SecureCast: Encrypted Messaging and Video Streaming Hub

Team Members:
Anmol Kumar sharma - CS22BT0006
Haridarshan R - EE22BT0025
Mallikarjun S Biradar - EE22BT0034
S. Sai Tharun - CS22BT057
Aman Meena - MC22BT0003

Computer Networks

April 20, 2025

1. Motivation

In today's interconnected world, secure communication is more important than ever. The motivation behind this project is to build a simple and effective encrypted messaging system where users can communicate safely without the risk of message interception or eavesdropping. This project also helped us understand the fundamentals of socket programming, threading, and cryptography in Python. We also include the feature of on-demand video streaming.

2. Goals

- Implement a client-server architecture for messaging.
- Ensure end-to-end encryption using public-key RSA cryptography.
- Broadcast public keys securely to all clients.
- Maintain a list of online users with their public keys.
- Design a video streaming service that allows clients to request and view videos over the network.
- Stream videos with varying resolutions (360p, 720p, 1080p) to simulate adaptive streaming based of HTTP (DASH).
- Allow clients to request available video files and play them using OpenCV.

3. Technology stack

- **Programming Language:** Python 3
- **Socket Programming:** Python's `socket` module
- **Threading:** `threading` module for handling multiple clients
- **Cryptography:** `cryptography` library (RSA encryption)
- **Serialization:** `pickle` module for transmitting complex data
- **Video Processing:** `OpenCV (cv2)` for capturing, resizing, and displaying video frames
- **Data Streaming:** Real-time video frame transmission with adaptive resolution switching
- **Media Formats Supported:** `.mp4`, `.avi`, `.mkv`

4. Features implemented

- Secure server that listens for client connections and handles encrypted message forwarding.
- Each client generates its own pair of RSA keys and sends the public key to the server.
- The server maintains and transmits an updated client list with public keys.
- Clients can send encrypted messages to other users using the recipient's public key.
- Clients decrypt received messages with their private key.
- Client requests from the list of the available video from the server.
- The server sends the video with different quality

5. How to Run Code

Server

1. Open a terminal.
2. Run the server using:

   ```
   make server
   ```

   Example:

   ```
   python server.py 127.0.0.1 12345
   ```

Client

1. Open a terminal.
2. Run the client using:
```
make client1(for client1)
make client2(for client2)
```
3. The client automatically connects to the server and exchanges public keys.
4. Once connected, clients can send messages to any listed client using their username.

Guided execution of Code

- When a client connects to the server, the server sends a message asking the client to choose a service: video or messaging.
- If the client selects messaging, the server prompts for the recipient's name, after which the client can start sending messages.
- If the client selects video, the server prompts the client to type `get_video` to view available services.
- The server sends a list of available videos, and the client selects one.
- The selected video starts playing.

6. Future Improvements

- Add GUI for user-friendly interaction.
- Improve error handling and session management.
- Implement authentication and digital signatures.
- Support for file transfer and multimedia messages.

7. Conclusion

This project successfully demonstrates a secure communication system using fundamental networking and cryptography concepts and this also contain video streaming system with it's database. It lays the groundwork for more sophisticated systems with real-world applications.

Link to Video