॥ सा विद्या या विमुक्तये ॥

Indian Institute of Technology Dharwad

भारतीय प्रौद्योगिकी संस्थान धारवाड

Secure P2P Chat Application with Video Calling

Instructor:	Prof. Tamal Das
Course:	Computer Networks
Date:	April 19, 2025

Team Members

Name	ID Number
HARISH R	220010018
RICKY ROY RAGOLU	220010047
SHIVAPRASAD F. N.	220010054
SIDDHARTH BALAI	CE23BT007
KAILAS S	MC23BT022

Secure P2P Chat Application with Video Calling

Introduction

Our Secure Peer-to-Peer Chat App is a private messaging tool that lets people communicate directly without using any central servers. Unlike regular chat apps that store your messages on company servers, our system keeps everything between you and the person you're talking to.

Key features include:

- **Private messaging**: All texts are locked with strong encryption that only you and your friend can open.
- **File sharing**: Send documents and photos securely, which automatically save to a "Downloads" folder.
- **Video calls**: Make face-to-face calls that stay private.

We built this because everyone deserves simple, private communication without worrying about who might be watching. The app runs on most computers and doesn't require special hardware or expensive subscriptions. Everything stays between you and your chat partners - no middleman, no data collection, and no hidden tracking.

Motivation

In an era of increasing digital communication and privacy concerns, we recognized the need for a secure, decentralized communication platform that doesn't rely on central servers. Our motivation was to create a private, peer-to-peer chat solution with end-to-end encryption that also supports rich media features like file sharing and video calls.

Goals

- Create a completely decentralized chat application
- Implement strong end-to-end encryption for all communications
- Support text messaging, file transfers, and video calls
- Automatically discover peers on the local network
- Ensure user-friendly operation with clear command interface

Tech Stack

Category	Technology	Description
Programming Language	Python 3	Primary language for the entire application
Cryptography	`cryptography` library	RSA-2048 with OAEP padding for end-to-end encryption
Networking	Python `socket`	TCP for encrypted messaging, UDP for peer discovery
Video Processing	OpenCV	Video capture, compression, and display
Audio Processing	sounddevice	Audio capture and playback
Data Processing	numpy	Handling audio/video data streams
Logging	Python `logging`	System activity and error logging
File Handling	Python `os`, `hashlib`	Secure file transfers with SHA-256 verification

```python
import socket
import threading
import time
import os
import hashlib
import json
from datetime import datetime
import logging
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import serialization, hashes
from cryptography.hazmat.backends import default_backend
import sys
import select
import cv2  # OpenCV used for video capture and display
import numpy as np  # For converting image bytes to numpy array
import sounddevice as sd
```

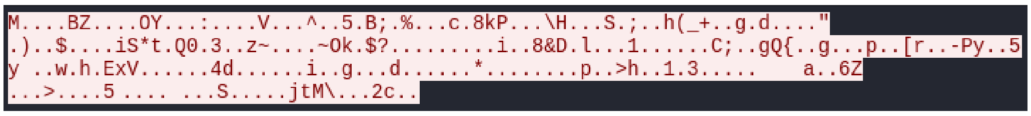
Key Features Implemented

1. Secure Peer Discovery

- Uses UDP broadcast on port 37020 to discover peers on the local network
- Each peer broadcasts their username, IP, listening port, and public key
- Automatically maintains a list of available peers

2. End-to-End Encryption

- RSA 2048-bit asymmetric encryption for all messages
- OAEP padding with SHA-256 for secure encryption
- Public key exchange during connection establishment
- Key verification to prevent man-in-the-middle attacks
- **Wireshark Verification**: Our network captures confirm all peer-to-peer communications are fully encrypted.
- **Traffic Analysis**: Wireshark monitoring shows the expected protocol behavior - UDP broadcasts for discovery followed by encrypted TCP streams for messaging, file transfers, and video calls, with no sensitive data exposed in transit.



```
M....BZ....OY...:....V...^..5.B;.%...c.8kP...\H...S.;..h(_+..g.d...."
.)..$....iS*t.Q0.3..z~....~Ok.$?.........i..8&D.l...1......C;..gQ{..g...p..[r..-Py..5
y ..w.h.ExV......4d......i..g...d......*........p..>h..1.3.....    a..6Z
...>....5 ....  ...S.....jtM\...2c..
```

Figure 1: Wireshark analysis showing encrypted payloads

3. Encrypted Communication Channels

- All text messages are encrypted before transmission
- File transfers are protected with:
  - Encrypted metadata (filename, size, hash)
  - SHA-256 hash verification for file integrity
  - Automatic creation of `downloads` directory on the receiver's side if it doesn't exist

4. Video Calling

- Bidirectional video streaming using OpenCV
- Audio streaming with sounddevice library
- Separate UDP ports for video and audio streams
- Quality compression with JPEG encoding for video

5. Logging System

- Comprehensive logging to `p2p_chat.log`
- Records peer discovery, connection events, and errors
- Helps with debugging and monitoring network activity

Encryption Implementation Details

The application uses RSA-2048 with OAEP padding for all secure communications. Here's how it works:

```python
# Key Generation:
private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048,
    backend=default_backend()
)
public_key = private_key.public_key()

# Message Encryption:
encrypted_msg = public_key.encrypt(
    message.encode('utf-8'),
    padding.OAEP(
        mgf=padding.MGF1(algorithm=hashes.SHA256()),
        algorithm=hashes.SHA256(),
        label=None
    )
)

# Message Decryption:
decrypted_msg = private_key.decrypt(
    encrypted_msg,
    padding.OAEP(
        mgf=padding.MGF1(algorithm=hashes.SHA256()),
        algorithm=hashes.SHA256(),
        label=None
    )
).decode('utf-8')
```

All network traffic except the initial peer discovery broadcasts are encrypted. The log file (`p2p_chat.log`) records connection events but never stores message content or encryption keys.

File Transfer Implementation

When receiving files, the application automatically creates a `downloads` directory if it doesn't exist:

```python
# Create downloads directory if it doesn't exist
os.makedirs('downloads', exist_ok=True)
filepath = os.path.join('downloads', filename)

# Receive file content in chunks
hasher = hashlib.sha256()
received = 0
with open(filepath, 'wb') as f:
    while received < filesize:
        chunk = self.peer_socket.recv(min(4096, filesize - received))
        if not chunk:
            break
        f.write(chunk)
        hasher.update(chunk)
        received += len(chunk)
```

Setup Instructions

Prerequisites

- Python 3.7 or higher
- Required libraries: `cryptography`, `opencv-python`, `sounddevice`, `numpy`

Installation

1. Clone the repository
2. Install dependencies:

```
    pip install cryptography opencv-python sounddevice numpy

```

3. Run the application:

```
    python p2p_chat.py [username] OR python3 p2p_chat.py [username]

```

Usage

1. Start the application with a unique username
2. Discover peers automatically on your local network
3. Use commands:
    - `/help` - To see commands
    - `/connect [username]` - Connect to a peer
    - `/msg [message]` - Send encrypted message
    - `/sendfile [path]` - Send encrypted file (will be saved in receiver's `downloads` folder)
    - `/videocall` - Start video call
    - `/disconnect` - End current connection
    - `/exit` - Quit application

Demonstration Video

Click here to view the demonstration video.

The video demonstrates:

- Peer discovery and connection establishment
- Encrypted text messaging
- Secure file transfer (including automatic `downloads` folder creation)
- Video call initiation and operation
- Network behavior visible in Wireshark (encrypted traffic)

Security Considerations

- All communications are end-to-end encrypted
- Public keys are verified during connection
- No central server means no single point of failure
- File transfers include hash verification
- Video/audio streams use separate UDP ports
- Downloaded files are saved in a dedicated `downloads` directory

Future Enhancements

- Group chat functionality
- Message history with local encryption
- Better video compression and quality adjustment
- NAT traversal for internet-wide communication
- Mobile client support
- Configurable download directory location

This application provides a secure, private communication platform that respects user privacy while offering rich media features, all without relying on any central infrastructure. The automatic creation of the `downloads` directory ensures a seamless file transfer experience for users.