A

Project Report

On

Secure Spam Protected Messaging Platform

Submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology

In

Computer Science and Engineering

By

Harshit Tewari	**2261255**
Tanmay Shah	**2261569**
Harshita Jalal	**2261257**
Shivam Rawat	**2261527**

Under the Guidance of

Mr. Anubhav Bewerwal

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

GRAPHIC ERA HILL UNIVERSITY, BHIMTAL CAMPUS

SATTAL ROAD, P.O. BHOWALI,

DISTRICT- NAINITAL-263132

2024-2025

STUDENT'S DECLARATION

We,**Harshit Tewari , Tanmay Shah , Harshita Jalal , Shivam Rawat** hereby declare the work, which is being presented in the project, entitled **"Secure Spam Protected Messaging Platform"** in partial fulfillment of the requirement for the award of the degree **Bachelor of Technology (B.Tech.)** in the session **2024-2025**, is an authentic record of my work carried out under the supervision of of **Mr. Anubhav *Bewerwal, Assistant Professor***, Department of Computer Science & Engineering, Graphic Era Hill University, Bhimtal.

The matter embodied in this project has not been submitted by me for the award of any other degree.

Date:

Harshit Tewari
Tanmay Shah
Harshita Jalal
Shivam Rawat

CERTIFICATE

The project report entitled "Secure Spam Protected Messaging Platform" being submitted by Harshit Tewari(2261255) S/o Harish Chandra Tewari,Tanmay Shah(2261569) S/o Vijay Shah,Shivam Rawat(2261527) S/o Vishwanbhar Singh Rawat, Harshita Jalal(2261257) D/o Bhagwan Singhof B.Tech.(CSE)to Graphic Era Hill University Bhimtal Campus for the award of bonafide work carried out by them.They have worked under my guidance and supervision and fulfilled the requirement for the submission of a report.

Mr. Anubhav Bewerwal
(Project Guide)

Dr. Ankur Singh Bisht
(Head, CSE)

ACKNOWLEDGEMENT

We take immense pleasure in thanking the Honorable Director**'Prof. (Col.)Anil Nair (Retd.)',** GEHU Bhimtal Campusto permit me and carry out this project work with his excellent and optimistic supervision. This has all been possible due to his novel inspiration, able guidance, and useful suggestions that helped me to develop as a creative researcher and complete the research work, in time.

Words are inadequate in offering my thanks to GOD for providing me with everything that we need. We again want to extend thanks to our president**'Prof. (Dr.) Kamal Ghanshala'** for providing us with all infrastructure and facilities to work in need without which this work could not be possible.

Many thanks to **'Dr. Ankur Singh Bisht'** (Head, Department of Computer Science and Engineering, GEHU Bhimtal Campus), our project guide**'Secure Spam Protected Messaging Platform'**(Assistant Professor, Department of Computer Science and Engineering, GEHU Bhimtal Campus)and other faculties for their insightful comments, constructive suggestions, valuable advice, and time in reviewing this report.

Finally, yet importantly, We would like to express my heartiest thanks to our beloved parents, for their moral support, affection, and blessings. We would also like to pay our sincere thanks to all my friends and well-wishers for their help and wishes for the successful completion of this project.

Harshit Tewari ,2261255

Tanmay Shah, 2261569

Harshita Jalal, 2261257

Shivam Rawat,2261527

ABSTRACT

In the digital age, real-time communication has become an essential part of personal and professional life. However, most existing messaging systems are centralized, making them prone to single points of failure, privacy breaches, and unauthorized data access. To address these challenges, this project presents an **Encrypted Peer-to-Peer (P2P) Chat Application with Spam Detection**, which enables secure, decentralized communication between users without reliance on a central server.

The application is developed using Python and incorporates **DES (Data Encryption Standard)** for securing messages exchanged over the network. It features a basic but effectivespamdetection system that filters out unwanted or potentially harmful messages based on predefined heuristics. Each message is tagged with a unique identifier (UUID), preventing duplication and ensuring that peers do not reprocess the same message.

Communication between peers is achieved using **socket programming** with support for **multi-threading**, allowing the system to handle multiple concurrent peer connections efficiently. Each peer functions as both a client and a server, creating a true peer-to-peer architecture. The application is lightweight, scalable for small networks, and demonstrates key concepts in network security, concurrent programming, and content filtering**.**

This project serves as a practical foundation for building more robust decentralized communication systems. Future enhancements may include stronger encryption algorithms, machine learning-based spam detection, dynamic peer discovery, offline messaging, and a graphical user interface for broader usability.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

P2P	Peer-to-Peer
DES	Data Encryption Standard
AES	Advanced Encryption Standard
RSA	Rivest-Shamir-Adleman (Public-Key Encryption)
UUID	Universally Unique Identifier
GUI	Graphical User Interface
CLI	Command Line Interface
IP	Internet Protocol
TCP	Transmission Control Protocol
SDLC	Software Development Life Cycle
ML	Machine Learning
OS	Operating System
IDE	Integrated Development Environment
DHT	Distributed Hash Table
I/O	Input / Output
SVM	Support Vector Machine

INTRODUCTION

1.1 Prologue

The advancement of digital technologies has significantly transformed the way humans communicate. From emails to instant messaging platforms, the evolution of communication tools has been swift and far-reaching. However, with this growth comes a growing concern for data privacy, network reliability, and protection against malicious content like spam. Centralized messaging platforms are susceptible to server failures, data breaches, and surveillance, thereby compromising user trust and data integrity.

This project aims to address these concerns by developing a secure, encrypted Peer-to-Peer (P2P) Chat Application that not only provides end-to-end communication without relying on a centralized server but also filters out spam messages using a built-in detection mechanism. The use of Data Encryption Standard (DES) for encryption and Python's `socket` and `threading` libraries ensures both security and concurrent communication capabilities. The application is lightweight, extensible, and designed for real-time communication among distributed nodes.

1.2 Background and Motivations

Peer-to-Peer networking has gained popularity due to its decentralized nature, which enhances system reliability and fault tolerance. Unlike traditional client-server models, a P2P architecture allows each node (peer) to function as both a client and a server. This ensures that no single point of failure exists, making the system more robust and scalable.

At the same time, cybersecurity remains a major challenge. Data transmitted over networks can be intercepted, altered, or misused. Thus, encryption is essential for protecting user messages. Additionally, spam messages—ranging from unsolicited advertisements to malicious links—further reduce the effectiveness of communication systems and expose users to potential threats.

The motivation for this project stems from the need for a system that:

- Does not rely on a central authority.
- Provides secure, encrypted communication.

- Prevents spam messages from being transmitted.
- Supports multiple simultaneous peer connections.

This project leverages core networking concepts, encryption techniques, and basic artificial intelligence principles (in spam filtering) to build a secure and intelligent messaging solution.

1.3 Problem Statement

Existing messaging platforms are predominantly centralized, which makes them vulnerable to:

- Server downtimes and outages.
- Unauthorized surveillance and privacy violations.
- Single points of failure affecting all users.
- Widespread dissemination of spam and harmful content.

There is a need for a decentralized chat application that provides the following:

- Encrypted messaging to ensure data privacy.
- A peer-to-peer network structure with no central control.
- A spam detection system to block unwanted or potentially harmful messages.
- Multi-user, concurrent messaging support using threading.

This project proposes a solution that addresses all the above challenges using a Python-based peer-to-peer encrypted chat system with spam filtering.

1.4 Objectives and Research Methodology

Objectives:

- To develop a real-time chat application based on peer-to-peer communication.
- To secure communication using the DES (Data Encryption Standard) encryption algorithm.
- To implement a basic yet functional spam detection algorithm for filtering malicious or unwanted content.

- To utilize Python's socket and threading libraries for efficient and scalable network communication.
- To test message propagation, duplicate filtering, and peer connectivity under multiple concurrent scenarios.

Research Methodology:

- **Literature Survey:** Study existing P2P systems, encryption protocols, and spam detection methods.
- **Design Phase:** Design the application's architecture including modules for networking, encryption, spam filtering, and user I/O.
- **Development:** Implement the chat system using Python, integrating DES and a spam filter module.
- **Testing:** Simulate multiple peer connections and test for encryption accuracy, spam detection reliability, and duplicate message handling.
- **Evaluation:** Assess performance based on criteria such as speed, reliability, and security.

1.5 Project Organization

The project consists of modules for peer connection, encrypted message exchange, spam detection, and user interaction. These are implemented in Python and structured using modular programming practices.

HARDWARE AND SOFTWARE REQUIREMENTS

2.1Hardware Requirement

Windows	OS X	Linux
Microsoft Windows 10/8/7 (32 or 64 bit	macOS 10.13 (High Sierra) or higher	Ubuntu 18.04+, Fedora, Debian with GNOME/KDE/Unity
intel Core i3 or higher	Intel-based Mac	Intel/AMD processor
4 GB minimum, 8 GB recommended	4 GB minimum	4 GB minimum
400 MB disk space for code + 100 MB for logs and cache	400 MB disk space for code + 100 MB for logs and cache	400 MB disk space for code + 100 MB for logs and cache
Ethernet/Wi-Fi connection	Ethernet/Wi-Fi connection	Ethernet/Wi-Fi connection

2.2 Software Requirement

Programming Language Python 3.10 or higher

Development Environment PyCharm, VS Code, or any Python IDE

Python Libraries `socket`, `threading`, `uuid`, `time`, `sys`, `select`

Custom Modules `spam_filter.py` for spam detection, `secure.py` for DES encryption

Encryption Algorithm Data Encryption Standard (DES)

Operating System Support Cross-platform: Windows, macOS, and Linux

Execution Mode CLI-based (Command Line Interface)

2.3 Software Development Lifecycle (SDLC) Phases

The development process followed a simplified version of the Waterfall Model, focusing on the following phases:

1. Requirement Analysis

The goal was to determine the functionalities the system should provide:

- Real-time text-based peer-to-peer communication.
- Secure message exchange using encryption.
- Spam detection and message filtering.
- Concurrent connections using multithreading.

2. System Design

The system was modularized into:

- Networking Module (handling connections and communication)
- Security Module (DES encryption and decryption)
- Spam Filter Module
- User Interaction Module

This modular design enabled parallel development and testing of components.

3. Implementation

The application was implemented using Python. Key tasks included:

- Creating socket-based server and client logic.
- Integrating encryption/decryption with DES.
- Implementing spam detection logic.
- Handling threading for concurrent communication.

4. Testing

The system was tested for:

- Secure transmission across multiple peers.
- Accuracy of spam detection.
- Performance under simultaneous connections.
- Prevention of duplicate message processing.

5. Maintenance and Debugging

Bugs and crashes during peer disconnection, broken pipes, and encryption errors were resolved during testing and peer reviews. Logs and exception handling were added to improve fault tolerance.

CODING OF FUNCTIONS

Handle Client Function

```
def handle_client(conn):
    while True:
        try:
            data = conn.recv(1024).decode()
            if not data:
                break
            decrypted = des_decrypt(data, KEY)
            message_id, sender_id, sender_username, actual_msg = decrypted.split(":", 3)

            if message_id in processed_messages or sender_id == PEER_ID:
                continue

            processed_messages.add(message_id)
            print(f"\n[RECEIVED from {sender_username}]: {actual_msg}\nEnter message: ", end='', flush=True)
        except Exception as e:
            print(f"[ERROR] Connection lost: {e}")
            break
```

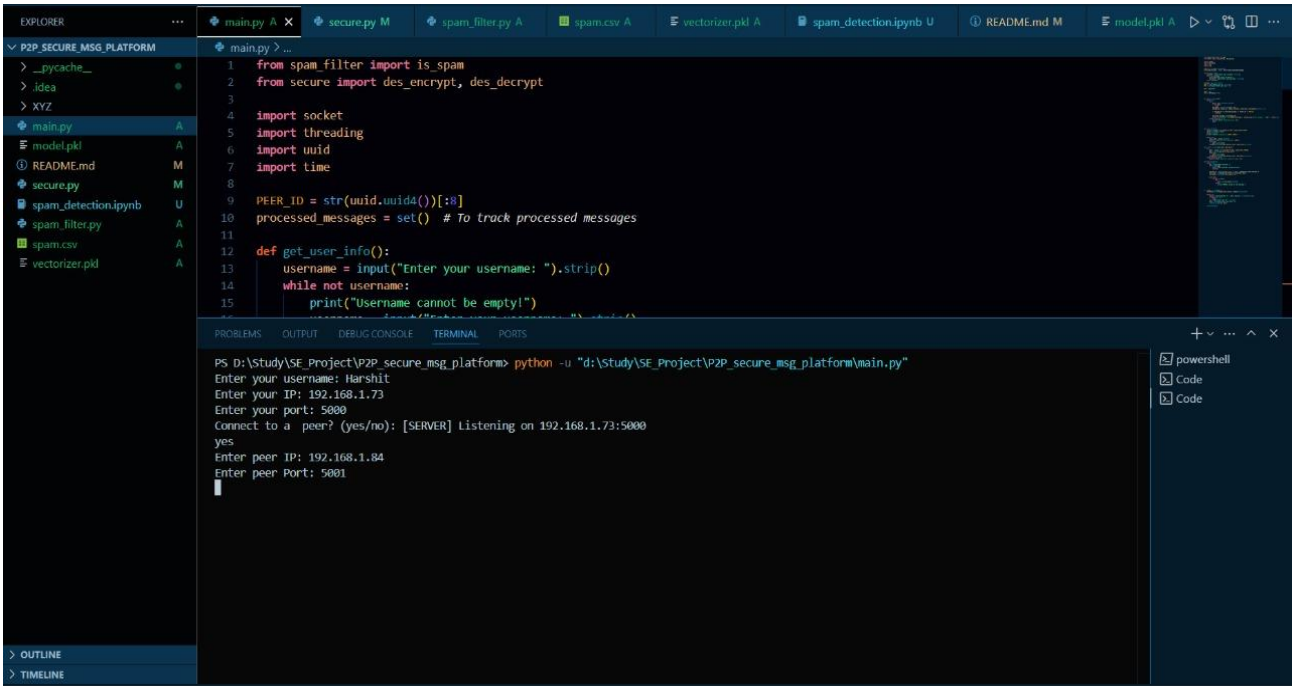
Server start and connecting to peers function

```
def start_server():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind((HOST, PORT))
    server.listen(5)
    print(f"[SERVER] Listening on {HOST}:{PORT}")

    while True:
        conn, addr = server.accept()
        print(f"[CONNECTED] Peer connected: {addr}")
        with lock:
            peers.append(conn)
        threading.Thread(target=handle_client, args=(conn,)).start()

def connect_to_peer(peer_host, peer_port):
    try:
        peer = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        peer.connect((peer_host, peer_port))
        with lock:
            peers.append(peer)
        threading.Thread(target=handle_client, args=(peer,)).start()
    except Exception as e:
        print(f"[ERROR] Could not connect to peer: {e}")
```

DES functions

```python
def permute(block, table):
    return ''.join(block[i - 1] for i in table)

def xor(a, b):
    return ''.join('0' if i == j else '1' for i, j in zip(a, b))

def sbox_substitution(bits):
    result = ''
    for i in range(8):
        block = bits[i * 6:(i + 1) * 6]
        row = int(block[0] + block[5], 2)
        col = int(block[1:5], 2)
        val = S_BOXES[i][row][col]
        result += f'{val:04b}'
    return result

def feistel(right, subkey):
    expanded = permute(right, E)
    xored = xor(expanded, subkey)
    substituted = sbox_substitution(xored)
    return permute(substituted, P)

def key_schedule(key):
    return [key] * 16  # Simple repetition

def des_encrypt_block(plain_bin, key_bin):
    perm = permute(plain_bin, IP)
    left, right = perm[:32], perm[32:]
    keys = key_schedule(key_bin)

    for i in range(16):
        temp = right
        right = xor(left, feistel(right, keys[i]))
        left = temp
```

SNAPSHOTS


EXPLORER
P2P_SECURE_MSG_PLATFORM
__pycache__
.idea
XYZ
main.py
model.pkl
README.md
secure.py
spam_detection.ipynb
spam_filter.py
spam.csv
vectorizer.pkl
OUTLINE
TIMELINE
main.py A
secure.py M
spam_filter.py A
spam.csv A
vectorizer.pkl A
spam_detection.ipynb U
README.md M
model.pkl A
from spam_filter import is_spam
from secure import des_encrypt, des_decrypt
import socket
import threading
import uuid
import time
PEER_ID = str(uuid.uuid4())[:8]
processed_messages = set() # To track processed messages
def get_user_info():
username = input("Enter your username: ").strip()
while not username:
print("Username cannot be empty!")
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS D:\Study\SE_Project\P2P_secure_msg_platform> python -u "d:\Study\SE_Project\P2P_secure_msg_platform\main.py"
Enter your username: Harshit
Enter your IP: 192.168.1.73
Enter your port: 5000
Connect to a peer? (yes/no): [SERVER] Listening on 192.168.1.73:5000
yes
Enter peer IP: 192.168.1.84
Enter peer Port: 5001
powershell
Code
Code

LIMITATIONS

While the developed system successfully demonstrates secure, encrypted, peer-to-peer communication with basic spam detection, it is not without limitations. These constraints arise due to the project scope, time frame, and chosen technologies. Addressing these limitations would be an important step toward production-level deployment.

5.1 Lack of GUI Interface

The application operates entirely through the command-line interface (CLI), which can be less intuitive for non-technical users. A graphical user interface (GUI) would improve usability and accessibility.

5.2 Use of DES Encryption

The current implementation uses the Data Encryption Standard (DES) algorithm, which is now considered outdated and vulnerable to brute-force attacks. For stronger security, advanced algorithms like AES (Advanced Encryption Standard) or RSA should be considered.

5.3 Static Peer Discovery

Peers must manually input IP addresses and port numbers to connect with others. This method does not scale well and limits automation. A dynamic peer discovery protocol (like Distributed Hash Tables) would enhance usability in larger networks.

5.4 Basic Spam Detection

The spam detection algorithm is simple and relies on keyword matching. It may miss context-based spam or produce false positives/negatives. A more intelligent, ML-based filtering mechanism would provide better accuracy.

5.5 No Offline Message Handling

The system does not store or queue messages when a peer is offline. If a recipient is unavailable during transmission, the message is lost.

5.6 No Message History or Logging

The application does not retain chat history or logs. Once a message is received or sent, it is not stored locally or in a database. This limits usability in scenarios requiring message retrieval or auditing.

5.7 No File Transfer Support

The current system is limited to text-based communication. Media and file sharing are not supported, which is a common feature in modern messaging systems.

5.8 Limited Scalability and Fault Tolerance

Although multithreaded connections are used, the application has not been stress-tested for large-scale deployment. Performance may degrade as the number of concurrent peers increases.

ENHANCEMENTS

While the developed system successfully implements the core features of secure communication and spam filtering in a peer-to-peer environment, there are several areas where enhancements can be made to improve functionality, security, scalability, and user experience.

The following suggestions outline potential future work and improvements:

6.1 Integration of Stronger Encryption Algorithms

- Replace the current DES encryption mechanism with more robust and modern algorithms such as AES (Advanced Encryption Standard) or RSA (Rivest-Shamir-Adleman).
- This would improve resistance against brute-force attacks and enhance the overall security of message transmission.

6.2 Development of a Graphical User Interface (GUI)

- Implementing a user-friendly GUI using frameworks like Tkinter, PyQt, or Kivy would make the application more accessible to general users.
- The GUI can include features like contact lists, chat history, file transfer buttons, and peer status indicators.

6.3 Dynamic Peer Discovery

- Introduce automatic peer discovery mechanisms using:
    - Broadcasting techniques on local networks
    - Distributed Hash Table (DHT) architecture
    - QR code-based pairing for manual addition
- This eliminates the need to manually enter IP addresses and ports, improving scalability.

6.4 Advanced Spam Detection

Upgrade the keyword-based spam filter to a machine learning-based classifier (e.g., using Naive Bayes, SVM, or Neural Networks).

- Train the model using a dataset of labeled spam and ham messages to improve detection accuracy and reduce false positives.

6.5 Support for Message History and Logging

- Implement a feature to store sent and received messages in local logs or a lightweight database like SQLite.
- This enables users to access past conversations and maintain an audit trail of communication.

6.6 Offline Messaging Support

- Incorporate a message queuing mechanism to store messages when a peer is offline.
- Automatically deliver stored messages once the peer reconnects.

6.7 File Sharing and Media Support

- Extend the current text-only communication to support file sharing, including documents, images, and videos.
- Ensure encryption is applied to shared files for end-to-end security.

6.8 Improved Scalability and Performance Optimization

- Optimize the threading model and socket buffer size to improve efficiency when handling many peers simultaneously.
- Implement asynchronous I/O or event-driven programming (e.g., using `asyncio`) for better performance.

6.9 Cross-Platform Deployment

- Package the application using tools like PyInstaller or cx_Freeze to create executable installers for Windows, Linux, and macOS.
- This will help in distributing the application to non-developer users.

By implementing these enhancements, the project can evolve from a prototype academic solution to a practical, production-ready communication tool. These improvements will not only expand the usability of the application but also align it with current standards in secure and efficient messaging systems.

CONCLUSION

In this project, asecure, decentralized peer-to-peer (P2P) chat application with built-in spam detectionwas designed and implemented using Python. The primary objective was to enable encrypted communication between users without relying on any centralized server infrastructure, thereby enhancing privacy, reliability, and resilience.

The application successfully integrates DES-based encryption to ensure that messages exchanged between users are secure and cannot be intercepted or tampered with during transmission. To improve the quality of communication, a basic spam detection module was also developed, which blocks predefined spam content before it reaches any peer.

Using socket programming and multithreading, the system supports real-time, full-duplex messaging between multiple peers concurrently. Each peer acts both as a server and a client, demonstrating the true nature of peer-to-peer architecture. The application was tested under different scenarios involving multiple peers, encrypted message exchanges, spam filtering, and message deduplication.

Despite its achievements, the project also exhibits certain limitations, such as the use of a command-line interface, static peer discovery, and basic keyword-based spam detection. These shortcomings present an opportunity for future enhancement and continued development. Features likeGUI integration, stronger encryption algorithms, intelligent spam filtering using machine learning, message persistence, offline messaging**,** andautomated peer discovery can be explored as next steps.

Overall, this project provides a foundational implementation of a secure, lightweight, and privacy-focused messaging system. It demonstrates a practical application of core computer science concepts including cryptography, networking, concurrent programming, and cybersecurity. The skills and insights gained during this project are valuable for developing real-world secure communication systems and for advancing research in peer-to-peer technologies.

REFERENCES

[1] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 7th ed., Pearson Education, 2017.

[2] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks*, 5th ed., Pearson Education, 2013.

[3] Python Software Foundation, "Python 3.10 Documentation," Available: https://docs.python.org/3/

[4] M. Lutz, *Programming Python*, 4th ed., O'Reilly Media, 2010.

[5] B. A. Forouzan, *Data Communications and Networking*, 5th ed., McGraw-Hill Education, 2012.

[6] J. Goerzen and B. D. Curtis, *Foundations of Python Network Programming*, 3rd ed., Apress, 2014.

[7] S. Garfinkel and G. Spafford, *Practical UNIX and Internet Security*, 3rd ed., O'Reilly Media, 2003.

[8] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz, "A Bayesian approach to filtering junk e-mail," in *Proc. AAAI Workshop on Learning for Text Categorization*, Madison, WI, USA, Jul. 1998.

[9] The OpenSSL Project, "OpenSSL: Cryptography and SSL/TLS Toolkit," [Online]. Available: https://www.openssl.org/

[10] R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

.