

Secure Chat Application with End-to-End Encryption

1. Introduction

In today's digital age, real-time messaging has become an integral part of communication, but many chat platforms still lack strong encryption, leaving user data vulnerable to attacks. This project addresses this concern by developing a secure, encrypted chat application that ensures private communication between users over a network. It is designed using Python, with a GUI-based client interface and a command-line server backend, supporting both user-to-user and server-to-client communication.

2. Abstract

The Secure Chat Application is a Python-based tool that enables encrypted real-time messaging between clients using RSA encryption. It incorporates socket programming for communication, PyCryptodome for encryption, and Tkinter for the GUI. The project features secure user authentication with persistent login/registration support, a multi-threaded chat server, and encrypted broadcast messages. The server can also send messages through a terminal interface and announces when a new user joins the chat. This system ensures end-to-end confidentiality, making it suitable for internal communication in secure environments or as a demonstration of cryptographic principles in networked applications.

3. Tools and Technologies Used

Tool/Technology	Purpose
Python 3.x	Core programming language
Socket Programming	Real-time network communication
PyCryptodome	RSA encryption and decryption
Tkinter	GUI framework for the chat interface
JSON	Persistent user credential storage

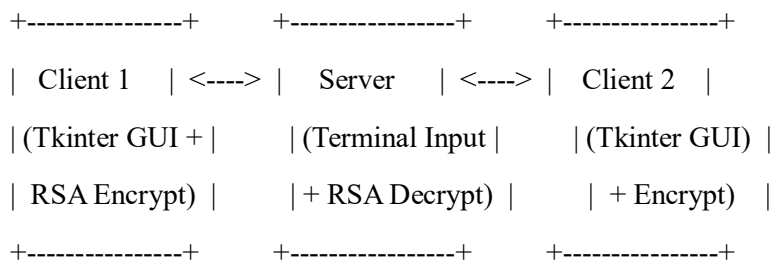
4. Objectives

- To develop a real-time chat system using socket programming.
- To ensure data privacy and confidentiality using RSA-based encryption.
- To support secure user authentication with persistent storage.
- To provide a simple and functional GUI for users.
- To enable the server to broadcast messages and interact with users.

5. System Features

- **End-to-End Encryption:** All messages are encrypted using RSA.
- **Real-Time Communication:** Messages are instantly sent and received using sockets.
- **User Authentication:** Users can register and log in securely.
- **Persistent User Data:** Registered credentials are stored in a JSON file.
- **GUI Interface:** Built with Tkinter for a scrollable chat window and input box.
- **Server Messaging:** Server can type and broadcast messages from its console.
- **User Join Notification:** The server automatically announces when a new user joins.
- **Graceful Error Handling:** Handles disconnects and login failures robustly.

6. Architecture Diagram



7. Steps Involved in Building the Project

1. **Designed the Encryption Module (crypto_utils.py)**
Implemented RSA-based key generation, message encryption, and decryption using PyCryptodome.
2. **Developed the Server (server.py)**
Built a socket-based multi-client server that manages login/registration and broadcasts messages. Added JSON-based user storage and server-side terminal chat.
3. **Built the Client Interface (client.py)**
Designed a GUI using Tkinter that allows users to send encrypted messages and register/login securely.

4. Handled Message Flow

Ensured clients encrypt messages with the server's public key and the server decrypts with its private key, then re-broadcasts the messages.

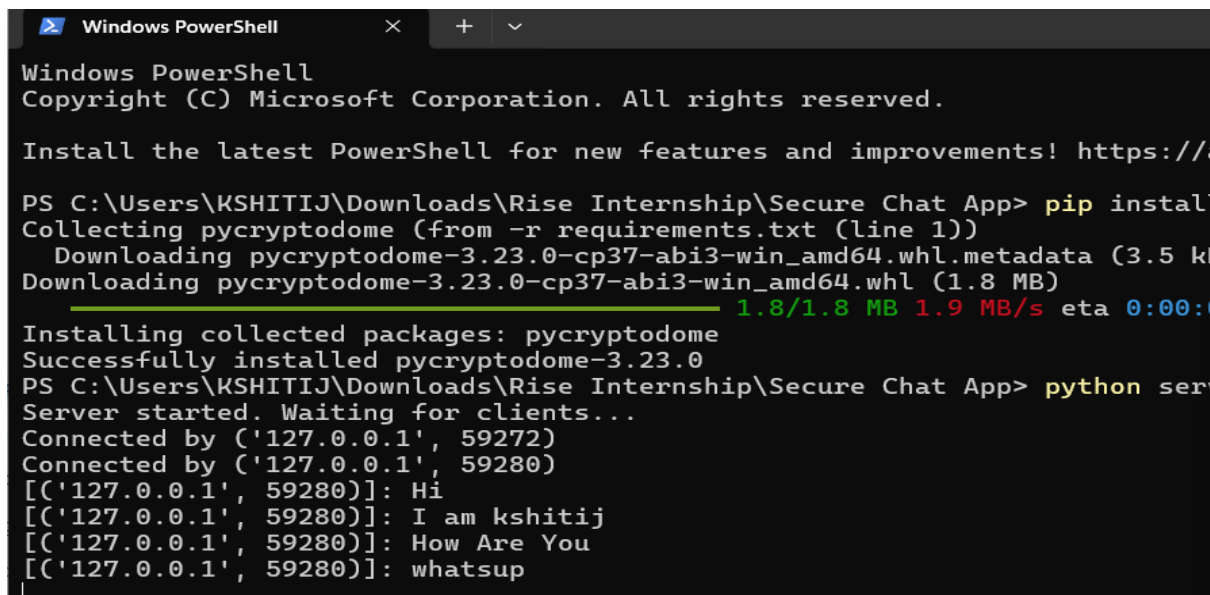
5. Implemented Features

- Console-based server input
- Welcome message on user join
- Error handling for client disconnects
- Localhost and LAN compatibility

8. Challenges Faced

- **User Persistence:** Initially, user data was stored in memory. This was resolved by using a JSON file (users.json) to save credentials.
- **Socket Errors:** Unexpected disconnects led to crashes. Added exception handling to maintain server stability.
- **Server Messaging:** Allowing server interaction required a separate thread to read terminal input.
- **GUI Stability:** Handling multithreaded message updates in Tkinter was tricky but resolved with proper widget locking.

9. Screenshot



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PowerShellLatest

PS C:\Users\KSHITIJ\Downloads\Rise Internship\Secure Chat App> pip install pycryptodome
Collecting pycryptodome (from -r requirements.txt (line 1))
  Downloading pycryptodome-3.23.0-cp37-abi3-win_amd64.whl.metadata (3.5 kB)
  Downloading pycryptodome-3.23.0-cp37-abi3-win_amd64.whl (1.8 MB)
    1.8/1.8 MB 1.9 MB/s eta 0:00:00
Installing collected packages: pycryptodome
Successfully installed pycryptodome-3.23.0
PS C:\Users\KSHITIJ\Downloads\Rise Internship\Secure Chat App> python server.py
Server started. Waiting for clients...
Connected by ('127.0.0.1', 59272)
Connected by ('127.0.0.1', 59280)
[('127.0.0.1', 59280)]: Hi
[('127.0.0.1', 59280)]: I am kshitij
[('127.0.0.1', 59280)]: How Are You
[('127.0.0.1', 59280)]: whatsup
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\KSHITIJ\Downloads\Rise Internship\Secure Chat App> pip install -r requirements.txt
Collecting pycryptodome (from -r requirements.txt (line 1))
  Downloading pycryptodome-3.23.0-cp37-abi3-win_amd64.whl.metadata (3.5 kB)
  Downloading pycryptodome-3.23.0-cp37-abi3-win_amd64.whl (1.8 MB)
    1.8/1.8 MB 1.9 MB/s eta 0:00:00
Installing collected packages: pycryptodome
Successfully installed pycryptodome-3.23.0
PS C:\Users\KSHITIJ\Downloads\Rise Internship\Secure Chat App> python server.py
Server started. Waiting for clients...
Connected by ('127.0.0.1', 59272)
Connected by ('127.0.0.1', 59280)
[('127.0.0.1', 59280)]: Hi
[('127.0.0.1', 59280)]: I am kshitij
[('127.0.0.1', 59280)]: How Are You
```

Secure Chat

whatsup

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\KSHITIJ\Downloads\Rise Internship\Secure Chat App> python server.py
Server started. Waiting for clients...
Connected by ('127.0.0.1', 59904)
[('127.0.0.1', 59904)]: hi
[('127.0.0.1', 59904)]: i am kshitij
[('127.0.0.1', 59904)]: whatsup
[('127.0.0.1', 59904)]: hello
Connected by ('127.0.0.1', 60034)
[('127.0.0.1', 60034)]: hello sir
[('127.0.0.1', 59904)]: forcibly disconnected.
[('127.0.0.1', 60034)]: forcibly disconnected.
Connected by ('127.0.0.1', 60148)
[('127.0.0.1', 60148)]: disconnected during login/registration.
Connected by ('127.0.0.1', 60166)
[('127.0.0.1', 60166)]: disconnected during login/registration.
```

10. Conclusion

The Secure Chat App provides a functional demonstration of how cryptographic techniques can be integrated into real-time communication systems. It combines the essential principles of network programming, data security, and GUI design. This project not only improves understanding of secure socket communication but also serves as a robust tool for encrypted messaging. It's suitable for academic evaluation, internal secure communications, and as a foundational concept for enterprise-level secure messaging apps.

11. Future Scope

- Add message timestamps and typing indicators
- Extend to group chats with rooms
- Support file transfer with encryption
- Deploy the app as a cross-platform executable
- Integrate with a database for advanced user management