CMPUT 361 – Assignment 3
Secure File Transfer Protocol

Weight: 13% Total points: 130 points Due: November 30th, 2019 and before 11 PM

I. Objectives:

In this assignment you will work in teams. Each team **MUST** have **THREE** members. The members **MUST** belong to the same lab section. ONLY one group can have TWO members from X01L lab and ONE member from X02L lab. Each group must register their names using the posted group registration links on the BlackBoard.
Each team will develop a reasonable secure file transfer protocol. In addition, each team **MUST** solve section **V**. During your work in this assignment, you will learn:

1- How to program both clients and servers in a UNIX-like environment
2- How to allow the server to handle multiple clients using fork function to create multiple processes
3- How to reasonably secure your application
4- How to identify some possible attacks against your developed protocol and how to enhance this protocol to defend against these attacks.

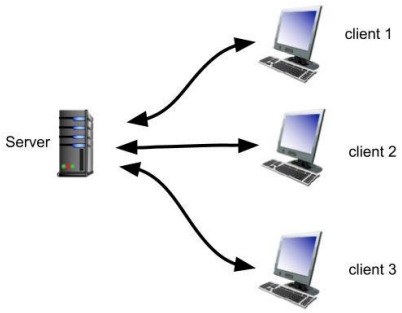
II. Suggestions for group work:

Here are some suggestions for working in a group:

- be **respectful** of your peers: different people have different backgrounds
- agree on some **expectations** (or rules) for your group
  - discuss the possibility of non- or under-contributing members
- work in **parallel**
  - define modules and function headers early, including function names, arguments, behaviour, and return values
  - implement the functions in **parallel**
- set agreeable and reasonable **deadlines** for tasks
  - have each member deliver one or two reasonable tasks every week
  - try to make significant progress early in the project

III. Description:

In this assignment, you are going to develop a reasonably secure file transfer protocol that works between a server and three known clients to this server. The known clients wish to securely upload some text files to the server by using TCP connections as shown in the following figure:



During your work in this part, you will develop two python 3 programs:

- Server.py: which runs in the server machine
- Client.py: which runs in the clients' machines

A. Definitions

- Known client: a user of the client machine who is authorized to upload files to the server and is using client.py
- Unknown client: a user who is **NOT** authorized to upload files to the server
- Client application (Client.py): the client program that runs on the known client machines.
- Server program (Server.py): the program running on the server machine
- Server user: the user who is running the server program.
- MAX_FILE_SIZE: The maximum file size (in terms of the number of characters stored inside this file) that the server can accept in one upload connection.

B. Server-side description:

- The known clients have the following names: client1, client2 and client3
- The server machine has:
  - Prestored public and private keys for this machine called: server_public.pem and server_private.pem
- For each of the 3 known clients, the server machine has:
  - the client's public key stored as [client_name]_public.pem
  - a folder with the client name
- All server keys, server program, clients' folders and keys should be stored in the same directory of the server program.
- The server should listen to clients' connections on port number 13000
- The server **MUST** be able to serve all known clients simultaneously. You **MUST** use fork to achieve this purpose.
- At the start of running the server program (Server.py), the server user should specify MAX_FILE_SIZE

C. Known client-side description:

- Each known client **MUST**:
  - Know the server IP address (or hostname) and the server public key

  - Have a private key and a public key with the names *[client_name]_private.pem* and *[client_name]_public.pem*
  - Know his/her client name that is kept in the server machine
- Each client machine should store in the same directory the following:
  - the client public key and private key (called *[client_name]_private.pem* and *[client_name]_public.pem*)
  - The server public key called *server_public.pem*
  - The client program called *Client.py*
  - The text file(s) that the client wish to transmit to the server.
- At the start of running the client program, the client should specify the server's IP address (or hostname)

D. <u>Protocol:</u>

After the connection is established between a known client and the server. The following protocol takes place:

1- **[Client side]** The client program asks the client to enter his/her name. Then the client program encrypts this name with **the server public key**, sends the encrypted name to the server side, and prints the following message to the client *"Client name [client_name] is sent to the server."*

2- **[Server side]** After receiving the client name and decrypting it, the server checks whether this name is one of the known client folders' names:

   a. If yes,
      - The server program generates a 256 AES key (called *sym_key*) and send it to the client encrypted with the corresponding **client public key**.
      - Then the server program prints on the server screen the following message "Connection Accepted and Symmetric Key Generated for client: [client_name]"
   b. If no,
      - The server program sends the following unencrypted message to the client: "Invalid clientName"
      - The server then prints the following message "The received client: [client_name] is invalid (Connection Terminated)." And terminate the connection with the client.

3- **[Client side]** after the client receives the server response:

   a. If the received response is "Invalid clientName", the client program prints to the client the message "Invalid client name.\nTerminating." and terminate the connection with the server.
   b. Otherwise, the client decrypts the message and stores the received symmetric key (as *sym_key)*, print "Received the symmetric key from the server." and wait for the next server message.

4- **[Server side]** After validating the client identity, the server sends the client the message "Enter filename:" encrypted with ***sym_key*** and prints to the server screen "Server is now processing client [client_name] request"

5- **[Client side]** After receiving server request of step 4:
   a. the client program decrypts the message using ***sym_key*** and print this message to ask the client to enter a file name.
   b. after the client enters the file name to be uploaded, the client program sends this file name encrypted with ***sym_key*** to the server.
6- **[Server side]** after receiving the file name and decrypting it, the server program prints "The server received the file name *[received file name]* from client: *[client_name]*" and sends to the client the message "Server Requested file size" encrypted with ***sym_key.***
7- **[Client side]** After receiving server request of step 6:
   a. The client program prints the server's decrypted message
   b. The client program reads the referenced file, counts the number of characters inside the file and sends this count encrypted with ***sym_key*** to the server.
8- **[Server side]** After receiving the file size and decrypting it:
   a. If file size > MAX_FILE_SIZE:
      - The server sends to the client the message "No" encrypted with ***sym_key,*** print the message *"File size exceed the threshold. Terminating Connection with [client_name]"* to the server screen and terminate the connection with the client.
   b. Otherwise,
      - The server sends to the client the message "OK" encrypted with ***sym_key,*** prints the message *"Uploading data from: [client_name]"* to the server screen.
9- **[Client side]** after the client receives the server response and decrypts it:
   a. If the received response is *"No",* the client program prints to the client the message *"The file size is too large.\nTerminating"* and terminate the connection with the server.
   b. Otherwise, the client program encrypts the file contents with ***sym_key*** and sends it to the server. Then, the client program prints the message *"The file size is OK.\nSending the file contents to the server.\n The file is saved."* To the client screen.
10- **[Server side]** After receiving the full file contents and decrypting it, the server program should write the file with its contents to the appropriate client directory and print the following message to the client screen *"Upload complete for: [client_name] Terminating connection"*
11- Both the server and the client program should terminate the connection.

Important notes:

- While using symmetric key encryption, you must use it in Electronic Code Book (ECB).
- In step 10, the server program should not terminate the connection until it makes sure that it receives the full file. Therefore, the number of received characters of the file is equal to the previously received file size.
- [client_name] should be replaced with the corresponding client name.

IV. Sample output:

1- Client 4 (Unknown client):

```
$ python3 client.py
Enter the server IP or name: localhost
Enter your client name: client4
Client name client4 is sent to the server.
Invalid client name.
Terminating.
```

2- Client 2 (File size exceeds threshold):

```
$ python3 client.py
Enter the server IP or name: localhost
Enter your client name: client2
Client name client2 is sent to the server.
Received the symmetric key from the server.
Enter filename:
b.txt
Server Requested file size
The file size is too large.
Terminating
```

3- Client 1 (known client with acceptable file size):

```
$ python3 client.py
Enter the server IP or name: localhost
Enter your client name: client1
Client name client1 is sent to the server.
Received the symmetric key from the server.
Enter filename:
a.txt
Server Requested file size
The file size is OK.
Sending the file contents to the server.
The file is saved.
```

4- Server:

```
$ python3 Server.py
Enter the permitted file size: 40
The server is ready to accept connections
The received client: client4 is invalid (Connection Terminated).
Connection Accepted and Symmetric Key Generated for client: client2
Server is now processing client client2 request
Connection Accepted and Symmetric Key Generated for client: client1
Server is now processing client client1 request
The server received the file name b.txt from client: client2
File size exceed the threshold. Terminating Connection with client2
The server received the file name a.txt from client: client1
Uploading data from: client1
Upload complete for: client1. Terminating connection
```

V. Protocol analysis and Enhancement

The developed protocol described in section III provides a reasonable security level. Although, some attacks still can work against it and may cause problems.

1- Identify ONE type of these attacks and show how this attack can affect the client and the server sides.
2- For the identified attack type, describe a modification to the developed protocol to defend against this attack.
3- Use your developed programs in section 3 to develop two programs *Server_enhanced.py* and *Client_enhanced.py* that have your suggested modified protocol.

VI. Instructions:

1- The developed programs (*Server.py, Client.py, Server_enhanced.py and Client_enhanced.py*):
   a. Must run with Python 3
   b. Must be well documented. The grader should be able to know the detailed logic of your code by reading the comments provided.
   c. Must ONLY import the following modules (Your developed programs MUST NOT import any other module):
      import socket
      import os
      import sys
      Any module from Crypto library (https://pycryptodome.readthedocs.io/en/latest/src/installation.html)
   d. Must be able to transfer any file size if it is less than or equal to MAX_FILE_SIZE
   e. The server program MUST be able to handle the three known clients concurrently using fork.

2- You MUST write a python script called key_generator.py to generate public/private keys for the server and the trusted clients and store these keys in files with the names provided in section C.

3- Your submission should contain the following (**ONLY ONE** member from each group **MUST** submit them to his/her lab Black Board):
   a. The developed programs (*Server.py, Client.py, Server_enhanced.py and Client_enhanced.py*)
   b. A word/PDF file indicating:
      - The names of the group members
      - the status of your developed programs and any deficiencies in these developed programs.
      - The tests conducted to verify the programs operations (You must conduct some tests with multiple lab machines)
      - For section **V,** The identified attack type and the proposed enhanced protocol
   c. key_generator.py and ALL the generated public/private keys.

4- The submission should be done to the BlackBoard before the announced due date and time.

5- Each group will present a demonstration for their developed work to their lab instructor during the lab time on the week of Monday 2nd December 2019.

VII. Grading

Item	Amount of points
Implementation of the described features	80
Documentation Quality	12
key_generator.py with the generated public/private keys with their appropriate names.	3
Accuracy of the submitted file and the quality of the conducted tests	10
Part V	15
Demo	10
Total	130

Note:

- Developed programs that fail to run will not receive more than 50% of the available points.
- Each team member should send the lab instructor an e-mail that clearly assesses every group member's contribution to the technical (coding) component and the documentation.
- The e-mail must indicate each member's contribution as a percentage. It is expected that every member will contribute (roughly) equally to each component. If the actual contributions vary, the instructor may scale individual marks within a team.