Introduction

The Secure Communication System is a Python-based framework developed by Krishna Ganta, a student of Northshore School District (Student ID: 2032673). This system allows secure communication between a server and a client over a network by utilizing cryptographic techniques and encryption protocols. This document serves as a guide to understand the purpose of this system, why it's needed, and how to use it effectively.

Purpose

The Secure Communication System was developed to address the need for secure data transmission over untrusted networks, such as the internet. In many applications, ensuring the confidentiality and integrity of data is critical. This system provides a solution for establishing secure connections and encrypting data between a server and a client, making it suitable for scenarios where sensitive information needs to be protected from unauthorized access or tampering.

Key Features

1. **Encryption**: The system uses strong encryption techniques to protect data during transmission. It employs RSA for key exchange and the Fernet symmetric encryption algorithm for encrypting and decrypting the actual data.
2. **Challenge-Response Authentication**: The client and server engage in a challenge-response authentication process to ensure that the server that they are communicating with does indeed possess the RSA key pair they claim to have.

IMPORTANT: THIS PROTOCOL DOES NOT ENSURE THE VALIDITY OF THE IP BEING CONNECTED TO. IT ONLY SECURES DATA IN TRANSIT.

IT IS THE USER'S RESPONSIBILITY TO GUARD AGAINST IP SPOOFING, ARP POISONING AND OTHER IP ATTACKS.

THE AUTHOR RECOMMENDS ARP BINDING AND DHCP ADDRESS RESERVATION ON THE ACCESS POINT, ALONG WITH PHYSICALLY ASCERTAINING THE IP ADDRESS OF THE REMOTE MACHINE.

How to Use

Server

1. **Import Necessary Libraries**: Ensure that the required libraries are installed by running the script. The script will raise an error if any dependencies are missing.
2. **SecureSession Class**: This class provides an interface for encrypted communication. To use it, follow these steps:

a. Initialize a `SecureSession` object by passing a socket connection and a session.

b. Use the `send` method to send data securely. It encrypts the data and sends it to the client.

c. Use the `recv` method to receive and decrypt data from the client.

d. Close the session and the connection when done using the `close` method.

3. **RSA Key Generation**: The script generates RSA key pairs for secure communication.
4. **Initialization**: In the example block, the server initializes a socket and waits for a connection from a client. Once connected, it initializes a SecureSession object, sends a welcome message, and closes the session when done. For use in other code, call the `init_session(connection)` and pass it the socket that you received from the `serversocket.accept()` call.

Client

1. **Import Necessary Libraries**: Ensure that the required libraries are installed by running the script. The script will raise an error if any dependencies are missing.
2. **SecureSession Class**: Similar to the server, this class provides an interface for encrypted communication. To use it, follow these steps:

   a. Initialize a SecureSession object by passing a socket connection and a session.

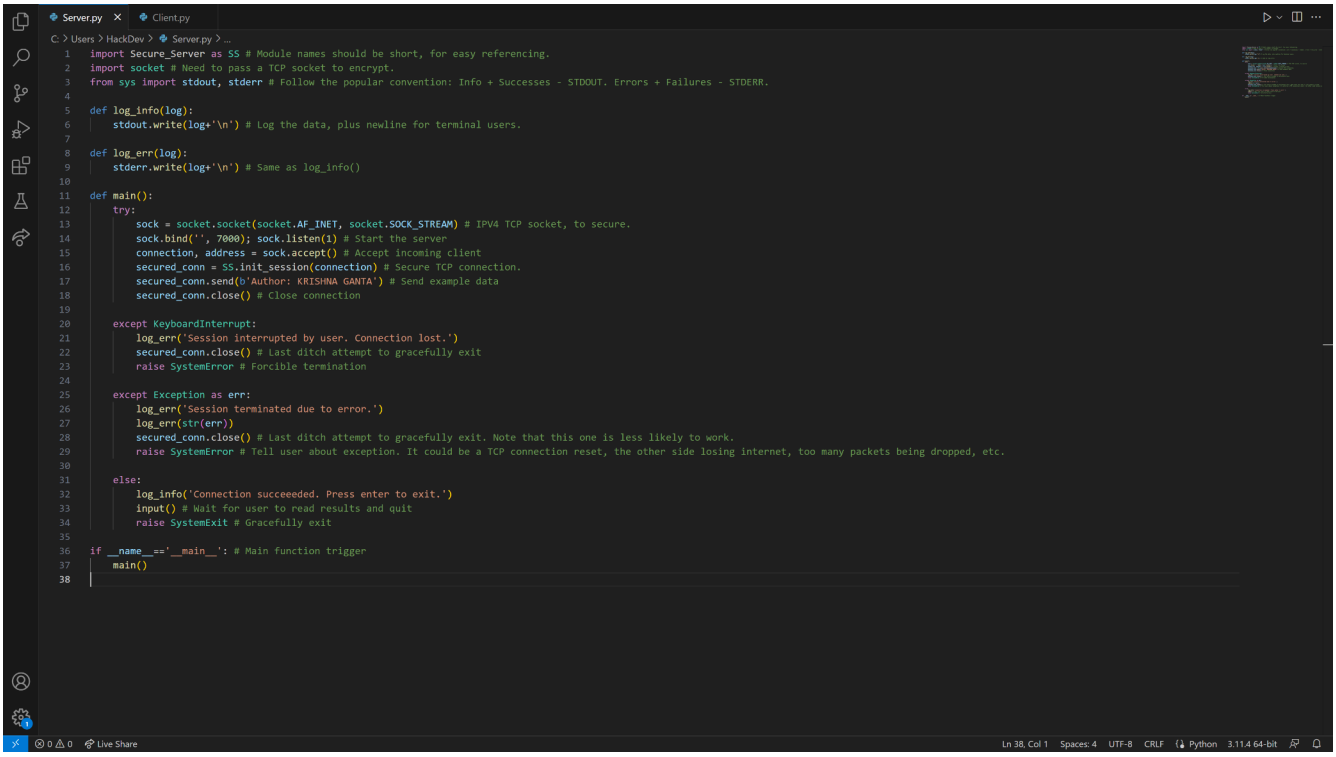
   b. Use the `send` method to send data securely. It encrypts the data and sends it to the server.

   c. Use the `recv` method to receive and decrypt data from the server.

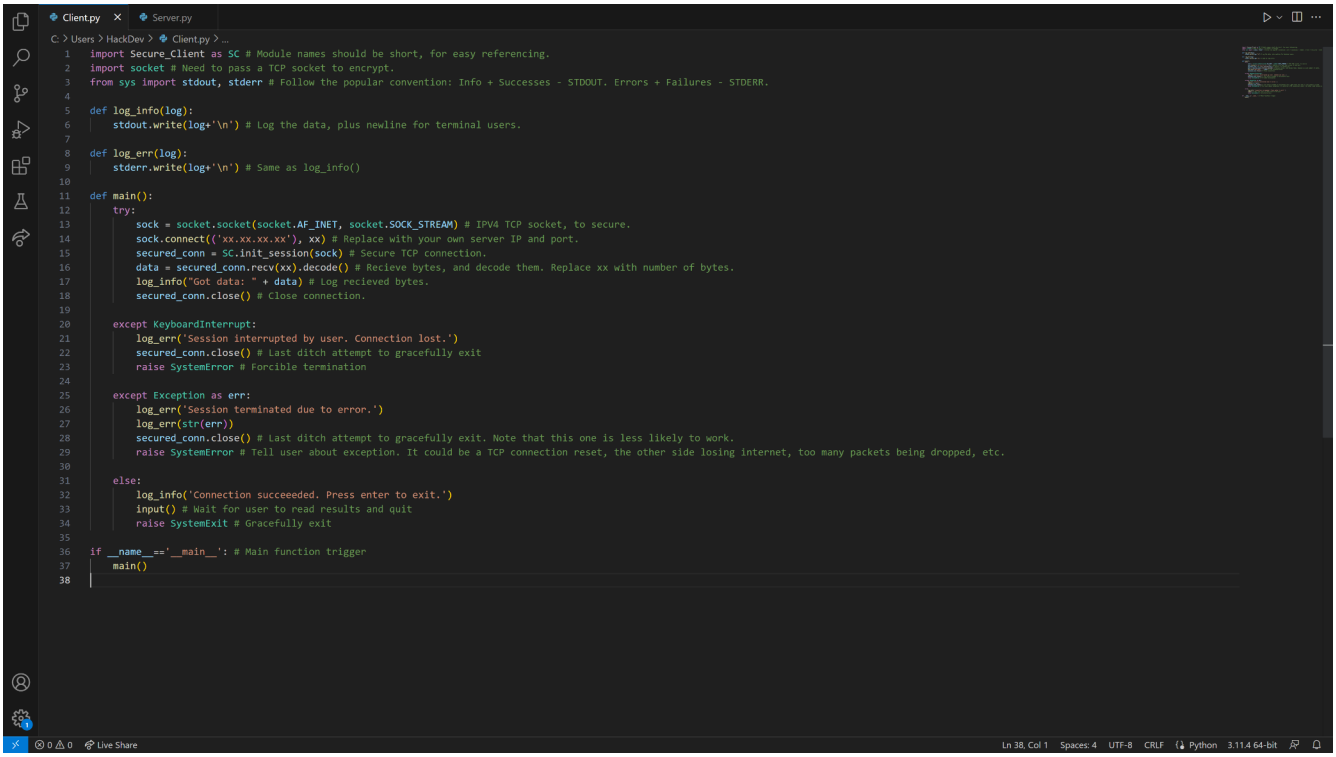
   d. Close the session and the connection when done using the `close` method.

3. **Initialization**: In the example block, the client initializes a socket and connects to the server. It then initializes a SecureSession object, receives a welcome message from the server, and closes the session when done. For use in other code, call the `init_session(socket)` and pass it the socket that you received from the `socket.connect()` call.

Example Usage:



```python
import Secure_Server as SS # Module names should be short, for easy referencing.
import socket # Need to pass a TCP socket to encrypt.
from sys import stdout, stderr # Follow the popular convention: Info + Successes - STDOUT. Errors + Failures - STDERR.

def log_info(log):
    stdout.write(log+'\n') # Log the data, plus newline for terminal users.

def log_err(log):
    stderr.write(log+'\n') # Same as log_info()

def main():
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # IPV4 TCP socket, to secure.
        sock.bind('', 7000); sock.listen(1) # Start the server
        connection, address = sock.accept() # Accept incoming client
        secured_conn = SS.init_session(connection) # Secure TCP connection.
        secured_conn.send(b'Author: KRISHNA GANTA') # Send example data
        secured_conn.close() # Close connection

    except KeyboardInterrupt:
        log_err('Session interrupted by user. Connection lost.')
        secured_conn.close() # Last ditch attempt to gracefully exit
        raise SystemError # Forcible termination

    except Exception as err:
        log_err('Session terminated due to error.')
        log_err(str(err))
        secured_conn.close() # Last ditch attempt to gracefully exit. Note that this one is less likely to work.
        raise SystemError # Tell user about exception. It could be a TCP connection reset, the other side losing internet, too many packets being dropped, etc.

    else:
        log_info('Connection succeeeded. Press enter to exit.')
        input() # Wait for user to read results and quit
        raise SystemExit # Gracefully exit

if __name__=='__main__': # Main function trigger
    main()
```



```python
import Secure_Client as SC # Module names should be short, for easy referencing.
import socket # Need to pass a TCP socket to encrypt.
from sys import stdout, stderr # Follow the popular convention: Info + Successes - STDOUT. Errors + Failures - STDERR.

def log_info(log):
    stdout.write(log+'\n') # Log the data, plus newline for terminal users.

def log_err(log):
    stderr.write(log+'\n') # Same as log_info()

def main():
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # IPV4 TCP socket, to secure.
        sock.connect(('xx.xx.xx.xx'), xx) # Replace with your own server IP and port.
        secured_conn = SC.init_session(sock) # Secure TCP connection.
        data = secured_conn.recv(xx).decode() # Recieve bytes, and decode them. Replace xx with number of bytes.
        log_info("Got data: " + data) # Log recieved bytes.
        secured_conn.close() # Close connection.

    except KeyboardInterrupt:
        log_err('Session interrupted by user. Connection lost.')
        secured_conn.close() # Last ditch attempt to gracefully exit
        raise SystemError # Forcible termination

    except Exception as err:
        log_err('Session terminated due to error.')
        log_err(str(err))
        secured_conn.close() # Last ditch attempt to gracefully exit. Note that this one is less likely to work.
        raise SystemError # Tell user about exception. It could be a TCP connection reset, the other side losing internet, too many packets being dropped, etc.

    else:
        log_info('Connection succeeeded. Press enter to exit.')
        input() # Wait for user to read results and quit
        raise SystemExit # Gracefully exit

if __name__=='__main__': # Main function trigger
    main()
```