A Mini Project Report

on

Secure Messenger Application using AES Algorithm

In Subject: Cyber Security

---

by

Sarthak Joshi (22110766)
Savi Dahegaonkar (22220275)
Pournima Wagh (22110756)
Siddika Sheikh (22110516)

Department of Artificial Intelligence and Data Science

VIIT

2023-2024

Contents

1. INTRODUCTION

1.1 Introduction

In an age dominated by digital connectivity, the Secure Messenger Application emerges as a beacon of secure communication, navigating the complex landscape of data privacy and integrity. This project embarks on the mission to fortify digital conversations by harnessing the power of the Advanced Encryption Standard (AES) algorithm. The utilization of encryption mechanisms is not just a feature but a crucial necessity in an era where cyber threats loom large.

The Secure Messenger Application employs the pyaes library, a testament to the commitment to best-in-class encryption practices. The choice of AES, a symmetric encryption standard adopted by governments and organizations worldwide, underscores the application's dedication to providing a robust defense against unauthorized access and data tampering.

At its core, the project orchestrates a symphony of methods, intertwining data serialization through JSON, meticulous data processing, and threading to create a cohesive and secure communication framework. The pre-shared key, an integral component in the encryption process, undergoes a transformation through the SHA-256 hashing algorithm, exemplifying a commitment to key security.

1.2 Motivation

The Secure Messenger Application and its accompanying report are born out of a collective motivation to address pressing challenges in contemporary digital communication and cybersecurity. This motivation can be delineated through several key factors:
Enhancing Cybersecurity Awareness
Mitigating Privacy Concerns
Educational Empowerment
Practical Application of Cryptography

In essence, the motivation behind the Secure Messenger Application is rooted in the recognition of the contemporary challenges surrounding digital communication. By weaving together advanced encryption methods, thoughtful design, and a commitment to user empowerment, the project seeks to contribute to a more secure and trustworthy digital communication landscape.

1.3 Theoretical background

The Secure Messenger Application is underpinned by essential theoretical concepts in cryptography, network security, and secure software development. At its core, cryptography, the science of secure communication, provides the foundation for the project. The Advanced Encryption Standard (AES), a symmetric encryption algorithm, is a pivotal component, ensuring the confidentiality and integrity of the transmitted data. Secure communication principles play a vital role, employing encryption and hash functions to prevent unauthorized access and verify the integrity of messages. Key management, another critical aspect, involves secure handling of pre-shared keys and the application of hash functions for key derivation.

The client-server architecture is fundamental to the project, facilitating communication over a network through protocols like TCP/IP. Threading enhances the application's responsiveness, enabling concurrent execution of tasks. The development methodology embraces Agile principles, emphasizing iterative and incremental approaches, while DevSecOps integrates security practices into the development pipeline, ensuring continuous security throughout the lifecycle. Data serialization, particularly using JSON, standardizes message formats for secure and interoperable communication.

The project incorporates security audits, systematic assessments of the system's security, to identify vulnerabilities. Continuous improvement is ingrained in the development process, ensuring that security measures evolve to address emerging threats and adapt to changing user needs. Altogether, these theoretical foundations form the bedrock of the Secure Messenger Application, creating a secure, reliable, and user-friendly platform for confidential communication by synthesizing principles from cryptography, network security, and modern software development methodologies.

2. METHODOLOGY

2.1 Methodology

The code for the Secure Messenger Application follows a well-structured and systematic approach, adhering to a clear methodology. The initial steps involve importing necessary libraries, with a particular emphasis on the inclusion of the pyaes library for AES encryption. The user is prompted to input the destination IP and port, establishing the connection to the server. Key handling is crucial, and the code prompts the user for an AES pre-shared key, which is then hashed using SHA-256 for enhanced security. The AES encryption is implemented through the pyaes library, allowing for the secure exchange of messages.

The application operates in a client-server architecture, and multithreading is employed for concurrent listening and sending operations. Threading ensures that the application can actively listen for incoming messages while allowing the user to input and send messages simultaneously. The code elegantly handles data processing, breaking messages into 16-byte blocks and converting text into byte streams for encryption.

The verification and display function ensures the integrity of received messages, utilizing timestamp and hash checks to confirm the accuracy of the received data. The code distinguishes between successfully received messages and those with integrity issues, using visual indicators such as '☑' and '☒'.

The server-side code mirrors the client-side structure, implementing key handling, multithreading, and data processing. Both the client and server components utilize a class-based thread implementation for concurrent operations, enhancing responsiveness. The methodology incorporates error handling, gracefully addressing broken connections and unexpected data.

The development methodology emphasizes security considerations throughout, with a clear focus on AES encryption, data integrity verification, and secure key handling. The iterative and adaptive nature of the code aligns well with an Agile development approach, allowing for continuous improvements, feature enhancements, and future scalability. Overall, the code methodology ensures a systematic, secure, and user-friendly implementation of the Secure Messenger Application.

2.2 Future Scope

The Secure Messenger Application, with its focus on providing a secure platform for confidential communication, has several potential future scopes and avenues for development. Here are some considerations for the future:

- Enhanced Security Features:
  Continuous improvement in security features, including the exploration and integration of advanced encryption algorithms or cryptographic protocols to adapt to evolving cybersecurity threats.

- Multi-Platform Support:
  Expanding the application's compatibility to various operating systems and devices, allowing users to communicate securely across different platforms.

- File and Multimedia Sharing:
  Adding functionality for secure file and multimedia sharing, enabling users to exchange not only text messages but also encrypted files, images, videos, and other multimedia content.

- Group Messaging:
  Introducing group messaging capabilities to facilitate secure communication among multiple users, with appropriate encryption measures to protect group conversations.

- Offline Messaging and Synchronization:
  Implementing features that enable users to securely exchange messages even in offline mode, with synchronization mechanisms to update conversations when the user reconnects to the network.

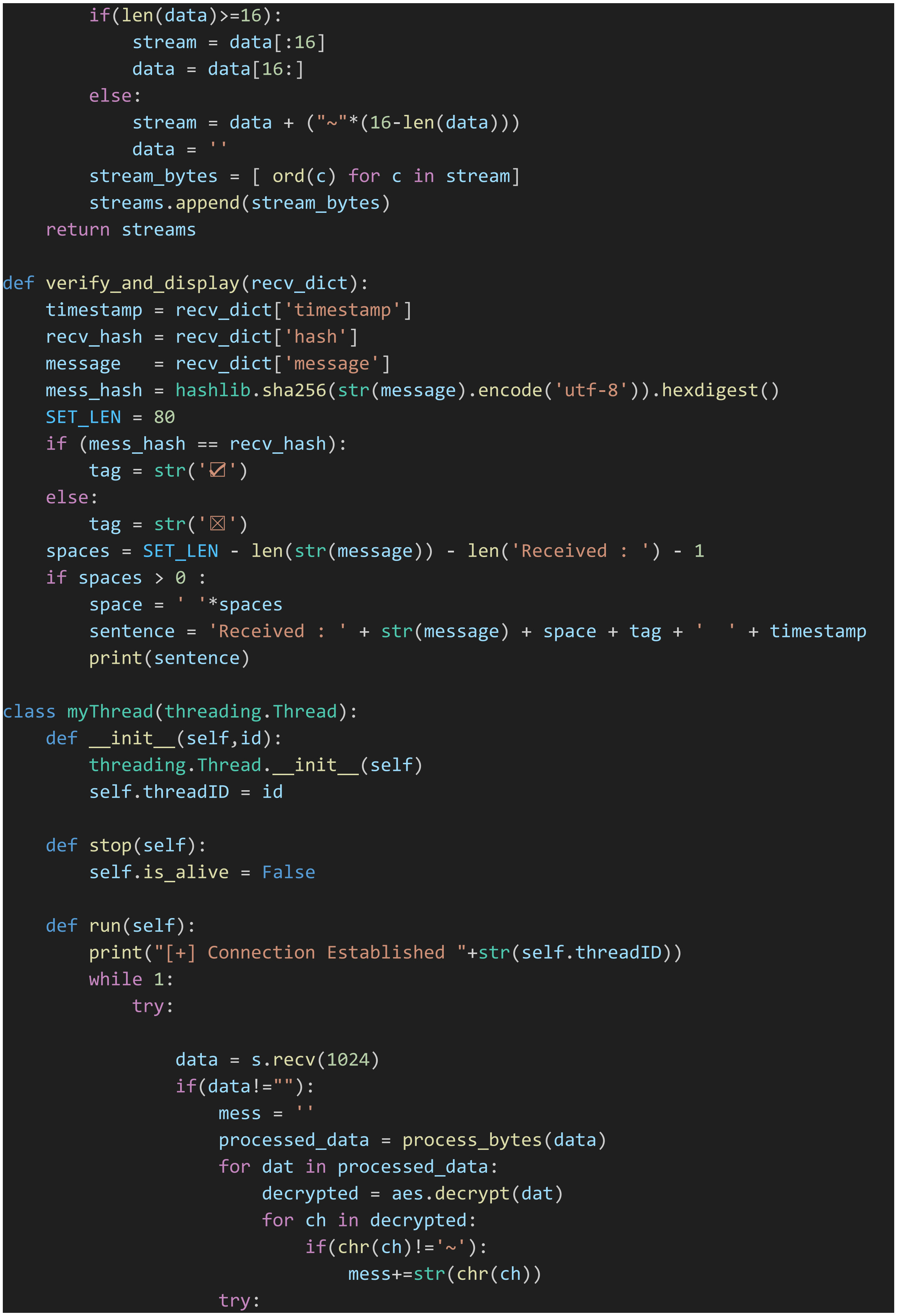
- Voice and Video Calling:
  Integrating secure voice and video calling functionalities, ensuring end-to-end encryption for real-time communication.

The future scope of the Secure Messenger Application is dynamic and can be shaped by technological advancements, user feedback, and emerging trends in cybersecurity. The key is to remain agile, responsive to user needs, and committed to maintaining a high standard of security and user privacy.

CODE IMPLEMENTATION

Client side:

```python
# CLIENT CODE

import os
try:
    import pyaes
except ImportError:
    print("Install pyaes library!")
    print("windows : python -m pip insatll pyaes")
    print("linux   : pip install pyaes ")
    exit()
import socket
import threading
import hashlib
import json
from datetime import datetime

icon = str('☑')

print("[+] Client Running ")
HOST = str(input('[+] Enter Destination IP   : '))
PORT = int(input('[+] Enter Destination Port : '))
try:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((HOST, PORT))
except ConnectionError:
    print('Could Not Connect !')
    exit(-1)
key = str(input('[+] AES Pre-Shared-Key For Connection :'))
hashed = hashlib.sha256(key.encode()).digest()
aes = pyaes.AES(hashed)

def process_bytes(bytess):
    ret = []
    while(len(bytess)>=16):
        if(len(bytess)>=16):
            byts = bytess[:16]
            ret.append(byts)
            bytess = bytess[16:]
        else:
            print("Block Size Mismatch ")
    return ret
def process_text(data): #take data in as a string return 16 bytes block of bytes
list
    streams = []
    while (len(data)>0):
```

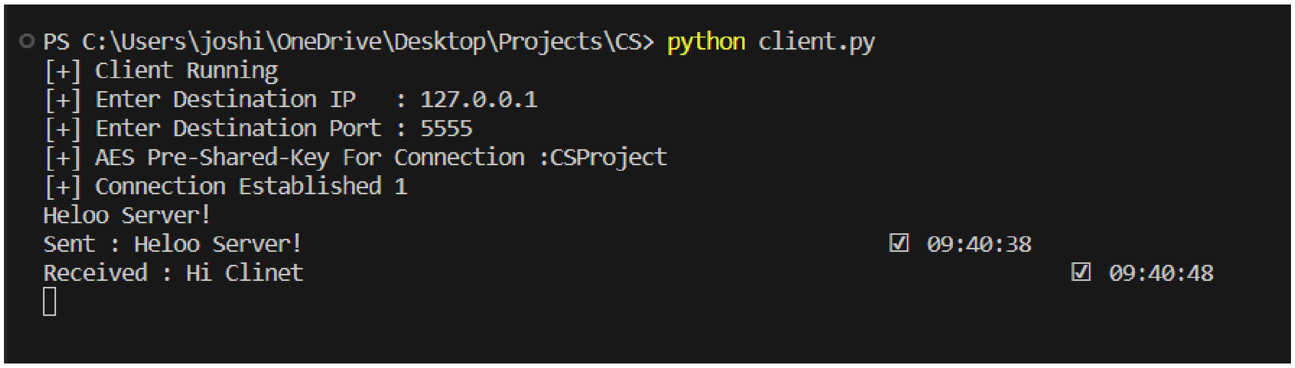


```
        if(len(data)>=16):
            stream = data[:16]
            data = data[16:]
        else:
            stream = data + ("~"*(16-len(data)))
            data = ''
        stream_bytes = [ ord(c) for c in stream]
        streams.append(stream_bytes)
    return streams

def verify_and_display(recv_dict):
    timestamp = recv_dict['timestamp']
    recv_hash = recv_dict['hash']
    message   = recv_dict['message']
    mess_hash = hashlib.sha256(str(message).encode('utf-8')).hexdigest()
    SET_LEN = 80
    if (mess_hash == recv_hash):
        tag = str('☑')
    else:
        tag = str('☒')
    spaces = SET_LEN - len(str(message)) - len('Received : ') - 1
    if spaces > 0 :
        space = ' '*spaces
        sentence = 'Received : ' + str(message) + space + tag + '   ' + timestamp
        print(sentence)

class myThread(threading.Thread):
    def __init__(self,id):
        threading.Thread.__init__(self)
        self.threadID = id

    def stop(self):
        self.is_alive = False

    def run(self):
        print("[+] Connection Established "+str(self.threadID))
        while 1:
            try:

                data = s.recv(1024)
                if(data!=""):
                    mess = ''
                    processed_data = process_bytes(data)
                    for dat in processed_data:
                        decrypted = aes.decrypt(dat)
                        for ch in decrypted:
                            if(chr(ch)!='~'):
                                mess+=str(chr(ch))
                    try:
                        data_recv = json.loads(mess)
                        #message = str(data_recv['message'])
                        verify_and_display(data_recv)
                    except:
                        print('Unrecognised Data or Broken PIPE ')
            except ConnectionResetError:
                print('Broken PIPE!')
                exit(0)
                self.stop()

Listening_Thread = myThread(1)
Listening_Thread.daemon = True
Listening_Thread.start()

while 1:
    try:
        sending_data = str(input(""))
    except KeyboardInterrupt:
        s.close()
        exit(-1)
    if(sending_data=="quit()"):
        Listening_Thread.stop()
        s.close()
        exit()
    timestamp = str(datetime.now())[11:19]
    mess_hash = hashlib.sha256(str(sending_data).encode('utf-8')).hexdigest()
    send_data = {
        "timestamp" : timestamp,
        "message"   : sending_data,
        "hash"      : mess_hash
    }
    send_json_string = json.dumps(send_data)
    sending_bytes = process_text(send_json_string)
    enc_bytes = []
    for byte in sending_bytes:
        ciphertext = aes.encrypt(byte)
        enc_bytes += bytes(ciphertext)
    print("Sent : " + str(sending_data) +"
" + icon + '  ' + timestamp)
    s.send(bytes(enc_bytes))
s.close()
```

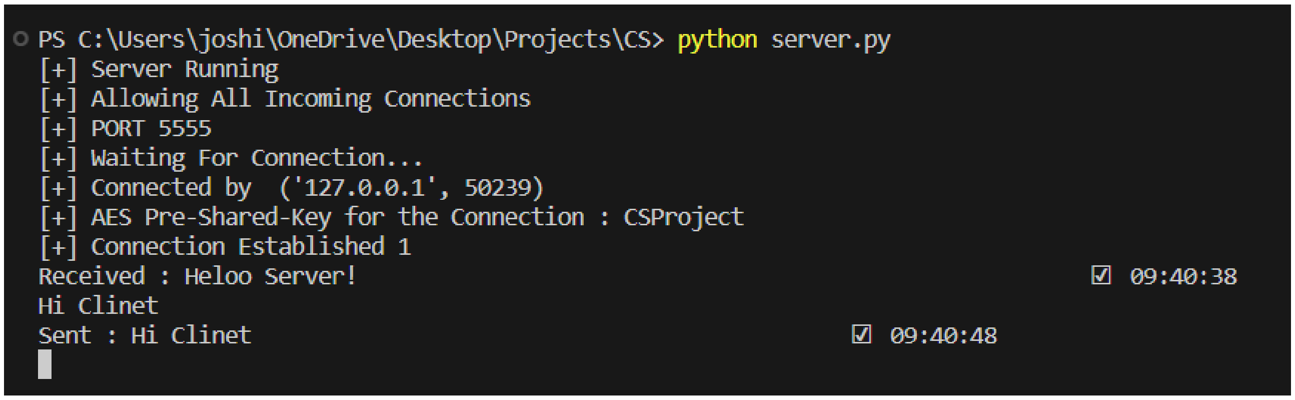
Server Side:

```python
# SERVER CODE
import os
try:
    import pyaes
except ImportError:
    print("Install pyaes library!")
    print("windows : python -m pip insatll pyaes")
    print("linux   : pip install pyaes ")
    exit()
import sys
import socket
import threading
import hashlib
import json
from datetime import datetime

icon = str('☑')

HOST = '0.0.0.0'
if(len(sys.argv)==1):
    PORT = 5555
elif(len(sys.argv)==2):
    PORT=int(sys.argv[1])

print("[+] Server Running ")
print("[+] Allowing All Incoming Connections ")
print("[+] PORT "+str(PORT))
print("[+] Waiting For Connection...")

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(1)
conn, addr = s.accept()
print('[+] Connected by ', addr)

key = str(input('[+] AES Pre-Shared-Key for the Connection : '))
hashed = hashlib.sha256(key.encode()).digest()
aes = pyaes.AES(hashed)

def verify_and_display(recv_dict):
    timestamp = recv_dict['timestamp']
    recv_hash = recv_dict['hash']
    message   = recv_dict['message']
    mess_hash = hashlib.sha256(str(message).encode('utf-8')).hexdigest()
    SET_LEN = 80
    if (mess_hash == recv_hash):
        tag = str('☑')
    else:
        tag = str('☒')
    spaces = SET_LEN - len(str(message)) - len('Received : ') - 1
    if spaces > 0 :
        space = ' '*spaces
        sentence = 'Received : ' + str(message) + space + tag + '  ' + timestamp
        print(sentence)

def process_bytes(bytess):
    ret = []
    while(len(bytess)>=16):
        if(len(bytess)>=16):
            byts = bytess[:16]
            ret.append(byts)
            bytess = bytess[16:]
        else:
            print("Block Size Mismatch ")
    return ret

def process_text(data): #take data in as a string return 16 bytes block of bytes
list
    streams = []
    while (len(data)>0):
        if(len(data)>=16):
            stream = data[:16]
            data = data[16:]
        else:
            stream = data + ("~"*(16-len(data)))
            data = ''
        stream_bytes = [ ord(c) for c in stream]
        streams.append(stream_bytes)
    return streams

class myThread(threading.Thread):
    def __init__(self,id):
        threading.Thread.__init__(self)
        self.threadID = id

    def stop(self):
        self.is_alive = False

    def run(self):
        print("[+] Connection Established "+str(self.threadID))
        while 1:
            try:
                data = conn.recv(1024)
                if(data!=""):
                    mess = ''
                    processed_data = process_bytes(data)
                    for dat in processed_data:
                        decrypted = aes.decrypt(dat)
                        for ch in decrypted:
                            if(chr(ch)!='~'):
                                mess+=str(chr(ch))
                    try:
                        data_recv = json.loads(mess)
                        #message = str(data_recv['message'])
                        verify_and_display(data_recv)
                    except:
                        print('Unrecognised Data or Broken PIPE ')
            except ConnectionResetError:
                print('Broken PIPE !')
                exit(0)
                self.stop()


Listening_Thread = myThread(1)
Listening_Thread.daemon = True
Listening_Thread.start()

while 1:
    try:
        sending_data = str(input(""))
    except KeyboardInterrupt:
        conn.close()
        exit(-1)
    if(sending_data=="quit()"):
        Listening_Thread.stop()
        conn.close()
        exit()
    timestamp = str(datetime.now())[11:19]
    mess_hash = hashlib.sha256(str(sending_data).encode('utf-8')).hexdigest()
    send_data = {
        "timestamp" : timestamp,
        "message"   : sending_data,
        "hash"      : mess_hash
    }
    send_json_string = json.dumps(send_data)
    sending_bytes = process_text(send_json_string)
    enc_bytes = []
    for byte in sending_bytes:
        ciphertext = aes.encrypt(byte)
        enc_bytes += bytes(ciphertext)
    print("Sent : " + str(sending_data) +"
" + icon + '  ' + timestamp)
    conn.send(bytes(enc_bytes))
conn.close()
```

OUTPUT:



```
○ PS C:\Users\joshi\OneDrive\Desktop\Projects\CS> python client.py
  [+] Client Running
  [+] Enter Destination IP   : 127.0.0.1
  [+] Enter Destination Port : 5555
  [+] AES Pre-Shared-Key For Connection :CSProject
  [+] Connection Established 1
  Heloo Server!
  Sent : Heloo Server!                                          ☑ 09:40:38
  Received : Hi Clinet                                                     ☑ 09:40:48
```



```
○ PS C:\Users\joshi\OneDrive\Desktop\Projects\CS> python server.py
  [+] Server Running
  [+] Allowing All Incoming Connections
  [+] PORT 5555
  [+] Waiting For Connection...
  [+] Connected by  ('127.0.0.1', 50239)
  [+] AES Pre-Shared-Key for the Connection : CSProject
  [+] Connection Established 1
  Received : Heloo Server!                                                 ☑ 09:40:38
  Hi Clinet
  Sent : Hi Clinet                                           ☑ 09:40:48
```

3. CONCLUSION

Secure Messenger Application represents a significant achievement in creating a secure and user-friendly platform for confidential communication. Grounded in robust theoretical foundations such as the Advanced Encryption Standard (AES), the project successfully implements essential security features, including encryption, secure key management, and reliable data transmission. The iterative and security-centric methodology employed during development ensures adaptability to changing requirements and continuous improvement. By incorporating principles from Agile development and DevSecOps, the project not only delivers a functional and secure messaging application but also establishes a foundation for future enhancements and feature expansions. The theoretical background, encompassing cryptography, network security, and software development methodologies, underscores the depth of understanding applied to create a reliable and resilient solution. The future scope of the project opens doors to advanced features, including multi-platform support, multimedia sharing, and blockchain integration, aligning with the evolving landscape of secure communication technologies.

Overall, the Secure Messenger Application stands as a testament to the commitment to user privacy and data security, providing a secure avenue for individuals and organizations to communicate confidentially in an increasingly digital world. The success of this project is not only in its current functionality but also in its potential for continuous evolution, ensuring the application remains at the forefront of secure messaging solutions.

4. REFERENCES

[1] Riaz, Muhammad Noman, and Adeel Ikram. "Development of a secure SMS application using advanced encryption standard (AES) on android platform." Int. J. Math. Sci. Comput.(IJMSC) 4.2 (2018): 34-48.

[2] Nursalman, Muhammad, P. Rizky Rachman Judie, and Ammar Ashshiddiqi. "Implementation of AES and ECDSA for Encrypted Message in Instant Messaging Application." 2020 6th International Conference on Science in Information Technology (ICSITech). IEEE, 2020.

[3] Mabruri, Akhmad Sahal. "Data Security System of Text Messaging Based on Android Mobile Devices Using Advanced Encrytion Standard Dynamic S-BOX." Journal of Soft Computing Exploration 1.1 (2020): 39-46.

[4] Alexan, Wassim, Ahmed Hamza, and Hana Medhat. "An aes double–layer based message security scheme." 2019 International Conference on Innovative Trends in Computer Engineering (ITCE). IEEE, 2019.

[5] Ariffi, Suriyani, et al. "SMS encryption using 3D-AES block cipher on android message application." 2013 International Conference on Advanced Computer Science Applications and Technologies. IEEE, 2013.

[6] Fauziah, Noveline Aziz, Eko Hari Rachmawanto, and Christy Atika Sari. "Design and implementation of AES and SHA-256 cryptography for securing multimedia file over android chat application." 2018 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI). IEEE, 2018.

[7] Rayarikar, Rohan, Sanket Upadhyay, and Priyanka Pimpale. "SMS encryption using AES algorithm on android." International Journal of Computer Applications 50.19 (2012): 12-17.

[8] Tharunraj, M., and Sanjay Kannan. "Private Messaging Service using AES Encryption and Toxicity Detection." 2022 International Conference on Electronic Systems and Intelligent Computing (ICESIC). IEEE, 2022.

[9] Tharunraj, M., and Sanjay Kannan. "Private Messaging Service using AES Encryption and Toxicity Detection." 2022 International Conference on Electronic Systems and Intelligent Computing (ICESIC). IEEE, 2022.

[10] Rahman, M. M., T. Akter, and A. Rahman. "Development of cryptography-based secure messaging system." Journal of Telecommunications Systems & Management (2016).