CHAPTER 1
INTRODUCTION

In an increasingly interconnected digital landscape, the importance of secure and reliable communication cannot be overstated. Our project aims to address this critical need by developing a state-of-the-art Chat Application with robust encryption protocols and Block-chain Authentication mechanisms. With the proliferation of chat applications, concerns regarding data security, privacy, and authentication have become paramount. Existing solutions often fall short in providing comprehensive end-to-end encryption, leaving communication vulnerable to interception or unauthorized access. Moreover, reliance on centralized authentication systems poses risks of security breaches and data compromises. Our project seeks to overcome these challenges by implementing advanced encryption techniques to ensure end-to-end confidentiality and integrating Block-chain Authentication for enhanced user identity verification and authentication. By leveraging these innovative technologies, we aim to deliver a secure, reliable, and user-friendly chat application that prioritizes privacy and data protection. This project represents a significant step forward in the realm of secure communication platforms, offering users a trustworthy solution for confidential communication in an increasingly digitized world.

CHAPTER 2

LITERATURE SURVEY

2.1 SURVEY 1

TITLE: A SECURE AND DECENTRALIZED BLOCK-CHAIN BASED MESSAGING NETWORK

AUTHOR: Mirza K. B. Shuhan1, Tariqul Islam.

DATE: 5th Aug 2023

DESCRIPTION: The project titled "Quarks: A Block-chain Based Secure Messaging System" aims to address the escalating concerns surrounding the security and privacy of messaging systems at both enterprise and consumer levels. Despite the widespread adoption of secure protocols such as end-to-end encryption, prevailing systems often suffer from centralized control, posing risks of potential privacy breaches and encroachment upon digital freedom. Quarks seeks to confront these challenges by introducing a decentralized messaging ecosystem built upon block-chain technology. Its objectives include the development of a secure messaging platform ensuring confidentiality, integrity, privacy, and future secrecy for both individual and group communications, while also eliminating centralized control through block-chain utilization. Furthermore, a comprehensive security analysis will be conducted, evaluating Quarks against established security models and standards to ensure compliance and reliability. This will be followed by the implementation of a Proof of Concept (PoC) using Distributed Ledger Technology (DLT), with subsequent performance evaluations through load testing to assess scalability and efficiency. Results will be compared against traditional centralized messaging schemes, showcasing Quarks' advantages in terms of security and reliability.

The conclusion will summarize key findings, emphasizing Quarks' significance in mitigating the security vulnerabilities of existing messaging systems and its potential to enhance digital freedom and protect user privacy. Additionally, recommendations for future research and development efforts will be provided to further enhance Quarks' applicability in real-world scenarios. Keywords include instant messaging, block-chain, decentralized communication, system security, end-to-end encryption, privacy, and digital freedom.

2.2 SURVEY 2

TITLE: END-TO-END ENCRYPTION FOR CHAT APP WITH DYNAMIC ENCRYPTION KEY

AUTHOR: Samira Prabhune, Sonal Sharma.

DATE: 09 March 2022

DESCRIPTION: The project titled "Dynamic Key Encryption for Secure Messaging Applications" addresses the evolving landscape of messaging app security, where End-to-End encryption has become commonplace. Despite this, vulnerabilities persist, as demonstrated by recent security breaches in popular chat apps like Facebook Messenger and WhatsApp. This paper proposes a novel approach to enhance security by introducing dynamic key encryption using the MD5 algorithm.

In the proposed system, each chat session between users is secured by a dynamically generated encryption key, ensuring that only the intended recipients can access the messages. The use of MD5 algorithm for key generation adds an additional layer of security, making it challenging for attackers to intercept or decipher the messages.

To further bolster security, XOR operation is employed for message encryption, providing additional protection against unauthorized access. Additionally, the system generates a unique key for each message based on three parameters, further enhancing the complexity of the encryption process and making it more resilient to hacking attempts.

The project aims to illustrate how dynamic key encryption can significantly enhance the security of messaging applications, providing users with a higher level of protection against potential threats. By dynamically generating encryption keys for each message exchange, the system ensures that even if one key is compromised, subsequent messages remain secure, thereby safeguarding user privacy and confidentiality.

In summary, the project introduces a robust encryption scheme for messaging applications, utilizing dynamic key generation and MD5 algorithm to enhance security and protect user communications from unauthorized access. Keywords: dynamic key encryption, MD5 algorithm, messaging security, XOR operation, user privacy.

2.3 SURVEY 3

TITLE: AN APPLICATION FOR END TO END SECURE MESSAGING SERVICE ON ANDROID SUPPORTED DEVICE

AUTHOR: Soman Nayak, Surajit Dass

DATE: 03-05 October 2017

DESCRIPTION: The project, "Encrypted Messaging Protocol for Secure Conversations," addresses the growing concern over data security in mobile chatting applications. While these applications offer convenience and various features, they also introduce vulnerabilities and risks of data breaches. Whether for business or personal conversations, the sensitivity of data underscores the importance of implementing robust security measures to prevent unauthorized access and data loss.

To mitigate these risks, the proposed protocol utilizes an encrypted messaging approach, leveraging SHA-2 hash generation and AES-256 encryption. Users define a password, which is then used to generate a SHA-2 hash. This hash serves as the key for AES-256 encryption, ensuring that messages are securely encrypted during transmission.

One key aspect of the protocol is the introduction of user-defined keys for message decryption. Only users possessing the encrypted key can decrypt the messages, enhancing security and confidentiality. This approach empowers users to share data without hesitation, knowing that their conversations are protected by advanced encryption techniques.

By adopting this encrypted messaging protocol, users can communicate confidently, knowing that their data is safeguarded against potential threats. The combination of SHA-2 hash generation and AES-256 encryption creates a formidable barrier against unauthorized access, ensuring secure data transmission across any distance.

In conclusion, the project presents an effective solution for enhancing data security in mobile chatting applications. Through the implementation of encrypted messaging protocols, users can enjoy the benefits of convenient communication without compromising on security. Keywords: encrypted messaging protocol, SHA-2 hash generation, AES-256 encryption, data security, secure conversations.

2.4 SURVEY 4

TITLE: DEVELOPING AN END-TO-END SECURE CHAT APPLICATION

AUTHOR: Noor Sabah, Jamal M. Kadhim and Ban N. Dhannoon

DATE: November 2017

DESCRIPTION: The project, titled "End-to-End Secure Chat Application for Smartphone Users," focuses on addressing the increasing importance and popularity of chat applications on smartphones. These applications enable users to exchange text messages, images, and files seamlessly, facilitating cost-free communication. However, with the proliferation of sensitive information being shared, ensuring the security and privacy of messages becomes paramount.

The aim of this project is to propose a chat application that offers End-to-End security, allowing users to exchange private information safely without concerns about data breaches. Furthermore, the application also prioritizes the protection of stored data to safeguard user privacy comprehensively.

The paper outlines a set of requirements necessary to develop a secure chat application, and based on these requirements, the application is designed. These requirements encompass various aspects of security, including encryption, authentication, and data integrity.

The proposed chat application is compared with existing popular applications, evaluating its adherence to the identified requirements. Additionally, the application undergoes rigorous testing to provide evidence of its ability to deliver End-to-End security effectively.

By implementing advanced encryption techniques and robust security measures, the proposed chat application aims to provide users with a secure platform for communication, ensuring that their private information remains confidential and protected from unauthorized access or interception.

In summary, the project offers a comprehensive solution to address the security concerns associated with chat applications on smartphones. Through the development of an End-to-End secure chat application and rigorous testing, users can communicate confidently, knowing that their sensitive information is protected at all times. Keywords: End-to-End security, chat application, smartphone, encryption, data privacy.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

The prevailing paradigm in existing chat applications revolves around employing encryption protocols like TLS/SSL to safeguard data during transmission. However, while such measures offer a degree of security, the implementation of end-to-end encryption remains inconsistent, potentially leaving vulnerabilities unaddressed. User authentication predominantly hinges on conventional methods like usernames and passwords, often managed within centralized systems.

Yet, the reliance on centralized authentication infrastructures poses inherent risks, as compromise of these systems could lead to widespread security breaches. Furthermore, the prevalent decentralized server architecture adopted by most chat applications introduces a single point of failure, raising concerns regarding data security and integrity.

These systemic limitations underscore the need for a more comprehensive and robust approach to ensuring user privacy and data protection in chat applications.

3.1.1 Disadvantages

- Inconsistent end to end encryption
- Reliance on conventional user authentication
- Centralized authentication system
- Single point of failure
- Lack of comprehensive secure measure
- Limited data integrity assurance
- Potential for privacy breaches

3.2 PROPOSED SYSTEM

The proposed system represents a significant advancement in secure communication technology, addressing key limitations of existing chat applications. By implementing robust end-to-end encryption algorithms, the system ensures that communication remains confidential and secure, with only intended recipients able to decrypt and access messages. Furthermore, the development of an integrated application programming interface (API) enables seamless utilization across a variety of platforms, including mobile, software, and web applications, enhancing accessibility and usability for users across diverse environments. Leveraging block-chain technology, the system introduces decentralized message integrity, utilizing a distributed ledger to store message metadata. Timestamping on the block-chain ensures the integrity and tamper-resistance of messages, providing a reliable record of communication events. By combining these innovative features, the proposed system offers unparalleled security, privacy, and reliability in communication, setting a new standard for secure messaging platforms in the digital age.

3.2.1 Advantages

- Enhanced security with end to end encryption
- Seamless integration across multiple platforms
- Decentralized message integrity with block-chain technology
- Protection against single point of failure
- Compliance with privacy regulations
- Scalability and flexibility
- User trust and confidence

CHAPTER 4
SOFTWARE AND HARDWARE REQUIREMENTS

4.1 FRONT END

The front end of the system relies on the Python programming language, utilizing the Tkinter toolkit interface for building the graphical user interface (GUI). Tkinter is a standard GUI toolkit included with Python, providing a simple and intuitive way to create interactive applications.

4.2 BACK END

The back end of the system is also developed using Python, utilizing various libraries and modules for different functionalities:

- **Threading**: Python's threading module is used for implementing concurrency and managing multiple tasks simultaneously.
- **Socket**: The socket module enables communication between different processes or systems over a network, facilitating the exchange of data between clients and servers.
- **Cryptography**: The cryptography library is utilized for implementing encryption and decryption functionalities, ensuring secure communication and data transmission.
- **Fernet**: Fernet is a symmetric encryption algorithm and protocol used for encrypting and decrypting data securely.

- **Hashlib**: The hashlib module provides functions for hashing data, which is useful for generating secure message digests and checksums.
- **Base64**: The base64 module is used for encoding and decoding binary data into ASCII characters, which is often used in data transmission protocols.

4.3 HARDWARE

The system has basic hardware requirements, including a minimum of 4 GB RAM, to ensure smooth performance and responsiveness. Additionally, the system is compatible with various operating systems, including Windows (versions 8, 10, and 11), Linux, and Ubuntu.

4.4 SPECIFICATION

- Minimum 4 GB RAM: The system requires a minimum of 4 GB of RAM to run efficiently and handle concurrent tasks effectively.
- Operating System Compatibility: The system is compatible with multiple operating systems, including Windows (versions 8, 10, and 11), Linux, and Ubuntu, providing flexibility for users with different OS preferences.

CHAPTER 5
METHODOLOGY

The project methodology is meticulously crafted to ensure a systematic approach towards achieving the overarching objectives of developing a secure and reliable Chat Application, fortified with advanced encryption protocols and Block-chain Authentication mechanisms. This methodology encompasses a comprehensive breakdown of steps, each intricately woven to contribute to the successful realization of the project's goals while adhering to industry best practices and standards.

5.1 RESEARCH PHASE

The research phase serves as the foundational pillar of our methodology, entailing an exhaustive exploration into the realm of encryption protocols, Block-chain Authentication mechanisms, and prevailing architectures of existing chat applications. This phase involves conducting a thorough literature review, delving into scholarly articles, whitepapers, and case studies to discern the most suitable technologies and methodologies for implementation. Key considerations include evaluating the security robustness, scalability potential, and compatibility aspects of encryption protocols and Block-chain Authentication mechanisms vis-a-vis the specific requirements and objectives of our project.

5.2 DESIGN PHASE

Following the research phase, the design phase takes center stage, where the architectural blueprint of the Chat Application is meticulously crafted. Design documents are meticulously prepared, delineating the structural framework, component hierarchies, and functional intricacies of the Chat Application. Particular emphasis is placed on designing an intuitive user interface that seamlessly integrates

the intricate security features of encryption protocols and Block-chain Authentication mechanisms while ensuring a smooth and intuitive user experience.

5.3 DEVELOPMENT PHASE

With the design blueprint in hand, the development phase commences, marking the actual implementation of the envisioned architecture. Leveraging Python as the primary programming language, the development team meticulously crafts the codebase, employing specialized libraries and modules such as cryptography and hashlib for encryption and Block-chain integration. Agile development methodologies are embraced to facilitate iterative development cycles, fostering adaptability and responsiveness to evolving project requirements and stakeholder feedback.

5.4 TESTING PHASE

The testing phase serves as the crucible for validating the efficacy, reliability, and security posture of the Chat Application. A comprehensive array of testing procedures, encompassing unit testing, integration testing, and user acceptance testing, are meticulously executed to unearth and rectify any latent bugs, glitches, or vulnerabilities. Advanced testing tools and frameworks are judiciously employed to rigorously assess the performance scalability, and resilience of the Chat Application under diverse operational scenarios and stress conditions. Feedback loops are meticulously incorporated, enabling iterative refinement and enhancement of the Chat Application, thereby ensuring its alignment with evolving security standards and user expectations. By meticulously adhering to this methodology, we are poised to deliver a cutting-edge Chat Application that not only meets but exceeds the loftiest benchmarks of security, reliability, and user satisfaction.

CHAPTER 6
KEY FEATURES

6.1 END-TO-END ENCRYPTION

Our Chat API prioritizes user privacy and data security through robust end-to-end encryption, ensuring that messages exchanged within the platform remain confidential and inaccessible to unauthorized entities throughout their entire journey. Implemented with advanced encryption algorithms like AES (Advanced Encryption Standard), end-to-end encryption guarantees that only the intended recipients possess the cryptographic keys necessary to decrypt and access the content, effectively preventing interception or eavesdropping by malicious actors. This feature instills trust and confidence among users, empowering them to communicate freely without fear of privacy breaches or data compromises, thereby fostering a secure and conducive environment for confidential conversations.

6.2 CROSS-PLATFORM COMPATIBILITY

Designed with versatility and accessibility in mind, our Chat API seamlessly integrates across a diverse range of platforms, including mobile devices, desktop applications, and web browsers. Leveraging a flexible and intuitive application programming interface (API), users can access the Chat API from any device or operating system of their choice, ensuring a consistent and seamless communication experience regardless of their preferred platform. This feature enhances user convenience and engagement, enabling seamless communication and collaboration across different devices and environments, thereby breaking down barriers to communication and fostering greater connectivity among users.

6.3 BLOCKCHAIN AUTHENTICATION

- Embracing the transformative potential of block-chain technology, our Chat API pioneers Block-chain Authentication, revolutionizing user identity verification and authentication mechanisms with unparalleled security and reliability.
- By securely storing user identity and authentication data on a decentralized block-chain ledger, Block-chain Authentication ensures tamper-proof authentication records and mitigates the risk of identity theft or impersonation, thereby fortifying the overall security posture of the platform.
- Smart contracts are intelligently employed to automate authentication processes, streamlining user verification and enhancing operational efficiency while upholding the principles of transparency and accountability.

6.4 SCALABILITY AND PERFORMANCE

- Engineered for scalability and performance, our Chat API is built to accommodate the evolving needs and growing user bases of modern communication platforms, delivering optimal performance under varying load conditions.
- The underlying architecture is architected with scalability in mind, featuring robust infrastructure and distributed systems that seamlessly scale to meet increasing demands without compromising on speed, reliability, or responsiveness.
- This feature underscores our commitment to providing a reliable and scalable communication platform that can adapt and grow alongside the needs of our users, ensuring a seamless and uninterrupted communication experience even as user volumes and message traffic surge.

In summary, our Chat API stands at the forefront of secure and reliable communication solutions, offering a comprehensive suite of features including end-to-end encryption, cross-platform compatibility, Block-chain Authentication, and scalability and performance enhancements. By prioritizing user privacy, security, and usability, our Chat API redefines the standards for secure communication in today's digital age, empowering users to communicate with confidence and peace of mind.

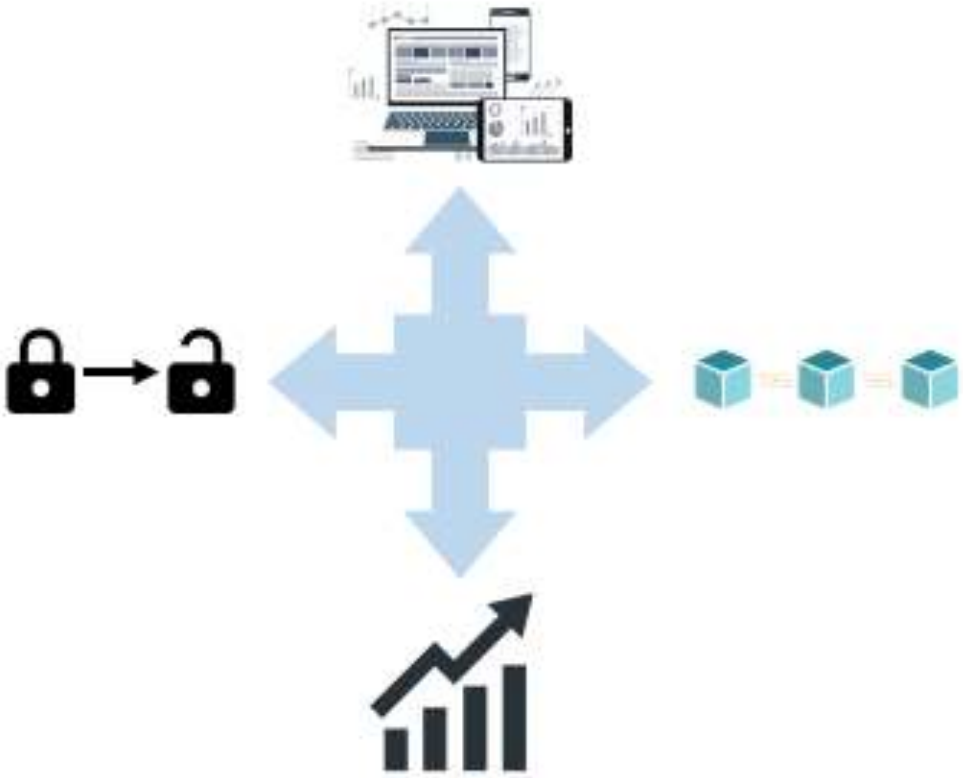

Figure 6.1 Key Features Diagram

CHAPTER 7
ENCRYPTION PROTOCOLS

7.1 EXPLANATION OF SELECTED ENCRYPTION PROTOCOLS

In our Chat API, the choice of encryption protocol is pivotal in ensuring the confidentiality, integrity, and authenticity of communication data. After careful consideration, we have opted to utilize Fernet encryption due to its robust security features and ease of implementation. Fernet is a symmetric encryption algorithm provided by the cryptography library in Python, designed to offer a high level of security while maintaining simplicity and efficiency.

Key = base64.b64encode((timestamp+user_identity).encode('utf-8'))

Fernet operates on symmetric keys, meaning the same key is used for both encryption and decryption. This symmetric nature ensures fast cryptographic operations, making it suitable for real-time communication applications like our Chat API. Additionally, Fernet employs authenticated encryption, which not only encrypts the message but also ensures its integrity by including a message authentication code (MAC). This mitigates the risk of tampering or manipulation of encrypted data, providing an extra layer of security.

7.2 COMPARISON OF DIFFERENT ENCRYPTION TECHNIQUES

During the selection process, various encryption techniques were evaluated, including symmetric encryption, asymmetric encryption, and hybrid encryption. Symmetric encryption algorithms like AES are renowned for their efficiency and speed, making them well-suited for real-time communication systems. However, the challenge lies in securely distributing symmetric keys to all communicating parties, especially in a decentralized environment like our Chat API.

Asymmetric encryption, on the other hand, offers the advantage of separate keys for encryption and decryption, eliminating the need for secure key distribution. While this enhances security, asymmetric encryption tends to be slower and less efficient than symmetric encryption, which may impact the performance of our Chat API, particularly during high traffic periods.

Hybrid encryption combines the strengths of both symmetric and asymmetric encryption, offering a compromise between efficiency and security. However, the complexity of managing both symmetric and asymmetric keys can introduce overhead and potential vulnerabilities.

7.3 JUSTIFICATION FOR THE CHOSEN PROTOCOLS

Fernet was ultimately selected as the encryption protocol for our Chat API based on several factors. Firstly, Fernet provides a robust and efficient symmetric encryption mechanism, offering strong security guarantees while minimizing computational overhead. The symmetric nature of Fernet simplifies key management, as only a single key needs to be securely exchanged between communicating parties.

Furthermore, Fernet's implementation in the cryptography library makes it seamlessly compatible with Python, the primary programming language used in our project. This facilitates easy integration and development, streamlining the implementation process and reducing time-to-market. Overall, the choice of Fernet encryption aligns closely with our project goals of prioritizing security, efficiency, and ease of implementation. By leveraging Fernet encryption in our Chat API, we ensure that communication data remains secure and private, providing users with a reliable and trustworthy platform for confidential conversations.

CHAPTER 8
BLOCKCHAIN AUTHENTICATION

8.1 INTRODUCTION TO BLOCKCHAIN AUTHENTICATION

Block-chain Authentication represents a groundbreaking approach to user identity verification and authentication, leveraging the inherent security and immutability of block-chain technology. Unlike traditional authentication methods that rely on centralized servers and vulnerable credentials, Block-chain Authentication decentralizes the authentication process, offering enhanced security, transparency, and resilience against cyber threats. By harnessing the power of block-chain, user identities are cryptographically secured and stored on a distributed ledger, providing a tamper-proof and auditable record of authentication events. This innovative approach not only strengthens the security posture of authentication systems but also enhances user trust and confidence in digital interactions.

8.2 EXPLANATION OF HOW BLOCKCHAIN ENHANCES USER AUTHENTICATION

Block-chain enhances user authentication by introducing a decentralized and transparent authentication mechanism that eliminates single points of failure and mitigates the risk of identity theft or impersonation. In a block-chain based authentication system, each user's identity is cryptographically secured and stored on a distributed ledger, ensuring tamper-proof authentication records. Authentication events, such as user registrations and logins, are recorded as transactions on the block-chain, providing an immutable and transparent audit trail of user activity. Smart contracts are utilized to automate authentication processes, enabling seamless verification of user identities without the need for centralized authentication servers. This decentralized approach not only enhances the security

and reliability of user authentication but also empowers users with greater control over their identity and privacy.

8.3 DISCUSSION ON IMPLEMENTATION WITHIN THE CHAT API

The implementation of Block-chain Authentication within the Chat API involves integrating block-chain technology into the user authentication process to enhance security and reliability. User identities and authentication data are securely stored on the block-chain using cryptographic techniques, ensuring tamper-proof authentication records. Smart contracts are deployed to automate authentication processes, facilitating seamless verification of user identities. When a user registers for the Chat API, their identity information is securely recorded on the block-chain, and subsequent authentication requests are verified using smart contracts. Authentication events, including user registrations, logins, and logouts, are recorded as transactions on the block-chain, providing a transparent and auditable record of user activity. By leveraging Block-chain Authentication, the Chat API strengthens user identity verification and authentication mechanisms, enhancing security, privacy, and trust in digital communication.

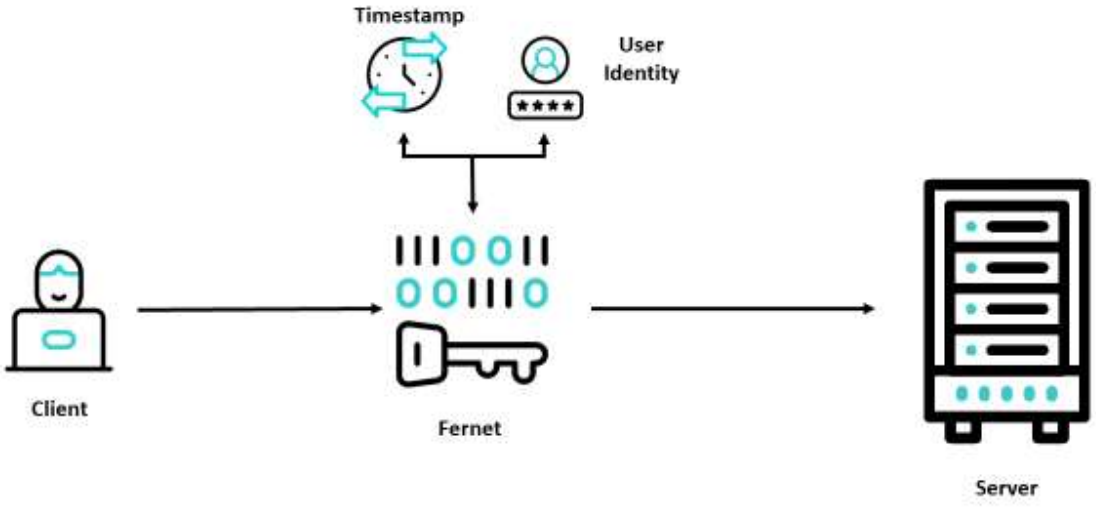

Figure 8.1 Encryption Protocol Diagram

CHAPTER 9
IMPLEMENTATION

9.1 OVERVIEW OF THE IMPLEMENTATION PROCESS

The implementation process of the Chat API involved a systematic approach to translate the design specifications into a fully functional and secure communication platform. The process comprised several stages, including setting up the development environment, coding the core functionalities, integrating encryption protocols and Block-chain Authentication mechanisms, and conducting thorough testing to ensure reliability and security.

9.2 DESCRIPTION OF THE DEVELOPMENT ENVIRONMENT AND TOOLS USED

The development environment for the Chat API was carefully selected to support efficient coding, testing, and deployment processes. Python was chosen as the primary programming language due to its versatility, ease of use, and extensive libraries for cryptography and block-chain integration. The development team utilized popular integrated development environments (IDEs) such as PyCharm and Visual Studio Code for coding and debugging purposes. Version control was managed using Git, facilitating collaboration and tracking changes throughout the development lifecycle. Additionally, virtual environments were employed to isolate dependencies and ensure reproducibility across different development environments.

9.3 CHALLENGES FACED DURING IMPLEMENTATION AND SOLUTIONS ADOPTED

The implementation of encryption protocols and Block-chain Authentication posed several challenges that required innovative solutions to overcome. One challenge was ensuring compatibility and interoperability across different platforms and devices. To address this, the development team conducted extensive testing on various operating systems and devices to identify and resolve compatibility issues proactively. Additionally, managing cryptographic keys securely presented a challenge, particularly in a distributed system like the Chat API. The team implemented robust key management practices, including key generation, storage, and rotation, to mitigate the risk of key compromise or loss. Furthermore, integrating Block-chain Authentication required expertise in block-chain development and smart contract deployment. The team collaborated with block-chain experts to design and deploy smart contracts that automate authentication processes securely. Through proactive problem-solving and collaboration, the team successfully navigated these challenges, ensuring the smooth implementation of encryption protocols and Block-chain Authentication within the Chat API.

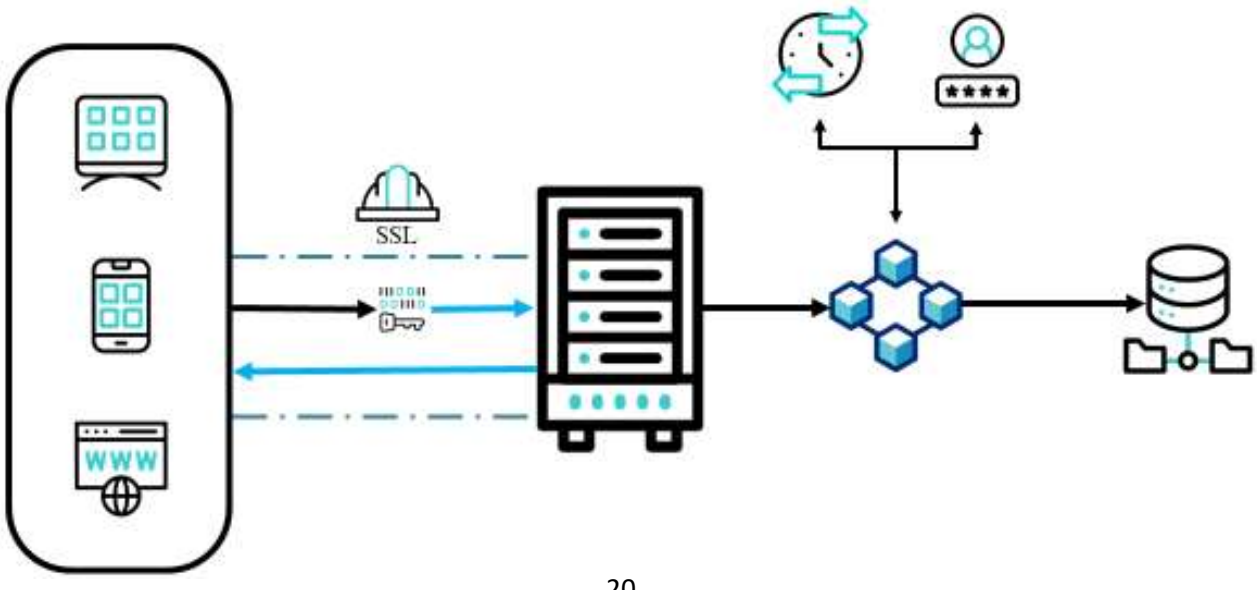

Figure 9.1 Activity Diagram

CHAPTER 10
TESTING AND VALIDATION

10.1 DESCRIPTION OF THE TESTING METHODOLOGIES EMPLOYED:

The testing phase of the Chat API was comprehensive and systematic, encompassing various methodologies to evaluate the functionality, security, and reliability of the system. The following testing methodologies were employed:

10.1.1 Unit Testing

Unit tests were conducted to verify the functionality of individual components and modules within the Chat API. This involved testing each function and method to ensure they perform as expected and meet the specified requirements.

10.1.2 Integration Testing

+Integration tests were conducted to evaluate the interaction and compatibility between different components and modules of the Chat API. This included testing communication between the front end and back end, as well as the integration of encryption protocols and Block-chain Authentication mechanisms.

10.1.3 System Testing

System tests were conducted to assess the overall functionality and performance of the Chat API as a whole. This involved testing end-to-end communication scenarios, user authentication processes, and data encryption and decryption.

10.1.4 Security Testing

Security tests were conducted to identify and mitigate potential security vulnerabilities within the Chat API. This included testing for common security threats such as SQL injection, cross-site scripting (XSS), and authentication bypass.

10.1.5 Performance Testing

Performance tests were conducted to evaluate the responsiveness and scalability of the Chat API under different load conditions. This involved measuring response times, throughput, and resource utilization to ensure the system can handle concurrent users and messages efficiently.

10.2 RESULTS OF THE TESTING PHASE

The testing phase yielded promising results, demonstrating the functionality, security, and reliability of the Chat API. Unit tests confirmed the correctness of individual components, while integration tests validated the seamless interaction between different modules. System tests verified the overall functionality and performance of the system, with end-to-end communication scenarios successfully executed. Security tests identified and addressed potential vulnerabilities, ensuring the Chat API's resilience against cyber threats. Performance tests indicated that the system could handle a significant load of concurrent users and messages without compromising responsiveness or stability.

10.3 VALIDATION OF THE CHAT API'S SECURITY AND RELIABILITY

The validation of the Chat API's security and reliability was a critical aspect of the testing phase. Security validation involved assessing the effectiveness of encryption protocols and Block-chain Authentication mechanisms in safeguarding user data and preventing unauthorized access. Through rigorous testing and validation processes, the Chat API demonstrated its ability to provide secure and reliable communication, instilling confidence in users regarding the confidentiality and integrity of their interactions. Ongoing monitoring and evaluation will further validate the Chat API's security and reliability, ensuring continued effectiveness in mitigating emerging threats and meeting evolving user expectations.

CHAPTER 11

CONCLUSION AND FUTURE ENHANCEMENTS

11.1 CONCLUSION

In summary, the development of the Chat API represents a significant milestone in the realm of secure and reliable communication platforms. Throughout the project lifecycle, we have successfully addressed key challenges and implemented innovative solutions to deliver a robust and feature-rich chat application that prioritizes user privacy and data security.

Furthermore, our contributions extend beyond technical implementation. We have conducted thorough testing and validation processes to ensure the reliability and security of the Chat API, instilling confidence in users regarding the confidentiality and integrity of their interactions. Additionally, our commitment to continuous improvement and innovation has paved the way for future enhancements and advancements in secure communication technologies.

11.2 POTENTIAL ENHANCEMENT

Looking ahead, the Chat API holds immense potential for further development and enhancement. Future scope includes expanding the feature set to incorporate additional communication functionalities such as multimedia messaging, file sharing, and voice/video calling. Additionally, further research and development efforts can focus on enhancing encryption protocols and Block-chain Authentication mechanisms to adapt to evolving cybersecurity threats and user requirements.

11.3 FEATURE ENHANCEMENTS

11.3.1 MULTIMEDIA MESSAGING

- Introduce support for multimedia messaging capabilities within the Chat API, allowing users to exchange images, videos, and audio files securely.
- Implement encryption protocols tailored for multimedia content to ensure the confidentiality and integrity of media files transmitted over the platform.
- Enhance the user interface to accommodate multimedia messaging functionalities, providing a seamless and intuitive user experience for sharing diverse types of content.

11.3.2 VOICE AND VIDEO CALLING

- Integrate voice and video calling features into the Chat API, enabling users to initiate secure audio and video calls with contacts within the application.
- Implement end-to-end encryption protocols for voice and video streams to safeguard the privacy of communication and prevent interception by unauthorized parties.
- Optimize network protocols and codecs to ensure high-quality audio and video transmission, prioritizing reliability and low latency for seamless communication experiences.

11.3.3 ADVANCED AUTHENTICATION MECHANISMS

- Enhance user authentication mechanisms by incorporating advanced biometric authentication methods such as fingerprint recognition or facial recognition.

- ❖ Leverage biometric data for secure and convenient user authentication, providing an additional layer of identity verification beyond traditional username/password authentication.
- ❖ Implement multi-factor authentication (MFA) capabilities, allowing users to combine multiple authentication factors such as passwords, biometrics, and one-time passcodes for enhanced security.

In conclusion, the Chat API represents a culmination of innovation, collaboration, and dedication towards creating a secure and reliable communication platform for users worldwide. With a strong foundation in place and a clear vision for future enhancements, the Chat API is poised to redefine the standards of secure communication in the digital age.

APPENDICES

Appendix 1:

SAMPLE CODE

server.py

```
from socket import *
import threading
import mysql.connector
import json

class ZchatDB:
    def __init__(self) -> None:
        self.connect = mysql.connector.connect(host='127.0.0.1', user='root',
passwd='smfsql123',database="zchat")
        self.cur = self.connect.cursor()
        print("Database connected successfully")
    def add_message(self):
        pass

    def check_user(self,mobile_number,active_ip):
        try:
            self.cur.execute(f"SELECT * FROM users where mobile_number =
dataenc({mobile_number})")
            result=self.cur.fetchall()
        except:
            return False
```

```
        if len(result)>0:
            self.cur.execute(f"UPDATE users SET active_ip = '{active_ip}' where
mobile_number = '{mobile_number}'")
            self.connect.commit()
            try:
                self.connect.close()
            except:
                pass
            self.__init__()
            return True
        else:
            return False

    def get_username(self,mobile_number):
        self.cur.execute(f"SELECT username FROM users where mobile_number =
dataenc({mobile_number})")
        result=self.cur.fetchone()
        if len(result)==1:
            return result[0][0]
        else:
            return "Nick"

    def new_user(self,username,mobile_number,active_ip):

        insert_query = '''
            INSERT INTO users(username,mobile_number,active_ip)
            VALUES (%s, %s, %s)
```

```
        '''
        print(f"New use created.\nusername: {username}\nMobile Number:
{mobile_number}\nActive IP: {active_ip}")
        data = (username,mobile_number,active_ip)

        try:
            self.cur.execute(insert_query, data)

        except:

            self.__init__()
            self.new_user()

        self.connect.commit()
        try:
            self.connect.close()
        except:
            pass
        self.__init__()

        return True

db=ZchatDB()

class server:
    def __init__(self) -> None:
        server = socket(AF_INET, SOCK_STREAM)
```

```
        server.bind(("127.0.0.1", 5555))
        server.listen()

        print("Server is listening for incoming connections...")

        self.clients = []

        while True:
            client_socket, client_addr = server.accept()
            print(f"Accepted connection from {client_addr}")
            print(client_addr,client_socket)

            self.clients.append(client_socket)

            client_handler = threading.Thread(target=self.handle_client,
args=(client_socket,))
            client_handler.start()

    def handle_client(self,client_socket:socket):
        while True:
            try:
                # Receive data from the client
                data = client_socket.recv(1024).decode('utf-8')
                if not data:
                    break
                # Broadcast the received message to all connected clients
                self.process_data(client_socket,data)
```

```
            except Exception as e:
                print(f"Error: {e}")
                break

        # Remove the disconnected client
        self.clients.remove(client_socket)
        client_socket.close()

    def process_data(self,client_socket:socket,data):
        data_json={}
        try:
            dict_data=json.loads(data)
            if dict_data['process']=="user_check":
                if
db.check_user(dict_data['mobile_number'],client_socket.getsockname()[0]):
                    data_json['process']="found"
                    client_socket.send(json.dumps(data_json).encode('utf-8'))
                else:
                    data_json['process']="notfound"
                    client_socket.send(json.dumps(data_json).encode('utf-8'))
                return

            if dict_data['process']=="new_user":
                if
db.new_user(dict_data['username'],dict_data['mobile_number'],client_socket.getso
ckname()[0]):
                    dict_data={
```

```
                "process":"new_user_created",
                "status":True
            }
            client_socket.send(json.dumps(dict_data).encode("utf-8"))
        return

    if dict_data['process']=="message_boardcast":
        dict_data['username']=db.get_username(dict_data['mobile_number'])
        self.broadcast(dict_data,client_socket)
        return

  except Exception as e:
    data_json['process']="error"
    client_socket.send(json.dumps(data_json).encode('utf-8'))
    raise e
def broadcast(self,message,client_socket):
  for client in self.clients:
    if client!=client_socket:
      try:
        client.send(json.dumps(message).encode('utf-8'))
        print("Sended")

      except Exception as e:
        print(f"Error: {e}")
        client.close()
        self.clients.remove(client)
server()
```

client.py

```
from tkinter import *
import datetime
import json
from tkinter import messagebox
import socket
import threading

class VerticalScrolledFrame:
    def __init__(self, master, **kwargs):
        width = kwargs.pop('width', None)
        height = kwargs.pop('height', None)
        bg = kwargs.pop('bg', kwargs.pop('background', None))
        self.outer = Frame(master, **kwargs)

        self.vsb = Scrollbar(self.outer, orient=VERTICAL)
        self.vsb.pack(fill=Y, side=RIGHT)
        self.canvas = Canvas(self.outer, highlightthickness=0, width=width,
height=height, bg=bg)
        self.canvas.pack(side=LEFT, fill=BOTH, expand=True)
        self.canvas['yscrollcommand'] = self.vsb.set

        self.canvas.bind("<Enter>", self._bind_mouse)
        self.canvas.bind("<Leave>", self._unbind_mouse)
        self.vsb['command'] = self.canvas.yview

        self.inner = Frame(self.canvas, bg=bg)
```

```
        self.canvas.create_window(4, 4, window=self.inner,
anchor='nw',width=width)
        self.inner.bind("<Configure>", self._on_frame_configure)

        self.outer_attr = set(dir(Widget))

    def __getattr__(self, item):
        if item in self.outer_attr:
            return getattr(self.outer, item)
        else:
            return getattr(self.inner, item)

    def _on_frame_configure(self, event=None):
        x1, y1, x2, y2 = self.canvas.bbox("all")
        height = self.canvas.winfo_height()
        self.canvas.config(scrollregion = (0,0, x2, max(y2, height)))

    def _bind_mouse(self, event=None):
        self.canvas.bind_all("<4>", self._on_mousewheel)
        self.canvas.bind_all("<5>", self._on_mousewheel)
        self.canvas.bind_all("<MouseWheel>", self._on_mousewheel)

    def _unbind_mouse(self, event=None):
        self.canvas.unbind_all("<4>")
        self.canvas.unbind_all("<5>")
        self.canvas.unbind_all("<MouseWheel>")
```

```
    def _on_mousewheel(self, event:Event):
        if event.num == 4 or event.delta > 0:
            self.canvas.yview_scroll(-1, "units" )
        elif event.num == 5 or event.delta < 0:
            self.canvas.yview_scroll(1, "units" )

    def __str__(self):
        return str(self.outer)



class Software:
    def __init__(self):
        if self.connect():
            threading.Thread(target=self.software).start()

    def software(self):
        self.main=Tk()
        self.main.title("Z Chat")
        self.main.geometry("1200x700")
        self.main.iconbitmap("icon.ico")
        self.main.resizable(False,False)
        self.main.withdraw()

        self.main.bind("<Key>",self.clicker)

        self.message_entry = Text(self.main,height=4,width=80,font=("Arial",15))
        self.message_entry.place(x=200,y=610)
```

```
        self.message_entry.bind("<Return>",self.send_message)

        info_frame =
Frame(self.main,background="#AFEEEE",width=200,height=700)
        info_frame.place(x=1,y=1)


self.usernamelabel=Label(info_frame,text="Username",font=(("Arial",18)),backgr
ound="#AFEEEE",foreground="#000080")
        self.usernamelabel.place(x=10,y=40)


self.mobilelabel=Label(info_frame,text="Mobile",font=(("Arial",18)),background
="#AFEEEE",foreground="#000080")
        self.mobilelabel.place(x=10,y=100)

        load_history_button=Button(info_frame,text="Load
History",font=("Arial",20),foreground="#000080",background="#87CEFA",comm
and=self.fake_history)
        load_history_button.place(x=10,y=620)

        self.chatframe =
VerticalScrolledFrame(self.main,background="#87CEFA",width=980,height=610)
        self.chatframe.place(x=200,y=1)
```

```
send_button=Button(self.main,text="Send",font=("Arial",20),foreground="#00008
0",background="#AFEEEE",command=lambda:self.send_message_thread)
        send_button.place(x=1090,y=625)
        self.login()

        self.loginpage.protocol("WM_DELETE_WINDOW", self.on_closing)
        self.main.mainloop()

    def send_message_thread(self):
        self.message_send_thread=threading.Thread(self.send_message)
        self.message_send_thread.start()

    def connect(self):
        self.client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        try:
            self.client.connect(("127.0.0.1", 5555))
            self.client_connected=True
            return True

        except Exception as error:
            print(error)
            self.nickname = messagebox.showwarning(title="Connection
Error",message="Please check your internet and try again.\nIf your internet isn't
problem Please contact admin to resolve this issue")
            self.on_closing()
            return False
```

```
    def showmainpage(self):
        self.main.deiconify()

    def login(self):
        self.loginpage=Toplevel(self.main,background="azure3")
        self.loginpage.title("Z Chat Login Page")
        self.loginpage.geometry("300x450")
        self.loginpage.iconbitmap("icon.ico")
        self.loginpage.resizable(False,False)

        usernamelabel=Label(self.loginpage,text="Enter User
Name",background="azure3")
        usernamelabel.place(relx=0.2, rely=0.3, anchor=W)

        self.usernameentry=Entry(self.loginpage,width=30)
        self.usernameentry.place(relx=0.21, rely=0.35, anchor=W)

        mobilenumberlabel=Label(self.loginpage,text="Enter Mobile
Number",background="azure3")
        mobilenumberlabel.place(relx=0.2, rely=0.4, anchor=W)

        self.number_entry=Entry(self.loginpage,width=30)
        self.number_entry.place(relx=0.21, rely=0.45, anchor=W)

        self.number_entry.bind("<Return>",self.check_user)
```

```
self.loginbutton=Button(self.loginpage,text="Login",background="azure3",width=
15,height=2,command=self.check_user,justify="center")
        self.loginbutton.place(x=100, y=300, anchor=W,bordermode="inside")
        self.usernameentry.focus_force()

        self.loginpage.protocol("WM_DELETE_WINDOW", self.on_closing)

    def connect_user(self,mobile_number,active_ip):
        dict_data={
            "process":"user_check",
            "mobile_number":mobile_number,
            "active_ip":active_ip
        }
        self.client.send(json.dumps(dict_data).encode('utf-8'))
        server_response = self.client.recv(1024).decode("utf-8")
        print(server_response)
        server_response=json.loads(server_response)
        if server_response['process']=="found":
            return True
        elif server_response['process']=="notfound":
            return False

    def check_user(self,event=None):
        if self.usernameentry.get()=="" or self.number_entry.get() == "" or
self.usernameentry.get().startswith(" ") or self.number_entry.get().startswith(" "):
```

```
            messagebox.showwarning(title="Invalid Entry",message="Please correct
the given Data")
            self.loginpage_reload()
            return

        mobile_number=self.number_entry.get()
        active_ip=socket.gethostbyname(socket.gethostname())
        username=self.usernameentry.get()

        try:
            int(mobile_number)
        except:
            messagebox.showerror(title="Invalid Mobile Number",message="Please
Enter a valid Mobile Number")
            return

        if self.connect_user(mobile_number=mobile_number,active_ip=active_ip):
            self.loginpage.destroy()
            self.main.deiconify()
            self.receive_thread=threading.Thread(target=self.receive)
            self.receive_thread.start()

            self.usernamelabel.configure(text=f"{username}")
            self.mobilelabel.configure(text=f"{mobile_number}")

        else:
```

```
        if messagebox.askokcancel(title="Register New User",message="Are you
want to register as new user ?"):
            dict_data={
                "process":"new_user",
                "username":username,
                "mobile_number":mobile_number
            }
            self.client.send(json.dumps(dict_data).encode("utf-8"))
            server_response = json.loads(self.client.recv(1024).decode("utf-8"))
            print(server_response)
            if server_response["status"]:
                messagebox.showinfo(title="New Login
Registered",message=f"Username: {username}\nMobile Number:
{mobile_number}\nActive ID: {active_ip}")
                self.loginpage_reload()
        else:
            self.loginpage_reload()

    def loginpage_reload(self):
        for i in self.loginpage.winfo_children():
            if isinstance(i,Entry):
                i.delete(0,END)

    def reload(self):
        try:
            self.chatframe.destroy()
        except:
```

```
            pass
        self.chatframe =
VerticalScrolledFrame(self.main,background="#87CEFA",width=980,height=610)
        self.chatframe.place(x=200,y=1)

        self.update_scrollbar

    def update_scrollbar(self):
        self.chatframe.canvas.yview_moveto(1)

    def send_message(self,event=None):
        message=self.message_entry.get("1.0",END)


        if message.startswith("\n") and len(message.split("\n"))<1:

            messagebox.showwarning(title="Empty Message",message="The first Line
is Empty")
            self.message_entry.after(10,self.clear_text)
            return

        message_json = {
            "process":"message_boardcast",
            "mobile_number":self.mobilelabel.cget("text")
            }

        if message.endswith("\n"):
```

```
            message=message[:-1]
            message_json["message"]=message
            message_json['hash']=hash(message)


labelframe=LabelFrame(self.chatframe,text=datetime.datetime.now().__format__(
"%d-%m-%Y %H:%M:%S"))
        labelframe.pack(padx=10,pady=10,anchor=E)

        Label(labelframe,text=message,font=("Arial",12)).pack()

        self.chatframe.outer.after(10,self.update_scrollbar)
        self.message_entry.after(10,self.clear_text)

        self.client.send(json.dumps(message_json).encode("utf-8"))
        self.clear_text

    def receive(self):
        print("Listening Server")
        while True:
            try:
                dict_data = json.loads(self.client.recv(1024).decode('utf-8'))
                if dict_data['process'] == 'message_boardcast':
                    if dict_data['mobile_number']!=self.mobilelabel.cget('text'):

labelframe=LabelFrame(self.chatframe,text=f"{dict_data['username']}
{datetime.datetime.now().__format__('%d-%m-%Y %H:%M:%S')}")
```

```
                labelframe.pack(padx=10,pady=10,anchor=W)

Label(labelframe,text=dict_data['message'],font=("Arial",12)).pack()

                self.chatframe.outer.after(10,self.update_scrollbar)

            except Exception as e:
                print(f"Error: {e}")
                break

    def on_closing(self):
        self.running = False
        try:
            self.client.close()
        except:
            pass
        try:
            self.main.destroy()
        except:
            pass

    def clear_text(self):
        self.message_entry.delete("1.0",END)

    def fake_history(self):
        self.reload()
        for i in range(1,110):
```

```
            if i%2==0:

labelframe=LabelFrame(self.chatframe,text=datetime.datetime.now().__format__(
"%d-%m-%Y %H:%M:%S"))
                labelframe.pack(padx=10,pady=10,anchor=E)

            else:

labelframe=LabelFrame(self.chatframe,text=datetime.datetime.now().__format__(
"%d-%m-%Y %H:%M:%S"))
                labelframe.pack(padx=10,pady=10,anchor=W)

            Label(labelframe,text=i,font=("Arial",12)).pack(anchor=E)

        self.chatframe.outer.after(10,self.update_scrollbar)

    def clicker(self,event:Event):
        if event.keysym=="slash":
            self.message_entry.focus()

Software()
```

Appendix 2:

SCREEN SHOTS

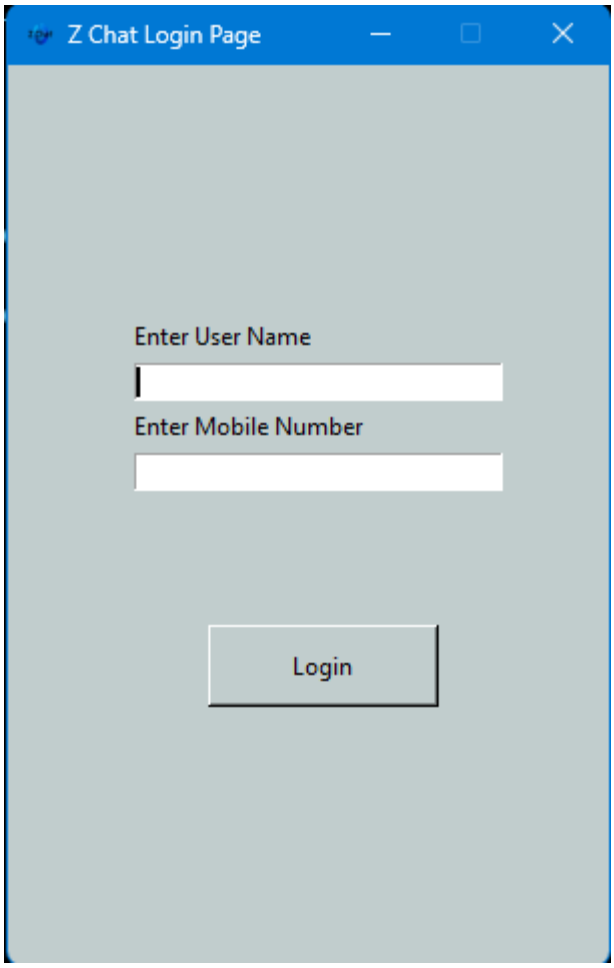

Figure No: A.2.1 Login Page

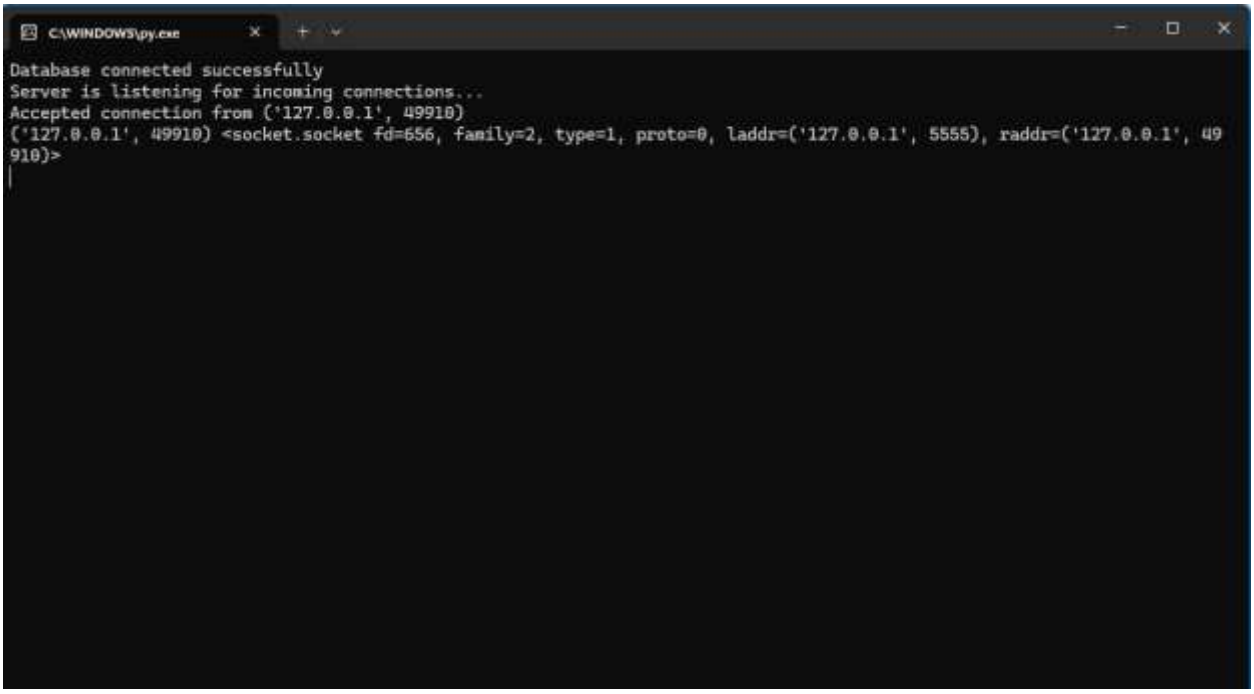

Figure No: A.2.2 Login Page

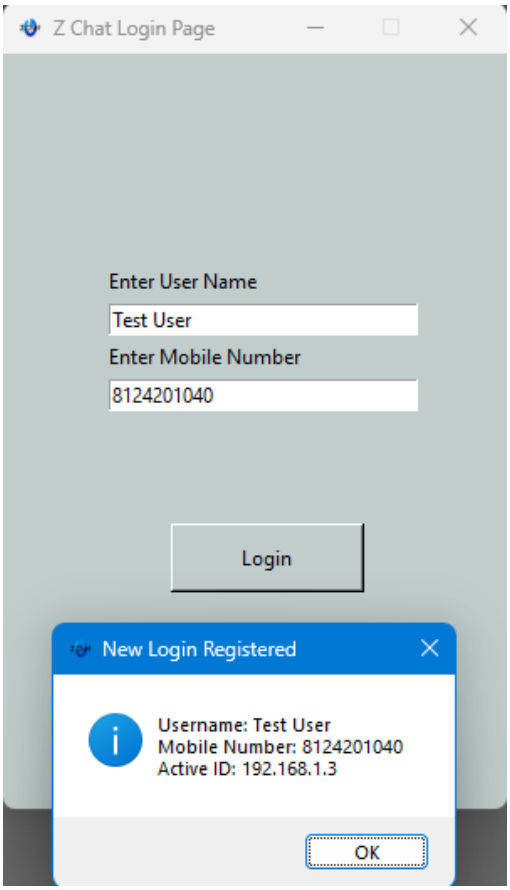

Figure No: A.2.3 New Registration Page

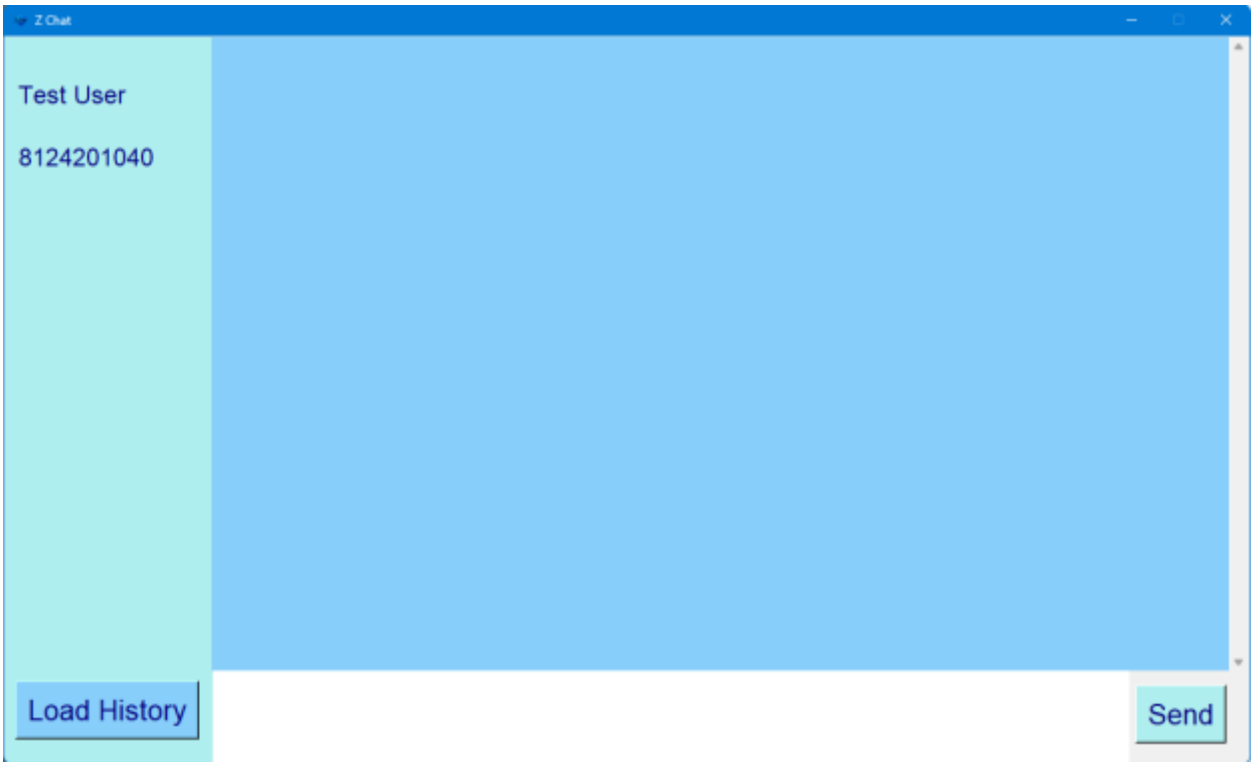

Figure No: A.2.4 Message Page

Figure No: A.2.5 Server side user verification

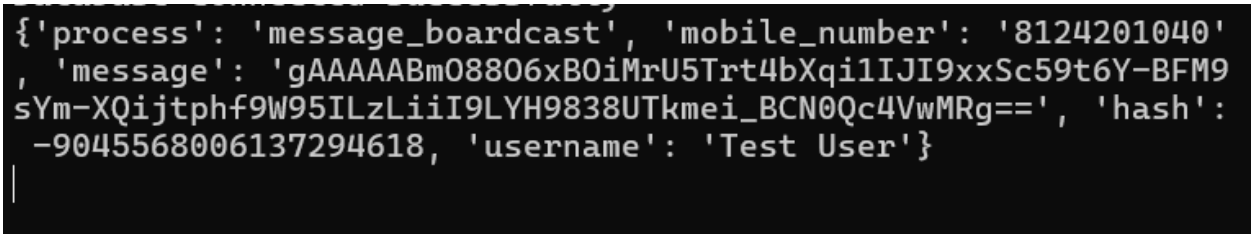

Figure No: A.2.6 Server side encrypted message

REFERENCE

[1]. Shuhan, Mirza K. B. & Islam, Tariqul. "A Secure and Decentralized Blockchain-Based Messaging Network." IEEE. 10195635,98. 2023.

[2]. Amey Tiwari, Rahul Talekar, Prof.S.M.Patil, “College Information Chat Bot System” International Journal of Engineering Research and General Science (IJERGS) Volume: 5, Issue: 2, Page no: 131-137| March-April 2017.

[3]. K. Jwala, G.N.V.G Sirisha, G.V. Padma Raju, “Developing a Chatbot using Machine Learning” International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878, Volume: 8 Issue: 1S3, Page no: 89-92| June 2019.

[4]. Noor Sabah, Jamal M. Kadhim, “Developing an End-to-End Chat Application” International Journal of Computer Science and Network Security, VOL., 17 No. 11, November 2017

[5]. Madhuri Chelamastti, Mallikharjuna Busani, “A Secure Double Encryption Chatting Application”, International Journal of Research Publication and Reviews. ISSN 2582-7421

INTERNATIONAL CONFERENCE PRESENTATION

KINGS COLLEGE OF ENGINEERING
An Autonomous Institution

Approved by AICTE, New Delhi
Affiliated to Anna University, Chennai
Recognized under 2(f) & 12B, UGC
NAAC Accredited Institution

Certificate of Participation

This Certificate is awarded to **GOKUL G**

of **UG SCHOLARS, M.I.E.T ENGINEERING COLLEGE** for

presenting a paper titled SECURE CHAT APT: ENCRYPTED CHAT APPLICATION PROGRAM INTERFACE WITH BLOCK CHAIN AUTHENTICATION

in the International Conference on Recent Trends in Engineering & Science (ICRTES-2024) on 2nd & 3rd May 2024, organized by the Centre for Promotion of Research (CPR), Kings College of Engineering, Pudukkottai.

Convener	Publication Chair	Conference Chair
Dr.P.P.Shantharaman	Dr.S.Sivakumar	Dr.J.Arputha Vijaya Selvi

GOKUL. G

KINGS COLLEGE OF ENGINEERING
An Autonomous Institution

Approved by AICTE, New Delhi
Affiliated to Anna University, Chennai
Recognized under 2(f) & 12B, UGC
NAAC Accredited Institution

Certificate of Participation

This Certificate is awarded to **PREMSRIDEV M**

of **UG SCHOLARS, M.I.E.T ENGINEERING COLLEGE** for

presenting a paper titled SECURE CHAT APT: ENCRYPTED CHAT APPLICATION PROGRAM INTERFACE WITH BLOCK CHAIN AUTHENTICATION

in the International Conference on Recent Trends in Engineering & Science (ICRTES-2024) on 2nd & 3rd May 2024, organized by the Centre for Promotion of Research (CPR), Kings College of Engineering, Pudukkottai.

Convener	Publication Chair	Conference Chair
Dr.P.P.Shantharaman	Dr.S.Sivakumar	Dr.J.Arputha Vijaya Selvi

PREMSRIDEV. M

KINGS
COLLEGE OF ENGINEERING
An Autonomous Institution

Approved by AICTE, New Delhi
Affiliated to Anna University, Chennai
Recognized under 2(f) & 12B, UGC
NAAC Accredited Institution

Certificate of Participation

This Certificate is awarded to RAHMATHULLAH A
of UG SCHOLARS, M.I.E.T ENGINEERING COLLEGE for
presenting a paper titled SECURE CHAT APT: ENCRYPTED CHAT APPLICATION PROGRAM INTERFACE WITH BLOCK CHAIN AUTHENTICATION
in the International Conference on Recent Trends in Engineering & Science (ICRTES-2024) on 2nd & 3rd May 2024, organized by the Centre for Promotion of Research (CPR), Kings College of Engineering, Pudukkottai.

Convener — Dr.P.P.Shantharaman
Publication Chair — Dr.S.Sivakumar
Conference Chair — Dr.J.Arputha Vijaya Selvi

RAHMATHULLAH. A

KINGS
COLLEGE OF ENGINEERING
An Autonomous Institution

Approved by AICTE, New Delhi
Affiliated to Anna University, Chennai
Recognized under 2(f) & 12B, UGC
NAAC Accredited Institution

Certificate of Participation

This Certificate is awarded to SHAIK MOHAMED FAHAD T
of UG SCHOLARS, M.I.E.T ENGINEERING COLLEGE for
presenting a paper titled SECURE CHAT APT: ENCRYPTED CHAT APPLICATION PROGRAM INTERFACE WITH BLOCK CHAIN AUTHENTICATION
in the International Conference on Recent Trends in Engineering & Science (ICRTES-2024) on 2nd & 3rd May 2024, organized by the Centre for Promotion of Research (CPR), Kings College of Engineering, Pudukkottai.

Convener — Dr.P.P.Shantharaman
Publication Chair — Dr.S.Sivakumar
Conference Chair — Dr.J.Arputha Vijaya Selvi

SHAIK MOHAMED FAHAD. T