CMPT 361 – Project
Secure Mail Transfer Protocol

Weight: 15% Total points: 100 points Due: Sunday December 1st, 2024 and before 11:59 PM

I. Objectives:

In this project, you will work in teams. Each team will develop a reasonable secure mail transfer protocol. In addition, each team **MUST** solve section **V**. During your work in this project, 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:

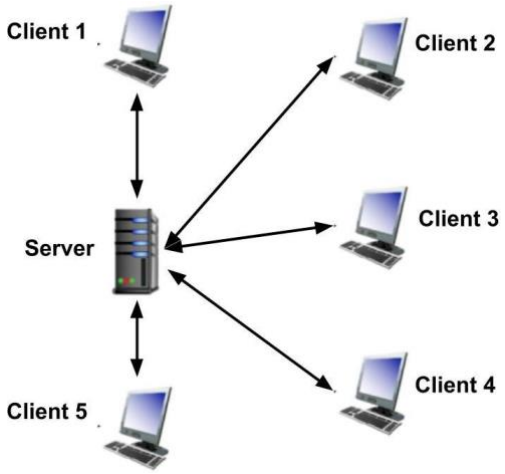
Here are some suggestions for working in a group:

- be **respectful** of your peers: different people have different backgrounds
- agree on some **expectations** (or rules) for your team
  - discuss the possibility of non- or under-contributing members
- work in **parallel**
  - define modules and function headers early, including function names, arguments, behavior, 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 project, you are going to develop a reasonably secure mail transfer protocol that works between a server and five known clients to this server. The known clients wish to securely send electronic mail (Emails) to each other through the mail server by using TCP connections as shown in the following figure:

**Section II is adapted from Dr. Nicholas Boers slides.



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 use the mail server services and can send emails to others.
- Unknown client: a user who is **NOT** authorized to use the mail server services.
- Client program (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.

B. **Server-side description:**

1. Each known client has a corresponding username and password that are known by the client and stored in the server machine.
2. The clients' usernames and passwords are stored in a JSON file with the name "user_pass.json"
3. The server machine has:
   - Prestored public and private keys for this machine called: server_public.pem and server_private.pem
4. For each of the 5 known clients, the server machine has:
   - the client's public key stored as [username]_public.pem
   - a folder with the client username that is used to store all the client incoming electronic messages (e-mail)
5. All server keys, server program, clients' folders and keys should be stored in the same directory of the server program
6. The server should listen to clients' connections on port number 13000

7. The server **MUST** be able to serve all known clients simultaneously. You **MUST** use fork to achieve this purpose

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 *[username]_private.pem* and *[username]_public.pem*
  - Know his/her username and password 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 *[username]_private.pem* and *[username]_public.pem*)
  - The server public key called *server_public.pem*
  - The client program called *Client.py*
  - The file(s) that contain the email contents that the client wish to send (more information is provided in the latter sections)
- At the start of running the client program, the client **MUST** specify the server's IP address (or hostname)

D. **Electronic Message Structure (Email):**

The electronic message, that a client wishes to send to other client(s), has the following structure:

> **From:** [The source client username who sent the message] **\n**
> **To:** [The list of destination clients' usernames separated by ";"] **\n**
> **Title:** [The title of the sent message with maximum length of 100 characters] **\n**
> **Content Length:** [Number of characters in the content field] **\n**
> **Content: \n**
> [message contents with a maximum length of 1000000 characters]

Notes:

1. The above bold strings **MUST** be part of the sent message and should not be changed.
2. Both Client.py and Server.py **MUST** check the title and content fields length to ensure that they follow the message specifications. Otherwise, the message should be rejected.

E. **Server/Client Communication Protocol:**

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

1. **[Client side]** The client program asks the client to enter his/her username and password. Then the client program encrypts both the client's username and password with **the server public key** and sends the encrypted username/password to the server side.
2. **[Server side]** After receiving the client information (username and password) and decrypting them, the server checks whether this user information is valid:
    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 username]**"
    b. If no,
        - The server program sends the following unencrypted message to the client: "Invalid username or password"
        - The server then prints the following message "The received client information: **[client_username]** 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 username or password", the client program prints to the client the message "Invalid username or password.\nTerminating." and terminate the connection with the server.
    b. Otherwise, the client decrypts the message, stores the received symmetric key (as ***sym_key***), sends the message "OK" to the server encrypted with the received symmetric key and wait for the next server message.
4. **[Server side]** after receiving the client message, the server sends to the client the following menu of operations encrypted with ***sym_key:***

```
'''Select the operation:
        1) Create and send an email
        2) Display the inbox list
        3) Display the email contents
        4) Terminate the connection
choice: '''
```

5. **[Client side]** after receiving and decrypting the server message using ***sym_key***, the client prints the received menu and asks the client user for his/her choice that can be one of the following values {1, 2, 3, 4}. The client program then encrypts the user choice using ***sym_key*** and sends it to the server.

6. **[Server side]:** After receiving the client response and decrypting it using ***sym_key***:
    a. If the received response is "1", **the sending email subprotocol** is performed between the client and the server.
    b. If the received response is "2", **the viewing inbox subprotocol** is performed between the client and the server.
    c. If the received response is "3", **the viewing email subprotocol** is performed between the client and the server.
    d. Otherwise, the **connection termination subprotocol** is performed between the client and the server.

7. **[Server side and Client side]:** If the connection is not terminated, steps 4 to 7 are repeated.

F. **Sending Email Subprotocol**

1. **[Server side]:** The server encrypts and sends the client the following message "Send the email" using ***sym_key***.
2. **[Client side]:** After the client program receives and decrypts the server message:
    - The client program asks the client user to enter the email destination clients' usernames and email title
    - The client program asks the user to enter either the message contents through the terminal or to get the message contents from a text file
    - The client program then constructs the email message according to section **D** email specifications, encrypts it using ***sym_key*** and sends it to the server side
    - The client program prints the following message "The message is sent to the server." to the client user
3. **[Server side]:** After receiving the encrypted email from the client and decrypting it using ***sym_key***:
    - prints the message "An email from **[client username]** is sent to **[destination usernames separated by ';']** has a content length of **[content length in terms of number of characters]**". Note that **[client username], [destination usernames]** and **[content length]** should be replaced with information contained in the received email.
    - The time and date of receiving the email is added to the received email as follows:

From: [The source client username who sent the message] **\n**
To: [The list of destination clients' usernames separated by ";"] **\n**
Time and Date: [The time and date of receiving the message]**\n**
Title: [The title of the sent message with maximum length of 100 characters] **\n**
Content Length: [Number of characters in the content field] **\n**
Content: \n
[message contents with a maximum length of 1000000 characters]

Note that [The time and date of receiving the message] should be replaced by the received time and date of receiving the complete message at the server side.

- The email is saved as a text file in each destination client directory with the name "[source client username]_[email title].txt" where [source client username] and [email title] should be replaced with their corresponding information from the email. Note, you may assume here that the client will not send duplicate emails with the same title to the same destination.

G. **Viewing Inbox Subprotocol**

1. **[Server side]:** The server encrypts a message using ***sym_key*** and sends it to the client side. The message contains a list of the client inbox emails' information sorted by the received time and date. For each email, the following information should be included [index, sending client username, date and time of receiving the email and title] where index is the email index in the returned list.
2. **[Client side]:** After receiving and decrypting the server message using ***sym_key***, the client prints the received menu and sends the server an encrypted message "OK".

H. **Viewing Email Subprotocol**

1. **[Server side]:** The server encrypts the message "the server request email index" using ***sym_key*** and sends it to the client side.
2. **[Client side]:** After receiving and decrypting the server message using ***sym_key***, the client program asks the client user to enter the email index that the client wish to view, encrypts this index using ***sym_key*** and sends it to the server side. Note that the index is chosen based on the last received email list from the server side.
3. **[Server side]:** After receiving and decrypting the client message, the server retrieves the email from the client folder, encrypts the email using ***sym_key*** and sends it to the client side.
4. **[Client side]:** After receiving and decrypting the server message using ***sym_key***, the client program prints the received decrypted email to the user.

I. **Connection Termination Subprotocol**

1. **[Server side]:** The server terminates the connection with the client and prints the following message "Terminating connection with [username]."
2. **[Client side]:** The client terminates the connection with the server and prints the following message "The connection is terminated with the server.".

IV. Sample output:

1- Client 10 (Unknown client):

```
$ python3 Client.py
Enter the server IP or name: cc5-212-05.macewan.ca
Enter your username: client10
Enter your password: pass10
Invalid username or password.
Terminating.
```

2- Client 1 (Known client, Sending a message):

```
$ python3 client.py
Enter the server IP or name: cc5-212-05.macewan.ca
Enter your username: client1
Enter your password: password1
Select the operation:
        1) Create and send an email
        2) Display the inbox list
        3) Display the email contents
        4) Terminate the connection

        choice:  1
Enter destinations (separated by ;): client2;client4;client5
Enter title: Test
Would you like to load contents from a file?(Y/N) N
Enter message contents: Welcome to CMPT 361 class.
The message is sent to the server.
Select the operation:
        1) Create and send an email
        2) Display the inbox list
        3) Display the email contents
        4) Terminate the connection

        choice:  4
The connection is terminated with the server.
```

3- Client 2 (Known client, Sending a message with loaded contents from a file):

```
$ python3 client.py
Enter the server IP or name: cc5-212-05.macewan.ca
Enter your username: client2
Enter your password: password2
Select the operation:
        1) Create and send an email
        2) Display the inbox list
        3) Display the email contents
        4) Terminate the connection

        choice:  1
Enter destinations (separated by ;): client1;client5
Enter title: Test2
Would you like to load contents from a file?(Y/N) Y
Enter filename: test.txt
The message is sent to the server.
Select the operation:
        1) Create and send an email
        2) Display the inbox list
        3) Display the email contents
        4) Terminate the connection

        choice:  4
The connection is terminated with the server.
```

4- Server:

```
$ python3 Server.py
The server is ready to accept connections
The received client information: client10 is invalid (Connection
Terminated).
Connection Accepted and Symmetric Key Generated for client: client1
Connection Accepted and Symmetric Key Generated for client: client2
Connection Accepted and Symmetric Key Generated for client: client5
An email from client2 is sent to client1;client5 has a content
length of 221070 .
An email from client1 is sent to client2;client4;client5 has a
content length of 26 .
Terminating connection with client1
Terminating connection with client2
```

5- Client 5 (Known client, receiving messages):

```
$ python3 Client.py
Enter the server IP or name: cc5-212-05.macewan.ca
Enter your username: client5
Enter your password: password5
Select the operation:
        1) Create and send an email
        2) Display the inbox list
        3) Display the email contents
        4) Terminate the connection

        choice:  2
Index   From        DateTime                        Title
1       client2     2022-07-21 19:29:35.768508      Test2
2       client1     2022-07-21 19:29:42.118132      Test


Select the operation:
        1) Create and send an email
        2) Display the inbox list
        3) Display the email contents
        4) Terminate the connection

        choice:  3
Enter the email index you wish to view: 2

From: client1
To: client2;client4;client5
Time and Date Received: 2022-07-21 19:29:42.118132
Title: Test
Content Length: 26
Contents:
Welcome to CMPT 361 class.

Select the operation:
        1) Create and send an email
        2) Display the inbox list
        3) Display the email contents
        4) Terminate the connection

        choice:
```

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.
4- Both *Server_enhanced.py* and *Client_enhanced.py* should print messages to their users that show and verify their operations.

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 json
      import socket
      import os,glob, datetime
      import sys
      Any module from Crypto library (https://pycryptodome.readthedocs.io/en/latest/src/installation.html)
   d. The client program **MUST** ensure that the email title length and content length are following the specifications.
   e. The server program **MUST** use the provided "user_pass.json" to verify the connected clients information.
   f. The server program **MUST** be able to handle the five known clients concurrently using fork.
   g. While using symmetric key encryption, you must use it in Electronic Code Book (ECB).
   h. While receiving an email, the server (or client program) **MUST** ensure that it receives the full email.
   i. Both server.py and client.py **MUST NOT** change the default socket properties of their used sockets (e.g. SO_SNDBUF).
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 to the MESKANAS should contain the following (**ONLY ONE** member from each group **MUST** submit them to the MESKANAS):
   a. The developed programs (*Server.py, Client.py, Server_enhanced.py and Client_enhanced.py*)
   b. A word (or PDF) report that contains:
      - The names of the group members
      - the status of your developed programs, any assumptions taken while developing the project and any deficiencies in these developed programs
      - The tests conducted for the programs (Server.py and Client.py) (You must conduct some tests using multiple lab machines)
      - For section **V:**
        - The identified attack type, how it can harm the original secure mail transfer protocol implemented in the client/server programs
        - The proposed enhanced protocol description and how it can defend against the proposed attack
        - The tests conducted for the programs (*Server_enhanced.py and Client_enhanced.py)* that show their operation
   c. key_generator.py and ALL the generated public/private keys.
4- The submission should be done to MESKANAS 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 2024.

VII. Grading

Item	Amount of points
Implementation of the described features	60
Documentation Quality	5
key_generator.py with the generated public/private keys with their appropriate names	2
Part V	17
Accuracy of the submitted report and the quality of the conducted tests	8
Demo	8
Total	100

Note:

- Developed programs that fail to run will not receive more than 50% of the available points.
- Each team member **MUST** send the course instructor an e-mail within 48 hours of the project **due date and time** that clearly assesses every group member's contribution to the technical (coding) component, Part V and the presentation.
- 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.