



RV Educational Institutions®
RV College of Engineering®

Autonomous
Institution Affiliated
to Visvesvaraya
Technological
University, Belagavi

Approved by AICTE,
New Delhi, Accredited
By NAAC, Bengaluru
And NBA, New Delhi

Go, change the world

**DEPARTMENT OF INFORMATION SCIENCE AND
ENGINEERING**

Encrypted SocketChat Application using python
CRYPTOGRAPHY & NETWORK SECURITY

Mini Project Report

(18IS63)

Submitted by

Maaz Afnan

1RV18IS022

Nehal N Shet

1RV18IS026

Under the guidance of

Dr. Padmashree T

Prof. Sushmitha N

Assistant Professor, ISE Dept.

Assistant Professor, ISE Dept.

Dept. of ISE

RV College of Engineering

In partial fulfilment for the award of degree

of

Bachelor of Engineering

In

Information Science and Engineering

2020-2021

RV COLLEGE OF ENGINEERING®, BENGALURU-59

(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING



CERTIFICATE

Certified that the cryptography and network security mini project work titled '*Encrypted SocketChat Application using python*' is carried out by **Maaz Afnan (1RV18IS022)** and **Nehal N Shet (1RV18IS026)** who are bonafide students of Sixth Semester, RV College of Engineering, Bengaluru, in partial fulfilment for the award of degree of **Bachelor of Engineering in Information Science and Engineering** of the Visvesvaraya Technological University, Belagavi during the year 2020-2021. It is certified that all corrections/suggestions indicated for the Internal Assessment have been incorporated in the mini project report deposited in the departmental library. The mini project report has been approved as it satisfies the academic requirements in respect of mini project work prescribed by the institution for the said degree.

Signature of Guide
Dr. Padmashree T

Signature of Guide
Prof. Sushmitha N

Signature of Head of the Department
Dr. Sagar B M

External Viva

Name of Examiners

Signature with Date

1

2

RV COLLEGE OF ENGINEERING®, BENGALURU-59

(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

DECLARATION

We, Maaz Afnan and Nehal N Shet, students of Sixth semester B.E, RV College of Engineering, Bengaluru, hereby declare that the cryptography and network security mini project titled **'Encrypted SocketChat Application using Python'** has been carried out by us and submitted in partial fulfilment for the award of degree of **Bachelor of Engineering** in Information Science and Engineering during the year 2020-21.

Further we declare that the content of the dissertation has not been submitted previously by anybody for the award of any degree or diploma to any other university.

We also declare that any Intellectual Property Rights generated out of this project carried out at RVCE will be the property of RV College of Engineering, Bengaluru and we will be one of the authors of the same.

Place: Bengaluru

Date:

Name

Signature

1. Maaz Afnan (1RV18IS022)
2. Nehal N Shet (1RV18IS026)

ACKNOWLEDGEMENT

We are indebted to our guide, **Dr. Padmashree T and Prof. Sushmitha N, Dept. of Information Science and Engineering**, for their wholehearted support, suggestions and invaluable advice throughout our project work and also helped in the preparation of this thesis.

Our sincere thanks to **Dr. Sagar B M**, Professor and Head, Department of Information Science and Engineering, RVCE for his support and encouragement.

We express sincere gratitude to our beloved Principal, **Dr. K. N. Subramanya** for his appreciation towards this project work.

We thank all the **teaching staff and technical staff** of the Information Science and Engineering department, RVCE for their help.

Lastly, we take this opportunity to thank our **family** members and **friends** who provided all the backup support throughout the project work.

Maaz Afnan (1RV18IS022)

Nehal N Shet (1RV18IS026)

ABSTRACT

Encrypted SocketChat is a simple chat room script in python. It uses AES encryption for secure data transfer over the public network. The server will be active and the client can connect from remote systems with the correct password. Multiple clients can connect to one single server from remote systems for a group chat. The server will be running at one end and will be waiting for a connection from the client. Clients will send connection requests and get connected to the server through sockets. The objective is to implement a simple chat room application in python and use AES encryption to encrypt and decrypt chat conversations.

Socket library in python is used here to establish the connection. The clients can easily communicate through socket connections, but this conversation is not secure and is subjected to cyber attacks. In order to make our conversation secure the AES encryption method to encrypt and decrypt the data will be used. Pycrypto is a collection of both secure hash functions (such as SHA256 and RIPEMD160), and various encryption algorithms (AES, DES, RSA, ElGamal, etc.).

The main objective of the project is to develop a Secure Chat Application. A wide range of literature review was done in order to achieve all the tasks, which gave information about some of the products that are existing in the market. The portability of the application has been achieved by using some of the libraries of python and the use of python virtual environment makes it portable to run on any system. The connections established using sockets are reliable without getting disconnected for many hours. Encryption performed using AES is very secure as the key space is large, due to its robust security brute force attack is not possible and is resilient against hacking attempts due to its higher length key sizes. The chat-room displays when the chat-room is empty and also whenever a user connects or disconnects from the server.

TABLE OF CONTENTS

	Page No
Abstract	
List of Tables	7
List of Figures	8
Chapter 1	
1. Introduction	10
Chapter 2	
2. Software Requirements Specifications	13
Chapter 3	
3. Design	15
Chapter 4	
4. Implementation & Testing	20
Chapter 5	
5. Conclusion and Future scope	32
References	33

LIST OF TABLES

Table No.	List of Tables	Page No
Table 4.1	Login	23
Table 4.2	Encryption and Decryption	25
Table 4.3	Server Side	27
Table 4.4	Client Side	29

List of Figures

Figure No.	List of Figures	Page No
Figure 3.1	Architecture block diagram	15
Figure 3.2	Architecture block diagram	16
Figure 3.3	Sequence diagram	16
Figure 3.4	Use Case diagram	17
Figure 3.5	Level 0 DFD	17
Figure 3.6	Level 1 DFD	18
Figure 4.1	Successful login	24
Figure 4.2	Incorrect port number	24
Figure 4.3	Decryption	26
Figure 4.4	Decryption	26
Figure 4.5	Encryption	26
Figure 4.6	New user joins, Encrypted texts, User disconnects	28
Figure 4.7	New User joins, User disconnects, Server down	30

CHAPTER-1

Introduction

CHAPTER 1

Introduction

A Chat Application is where many users or clients connect to have a conversation and share information. A server is set up which enables clients to connect to each other, the conversation the clients have should be encrypted to secure critical information. Sockets allow communication between two different processes on the same or different machines. To be more precise, it's a way to talk to other computers using standard Unix file descriptors. In Unix, every I/O action is done by writing or reading a file descriptor. A file descriptor is just an integer associated with an open file and it can be a network connection, a text file, a terminal, or something else. To a programmer, a socket looks and behaves much like a low-level file descriptor. This is because commands such as `read()` and `write()` work with sockets in the same way they do with files and pipes.

Socket.IO provides the ability to implement real-time analytics, binary streaming, instant messaging, and document collaboration. Notable users include Microsoft Office, Yammer, and Zendesk. Socket.IO handles the connection transparently. It will automatically upgrade to WebSocket if possible. This requires the programmer to only have Socket.IO knowledge. Socket.IO is not a WebSocket library with fallback options to other real time protocols. It is a custom real time transport protocol implementation on top of other real time protocols. A Socket.IO implementing server cannot connect to a non-Socket.IO WebSocket client. A Socket.IO implementing client cannot talk to a non-Socket.IO WebSocket or Long Polling Comet server. Socket.IO requires using the Socket.IO libraries on both client and server side. As of version 2.0, Socket.IO makes use of WebSockets as the underlying WebSocket library.

The Advanced Encryption Standard (AES) is a symmetric block cipher chosen by the U.S. government to protect classified information. AES is implemented in software and hardware throughout the world to encrypt sensitive data. It is essential for government computer security, cybersecurity and electronic data protection. The National Institute of Standards and Technology (NIST) started development of AES in 1997 when it announced the need for an alternative to the

Data Encryption Standard (DES), which was starting to become vulnerable to brute-force attacks. NIST stated that the newer, advanced encryption algorithm would be unclassified and must be "capable of protecting sensitive government information well into the [21st] century." It was intended to be easy to implement in hardware and software, as well as in restricted environments -- such as a smart card -- and offer decent defenses against various attack techniques. AES was created for the U.S. government with additional voluntary, free use in public or private, commercial or noncommercial programs that provide encryption services. However, nongovernmental organizations choosing to use AES are subject to limitations created by U.S. export control.

AES includes three block ciphers: AES-128, AES-192 and AES-256. AES-128 uses a 128-bit key length to encrypt and decrypt a block of messages, while AES-192 uses a 192-bit key length and AES-256 a 256-bit key length to encrypt and decrypt messages. Each cipher encrypts and decrypts data in blocks of 128 bits using cryptographic keys of 128, 192 and 256 bits, respectively. Symmetric, also known as secret key, ciphers use the same key for encrypting and decrypting, so the sender and the receiver must both know -- and use -- the same secret key. The government classifies information in three categories: Confidential, Secret or Top Secret. All key lengths can be used to protect the Confidential and Secret level. Top Secret information requires either 192- or 256-bit key lengths. There are 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. A round consists of several processing steps that include substitution, transposition and mixing of the input plaintext to transform it into the final output of ciphertext.

CHAPTER-2

Software Requirements Specifications

CHAPTER 2

Software Requirements Specifications

Hardware Requirements

- Ubuntu 18.04LTS and above or Windows 10
- 4GB RAM
- i5 7th gen processor and above or equivalent
- 40GB of Free Hard Disk Space

Software Requirement

- Python 2.7 and above
- Python virtual environment
- Pycrypto library
- Socket library
- Windows 8 and above, Linux 18.04 and above OS

Functional Requirement

1. The system must encrypt 128-bit of plain text.
2. Clients must be able to join the chatroom with password only.
3. Multiple clients should be able to join the server.
4. System should show all the clients that join the chatroom.
5. The chats of all the clients are encrypted.

Non Functional Requirements

1. Server must be running 24X7 to serve client requests.
2. The system must never store chats in plain text.
3. Application will be platform independent.
4. The system is highly secure.

CHAPTER-3

Design

CHAPTER 3

Design

The chat-room is set up using a server which sets up a socket on each end and allows multiple clients to interact with other clients via the server. The socket on the server side associates itself with some hardware port on the server side. Any client that has a socket associated with the same port can communicate with the server socket. The chat conversations between clients are encrypted using AES while sending the message in the chatroom, hence even the server can not receive the conversations in plain text.

Architecture Diagram

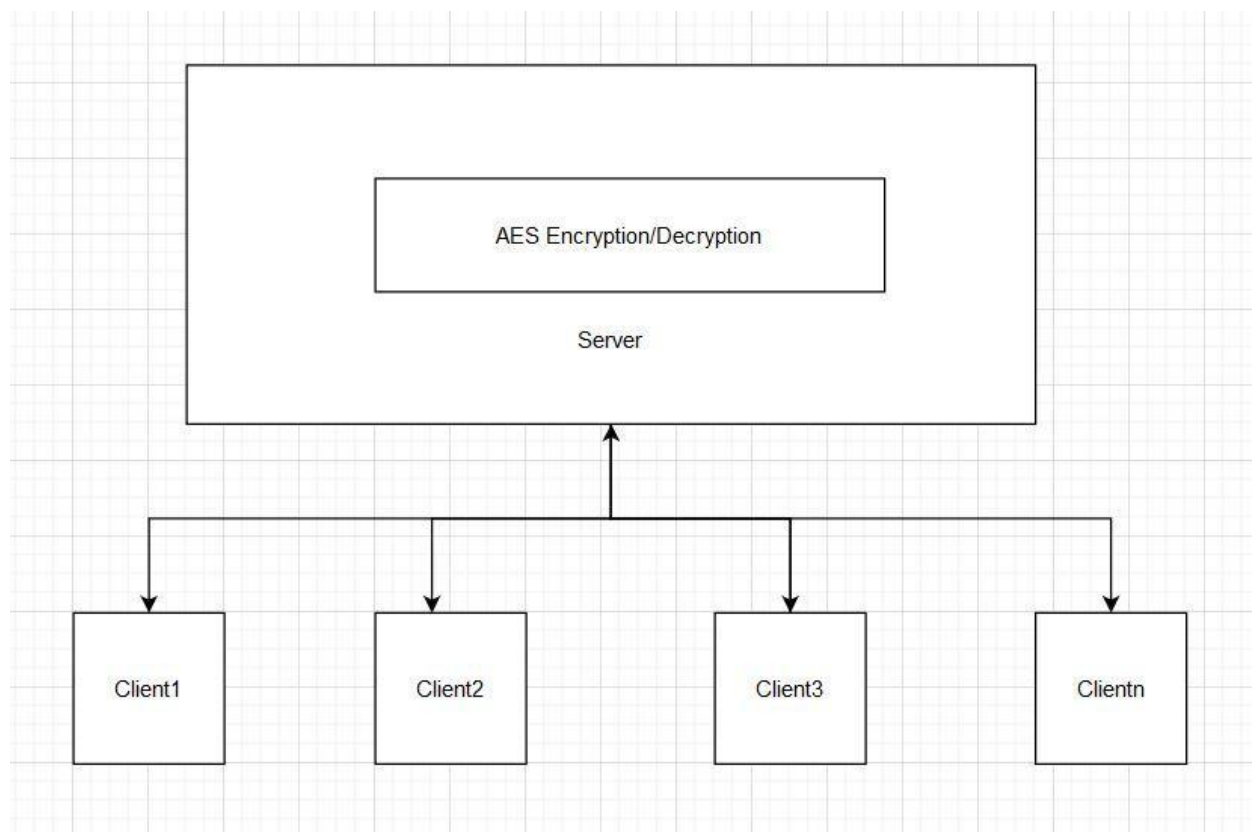


Fig 3.1 Architecture block diagram of SocketChat

Fig 3.1 describes the high level architecture of our socket chat application, where many users can connect to a server and communicate with each other and the AES Encryption/Decryption model

will make this communication channel secure. The below Fig 3.2 explains how the communication and its encryption and decryption workflow takes place.

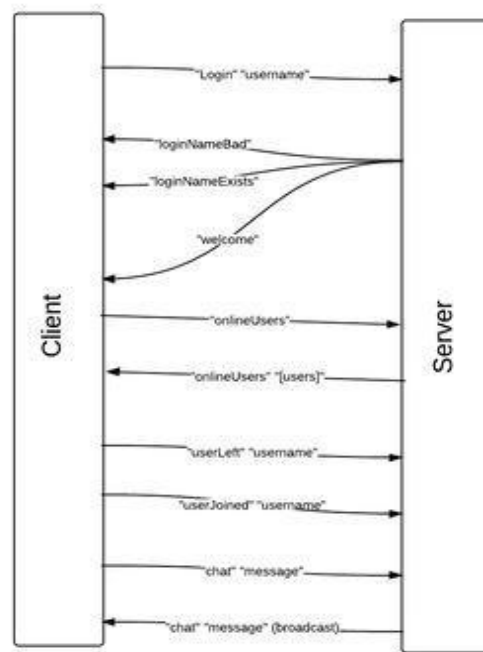


Fig 3.2 Communication model of SocketChat

Sequence Diagram

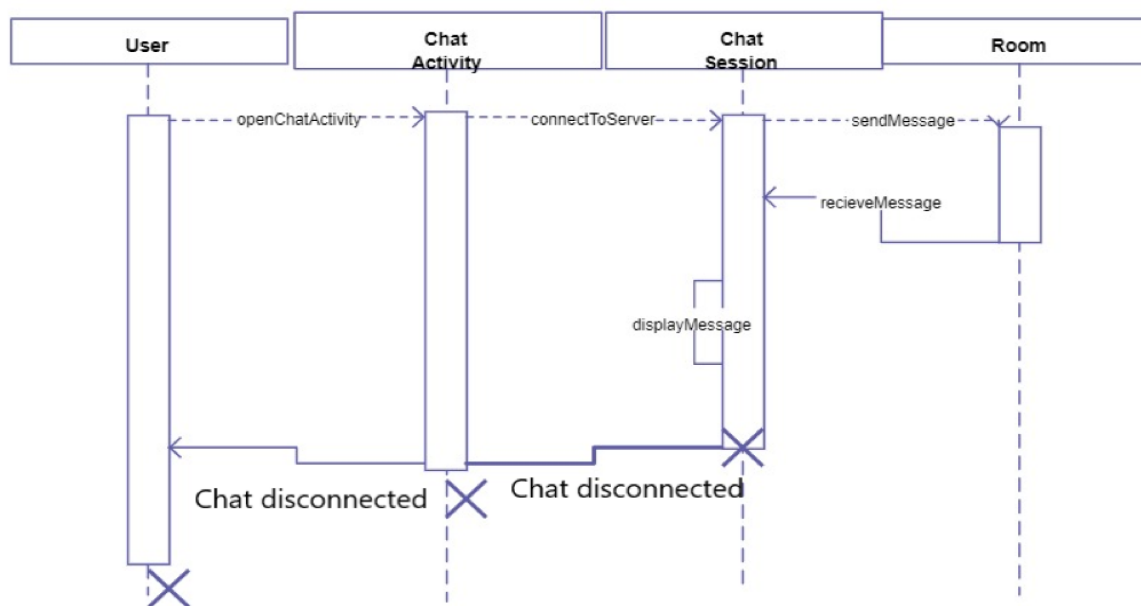


Fig 3.3 Sequence diagram of SocketChat

Fig 3.3 represents the sequence diagram of socketchat and Fig 3.4 represents the use case diagram. It describes how users can use the socket chat application and the timespan of each object that is involved with this application.

Use Case Diagram

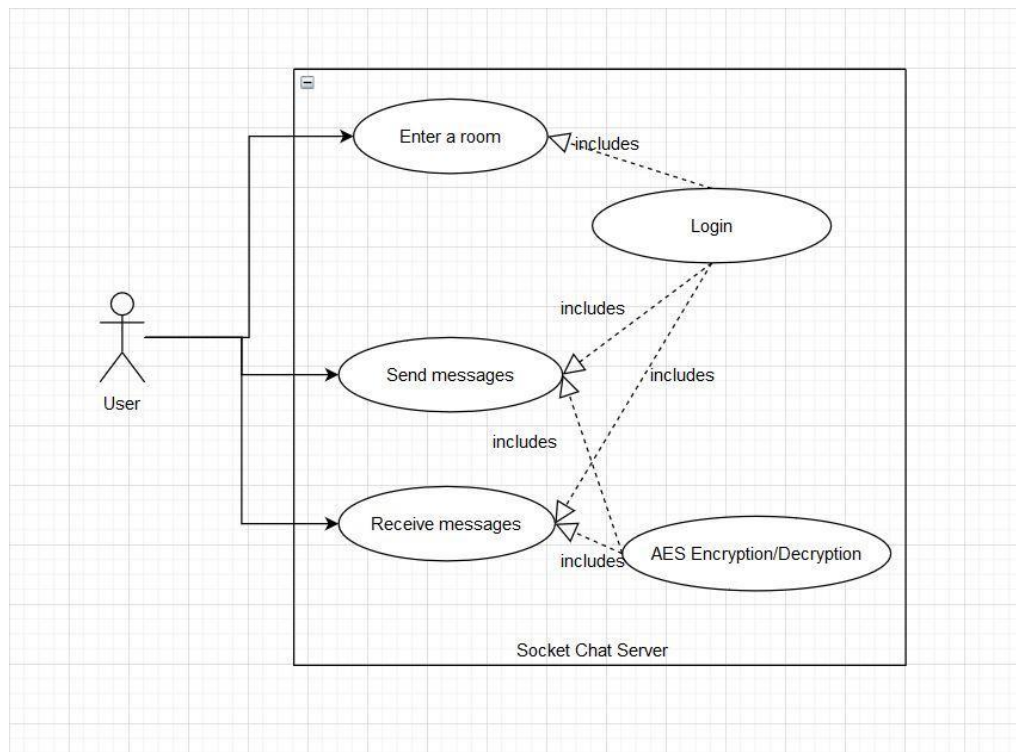


Fig 3.4 Use Case Diagram of SocketChat

Data Flow Diagram

Fig 3.5 is the 0 level DFD that explains high level data flow and functionalities of socketchat and Fig 3.6 describes many other functionalities and the data flow of the application in detail.

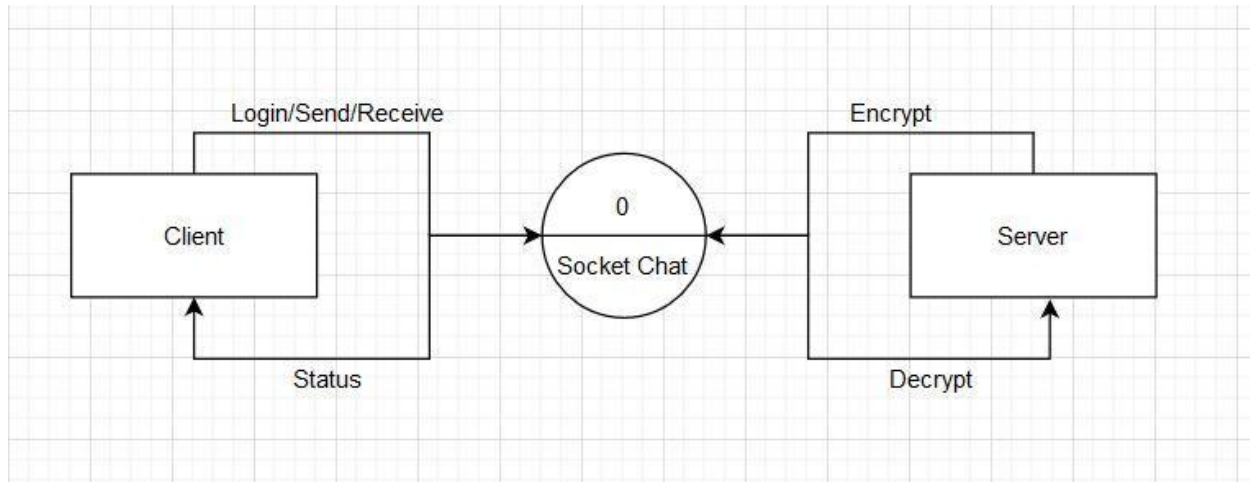


Fig 3.5 Level 0 DFD of SocketChat

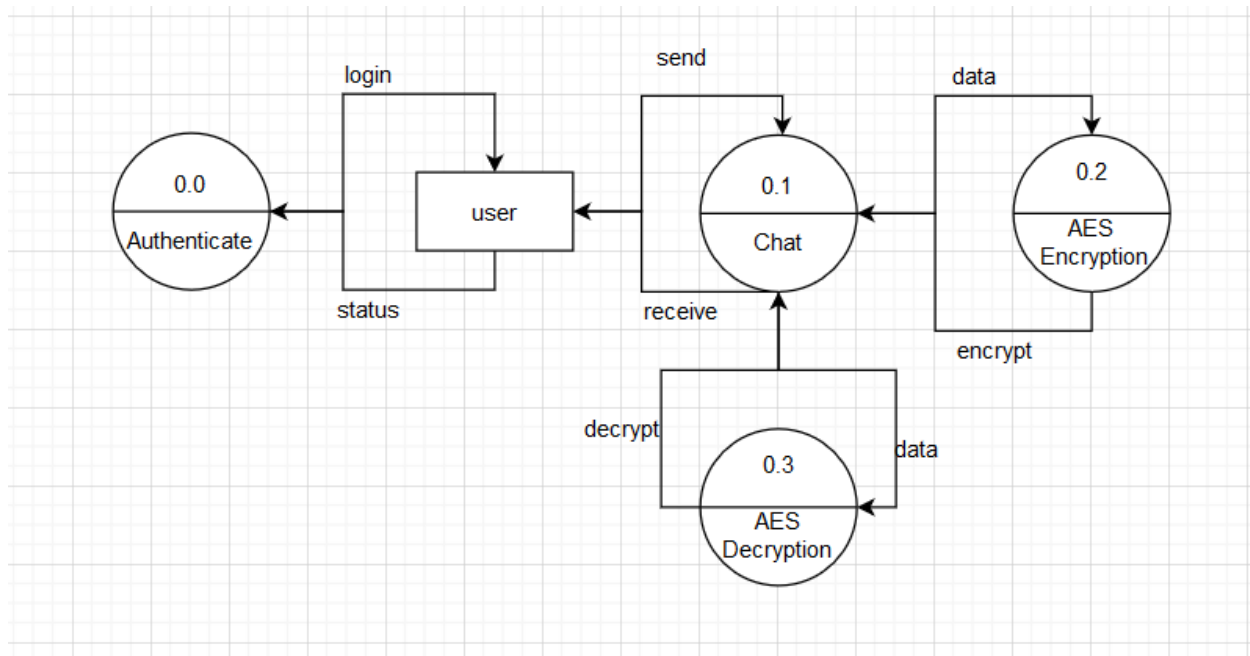


Fig 3.6 Level 1 DFD of SocketChat

CHAPTER-4

Implementation & Testing

CHAPTER 4

Implementation & Testing

Implementation:

Creating and setting up Socket

Socket library from python is used to set up the client and server. The process begins by creating the socket and to make this socket visible we use “server_socket.bind((HOST, PORT))”. The HOST specifies that the socket is reachable by any address the machine happens to have. The PORT is a number that denotes “well known” services (HTTP, SNMP etc). The port number used here is 7777. Finally the argument to “server_socket.listen(10)” tells the socket library that it requires to queue up as many as 10 connection requests before refusing outside connections.

Each “ClientSocket” is created in response to some other “Client” socket doing a connect() to the host and port that the app is bound to. As soon as the clientsocket has been created, it redirects back to listening for more connections. Now any multiple “Clients” who join the chat room are free to chat it up - they are using some dynamically allocated port which will be recycled when the conversation ends.

AES Encryption and Decryption

The chats sent through sockets are not secure hence these chats need to be encrypted. Pycrypto library is used to implement the AES encryption and decryption. AES has several rounds where each round is given a key through “key expansion”. In AES encryption and decryption each round comprises four sub-processes, in encryption round Byte Substitution, Shiftrows, MixColumns, Addroundkey takes place respectively with the given key whereas in decryption the inversion of the sub-processes is done in a reverse order.

Encryption pseudo code

```
def encrypt(secret,data):  
    BLOCK_SIZE = 32  
    PADDING = '{'  
    pad = lambda s: s + (BLOCK_SIZE - len(s) % BLOCK_SIZE) * PADDING  
    EncodeAES = lambda c, s: base64.b64encode(c.encrypt(pad(s)))  
    cipher = AES.new(secret)  
    encoded = EncodeAES(cipher, data)  
    return encoded.
```

Decryption pseudo code

```
def decrypt(secret,data):  
    BLOCK_SIZE = 32  
    PADDING = '{'  
    pad = lambda s: s + (BLOCK_SIZE - len(s) % BLOCK_SIZE) * PADDING  
    DecodeAES = lambda c, e: c.decrypt(base64.b64decode(e)).rstrip(PADDING)  
    cipher = AES.new(secret)  
    decoded = DecodeAES(cipher, data)  
    return decoded
```

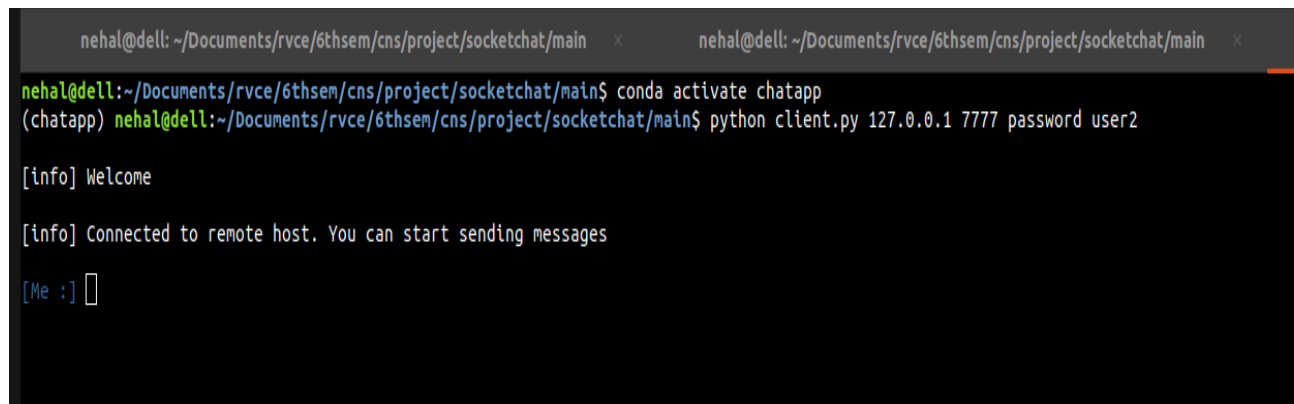
Testing:

The test plan keeps track of possible tests that will be run on the system after coding. This test plan is a document that is developed as the project is being developed. This document records tests as they come up and hence is very necessary. It's essential to identify test error prone parts of software development. The transformation of the abstract initial test plan to the final concrete test plan shows the development process of the code. When the system is in the design stage, it helps to identify the initial tests. During the detailed design or coding phase, it helps to identify the exact test cases. After coding, the test points are all identified and the entire test plan is exercised on the software. Test Plan ensures all Functional and Design Requirements are implemented as specified in the documentation such as user signup, user login etc. It provides a procedure for Unit and System Testing. It identifies the documentation process for Unit and System Testing. It identifies the test methods for Unit and System Testing. It serves as a guide to testing throughout the development. It serves as a valuable record of what testing was done.

LOGIN

Table 4.1 Login

Sl No	Action	Input	Expected Output	Actual Output	Result	Comments
1.	Login	127.0.0.1 7777 password user1	Welcome, Connected to the server	Welcome, Connected to the server	Pass	Login In to correct server address with correct password
2.	Login	127.0.0.1 7777 password user1	Welcome, Connected to the server	Welcome, Connected to the server	Pass	Login In to correct server address with correct password
3.	Login	127.0.12.1 7777 password user1	Unable to connect	Unable to connect	Pass	Incorrect server address
4.	Login	127.0.0.1 7779 password user1	Unable to connect	Unable to connect	Pass	Incorrect port number
5.	Login	127.0.0.1 7777 pass user	Unable to connect	Unable to connect	Pass	Incorrect Password



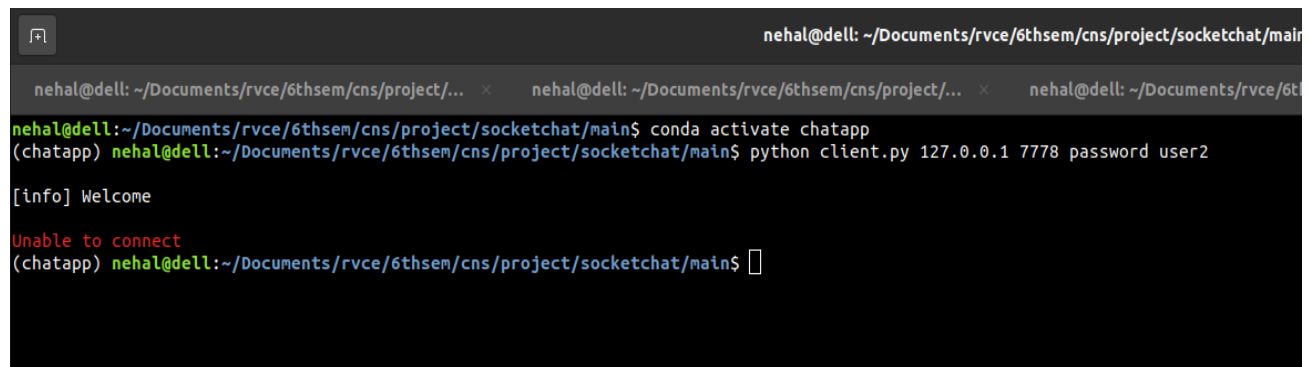
```
nehal@dell: ~/Documents/rvce/6thsem/cns/project/socketchat/main x nehal@dell: ~/Documents/rvce/6thsem/cns/project/socketchat/main x
nehal@dell:~/Documents/rvce/6thsem/cns/project/socketchat/main$ conda activate chatapp
(chatapp) nehal@dell:~/Documents/rvce/6thsem/cns/project/socketchat/main$ python client.py 127.0.0.1 7777 password user2

[info] Welcome

[info] Connected to remote host. You can start sending messages

[Me :]
```

Fig 4.1 Successful login



```
nehal@dell: ~/Documents/rvce/6thsem/cns/project/socketchat/main
nehal@dell: ~/Documents/rvce/6thsem/cns/project/... x nehal@dell: ~/Documents/rvce/6thsem/cns/project/... x nehal@dell: ~/Documents/rvce/6thsem/cns/project/... x
nehal@dell:~/Documents/rvce/6thsem/cns/project/socketchat/main$ conda activate chatapp
(chatapp) nehal@dell:~/Documents/rvce/6thsem/cns/project/socketchat/main$ python client.py 127.0.0.1 7778 password user2

[info] Welcome

Unable to connect
(chatapp) nehal@dell:~/Documents/rvce/6thsem/cns/project/socketchat/main$
```

Fig 4.2 Incorrect port number

ENCRYPTION AND DECRYPTION

Table 4.2 Encryption and Decryption

Sl No	Action	Input	Expected Output	Actual Output	Result	Comments
1.	Encryption	Hello	rN4pzMHRXt+H3yD3wjH83T3Q0kEK6tARTQWHNQVNR/o=	rN4pzMHRXt+H3yD3wjH83T3Q0kEK6tARTQWHNQVNR/o=	Pass	Raw text is encrypted
2.	Encryption	Hello world	aOB7rRqQrEWOj/fpsEi0UEem/6slAtCueg/R8nnIwQs=	aOB7rRqQrEWOj/fpsEi0UEem/6slAtCueg/R8nnIwQs=	Pass	Raw text is encrypted
3.	Decryption	rN4pzMHRXt+H3yD3wjH83T3Q0kEK6tARTQWHNQVNR/o=	Hello	Hello	Pass	Encrypted text is decrypted
4.	Decryption	aOB7rRqQrEWOj/fpsEi0UEem/6slAtCueg/R8nnIwQs=	Hello World	Hello world	Pass	Encrypted text is decrypted
5.	Encryption	Hello	rN4pzMHRXt+H3yD3wjH83T3Q0kEK6tARTQWHNQVNR/o=	rN4pzMHRXt+H3yD3wjH83T3Q0kEK6tARTQWHNQVNR/o=	Pass	Same text has same encrypted text
6.	Encryption	Hello	rN4pzMHRXt+H3yD3wjH83T3Q0kEK6tARTQWHNQVNR/o=	rN4pzMHRXt+H3yD3wjH83T3Q0kEK6tARTQWHNQVNR/o=	Pass	Same text has same encrypted text

```
nehal@dell: ~/Documents/rvce/6thsem/cns/project/socketchat/main x nehal@dell: ~/Documents/rvce/6thsem/cns/project/socketchat/main
nehal@dell:~/Documents/rvce/6thsem/cns/project/socketchat/main$ conda activate chatapp
(chatapp) nehal@dell:~/Documents/rvce/6thsem/cns/project/socketchat/main$ python client.py 127.0.0.1 7777 password user1

[info] Welcome
[info] Connected to remote host. You can start sending messages
[Me : ] [127.0.0.1:60558] entered our chatting room
[Me : ] Hello
[ user2: ] Hello world
[Me : ]
```

Fig 4.3 Decryption

```
nehal@dell: ~/Documents/rvce/6thsem/cns/project/socketchat/main x nehal@dell: ~/Documents/rvce/6thsem/cns/project/socketchat/main
nehal@dell:~/Documents/rvce/6thsem/cns/project/socketchat/main$ conda activate chatapp
(chatapp) nehal@dell:~/Documents/rvce/6thsem/cns/project/socketchat/main$ python client.py 127.0.0.1 7777 password user2

[info] Welcome
[info] Connected to remote host. You can start sending messages
[ user1: ] Hello
[Me : ] Hello world
[Me : ]
```

Fig 4.4 Decryption

```
nehal@dell: ~/Documents/rvce/6thsem/cns/project/socketchat/main x nehal@dell: ~/Documents/rvce/6thsem/cns/project/socketchat/main
nehal@dell:~/Documents/rvce/6thsem/cns/project/socketchat/main$ conda activate chatapp
(chatapp) nehal@dell:~/Documents/rvce/6thsem/cns/project/socketchat/main$ python server.py

[info] Server started
[info] Server started on port 7777
[info] user (127.0.0.1, 60556) connected
[info] user (127.0.0.1, 60558) connected
rN4pzMHRXt+H3yD3wjH83T3Q0kEK6tARTQWHNQVNR/o=
aOB7rRqQrEW0j/fpsEi0UEem/6slAtCueg/R8nnIwQs=

```

Fig 4.5 Encryption

SERVER SIDE

Table 4.3 Server Side

Sl No	Action	Input	Expected Output	Actual Output	Result	Comments
1.	Connect	User login	[info] user (127.0.0.1, 60528) connected	[info] user (127.0.0.1, 60528) connected	Pass	Keep track of users
2.	Connect	User login	[info] user (127.0.0.1, 60530) connected	[info] user (127.0.0.1, 60530) connected	Pass	Keep track of users
3.	Disconnect	User logoff	[info] user (127.0.0.1, 60530) disconnected	[info] user (127.0.0.1, 60530) disconnected	Pass	Keep track of users
4.	Disconnect	User logoff	[info] user (127.0.0.1, 60530) disconnected	[info] user (127.0.0.1, 60530) disconnected	Pass	Keep track of users
5.	Chatting	Hello	rN4pzMHRXt+H3yD3wjH83T3Q0kEK6tARTQWHNQVNR/o=	rN4pzMHRXt+H3yD3wjH83T3Q0kEK6tARTQWHNQVNR/o=	Pass	Complete Encryption
6.	Chatting	Hello world	aOB7rRqQrEWOj/fpsEi0UEem/6slAtCu eg/R8nnlwQs=	aOB7rRqQrEWOj/fpsEi0UEem/6slAtCu eg/R8nnlwQs=	Pass	Keep track of users

```

nehal@dell: ~/Documents/rvce/6thsem/cns/project/socketchat/main
nehal@dell:~/Documents/rvce/6thsem/cns/project/socketchat/main$ conda activate chatapp
(chatapp) nehal@dell:~/Documents/rvce/6thsem/cns/project/socketchat/main$ python server.py

[info] Server started
[info] Server started on port 7777
[info] user (127.0.0.1, 60556) connected
[info] user (127.0.0.1, 60558) connected
rN4pzMhRxt+H3yD3wjH83T3Q0kEK6tARTQWHNQVNR/o=
aOB7rRqRqREW0j/fpsEi0UEem/6s1AtCueg/R8nnIwQs=
[info] user (127.0.0.1, 60558) disconnected

```

Fig 4.6 New user joins, Encrypted texts, User disconnects

CLIENT SIDE

Table 4.4 Client Side

Sl No	Action	Input	Expected Output	Actual Output	Result	Comments
1.	New user joins	-	[Me :] [ip_address :port_number] entered our chatting room	[Me :] [ip_address :port_number] entered our chatting room	Pass	Other users can come to know
2.	User leaves	-	[Me :] [info] user (ip_address	[Me :] [info] user (ip_address	Pass	Other users can come to know

			'port_number) is offline	'port_number) is offline		
3.	Chatting	rN4pzMHR Xt+H3yD3 wjH83T3Q 0kEK6tAR TQWHNQ VNR/o=	Hello	Hello	Pass	Encrypted texts converted to original message
4.	Chatting	aOB7rRqQ rEWOj/fps Ei0UEem/6 slAtCueg/R 8nnlwQs=	Hello world	Hello world	Pass	Encrypted texts converted to original message
5.	Server down	-	[info] Disconnect ed from chat server	[info] Disconnect ed from chat server	Pass	Server down

```

nehal@dell: ~/Documents/rvce/6thsem/cns/project/socketchat/
nehal@dell:~/Documents/rvce/6thsem/cns/project/socketchat/main$ conda activate chatapp
(chatapp) nehal@dell:~/Documents/rvce/6thsem/cns/project/socketchat/main$ python client.py 127.0.0.1 7777 password user1
[info] Welcome
[info] Connected to remote host. You can start sending messages
[Me :] [127.0.0.1:60558] entered our chatting room
[Me :] Hello
[ user2: ] Hello world
[Me :] [info] user (127.0.0.1, 60558) is offline
[Me :] [info]
Disconnected from chat server
(chatapp) nehal@dell:~/Documents/rvce/6thsem/cns/project/socketchat/main$

```

Fig 4.7 New User joins, User disconnects, Server down

CHAPTER-5

CONCLUSION AND FUTURE SCOPE

CHAPTER 5

Conclusion and Future scope

Conclusion:

The main objective of the project is to develop a Secure Chat Application. A wide range of literature review was done in order to achieve all the tasks, which gave information about some of the products that are existing in the market. A detailed research in that path was made to cover the loopholes that existing systems are facing and to eradicate them in our application. In the process of research, understanding of latest technologies and different algorithms was achieved.

Analysis of various encryption algorithms (DES, AES), Integrity algorithms (MD5, SHA), key-exchange algorithms, authentication was done and implemented those functionalities in the application.

The portability of the application has been achieved by using some of the libraries of python and the use of python virtual environment makes it portable to run on any system.

As a result, the product has been successfully developed in terms of extendability, portability, and maintainability and tested in order to meet all requirements that are Authentication, Integrity, Confidentiality which are specified as the four basic concepts for the secure communication over a network.

Future Scope:

- Extending this application by providing Authorization service.
- Creating databases and maintaining users.
- Increasing the effectiveness of the application by providing Voice Chat.
- Extending it to Web Support.

References

1. “A cryptographic approach for secure client - server chat application using public key infrastructure (PKI)” by Işıl Karabey, Gamze Akman, Department of Computer Engineering, Hacettepe University, Ankara, Turkey , IEEE Xplore (2021).
2. “Ncryptr: a symmetric and asymmetric encryption application” by Gonçalo Ribeiro; Marin Grabovschi; Mário Antunes; Luís Frazão ,IEEE Xplore (2021).
3. “End-to-End Privacy Protection for Facebook Mobile Chat based on AES with Multi-Layered MD5” by Wibisono Sukmo Wardhono, Nurizal Dwi Priandani, Mahardeka Tri Ananta, Komang Candra Brata, Herman Tolle Brawijaya University, Malang, Indonesia, Researchgate.net (2018).
4. <https://pypi.org/project/pycrypto/>
5. <https://python-socketio.readthedocs.io/>
6. <https://docs.python.org/3/library/socket.html>
7. https://www.tutorialspoint.com/unix_sockets/what_is_socket.htm
8. <https://docs.python.org/3/tutorial/venv.html>
9. https://en.wikipedia.org/wiki/Advanced_Encryption_Standard