**Project [20 Points]: Fall2024**

*Instructions: This project is Group-based, each group may consist of (4-5 students). Each team member is expected to understand the programs and discuss them. Due date of submissions is on* ***Thursday. 5/12/2024****.*

**Team Members:**

1. Sanika Pandit 2. Zahra Ibrahim

3. Luna Asil Companioni 4. Meera Ansari

**Submission:** Each Team Leader submits the python client and server files and copy and paste the code of both client and server files into this document. *Files should be named as follow: teamleaderName\_client.py,teamleaderName\_server.py,teamleaderName\_projectDocument.docx*

**Rubric: (Each students might have different grade based on the discussions)**

|  |  |  |  |
| --- | --- | --- | --- |
| Criteria | Max Grade | Std. Grade | Feedback |
| Setup Phase | 5 | 5 | **Excellent, very well done indeed !!** |
| Closing Phase | 5 | 5 |
| Start-Packet | 2 | 2 |
| Encryption-Packet | 2 | 2 |
| Confirm-Connection-Packet | 2 | 2 |
| Using RSA including session key encryption | 5 | 5 |
| Using AES | 5 | 5 |
| Using Caesar | 5 | 5 |
| Prompt packets | 10 | 10 |
| openRead includes reading from files | 8 | 8 |
| openWrite incudes writing on files | 8 | 8 |
| Data Packet | 2 | **0** | **No DP is sent or received** |
| Exception packets | 8 | 8 | **Excellent, very well done indeed !!** |
| Server Multithreads | 5 | 5 |
| Comments | 5 | 5 |
| Running code | 8 | 8 |
| Integrating C client part | 5 | 5 |
| Discussions | 10 | 10 |
| Total | **/100** | **98 /100** | **19.60 /20** |

**Project Problem:**

Remote controlling protocols such as Telnet and SSH are used to remotely run commands on other devices without the need to physical direct interaction with these devices. In this project you will design and develop a simple application layer protocol that help folders and files management on servers.

You have to write Python3 code to implement the **Remote File Management Protocol (RFMP)** which is designed to serve a remote controlling application.

**RFMP** has three phases: ***setup phase***, ***operation phase***, and ***closing phase***.

**1. Setup-Phase:**

Once the client opens the application, the application sends a series of packets to the server device in the following order:

* **Start-Packet (from client to server):** contains the packet type, protocol name, protocol version, secured-communication, below are some examples:
  + (**SS**,RFMP,v1.0,0), where S*S* is the packet type (start), RFMP is the protocol name, v1.0 is the version, 0 means no secured communication is required, in this case no encryption for the packet is required.
  + (**SS**, RFMP,v1.0,1), where S*S* is the packet type (start), RFMP is the protocol name, v1.0 is the version, 1 means secured communication is required, in this case the server expects to receive an encryption-packet (EC) immediately after the start-packet (SS).
* **Confirm-Connection-Packet** **(from server to client)**: this packet is sent with one field if security is not required which is the packet-type (**CC**) from the server to inform the client to start sending information packets or with two fields if security is required (CC, Server\_public\_key).

* **Encryption-Packet (from client to server):** contains information about encryption algorithm, and credentials used to secure communication. This packet is sent in case the Start-Packet contains **1** in the secure-communication field. This packet type contains the following fields (packet-type, Algorithm, credentials): (**EC**, ALgorithm, session\_key, username:Client\_public\_key).

The following table shows the options of the algorithms and credentials sent with each:

|  |  |
| --- | --- |
| **Fields** | **Credentials** |
| **AES Cipher** | Both will use AES to encrypt files contents |
| **Caesar Cipher** | Both will use Caesarto encrypt files contents |
| **Session\_key** | Either AES or Caesar key encrypted using RSA private key |
| **Public\_Key** | Public key generated by RSA algorithm, used to decrypt session key |

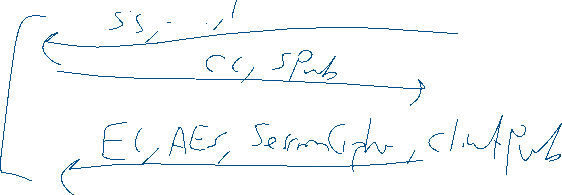
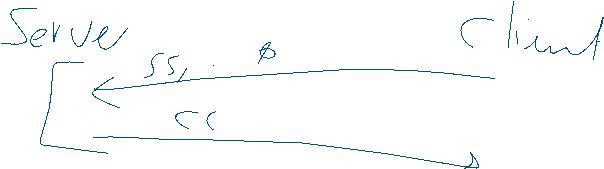
**In the Start-up phase the client prepares three keys:**

**1. Session Key that will be used later to encrypt data either by using Caesar or AES algorithms.**

**2. Public and Private Keys using RSA algorithm. That will be used later to encrypt and decrypt session key**

The server in the setup phase will generate both ***private key*** and ***public key*** using the *RSA* algorithm and send his public key to client in the confirmation packet (CC, Server\_publik\_Key).

The Client program will use the Server\_*public\_ key* to encrypt the *session\_key and send it to server*. The server will use his *private\_key* to decrypt the *session\_key,* then later in the operation phase the server will use the *session\_key* to decrypt data before saving the data into a file on server.



1. **Operation-Phase:**

This phase is started once the client starts typing commands and send them to the server. This packet has type filed (CM). The packet has only two fields the type and the command text.

Ex. (CM, ls), (CM, mkdir f1),.. See the below table for all commands.

You can send any system command to the server. Such as processor or memory status enquiry, or folder management commands. In this project you have to include the following commands for file and folder management:

|  |  |
| --- | --- |
| **mkdir** | Create new folder |
| **cd** | Change directory path command |
| **rmdir or rd** | To delete folders |
| **del** | To delete files |
| **ren** | To rename a folder |
| **openRead** | This command is not a prompt command, its keyword for the server program to use python file open command to create new file in write mode |
| **openWrite** | This command is not a prompt command, its keyword for the server program to use python file open command to create new file in read mode |
| Choose any 5 other system prompt commands you want to execute on the server |  |

* Each of the **prompt commands** requires argument which you should send in the packet (write full command text such as (CM,prompt,mkdir folder1), (CM,prompt,ren homework1 homework2). For these types of prompt commands you can use many python libraries to implement them:

os.system()

subprocess.run()

* The **openRead** and **openWrite** commands requires to send in the same packet file names you want to open in read mode or write mode for example: (CM,openRead, data.txt) or (CM, openWrite, data.txt)
* Data Packets are used to send data to be saved in the file created using openWrite command. This packet has the following format: (DP, text) where the text is any data you want the server to save.

If the communications are secured according to the setup phase then you have to encrypt the ***text*** field before sending. And the server will decrypt the text before saving it to the file.

* In the openRead, the server should open a file and send its contents to the client. Make sure if encryption is chosen in the setup phase then the server should use the session key to encrypt the text before sending it.
* After each command execution on the server, the server should returns either successful (SC) packet or Error packet(EE).

1. **Closing-Phase:**

Packet (**End**) is used in closing phase. In this phase a **Close-Packet** is sent to the server to confirm that the client has finished from using the application. And the server will expect no more messages from the client.

4. **Exception-Packets:** Your client should check each response from the server if the server sends an Exception Event (EE) packet-type message this means that there has an error occurred while processing client’s requests. Each EE packet contains the following fields: (EE, Error Code, Description). Error Code values are left to your implementation, you can suggest set of error codes (maximum 4 error codes).

**Note: You have to implement a client side with proper options that the user can choose from and remotely control the server and displays server responses.**

**5. Integrating with C client program:**

You have to integrate your project with a C client version in addition to the Python client version and Python server version. This C client part is simple version of the python client part, where you have to implement a nonsecured communication client in the C-part, (no encryption) and the C client program sends only one type command which is the openRead to save a text on server file.

**Python- Server Code:**

**import socket**

**import subprocess**

**import threading**

**from Crypto.PublicKey import RSA**

**from Crypto.Cipher import PKCS1\_OAEP, AES**

**import base64**

**import os**

**from datetime import datetime**

**# Caesar Cipher class**

**class CaesarCipher:**

**@staticmethod**

**def encrypt(text, shift):**

**return ''.join(**

**chr((ord(char) - 32 + shift) % 95 + 32) if 32 <= ord(char) <= 126 else char**

**for char in text**

**)**

**@staticmethod**

**def decrypt(text, shift):**

**return ''.join(**

**chr((ord(char) - 32 - shift) % 95 + 32) if 32 <= ord(char) <= 126 else char**

**for char in text**

**)**

**# Encryptor class for AES and session key management**

**class Encryptor:**

**@staticmethod**

**def aes\_encrypt(key, data):**

**cipher = AES.new(key, AES.MODE\_ECB)**

**ciphertext = b""**

**# Process the data in 16-byte chunks**

**while len(data) > 0:**

**chunk = data[:16]**

**if len(chunk) == 16:**

**ciphertext += cipher.encrypt(chunk.encode('utf-8'))**

**data = data[16:]  # Remove the processed chunk**

**return base64.b64encode(ciphertext).decode()  # Return the base64 encoded ciphertext**

**@staticmethod**

**def aes\_decrypt(key, data):**

**data = base64.b64decode(data)**

**cipher = AES.new(key, AES.MODE\_ECB)**

**decrypted\_data = b""**

**# Decrypt the data in 16-byte chunks**

**while len(data) > 0:**

**chunk = data[:16]**

**decrypted\_data += cipher.decrypt(chunk)**

**data = data[16:]  # Remove the processed chunk**

**return decrypted\_data.decode('utf-8')**

**class ServerThread(threading.Thread):**

**def \_\_init\_\_(self, client\_socket, client\_address, server\_private\_key, server\_public\_key):**

**super().\_\_init\_\_()**

**self.client\_socket = client\_socket**

**self.client\_address = client\_address**

**self.server\_private\_key = server\_private\_key**

**self.server\_public\_key = server\_public\_key**

**self.session\_key = None**

**self.cipher\_type = None**

**self.caesar\_shift = 3  # Default Caesar cipher shift**

**def send\_response(self, response\_type, message):**

**if self.cipher\_type == "AES" and self.session\_key:**

**message = Encryptor.aes\_encrypt(self.session\_key, message)**

**elif self.cipher\_type == "Caesar":**

**message = CaesarCipher.encrypt(message, self.caesar\_shift)**

**response\_packet = f"{response\_type},{message}"**

**print(f"Sent: {response\_packet}")**

**self.client\_socket.send(response\_packet.encode("utf-8"))**

**def run(self):**

**print(f"Connection from {self.client\_address}")**

**try:**

**while True:**

**request = self.client\_socket.recv(2048).decode("utf-8")**

**if not request:**

**print("Client disconnected.")**

**break**

**print(f"Received: {request}")**

**fields = request.split(",")**

**if fields[0] == "SS":  # Start Packet**

**self.cipher\_type = fields[1]  # Set cipher type (AES, Caesar, etc.)**

**public\_key\_encoded = base64.b64encode(self.server\_public\_key.export\_key()).decode()**

**self.send\_response("CC", public\_key\_encoded)**

**elif fields[0] == "EC":  # Encryption Packet**

**encrypted\_session\_key = base64.b64decode(fields[2])**

**self.session\_key = PKCS1\_OAEP.new(self.server\_private\_key).decrypt(encrypted\_session\_key)**

**print("Session key successfully decrypted.")**

**elif fields[0] == "CM":  # Command Packet**

**command = fields[1]**

**args = fields[2] if len(fields) > 2 else ""**

**self.execute\_command(command, args)**

**elif fields[0] == "End":  # Close Packet**

**print("Client has closed the connection.")**

**break**

**else:**

**self.send\_response("EE,1001", "Invalid packet type.")**

**except Exception as e:**

**print(f"Error: {e}")**

**self.send\_response("EE,1000", f"General error: {str(e)}")**

**finally:**

**self.client\_socket.close()**

**print(f"Connection with {self.client\_address} closed.")**

**def execute\_command(self, command, args):**

**try:**

**if self.cipher\_type == "Caesar" and args:**

**args = CaesarCipher.decrypt(args, self.caesar\_shift)**

**elif self.cipher\_type == "AES" and args and self.session\_key:**

**args = Encryptor.aes\_decrypt(self.session\_key, args)**

**if command == "mkdir":**

**os.makedirs(args, exist\_ok=True)**

**self.send\_response("SC", f"Directory '{args}' created successfully.")**

**elif command == "cd":**

**os.chdir(args)**

**self.send\_response("SC", f"Changed directory to '{args}'.")**

**elif command in ("rmdir", "rd"):**

**os.rmdir(args)**

**self.send\_response("SC", f"Directory '{args}' removed successfully.")**

**elif command == "del":**

**os.remove(args)**

**self.send\_response("SC", f"File '{args}' deleted successfully.")**

**elif command == "ren":**

**old\_name, new\_name = args.split(" ", 1)**

**os.rename(old\_name, new\_name)**

**self.send\_response("SC", f"Renamed '{old\_name}' to '{new\_name}'.")**

**elif command == "openRead":**

**with open(args, "r") as file:**

**data = file.read()**

**if self.cipher\_type == "AES" and self.session\_key:**

**data = Encryptor.aes\_encrypt(self.session\_key, data)**

**elif self.cipher\_type == "Caesar":**

**data = CaesarCipher.encrypt(data, self.caesar\_shift)**

**self.send\_response("SC", data)**

**elif command == "openWrite":**

**with open(args, "w") as file:**

**file.write("Sample content written by server.")**

**self.send\_response("SC", f"File '{args}' created in write mode.")**

**elif command == "date":**

**current\_date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")**

**self.send\_response("SC", f"Current Date and Time: {current\_date}")**

**elif command == "hostname":**

**hostname = subprocess.getoutput("hostname")**

**self.send\_response("SC", f"Hostname: {hostname}")**

**elif command == "echo":**

**self.send\_response("SC", f"ECHO: {args}")**

**elif command == "ls":**

**directory\_content = os.listdir(args if args else ".")**

**self.send\_response("SC", f"Directory Content: {', '.join(directory\_content)}")**

**elif command == "pwd":**

**current\_dir = os.getcwd()**

**self.send\_response("SC", f"Current Directory: {current\_dir}")**

**else:**

**self.send\_response("EE,1002", f"Unknown command '{command}'.")**

**except Exception as e:**

**print(f"Error executing command: {e}")**

**self.send\_response("EE,1003", f"Command execution error: {str(e)}")**

**def server\_main():**

**host = "127.0.0.1"**

**port = 5000**

**server\_private\_key = RSA.generate(2048)**

**server\_public\_key = server\_private\_key.publickey()**

**server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)**

**server\_socket.bind((host, port))**

**server\_socket.listen(5)**

**print(f"Server running on {host}:{port}")**

**try:**

**while True:**

**client\_socket, client\_address = server\_socket.accept()**

**thread = ServerThread(client\_socket, client\_address, server\_private\_key, server\_public\_key)**

**thread.start()**

**except KeyboardInterrupt:**

**print("Server shutting down.")**

**finally:**

**server\_socket.close()**

**if \_\_name\_\_ == "\_\_main\_\_":**

**server\_main()**

**Python- Client Code:**

**import socket**

**from Crypto.Cipher import AES**

**from Crypto.PublicKey import RSA**

**from Crypto.Cipher import PKCS1\_OAEP**

**from Crypto.Random import get\_random\_bytes**

**import base64**

**class CaesarCipher:**

**@staticmethod**

**def encrypt(text, shift):**

**return ''.join(**

**chr((ord(char) - 32 + shift) % 95 + 32) if 32 <= ord(char) <= 126 else char**

**for char in text**

**)**

**class StartPacket:**

**def \_\_init\_\_(self, packet\_type="SS", protocol\_name="RFMP", protocol\_version="v1.0", cipher\_type="None"):**

**self.packet\_type = packet\_type**

**self.protocol\_name = protocol\_name**

**self.protocol\_version = protocol\_version**

**self.cipher\_type = cipher\_type**

**def \_\_str\_\_(self):**

**return f"{self.packet\_type},{self.cipher\_type},{self.protocol\_name},{self.protocol\_version}"**

**class EncryptionPacket:**

**def \_\_init\_\_(self, algorithm="AES", session\_key=None, client\_pub\_key=None):**

**self.packet\_type = "EC"**

**self.algorithm = algorithm**

**self.session\_key = session\_key**

**self.client\_pub\_key = client\_pub\_key**

**def \_\_str\_\_(self):**

**return f"{self.packet\_type},{self.algorithm},{base64.b64encode(self.session\_key).decode()},{self.client\_pub\_key}"**

**def send\_command(sock, command\_type, command, args=""):**

**packet = f"{command\_type},{command},{args}"**

**sock.send(packet.encode("utf-8"))**

**response = sock.recv(2048).decode("utf-8")**

**print(f"Server Response: {response}")**

**def generate\_rsa\_keys():**

**private\_key = RSA.generate(2048)**

**public\_key = private\_key.publickey().export\_key()**

**return private\_key, public\_key**

**def encrypt\_session\_key(session\_key, server\_pub\_key):**

**cipher\_rsa = PKCS1\_OAEP.new(server\_pub\_key)**

**return cipher\_rsa.encrypt(session\_key)**

**def client\_main():**

**host = "127.0.0.1"**

**port = 5000**

**encrypt\_packets = input("Would you like to encrypt packets? (yes/no): ").strip().lower()**

**cipher\_type = "None"**

**session\_key = None**

**caesar\_shift = None**

**if encrypt\_packets == "yes":**

**cipher\_type = input("Choose encryption type (AES, Caesar): ").strip()**

**if cipher\_type == "AES":**

**session\_key = get\_random\_bytes(16)  # Example 16-byte AES key**

**elif cipher\_type == "Caesar":**

**caesar\_shift = int(input("Enter Caesar cipher shift value (e.g., 3): "))**

**client\_private\_key, client\_public\_key = generate\_rsa\_keys()**

**s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)**

**try:**

**s.connect((host, port))**

**print("Connected to the server.")**

**start\_packet = StartPacket(cipher\_type=cipher\_type)**

**s.send(str(start\_packet).encode("utf-8"))**

**print(f"Sent start packet: {start\_packet}")**

**response = s.recv(2048).decode("utf-8")**

**print(f"Received confirm-connection packet: {response}")**

**if cipher\_type == "AES":**

**fields = response.split(",")**

**server\_pub\_key = RSA.import\_key(base64.b64decode(fields[1]))**

**encrypted\_session\_key = encrypt\_session\_key(session\_key, server\_pub\_key)**

**encryption\_packet = EncryptionPacket(**

**algorithm="AES",**

**session\_key=encrypted\_session\_key,**

**client\_pub\_key=client\_public\_key.decode()**

**)**

**s.send(str(encryption\_packet).encode("utf-8"))**

**print(f"Sent encryption packet: {encryption\_packet}")**

**while True:**

**command\_input = input("Enter a command (or 'exit' to quit): ").strip()**

**if command\_input.lower() == "exit":**

**s.send("End".encode("utf-8"))**

**break**

**command\_parts = command\_input.split(" ", 1)**

**command = command\_parts[0]**

**args = command\_parts[1] if len(command\_parts) > 1 else ""**

**if cipher\_type == "Caesar" and args:**

**args = CaesarCipher.encrypt(args, caesar\_shift)**

**send\_command(s, "CM", command, args)**

**except Exception as e:**

**print(f"An error occurred: {e}")**

**finally:**

**s.close()**

**print("Connection closed.")**

**if \_\_name\_\_ == "\_\_main\_\_":**

**client\_main()**

**C- Client Code:**

#include <arpa/inet.h>

#include <stdio.h>

#include <string.h>

#include <sys/socket.h>

#include <unistd.h>

#define HOST "127.0.0.1" // defining server IP address (localhost)

#define PORT 5000 // defining the server's port number

int main(int argc, char const\* argv[]){

    int s; // file descriptor for the socket

    struct sockaddr\_in server\_addr; // struct to define the server address

    const char\* start\_packet = "SS,RFMP,v1.0,0"; // initial message sent to the server

    const char\* command\_input = "CM, openRead, example.txt"; // message to be sent

    s = socket(AF\_INET, SOCK\_STREAM, 0); // creating socket using ipv4 and tcp

    if (s<0){ // error handling for socket creation

        printf("Error in creating a socket.");

        return -1;

    }

    // configuring the server address structure

    server\_addr.sin\_family = AF\_INET; //Address: IPv4

    server\_addr.sin\_port = htons(PORT); //Converting the port number to network byte order

    server\_addr.sin\_addr.s\_addr = inet\_addr(HOST); //Converting the IP address to network format

    // server connection

    if (connect(s, (struct sockaddr\*)&server\_addr, sizeof(server\_addr)) < 0) {

        perror("Connection failed."); // Printing a detailed error message

        close(s); //Closing the socket

        return -1;

    }

    printf("Successful connection to server."); // confirmation message

    // start packet to server

    send(s, start\_packet, strlen(start\_packet),0);

    printf("Sent start packet: %s\n", start\_packet);

    // cmd input to server

    send(s, command\_input, strlen(command\_input), 0);

    printf("Sent start packet: %s\n", start\_packet);

    close(s); // socket closing

    printf("Connection killed.\n");

    return 0; // successful execution

}