacBook-Pro:Server matthewniemiec$ python3 server.py
starting up on localhost port 10001
waiting for a connection
connection from ('127.0.0.1', 49478)
matt fdsa
waiting for a connection
connection from ('127.0.0.1', 49480)
abigail cba
waiting for a connection
connection from ('127.0.0.1', 49485)
userdne pass
waiting for a connection
```

```
● ● ●                    Client — -bash — 80×24
[Matthews-MacBook-Pro:Solution matthewniemiec$ cd Client/
[Matthews-MacBook-Pro:Client matthewniemiec$ python3 client.py
What's your username? matt
What's your password? fdsa
connecting to localhost port 10001
User successfully authenticated!
closing socket
[Matthews-MacBook-Pro:Client matthewniemiec$ python3 client.py
What's your username? abigail
What's your password? cba
connecting to localhost port 10001
Password or username incorrect
closing socket
[Matthews-MacBook-Pro:Client matthewniemiec$ python3 client.py
What's your username? userdne
What's your password? pass
connecting to localhost port 10001
Password or username incorrect
closing socket
Matthews-MacBook-Pro:Client matthewniemiec$ ▯
```

Submission

Upload a zipped folder of your entire project. It should contain the following:

1. Folders for client and server with the required files
2. SSH keys you generated
3. Passfile.txt
4. All the other files you may add as part of your implementation.
5. A **report** explaining what you did, the libraries you used and finally add a section on what you learned. The report must include an easily readable **screenshot** depicting the client-server interaction and demonstrating that the interaction is indeed encrypted. Alternatively this could be done using Wireshark. A 1-2 page report explaining all of this is sufficient.