Problem: Design a secure client-server communication system for transferring sensitive data between multiple clients and a central server in real-time.

Objectives:

1. The client-server communication should be encrypted to protect the sensitive data being transferred.
2. The system should be scalable to accommodate multiple clients simultaneously.
3. The server should be able to handle clients connecting and disconnecting dynamically.
4. The server should store the data received from clients securely, and clients should be able to retrieve their data whenever they need.
5. The system should be able to handle failures in the network and recover gracefully.
6. The system should be designed in such a way that the server can process and respond to client requests efficiently.

Requirements:

1. Use the Python programming language to implement the system.
2. Use AES (Advanced Encryption Standard) for encryption and decryption of data.
3. Use TCP (Transmission Control Protocol) for reliable data transfer between clients and server.
4. Use the RSA (Rivest-Shamir-Adleman) algorithm for secure key exchange between clients and server.
5. Use a database (e.g., MySQL) to store the data received from clients at the server.
6. The system should allow clients to connect and disconnect dynamically.
7. The system should be able to handle failures in the network and recover gracefully.
8. The system should be designed in such a way that the server can process and respond to client requests efficiently.

Evaluation Criteria:

1. Security of the data transfer between clients and server.
2. Scalability of the system to accommodate multiple clients simultaneously.
3. Efficient handling of clients connecting and disconnecting dynamically.
4. Efficient storage of data received from clients at the server.
5. Efficient processing of client requests by the server.
6. Ability of the system to handle failures in the network and recover gracefully.
7. User-friendliness of the system.
8. Reliable Data Transfer: TCP (Transmission Control Protocol) can be used for reliable data transfer between clients and server. TCP is a reliable protocol that ensures that the data is transmitted correctly and in the correct order.
9. Dynamic Connection Management: To handle clients connecting and disconnecting dynamically, the server can maintain a list of connected clients. Whenever a client connects, it can be added to the list, and whenever a client disconnects, it can be removed from the list.
10. Data Storage: A database (e.g., MySQL) can be used to store the data received from clients at the server. The database can be designed in such a way that it can store the data efficiently and retrieve it quickly when needed.
11. Failure Handling: To handle failures in the network, the system can be designed in such a way that it can recover gracefully. For example, if the connection between a client and the server is lost, the client can attempt to reconnect after a certain period of time.
12. Efficient Processing: To ensure that the server can process and respond to client requests efficiently, the server can use multithreading. Each client request can be handled by a separate thread, which will run in parallel with other threads.

import socket

import threading

import RSA\_Encryption as RSA

import AES\_Encryption as AES

import mysql.connector

# Initialize the database connection

db = mysql.connector.connect(

host="localhost",

user="root",

password="password",

database="secure\_data\_storage"

)

cursor = db.cursor()

# Create a list to keep track of connected clients

clients = []

# Function to handle incoming clients

def handle\_client(client\_socket, client\_address):

# Generate a unique RSA key pair for each client

public\_key, private\_key = RSA.generate\_keypair()

# Send the public key to the client for encryption

client\_socket.send(public\_key.export\_key())

# Receive the encrypted AES key from the client

encrypted\_aes\_key = client\_socket.recv(1024)

# Decrypt the AES key using the private RSA key

aes\_key = RSA.decrypt(private\_key, encrypted\_aes\_key)

while True:

# Receive encrypted data from the client

encrypted\_data = client\_socket.recv(1024)

# Decrypt the data using the AES key

data = AES.decrypt(aes\_key, encrypted\_data)

# Store the data in the database

query = "INSERT INTO client\_data (client\_address, data) VALUES (%s, %s)"

values = (client\_address, data)

cursor.execute(query, values)

db.commit()

# Send a response back to the client

client\_socket.send("Data stored successfully.")

# Create a server socket

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

# Bind the socket to a specific address and port

server\_socket.bind(("0.0.0.0", 12345))

# Listen for incoming connections

server\_socket.listen(5)

print("Server started. Waiting for clients...")

# Continuously accept incoming connections

while True:

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

print("Accepted connection from", client\_address)

clients.append(client\_socket)

# Start a new thread for each incoming client

client\_thread = threading.Thread(target=handle\_client, args=(client\_socket, client\_address))

client\_thread.start()

# Close the server socket

server\_socket.close()

Client-Side :

import socket

import RSA\_Encryption as RSA

import AES\_Encryption as AES

# Connect to the server

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

client\_socket.connect(("localhost", 12345))

# Receive the server's public RSA key

public\_key = RSA.import\_key(client\_socket.recv(1024))

# Generate a unique AES key for this session

aes\_key = AES.generate\_