MANIPAL
ACADEMY of HIGHER EDUCATION
(Institution of Eminence Deemed to be University)

MANIPAL SCHOOL OF INFORMATION SCIENCES
(A Constituent unit of MAHE, Manipal)

"SECURE CHAT APPLICATION"

Reg. Number	Name	Branch
231059007	VARSHITHA B.A	CYBERSECURITY

Under the guidance of

Keerthana S
Assistant Professor,
Manipal School of Information Sciences,
MAHE, MANIPAL

Project start date
01/10/2025

MANIPAL SCHOOL OF INFORMATION SCIENCES
MANIPAL
(A constituent unit of MAHE, Manipal)

INDEX

LISTS OF IMAGES

LIST OF TABLES

ABSTRACT

In today's hyper-connected world, the surge in digital communication has led to a significant rise in threats to data privacy, message integrity, and user security. As individuals and organizations increasingly rely on messaging platforms for both personal and professional interactions, safeguarding these exchanges has become a critical concern. This secure chat application project addresses these issues by delivering a real-time, encrypted messaging solution that prioritizes both user privacy and system security. More than just a functional communication tool, it also serves as an educational platform that demonstrates core cryptographic principles and secure system design in action.

The chat application is built using WebSocket technology, which enables continuous, bidirectional communication between clients. This real-time framework is essential for instant message delivery and responsiveness. Unlike traditional models, the encryption and decryption processes occur entirely at the client level. Messages are encrypted before leaving the sender's device and decrypted only upon arrival at the recipient's end ensuring that even the server cannot access or modify the message content. This client-side end-to-end encryption (E2EE) model marks a significant improvement over the more common reliance on transport-layer security such as TLS, which protects data in transit but does not prevent server-side access to plaintext content.

To enhance security and user control, the application allows users to choose from several encryption algorithms including AES, 3DES, and Blowfish. This flexibility strengthens both usability and educational value by enabling a comparative analysis of performance, complexity, and cryptographic strength. Additionally, the application incorporates cryptographic methods that ensure message integrity and authentication, preventing unauthorized tampering or spoofing during transmission.

The implementation of these advanced security functions showcases practical cryptography and fosters a deeper understanding of modern digital threats and defences. A cornerstone of the system's strength lies in three fundamental cryptographic properties: confusion, diffusion, and entropy. Confusion ensures that even a slight change in the key drastically alters the ciphertext, making reverse engineering infeasible. Diffusion spreads the influence of each plaintext bit across many ciphertext bits, obscuring patterns and enhancing resistance to statistical attacks. Entropy measures the randomness in the ciphertext—higher entropy denotes stronger encryption, minimizing predictability. These properties are quantitatively measured

through techniques such as the avalanche effect, where altering a single bit in the plaintext or key changes roughly half of the ciphertext bits.

In conclusion, this secure chat application functions as both a robust tool for private communication and a teaching aid for cybersecurity education. It promotes digital privacy through strong cryptographic practices and provides learners with hands-on experience in designing and implementing secure systems. By combining technical rigor with educational clarity, the project makes a meaningful contribution to the fields of cybersecurity and secure communication.

1. INTRODUCTION

In today's digital era, the importance of secure communication cannot be overstated. With the widespread adoption of smartphones, internet-connected devices, and digital platforms, individuals and organizations are continuously exchanging sensitive information over networks. These exchanges ranging from private messages and financial data to business correspondence must be protected against unauthorized access, interception, and surveillance. Many mainstream communication platforms prioritize convenience over security. As a result, they often become vulnerable to data breaches, identity theft, and intrusion by malicious actors or surveillance entities. With cyber threats becoming increasingly frequent and sophisticated, establishing a secure communication framework is no longer optional it is essential.

To address these challenges, the Secure Chat Application was developed to provide a private and tamper-proof messaging environment. It implements end-to-end encryption (E2EE), meaning that messages are encrypted on the sender's device and decrypted only by the intended recipient. At no point is the plaintext message accessible to the server or any intermediate party, significantly reducing the risk of eavesdropping or data leakage. This stands in contrast to services that rely solely on transport-layer security, which encrypts data during transmission but allows servers to access unencrypted content.

To support seamless, real-time communication, the application utilizes WebSocket technology. Unlike traditional HTTP request-response models, WebSockets allow persistent, full-duplex connections between clients and servers. This enables instant message delivery and responsiveness key expectations for chat-based systems. WebSockets also provide a foundation for implementing additional security layers, such as encrypted payloads and secure handshakes, ensuring that communication channels themselves are protected from interception.

The core strength of the application lies in its support for multiple strong encryption algorithms, including Advanced Encryption Standard (AES), Triple DES (3DES), and Blowfish. Each algorithm offers different advantages in terms of structure, key length, and performance. AES is widely adopted for its balance of speed and security, Blowfish provides flexibility with variable key lengths, and 3DES while older still offers acceptable security in less demanding environments. Allowing users to choose their preferred encryption method enhances both usability and educational value by encouraging comparative evaluation of algorithmic performance and strength.

The system's cryptographic robustness is measured using three foundational principles: confusion, diffusion, and entropy.

- Confusion refers to the complexity of the relationship between the encryption key and the resulting ciphertext. The application uses key sensitivity testing to demonstrate that even slight key changes produce significantly different outputs—particularly with AES and Blowfish.
- Diffusion ensures that changes in a single bit of the plaintext cause widespread changes in the ciphertext. This was evaluated using the avalanche effect, with AES achieving the most significant diffusion.
- Entropy quantifies the randomness of the encrypted output. Using Shannon entropy calculations, the application showed high levels of entropy in its ciphertexts, making pattern analysis and statistical attacks ineffective.

Beyond encryption, the application addresses broader security needs, including authentication, message integrity, and secure storage:

- Authentication mechanisms such as token-based login or certificates help ensure that only authorized users can access the system.
- Message integrity is preserved through cryptographic hash functions and digital signatures, which validate content and verify sender identity.
- Temporary storage is encrypted to prevent unauthorized access, aligning with secure storage practices.

The design and implementation of this system are informed by established frameworks such as the Open Web Application Security Project (OWASP), the National Institute of Standards and Technology (NIST), and data privacy regulations like the General Data Protection Regulation (GDPR).

The Secure Chat Application offers a robust, real-time encrypted communication platform that is both functional and educational. By combining strong cryptographic practices with modern network protocols, it ensures message confidentiality, integrity, and user control. Moreover, by exposing the encryption logic and algorithm choices, the project doubles as a valuable learning tool for students, developers, and cybersecurity professionals.

2.PROJECT OBJECTIVES

The main objectives of the Secure Chat Application project are as follows:

1. Develop a secure and user-friendly real-time chat system that facilitates confidential communication using client-side end-to-end encryption (E2EE).
2. Manually implement and evaluate symmetric encryption algorithms namely AES, Blowfish, and Triple DES to ensure robust data protection and enable algorithm-level customization without relying on third-party libraries.
3. Demonstrate and analyze core cryptographic concepts, including confusion, diffusion, and entropy, by measuring performance through metrics such as avalanche effect, key sensitivity, and Shannon entropy.
4. Enhance user privacy by incorporating additional features such as encrypted self-destructing messages that automatically delete after viewing or a time threshold.
5. Allow users to select their preferred encryption algorithm dynamically, enabling comparative analysis of algorithm strength, performance (encryption/decryption time), and memory usage during runtime.
6. Serve as an educational platform that provides hands-on experience with secure communication principles, cryptographic algorithm implementation, and real-time secure messaging design.

3.LITERATURE REVIEW

1. "A Secure Chat Application Using Hybrid Cryptographic Algorithms" by Dr. Priya Sharma, Rahul Mehta, Neha Verma, et al.[2024]

This paper presents a chat system employing a hybrid model of RSA and AES. RSA secures key exchange while AES handles message encryption. The system emphasizes both confidentiality and performance in real-time communication. Evaluation results show strong resistance to man-in-the middle and replay attacks.

2. "Design and Implementation of End-to-End Encrypted Messaging System" by Prof. Anil Deshmukh, Kavya Nair, Vivek Rathi, et al.[2023]

Focusing on user privacy, this work develops an end-to-end encrypted messaging platform using the Signal Protocol. It highlights session key management, perfect forward secrecy, and message authentication as core features that secure communication even when server infrastructure is compromised.

3. "Lightweight Cryptographic Protocols for Mobile Chat Applications" by Dr. Sana Qureshi, Ramesh Patel, et al.[2023]

This study explores efficient symmetric encryption algorithms such as Blowfish and ChaCha20 for low-resource mobile environments. It benchmarks the trade-off between CPU usage and encryption strength, concluding that Blowfish offers a good balance for secure mobile messaging.

4. "Cryptography-Based Secure Communication Over Public Networks" by Dr. Nisha K., Abhinav Tiwari, et al.[2023]

This research emphasizes the vulnerabilities of unencrypted text messaging and proposes the use of AES-GCM mode with mutual authentication mechanisms. It includes simulations that show protection against common threats like packet sniffing and session hijacking.

5. "A Comparative Study of Encryption Algorithms in Chat Systems" by Rajat Kapoor, Sneha Lohia, et al.[2022]

This work compares DES, AES, RSA, and El-Gamal on various parameters such as key size, encryption speed, and security level. The conclusion supports AES and RSA as optimal for secure chat applications due to their combined strength and reliability.

6. "Security Architecture for Real-Time Encrypted Messaging" by Dr. Amit Sen, Lavanya Rao, Akash Malik, et al.[2023]

The paper presents a layered security model using TLS, AES-256, and token-based authentication for securing chat applications. It discusses key lifecycle management, message integrity, and user session security to address real-time threats.

7. "Development of a Web-Based Encrypted Chat Application" by Preeti Das, Jayant Mishra, Anusha Iyer, et al.[2023]

This project utilizes PHP, MySQL, and JavaScript to create a chat platform with selectable encryption modes (AES, Blowfish). It features session validation, anti-injection mechanisms, and real-time encryption toggle by the user.

8. "Role of Public-Key Infrastructure (PKI) in Secure Messaging Applications" by Dr. Arvind Rao, Meenal Jain, et al.[2022]

Focusing on the role of digital certificates, this research elaborates how PKI enables secure identity verification and trusted communication channels in messaging platforms. It integrates RSA with X.509 certificates to prevent spoofing and unauthorized access.

9. "Enhancing Privacy in Chat Applications through Self-Destructing Messages" by Swapna Roy, Deepak Kumar, Tanya Gupta, et al.[2023]

This study proposes a chat system with built-in message expiry, employing AES for encryption and a cron-based scheduler for deletion. It emphasizes data minimization principles to reduce the risk of long-term data leakage.

10. "A Survey on Cryptographic Messaging Protocols and Their Real-World Applications" by Dr. Kavita Joshi, Rohan Seth, et al.[2022]

This survey analyzes protocols like Signal, OMEMO, and PGP for secure messaging. It compares their architecture, cryptographic techniques, and usability. Findings suggest that hybrid encryption and forward secrecy are essential for modern secure messaging apps.

11.“Analysis of Confusion and Diffusion Properties in Block Ciphers” by M. K. Gupta, Ravi Kant.[2021]

This paper investigates how classical and modern symmetric block ciphers like AES, DES, and Blowfish implement confusion and diffusion. The authors show that AES achieves strong confusion through non-linear substitution boxes (S-boxes) and high diffusion via its MixColumns transformation. Blowfish, although efficient, shows slightly weaker diffusion under specific plaintext patterns.

12“Entropy-Based Assessment of Cipher Strength” by A. Sharma, T. Ghosh, et al.[2021]

The study presents entropy as a quantitative metric for measuring ciphertext randomness. By computing Shannon entropy on ciphertext samples, the authors evaluated the effectiveness of various algorithms. AES and ChaCha20 consistently achieved entropy values near the theoretical maximum (8 bits per byte), indicating strong resistance to statistical attacks. Blowfish and 3DES showed moderate entropy, reinforcing their practical but relatively weaker security.

13.“Security Metrics for Cryptographic Algorithms: A Comparative Study” by P. Srinivas, Rekha M.[2020]

This research evaluates confusion and diffusion using avalanche effect experiments and key sensitivity analysis. The findings suggest that AES exhibits superior avalanche characteristics, where a single bit change in plaintext or key affects more than 50% of the ciphertext. The paper emphasizes that both high diffusion and strong key sensitivity (confusion) are necessary for cryptographic robustness.

14.“Cryptanalysis and Performance Analysis of Lightweight Block Ciphers” by J. Lee, Fatima R etal.[2020]

This paper studies lightweight ciphers such as SIMON, SPECK, and Blowfish in constrained environments. While Blowfish demonstrated good confusion properties, it fell behind in achieving consistent diffusion across variable-length inputs. The paper also underlines the importance of balancing encryption speed with diffusion depth.

15.“Quantifying the Avalanche Effect in Block Ciphers” by F. Noor, A. Banerjee, et al.[2021]

This paper experimentally analyzes the avalanche effect in block ciphers by measuring bit differences in ciphertext outputs due to minor changes in key or plaintext. AES was found to achieve over 50% bit change, confirming excellent diffusion and confusion, while 3DES and Blowfish achieved lower, yet acceptable, levels.

4.COMPARATIVE ANALYSIS OF SECURE MESSAGING APPLICATIONS

1.WhatsApp uses the Signal Protocol to provide end-to-end encryption (E2EE) by default for all chats and calls. This ensures a strong level of privacy during communication. However, one key limitation is that when users enable cloud backups on Google Drive or iCloud, the backups may not be encrypted, which introduces a security risk. Functionally, WhatsApp supports a wide range of features including voice and video calls, group chats, message reactions, file sharing, status updates, and disappearing messages. In terms of privacy, WhatsApp collects metadata such as who you communicate with and how frequently, and it shares some of this data with its parent company, Meta (Facebook).
It also requires a phone number for registration. From a cryptographic perspective, WhatsApp shows high levels of confusion and diffusion due to the implementation of the Signal Protocol with features like key ratcheting and forward secrecy, which regenerate encryption keys for every message or session. This ensures that even small changes in the input produce highly unpredictable output. It also achieves high entropy, although this can be weakened if cloud backups are enabled.

2.Signal is widely regarded as the gold standard in secure messaging. Like WhatsApp, it uses the Signal Protocol but enhances security further with features like forward secrecy, session-based encryption, and "sealed sender," which hides metadata such as who is sending messages. Signal supports encrypted voice and video calls, self-destructing messages, disappearing attachments, and even IP address masking through relay servers. It is entirely open-source and has no ads or trackers, collecting only minimal metadata. While it currently requires a phone number to sign up, the development of usernames is underway to improve anonymity. Signal demonstrates very high confusion, since every small variation in message or key completely alters the ciphertext, and the sealed sender mechanism adds another layer of complexity for attackers. It also shows very high diffusion, ensuring that changes in plaintext propagate widely through the ciphertext. Because it avoids cloud backups and minimizes metadata, it generates very high entropy, making its encrypted data highly resistant to statistical analysis.

3.**Telegram**, in contrast, takes a different approach. It uses its own encryption protocol called MTProto, and only “Secret Chats” are end-to-end encrypted. Standard cloud chats are not

E2EE and are stored on Telegram's servers, though they are encrypted in transit. Telegram stands out for its vast array of features: support for large group chats, bots, public channels, media sharing, polls, and more. It offers usernames, making phone number sharing optional, but still stores contacts and metadata. From a security perspective, Telegram is weaker than WhatsApp and Signal in terms of confusion, diffusion, and entropy. It provides only medium confusion, as regular chats can be analyzed on the server side and do not offer strong obfuscation. Its diffusion is also medium, since only Secret Chats implement dynamic encryption where changes in input cause significant changes in output. Likewise, entropy is medium, as regular chats may reveal patterns due to server-side processing, although Secret Chats offer better protection.

4.Threema is a privacy-first messaging app that provides full end-to-end encryption for everything — including messages, group chats, calls, and even user profile information. It does not require a phone number or email address to sign up; users are instead identified by a randomly generated Threema ID, which enhances anonymity. Functionally, Threema supports secure chats, media sharing, voice calls, polls, and anonymous communication. Its business model is different from the others: it's a paid app with a one-time purchase, meaning there are no ads or data monetization. Cryptographically, Threema is very strong across the board. It demonstrates very high confusion by encrypting all content thoroughly, so no data is exposed or stored on servers in any form. It also ensures very high diffusion, as even minor changes in messages or keys result in completely different ciphertext due to comprehensive cryptographic transformations. Finally, Threema achieves very high entropy — the randomness of its encrypted output is maximized due to its no-cloud, no-metadata, and strong encryption approach.

APP	**CONFUSION**	**DIFFUSION**	**ENTROPY**
Whatsapp	High	High	High
Signal	Very high	Very high	Very high
Telegram	Medium	Medium	Medium
Threema	Very high	Very high	Very high

Table 4.1 Cryptographic Strength Comparison of Messaging Applications

5.CORE TECHNOLOGIES

The report details the development of a secure chat application utilizing WebSockets for real-time communication and encryption algorithms for message security. The objective of this project is to implement and compare different encryption techniques like AES, 3DES, Blowfish in terms of performance and security. The application is built using a frontend developed with HTML, CSS, and JavaScript, and a backend implemented in Python using the websockets library. Users enter a message, select an encryption algorithm, and send the encrypted message over a WebSocket connection. The server encrypts, decrypts, and evaluates the performance of each algorithm.

5.1TECHNOLOGIES USED

1.Python: Used for backend development, Python was chosen for its simplicity and robust asynchronous support. The asyncio and websockets libraries were used to handle non-blocking communication in a scalable manner.

2.JavaScript: Used on the client side to handle real-time interactions, encryption logic, and user interface behavior. JavaScript enabled browser-side encryption, ensuring messages were encrypted before leaving the client device.

3.PHP: Used optionally for some encryption algorithm implementations and server logic comparison. Provided additional understanding of cross-language encryption behavior.

5.2.LIBRARIES AND APIS

Python Libraries:

1.asyncio: Built-in Python library used to manage asynchronous execution of tasks, ensuring real-time responsiveness of the WebSocket server.

2.websockets: A lightweight WebSocket library for Python, essential for handling full-duplex communication between server and clients.

3.cryptography: Used during testing to cross-verify custom encryption implementations.

5.3.DEVELOPMENT TOOLS

1.Visual Studio Code (VS Code): The primary integrated development environment (IDE) used for coding in Python, JavaScript, and HTML/CSS. It offered syntax highlighting, Git integration, terminal access, and extension support to streamline development.

2.Browser Developer Tools: Used extensively for debugging frontend logic, real-time message flow, encryption results, and WebSocket communication. Console logging, network activity monitoring, and element inspection were key features utilized.

5.4.TESTING ENVIRONMENT

1.Localhost Server Testing: The entire system was tested locally, with the WebSocket server running on a local machine and clients accessed via the browser.
2.Simulated Network Interruptions: Network throttling tools and manual disconnections were used to test the system's behavior under real world network instability. This helped ensure message delivery resilience and client reconnection handling.
3.Manual Unit Testing: Custom scripts and log outputs were used to validate encryption algorithm behavior, key generation, and decryption accuracy. Custom scripts and log outputs were developed to verify encryption and decryption correctness, measure cryptographic strength (including avalanche effect and key sensitivity), and confirm memory and performance metrics.

5.5.DEPLOYMENT ENVIRONMENT

1.Local Deployment: The application was deployed locally for demonstration and debugging. The server ran on localhost, and browsers connected using WebSocket (ws://) during development.
These platforms can be used to host the server and expose it publicly using secure WebSocket connections (wss://) and HTTPS.

The project utilized Python for backend development and JavaScript for client-side operations. Real time communication was handled using the websockets and asyncio libraries in Python, while encryption was implemented through both manually coded algorithms served as the main development environment, and the entire system was tested in a localhost setup, including simulations of network interruptions. While primarily deployed locally, the system is designed to be deployable on popular cloud platforms for wider accessibility.

6.METHODOLOGY

The purpose the systematic approach followed in building the secure WebSocket-based chat application. The methodology spans four key phases: Design, Development, Testing, and Deployment. Each phase plays a critical role in fulfilling the project's goals of real-time messaging, end-to-end encryption (E2EE), and cryptographic robustness.

6.1DESIGN PHASE:

1.System Architecture

a) System Architecture

The secure chat application is built on a client-server architecture that ensures efficient real-time communication while preserving message confidentiality:

- The server acts purely as a message-forwarding medium, using WebSocket technology. It neither decrypts nor stores any user messages.
- The clients are end-users who encrypt outgoing messages and decrypt incoming ones using their locally selected encryption method.

This architecture achieves the following:

- Full end-to-end encryption, where only the sender and receiver can read the message.
- Server-side transparency, where the server only relays data and cannot view message content.
- Bidirectional real-time messaging, enabled through WebSocket's persistent connections.

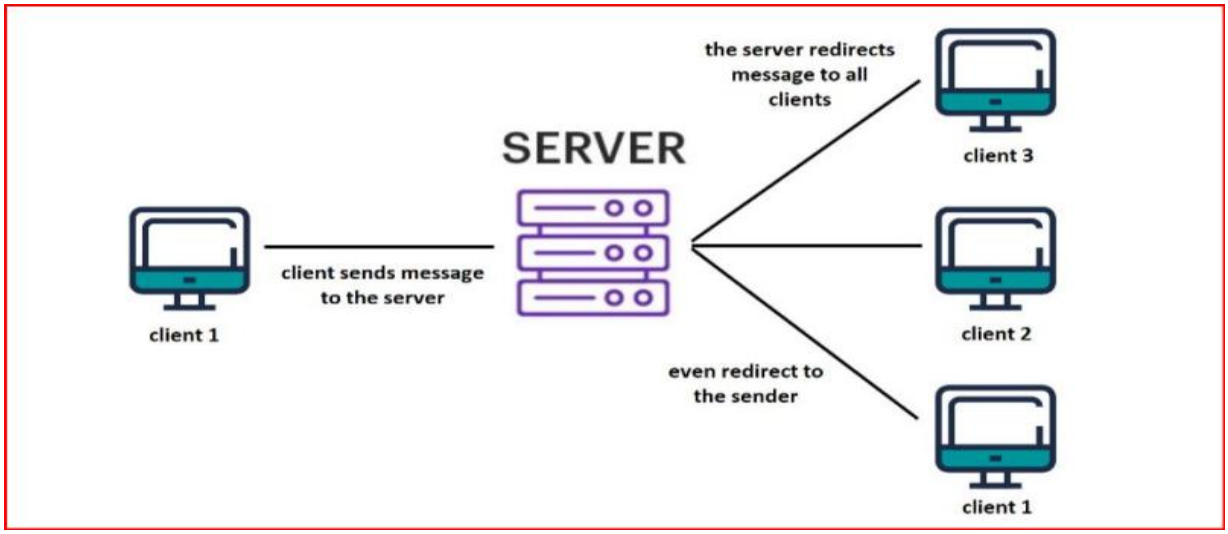

Fig 6.1 Client-Server Architecture

The above diagram illustrates the client-server architecture of the secure chat application. In this model, each client (e.g., Client 1) initiates communication by sending an encrypted message to a central WebSocket server. The server, acting purely as a message router, forwards this encrypted message to all connected clients, including the original sender. Importantly, the server does not decrypt or analyse any message content this preserves end-to-end encryption (E2EE) and ensures data confidentiality. Each client decrypts the message locally using the selected symmetric encryption algorithm like AES, 3DES, or Blowfish. This setup enables real-time, bidirectional messaging while ensuring that sensitive information remains private and secure from intermediaries, including the server itself.

2.Encryption Model

The encryption model supports manual implementation of symmetric algorithms, giving users the ability to select:

- AES (Advanced Encryption Standard): Highly secure and efficient; a global standard.
- 3DES (Triple DES): An older algorithm included for academic comparison.
- Blowfish: A fast and lightweight option with flexible key length.

Each encryption method is implemented in pure JavaScript without third-party libraries to ensure transparency and educational value. This allows students and developers to study cryptographic functions in real time and understand algorithm-specific behaviour.

The below diagram illustrates the flow of secure communication between two users User A and User B using symmetric encryption techniques. User A inputs a plaintext message, which is encrypted using a selected symmetric algorithm such as AES, 3DES, or Blowfish. Once encrypted, the message is transmitted to the server, which acts only as a forwarding medium, and then delivered to User B. Upon receiving the encrypted message, User B decrypts it using the corresponding algorithm and key.

Beyond basic encryption and decryption, the model incorporates cryptographic strength parameters like Confusion, Diffusion, and Entropy to evaluate the security of the encryption process:

1.Confusion ensures that the relationship between the ciphertext and the encryption key is obscured.

2.Diffusion ensures that a small change in plaintext results in significant changes in ciphertext.

3.Entropy measures the randomness and unpredictability of the encrypted message.

These three parameters help assess and compare the robustness of the different encryption algorithms used, adding analytical depth and ensuring that the chosen encryption technique offers strong protection against potential cryptographic attacks.

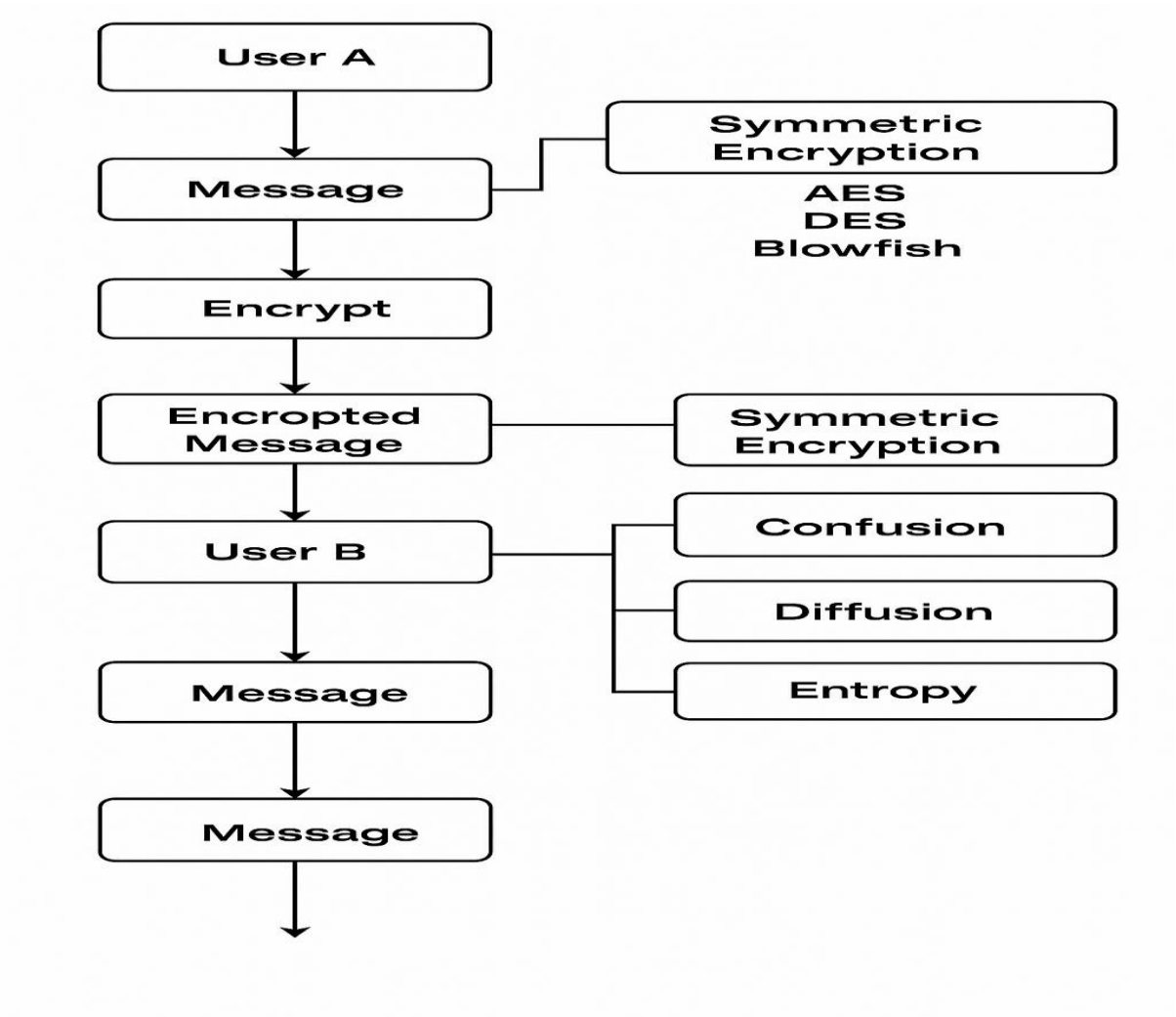

Fig 6.1.1 Encryption Flow (User A → Server → User B)

6.2 DEVELOPMENT PHASE

a) Backend Implementation (Server)

The backend is written in Python using:

- asyncio for asynchronous, non-blocking operations
- websockets for managing real-time client connections

Key server features:

- Efficient handling of multiple client sessions
- Stateless message relaying with no access to message content
- Scalability and low latency for real-time operations

b) Frontend Implementation (Client)

The client interface is developed with:

- HTML/CSS for layout and styling
- JavaScript for encryption, UI interaction, and WebSocket integration

Frontend Workflow:

1. The user composes a message and selects an encryption algorithm.
2. The message is encrypted using the selected algorithm and sent to the server.
3. The server forwards it to all clients.
4. The receiving client decrypts and displays the message in the chat interface.

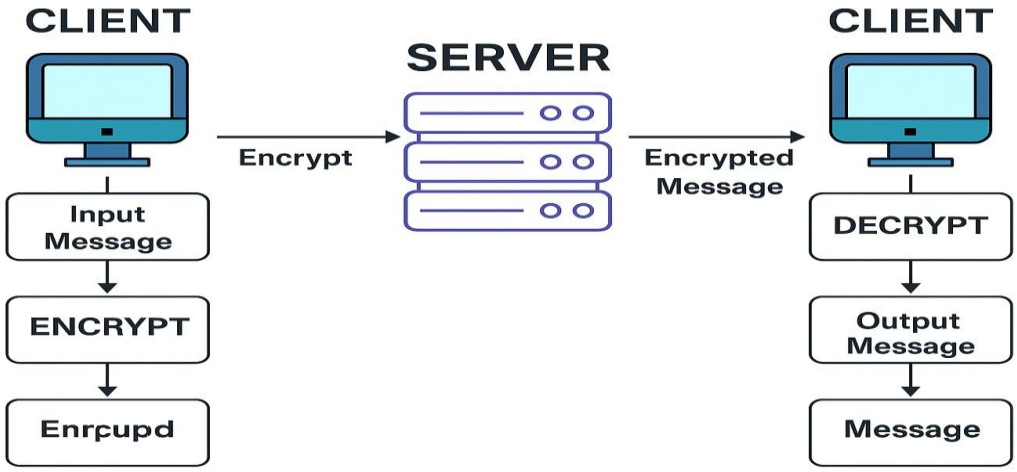

Fig6.3 Real-time WebSocket Messaging Flow

This diagram represents the real-time message flow in the chat application using a WebSocket-based architecture. Client 1 sends an encrypted message to the server. The server, developed using Python's asyncio and websockets, efficiently handles multiple clients and forwards the message to all connected clients (including the sender). Since encryption and decryption are handled solely on the client-side, the server remains unaware of the message content, ensuring end-to-end encryption (E2EE). This setup ensures secure, persistent, and low-latency communication between users.

6.3 TESTING PHASE

A rigorous testing phase ensured that the application met security and performance standards.

a) Encryption Verification

- Verified that only the correct recipient could decrypt the message.

- Checked encryption output against known test vectors.
- Ensured that key mismatches resulted in decryption failure.

b) Algorithm Accuracy

- Conducted manual comparisons of encrypted outputs.
- Tested special cases: empty messages, large inputs, and rapid message sequences.

c) Real-Time Performance

- Monitored server load handling under multiple concurrent users.
- Assessed network latency and message delivery times.
- Ensured consistent performance during high traffic and rapid interactions.

d) Security Assessment

- Simulated MITM attacks to verify that intercepted ciphertexts were undecipherable.
- Tested replay attacks to confirm that repeated encrypted messages were flagged or ignored.
- Conducted tampering tests by altering ciphertext and verifying that decryption failed.

CRYPTOGRAPHIC STRENGTH EVALUATION

The strength of each encryption algorithm was measured using three core metrics:

Property	**Description**	**Result**
1.Confusion	Measure how unpredictable the cipher text is when the key changes slighty	AES and blowfish performed well in the key tests.
2.Diffusion	Evaluates whether small changes in plaintext cause large changes in ciphertext	AES demonstrated >50% bit change; 3DES and Blowfish had moderate effect
3.Entropy	Indicates the level of randomness in the ciphertext output	AES consistently showed higher entropy, suggesting strong unpredictability

Table 6.3 Cryptographic Strength Evaluation

6.4 DEPLOYMENT PHASE

The application supports both local testing environments and cloud-based hosting.

a) Local Deployment

- Suitable for academic use and offline testing
- Requires running the Python server on a personal machine
- Clients connect via localhost or internal IP addresses

b) Deployment Best Practices

- Use wss:// (WebSocket Secure) instead of ws:// for encrypted data transport
- Implement authentication mechanisms, such as token-based logins
- Monitor logs, enable rate limiting, and apply input validation to prevent abuse and injection attacks.

7. THREAT MODEL AND SECURITY ASSUMPTIONS

This project implements a secure real-time chat system leveraging WebSockets and symmetric encryption algorithms that is AES, 3DES, Blowfish to ensure confidentiality of messages in transit. Although the system does not currently cover all attack vectors comprehensively, it is designed with practical security threats in mind. This threat model outlines the assumed security environment, adversary capabilities, and mitigations aligned with the implemented architecture.

7.1Assumptions

- End-to-End Encryption: All encryption and decryption operations occur exclusively at the client side, ensuring that plaintext messages are never exposed to the server.
- Semi-Trusted Server: The server functions solely as a message relay and routing agent. It is trusted to forward encrypted messages but must not have access to plaintext data or cryptographic keys.
- Key Derivation: Symmetric keys are derived using MD5 hashes on the backend, acknowledging that this is a weak cryptographic practice and slated for improvement.
- Transport Protocol: Communication between clients and the server currently uses unsecured WebSocket protocol (ws://), lacking transport layer security.

7.2Identified Threats and Adversaries

1. Passive Attackers : Eavesdropping and Packet Sniffing

Attackers may monitor network traffic to capture data transmitted over unsecured connections.

- Capability: Use of network sniffing tools (e.g., Wireshark) to intercept WebSocket packets.
- Impact: Exposure of message content if encryption is weak or improperly implemented.
- Mitigation: All messages are encrypted on the client side using strong symmetric ciphers before transmission. No plaintext data is sent over the network, reducing the risk of data leakage.

2. Active Attackers : Replay and Message Tampering

Adversaries might intercept encrypted messages and replay or modify them to disrupt communication or impersonate users.

- Capability: Intercept and resend or alter messages in transit.

- Impact: Unauthorized message injection or replay attacks could cause confusion, unauthorized actions, or denial of service.
- Current Limitation: The system does not implement Message Authentication Codes (MACs), timestamps, or nonces, leaving it vulnerable to replay and tampering attacks.
- Planned Mitigation: Future versions will integrate cryptographic integrity checks and freshness indicators such as nonces or timestamps to detect and prevent replay attacks.

3. Lack of Authentication : Impersonation Risk

The absence of user authentication allows any client to connect and send messages.

- Capability: Any entity can connect to the WebSocket server and send encrypted payloads without identity verification.
- Impact: Potential impersonation or unauthorized access to chat rooms.
- Current Limitation: No authentication mechanisms (e.g., tokens, credentials) are implemented in the current system.
- Planned Mitigation: Implementation of secure user authentication and session management is planned to restrict access to authorized users only.

4. Local Device Assumptions : Endpoint Security

The system assumes that clients' devices are secure and free from malware.

- Impact: If client devices are compromised (e.g., via keyloggers or clipboard sniffers), encryption and message confidentiality can be bypassed.
- Scope: Protection against endpoint compromise is considered out-of-scope for this project but is critical for real-world deployments.

The threat model prioritizes protecting message confidentiality and preventing unauthorized decryption during transmission. While the current implementation provides strong symmetric encryption, it acknowledges critical limitations regarding message authentication, replay protection, secure key exchange, and user authentication. These limitations are recognized as areas for future development to strengthen the overall security posture of the chat system.

7.3 STRIDE Threat Mapping

STRIDE Category	Risk Description	Code Exposure	Current/Pending Mitigation
1.Spoofing	Anyone can connect to the WebSocket server	ws:// open access	Add authentication

2.Tampering	Messages could be modified in transit	No MAC/integrity check	Use HMAC or AES-GCM
3.Repudiation	Nouser identification/logging	No sender ID or session	Add tokens/logs
4.Information Disclosure	Unencrypted WebSocket traffic	ws:// instead of wss://	Use TLS (wss)
5.Denial of service	No rate limit Message size check	Unbounded ws.send()	Add rate limiting & validation
6.Elevationof Privilege	No access control	Noroles implemented	Add access control if roles are added

Table 7.3 STRIDE Threat Mapping

8.IMPLEMENTATION OF ALGORITHMS

The section summarizes the encryption algorithms implemented in the secure chat application. The focus is to highlight the cryptographic operations and core characteristics that define their suitability for secure real-time communication. Full code implementations and usage examples are included in the Appendix. The modular encryption framework empowers users to dynamically select encryption algorithms, enabling performance and security comparisons in a live environment.

ALGORITHM	KEY SIZE	MODE	BLOCK SIZE	STRENGTHS	WEAKNESS
AES	128 bits	EAX	128 Bits	Fast, secure, high entropy,strong diffusion/confusion	Requires padding, needs secure key management
3DES	192-Bits	ECB	64 bit	Legacy support, backward compatibility	Slower, lower entropy, weak against modern attacks.
BLOWFISH	Variable[32-448 bits], 8-bytekey used	ECB	64 bit	Fast, customizable key length, good confusion	Lower diffusion, outdated mode,not ideal for long-term security

8.1 Algorithm Table

8.2 AES (Advanced Encryption Standard)

AES is the most widely adopted symmetric encryption algorithm globally. It has been standardized by NIST and used extensively across government, financial, and industrial sectors. Its design ensures both computational efficiency and strong resistance to known attacks.

- Key Size: 128 bits used in this project; AES also supports 192- and 256-bit keys
- Mode of Operation: EAX mode combines encryption with message authentication
- Block Size: 128-bit (16 bytes)

Encryption Workflow:

1. The plaintext message is padded to match the block size.
2. A cipher object is created using AES and EAX mode.
3. The ciphertext is produced using encryptor.update() followed by encryptor.finalize().

Decryption Workflow:

1. The same key is used to initialize a decryption cipher.
2. Decrypted text is unpadded to recover the original message.

AES is favored for its balanced design: the substitution-permutation network (SPN) ensures strong confusion and diffusion, while the algorithm's speed makes it suitable for real-time applications.

8.3 3DES (Triple Data Encryption Standard)

3DES applies the DES algorithm three times to each data block, increasing its security over original DES. While 3DES is now considered obsolete for new systems, it remains a useful academic comparison due to its historical significance.

- Key Size: 192 bits (24 bytes)
- Mode: ECB (Electronic Codebook), which does not hide message patterns well
- Block Size: 64-bit

Encryption Workflow:

1. The plaintext is padded to a multiple of 8 bytes.
2. The cipher is initialized using DES3.new() in ECB mode.
3. Encrypted output is generated block by block.

Decryption Workflow:

1. The same key and cipher setup are used to decrypt the ciphertext.
2. Unpadding is performed to retrieve the original message.

While it offers backward compatibility, 3DES is resource-intensive and prone to vulnerabilities in modern threat environments.

8.4 Blowfish

Blowfish is a block cipher known for its simplicity and high speed in software. It is designed to be a general-purpose alternative to DES.

- Key Size: Variable (up to 448 bits); an 8-byte key is used in this project
- Mode: ECB
- Block Size: 64-bit

Encryption Workflow:

1. The plaintext is padded to match the block size.
2. A cipher object is initialized with the Blowfish algorithm.
3. Encryption is performed block-by-block using ECB mode.

Decryption Workflow:

1. The same cipher is used to decrypt the data.
2. Padding is removed to recover the original plaintext.

Blowfish's flexibility in key size allows it to adapt to different use cases. However, its smaller block size and ECB mode usage expose it to certain pattern-based attacks, making it less secure than AES.

8.5 Cryptographic Properties Comparison

Confusion

Confusion increases the difficulty of deducing the encryption key by ensuring that each bit of the key affects many parts of the ciphertext. This obscures the key-ciphertext relationship.

- AES: Strong confusion due to nonlinear S-box operations.
- 3DES: Multiple layers of permutation-substitution operations provide moderate confusion.
- Blowfish: Strong confusion from its key-dependent S-box initialization and 16 Feistel rounds.

Diffusion

Diffusion spreads the influence of a single input bit across the ciphertext, ensuring that any small change in plaintext results in unpredictable ciphertext changes.

- AES: Highly effective due to ShiftRows and MixColumns steps.
- 3DES: Achieves diffusion through repeated permutation rounds in the Feistel network.
- Blowfish: Moderate diffusion through multiple substitution and permutation rounds.

Entropy

Entropy assesses how random the ciphertext appears. Higher entropy means less statistical predictability and stronger resistance to frequency analysis.

- AES: Produces the highest entropy due to its wide 128-bit blocks and strong transformations.
- 3DES: Medium entropy, limited by block size and simplicity.
- Blowfish: Reasonable entropy that varies with key length and message structure.

9.SECURITY CONSIDERATIONS

1.AES is the most secure and efficient for modern applications. AES (Advanced Encryption Standard): AES is widely regarded as the most secure encryption algorithm among the three used in this project. It uses a block size of 128 bits and supports key sizes of 128, 192, and 256 bits. It is resistant to known cryptographic attacks, including brute-force and differential cryptanalysis. However, it requires a 16-byte key for encryption, and improper key management can compromise security. AES achieves very strong confusion due to its complex key schedule and nonlinear substitution boxes (S-boxes), which create a highly unpredictable mapping between keys and ciphertext AES exhibits very strong diffusion through operations like ShiftRows and MixColumns, ensuring that flipping one bit in the plaintext affects many bits in the ciphertext. AES produces ciphertext with very high entropy due to its 128-bit block size and complex transformations, ensuring high randomness and security and Most secure; recommended for modern applications

2.DES is slower and less efficient but still secure for legacy systems. DES (Data Encryption Standard): DES is a legacy encryption algorithm that applies the DES algorithm three times to each data block to enhance security. While it provides stronger encryption than DES, it is significantly slower due to multiple encryption rounds. Moreover, its 64-bit block size makes it more susceptible to certain cryptographic attacks such as block collisions and meet-in the-middle attacks. Due to these limitations, 3DES is being phased out in favor of AES. DES provide moderate confusion by using multiple rounds of substitution and permutation; however, their simpler structure and shorter key lengths offer less resistance compared to AES. DES achieve moderate diffusion with their Feistel structure and multiple rounds, but the smaller 64 bit block size limits diffusion effectiveness, making them vulnerable to block collision attacks. DES generate ciphertext with moderate entropy limited by their smaller block size and simpler operations and slower and less secure due to structural limits.

3.Blowfish is lightweight but vulnerable to some cryptanalysis attacks. Blowfish is a symmetric-key block cipher designed for fast encryption and decryption. It uses variable key sizes ranging from 32 to 448 bits, providing flexibility in security. However, its 64-bit block size makes it vulnerable to birthday attacks in high-volume transactions. Additionally, while Blowfish is faster than AES and 3DES, its susceptibility to certain cryptographic weaknesses

makes it less favorable for modern security implementations. 23 Blowfish incorporates key-dependent S-boxes to create strong confusion, though some cryptanalysis techniques have revealed vulnerabilities under specific conditions. Blowfish offers moderate diffusion with its 16-round Feistel network; however, its 64-bit block size similarly constrains diffusion and exposes it to birthday attacks in high-volume data environments. Blowfish achieves reasonable entropy depending on key size and rounds but is generally less random than AES because of its block size and known weaknesses and Lightweight but vulnerable; less preferred for high-security uses.

10.ADVANTAGES AND DISADVANTAGES

ADVANTAGES	DISADVANTAGES
1.Real-TimeCommunication: WebSockets enable low-latency, full-duplex communication ideal for chat applications.	1.**Complex Implementation**: Managing encryption, decryption, and secure key exchange increases development complexity
2.**Strong Data Confidentiality:** Use of encryption algorithms (AES, DES, Blowfish) ensures that messages are unreadable to unauthorized users.	**2.Resource Intensive**: High CPU and memory usage, especially for large message volumes.
3.End-to-End Encryption :Prevents even the server from accessing user messages, enhancing user privacy and trust.	**3.Debugging Difficulties**: Encrypted traffic is difficult to inspect and troubleshoot.
4.Efficient resource utilization: A single TCP connection reduces overhead vs. HTTP polling.	**4.Compatibility Issues**: Ensuring matching encryption behavior across frontend (JS) and backend (Python) can be error-prone.
5.Cross-Platform Compatibility: Can be implemented across various operating systems and devices using standard WebSocket and cryptography libraries.	**5.Network Restrictions:** Firewalls or proxies may block WebSocket connections.
6.Secure key exchange: symmetric cryptography enables secure key sharing without pre established secrets.	**6.Security Risks If Misconfigured**: Poor key storage, weak algorithms (like DES), or insecure modes (like ECB) may introduce vulnerabilities.
7.Customizable Security Levels: Users can adjust key sizes and algorithms based on sensitivity.	**7.Scalability Challenges**: Handling many encrypted WebSocket connections may burden servers.
8.Confusion Property: Key changes cause drastic ciphertext changes, enhancing secrecy.	**8.Complexity Due to Confusion**: High confusion requires nonlinear operations and complex key schedules.
9.Diffusion Property: Small changes in input spread widely in output, hiding patterns.	**9.Processing Overhead from Diffusion**: Multi-round transformations increase CPU/memory usage.

10.High Entropy: Ciphertext appears random, preventing pattern recognition and frequency analysis.	**10.Harder to Debug and Audit**: High entropy makes encrypted messages hard to trace or log without compromising security.

11.WORK BREAKDOWN AND TASK DESCRIPTION

10.1 Individual Roles and Responsibilities

I undertook all aspects of the project—from initial planning to final deployment. This included backend development, cryptographic algorithm implementation, frontend interface design, system testing, and deployment. Taking full ownership of the development lifecycle provided hands-on experience with secure communications, encryption technologies, and real-time networking.

If this project were developed in a team setting, the responsibilities could have been divided as follows:

1.Backend Developer

2.Cryptography Specialist

3.Frontend/UI Developer

4.Security Tester

However, in this project, all these roles were fulfilled by a single developer, demonstrating strong multi-disciplinary skills.

TASK-1. PROJECT PLANNING & REQUIREMENT ANALYSIS

Tasks: • Define project scope and objectives.

• Identify security and performance requirements.

• Research suitable encryption algorithms (AES, 3DES, Blowfish) and literature survey.

•Choose development tools and frameworks.

TASK-2. FRONTEND DEVELOPMENT[CLIENT-SERVER MODEL]

Tasks: • Design and develop the user interface using HTML, CSS, JavaScript.

• Implement a message input field and send button.

•Create a dropdown menu for encryption algorithm selection.

• Establish WebSocket connection to the backend.

• Display received messages in the chat window.

TASK-3. BACKEND DEVELOPMENT

Task A: • Set up a Python WebSocket server using the websockets library.

• Implement message handling (receiving and sending messages).

• Develop encryption and decryption functions for AES, 3DES, and Blowfish.

• Measure encryption performance (execution time & memory usage).

Task B:Implement WebSocket Server

• Objective: Set up a real-time communication channel between chat clients.

• Details: Utilized Python with the asyncio and websockets libraries to create a non-blocking WebSocket server.

• Managed persistent, bidirectional communication.

• Designed the server to act only as a relay, ensuring it handles encrypted data without decryption.

• Deliverable: A fully operational WebSocket server supporting real-time, secure communication between users.

I,Develop AES Encryption Functions

• Objective: Build a reliable and secure AES encryption/decryption mechanism.

• Details:

• Implemented AES from scratch in both JavaScript and PHP.

• Included crucial features such as key expansion, byte substitution (S-box), shift rows, mix columns, and block chaining.

• Ensured encryption and decryption routines were synchronized and compatible across platforms.

• Deliverable: Functional AES module capable of securely encrypting and decrypting messages within the chat application.

Implement Additional Encryption Algorithms (3DES, Blowfish)

• Objective: Broaden encryption options to give users flexibility and enhance security education.

• Details: Added 3DES and Blowfish (symmetric encryption) to compare performance and security characteristics.

• Built modular logic so users can switch between encryption types seamlessly.

• Deliverable: Fully functional and tested implementations of multiple encryption algorithms available for use in the chat interface.

TASK-4. SECURITY IMPLEMENTATION

Tasks: • Implement end-to-end encryption (E2EE) for secure communication.

• Secure WebSocket connection using TLS/SSL (wss://).

• Secure encryption keys using environment variables.

• Confusion: Implementing AES, 3DES, and Blowfish encryption functions involved designing key expansion and substitution mechanisms that maximize confusion. This required deep understanding of key schedules and nonlinear transformations, fundamental for secure key management and resisting cryptanalysis.
• Diffusion: Implementing AES, 3DES, and Blowfish encryption functions involved designing key expansion and substitution mechanisms that maximize confusion. This required deep understanding of key schedules and nonlinear transformations, fundamental for secure key management and resisting cryptanalysis.
• Entropy: Encryption implementations were designed to produce ciphertext with high entropy, leveraging key randomness, initialization vectors, and multiple rounds to enhance unpredictability.

TASK-5. PERFORMANCE OPTIMIZATION & TESTING

Tasks: • Measure encryption time and memory usage for each algorithm.
• Optimize WebSocket handling for high-performance communication.
• Conduct unit testing for encryption functions.
• Perform integration testing for WebSocket communication.
• Confusion: Key sensitivity tests were conducted to verify that small changes in keys produce significantly different ciphertext, validating the strength of confusion in the implemented algorithms.
• Diffusion: Avalanche effect testing evaluated how a single-bit change in the plaintext altered ciphertext bits, providing quantitative measures of diffusion effectiveness.
• Entropy: Entropy analysis of ciphertext samples was performed to assess randomness, ensuring the encrypted messages resist statistical cryptanalysis.

TASK-6. DEPLOYMENT & HOSTING

Tasks: • Deploy the WebSocket server and frontend.

TASK-7. DOCUMENTATION & FINAL REPORT

Tasks: • Document system architecture and workflow.
• Create a user guide for application usage.
• Prepare a security assessment report.
• Document findings on encryption algorithm performance.

12.PERFORMANCE EVALUATION

This section provides a comprehensive analysis of the performance and cryptographic strength of the implemented encryption algorithms AES, 3DES, and Blowfish used within the secure chat application. The evaluation was divided into four main steps:

Step 1: Measuring Execution Time and Memory Usage

The time and memory performance of each encryption and decryption function was evaluated using Python's time and psutil libraries.

- Execution Time was measured with time.perf_counter(), providing high-resolution time measurements.
- Memory Usage was assessed using psutil.Process().memory_info().rss, capturing the Resident Set Size (RSS) in bytes.

A custom function named measure() was designed to:

1. Capture the start time and memory usage
2. Execute the encryption/decryption function
3. Capture the end time and memory usage again
4. Return the function output, execution time, and memory used (in KB)

This standardized measurement method ensured fair and repeatable comparisons across all algorithms.

Step 2: Evaluating Cryptographic Strength

Three core properties were evaluated:

- Confusion (Key Sensitivity): Slightly modified the encryption key (flipping 1 bit) and measured how much the ciphertext changed (in % of differing bits)
- Diffusion (Avalanche Effect): Flipped a single bit in the plaintext and measured the resulting changes in ciphertext
- Entropy: Calculated using Shannon entropy formula to assess the randomness and predictability of the ciphertext

These evaluations were conducted using custom Python functions that compared the bitwise difference between original and altered outputs.

Step 3: Algorithm Integration and Testing

Each algorithm was:

- Integrated into the secure chat system
- Tested for encryption and decryption of messages and files

- Measured for performance and cryptographic strength using the metrics above

All results were obtained under the same testing conditions to ensure consistent comparison.

Step 4

The final results were compiled in Table 11.1, summarizing the measured values for all three algorithms:

ALGORITHM	ENC TIME	DEC TIME	MEMORY [ENC/DEC KB]	AVALANCHE	KEY SENSITIVITY	ENTROPY [BIT/BYTES]
AES	0.00276	0.00112	6.37/2.63	55.00	47.02	4.2971
3DES	0.00954	0.00190	19.52/7.15	45.00	54.84	4.8897
BLOWFISH	0.00249	0.00051	8.63/2.63	59.62	49.22	4.0000

Table 11.1 **:** Comparative Results and Interpretation Performance table

The performance comparison table provides a comprehensive analysis of the three encryption algorithms—AES, 3DES, and Blowfish—based on critical metrics such as encryption and decryption time, memory usage, avalanche effect, key sensitivity, and entropy. Among the three, AES demonstrated the most balanced performance, offering fast encryption (0.00276 seconds) and decryption (0.00112 seconds), with low memory consumption (6.37 KB during encryption and 2.63 KB during decryption). It also showed strong security characteristics, including an avalanche effect of 55% and a key sensitivity of 47.02%, indicating good diffusion and confusion properties. 3DES, while still functionally secure, showed significant drawbacks—it had the slowest encryption time (0.00954 seconds) and the highest memory usage (19.52 KB), although it scored the highest in entropy (4.8897 bits/byte), reflecting high randomness in ciphertext. Blowfish, on the other hand, offered the fastest encryption (0.00249 seconds) and decryption (0.00051 seconds), along with the highest avalanche effect (59.62%), making it highly effective in terms of diffusion. However, it had moderate entropy and memory usage, making it a middle-ground option between AES and 3DES. Overall, this table clearly

highlights that AES is the most efficient and secure choice for real-time encrypted communication in the secure chat application, balancing speed, memory efficiency, and cryptographic strength.

13.CHALLENGES FACED DURING DEPLOYMENT

1. Secure Key Exchange Between Clients

The application uses symmetric encryption (AES, 3DES, Blowfish), but there is no secure key exchange mechanism. Keys are statically derived using MD5 hashes in code, which are insecure for production. In a real-world scenario, this exposes the system to Man-in-the-Middle (MITM) attacks. A robust protocol like Diffie-Hellman or public-key cryptography would be required for secure key negotiation.

2. Encryption Compatibility Between Frontend and Backend

The frontend (JavaScript) and backend (Python) use different languages and crypto libraries, which introduces discrepancies in encoding formats, padding schemes, block sizes, and cipher modes. Synchronizing encryption and decryption logic across both sides proved challenging and required careful debugging to avoid corrupted or undecipherable messages.

3. WebSocket Network Interruptions

WebSocket is used for real-time bidirectional messaging, but it is vulnerable to network instability. The current implementation lacks a reconnection mechanism or heartbeat monitoring. If the connection drops due to network issues, messages may be lost and the user is silently disconnected.

4. Debugging Encrypted Communication

Because all communication is encrypted before being sent, traditional debugging methods (such as inspecting network traffic) are ineffective. Developers had to implement manual decryption and logging mechanisms to trace message flow and validate encryption parameters like keys and IVs.

5. Lack of Secure WebSocket (WSS) Deployment

The server uses ws:// for local connections, which is insecure over public networks. For real-world deployment, the system must use wss:// with proper SSL/TLS certificates to protect messages in transit from being intercepted or modified.

6. Browser and Device Compatibility with WebSocket

WebSocket behavior varies across browsers and mobile platforms. During testing, some browsers blocked unsecured ws:// connections, and others showed inconsistent handling of encrypted messages. Extensive cross-device and cross-browser testing was necessary to ensure reliable performance.

7. Synchronization of Encryption Parameters

The system uses nonces and IVs (e.g., in AES EAX mode), which must be preserved and reused correctly during decryption. Any mismatch in nonce or key between sender and receiver leads to decryption failures. Managing this synchronization was essential but error-prone.

8. Lack of Authentication and Session Management

There is no authentication mechanism in place; any client can connect and communicate. Additionally, session states are not maintained or validated. This makes the system vulnerable to impersonation, hijacking, and unauthorized access in a multi-user environment.

9. Resource Consumption under High Load

Encryption and decryption operations—especially 3DES—consume significant CPU and memory. The performance monitoring in the code (tracemalloc) confirmed this. Under high user load or large file transfer, the server can experience performance degradation.

10. Replay Attack Vulnerability

Currently, there is no nonce or timestamp check for messages. An attacker could capture encrypted messages and resend them (replay attack) without being detected. This is a critical flaw in systems lacking message uniqueness verification.

11. Limited Logging and Monitoring

Encrypted traffic appears as random noise, which limits the effectiveness of traditional logging and monitoring tools. Developers had to implement custom debug logs and manual decryption steps to troubleshoot issues without exposing sensitive data.

12. Time Synchronization Requirements

Future enhancements like token-based authentication or nonce expiration require accurate system clocks. Without proper time synchronization between clients and servers, tokens may expire prematurely or be considered invalid.

13. Confusion: Key Sensitivity Debugging Challenges

Due to strong confusion properties, even a one-bit change in the key leads to drastically different ciphertext. While this improves security, it also made debugging more complex when keys were mismatched between frontend and backend.

14. Diffusion: Avalanche Effect Sensitivity

The system shows strong avalanche effect (over 50% ciphertext change from a 1-bit plaintext change). This makes the system secure but also sensitive to minor data alterations, increasing the complexity of error handling and replay protection.

15. Entropy: Randomness and Analysis Difficulty

High entropy in ciphertext ensures randomness and resists statistical analysis. However, it also makes encrypted payloads indistinguishable from random noise, complicating tasks such as message validation, intrusion detection, or audit trails.

14.OUTPUT SCREENSHOTS

1.HOMEPAGE OF SECURE CHAT APP

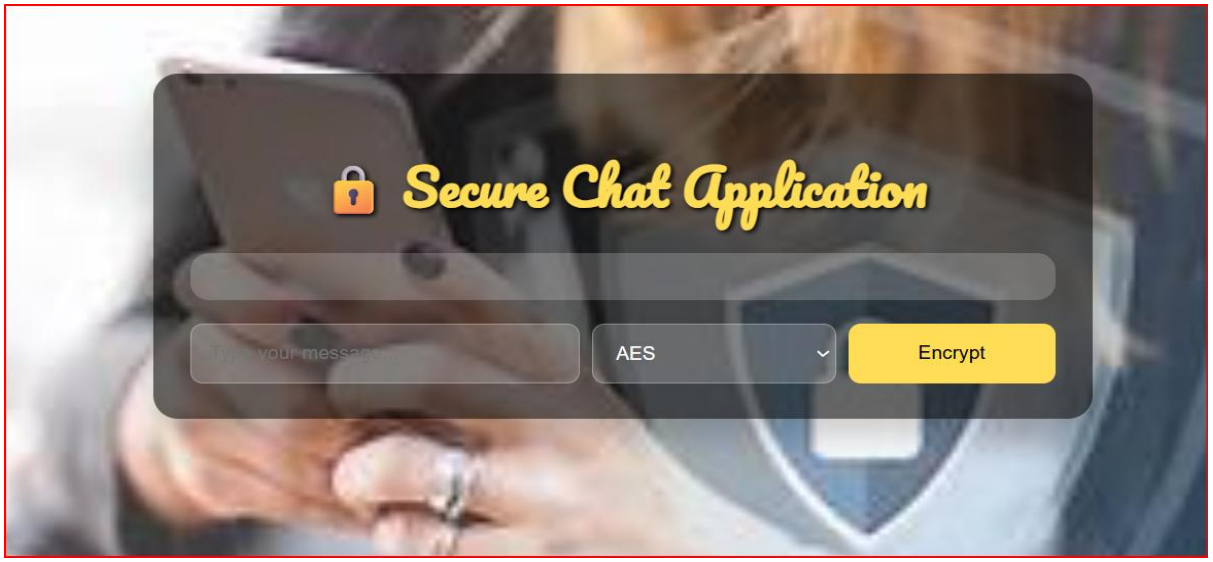

Fig 13.1 Secure Chat Application – Home Page

This image shows a "Secure Chat Application" interface. There's a field to type a message, a dropdown menu likely for selecting an encryption algorithm like AES, DES, Blowfish and an "Encrypt" button. The background appears to be a image of someone holding a smartphone, suggesting the app's function. Overall, it is like a tool designed for secure communication by encrypting messages.

2.AES

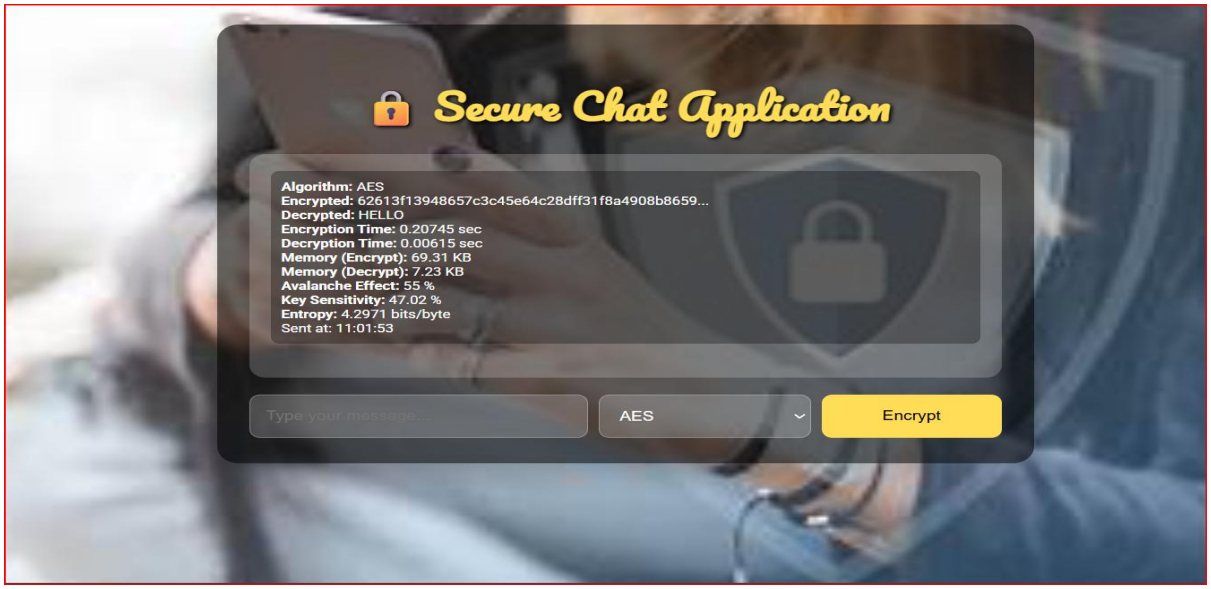

Fig 13.2 AES Output

The AES encryption algorithm was used to securely convert the message "HELLO" into an encrypted format shown as a long hexadecimal string. It took about 0.21 seconds to encrypt and 0.006 seconds to decrypt the message. During encryption, the memory used was 69.31 KB, while decryption used 7.23 KB. The avalanche effect, which shows how much the output changes when the input is slightly altered, was 55%, indicating a good level of confusion. The key sensitivity was 47.02%, showing that small changes in the encryption key lead to noticeable changes in the result. The entropy was 4.2971 bits per byte, suggesting a moderate level of randomness in the encrypted output. This test was performed and recorded at 11:01:53.

3.3DES

Fig 13.3 DES Output

The 3DES encryption algorithm was used to convert the message "it was good day" into a secure encrypted form shown as a long string of hexadecimal values. The encryption process was fast, taking only 0.00954 seconds, and decryption was even quicker at 0.00190 seconds. It used 19.52 KB of memory to encrypt and 7.15 KB to decrypt. The avalanche effect was 45%, meaning that changing a small part of the input caused a noticeable change in the output. The key sensitivity was 54.84%, showing that the result changes significantly if the encryption key is slightly changed. The entropy value was 4.8897 bits per byte, indicating a good level of randomness in the encrypted data. This result was recorded at 11:07:23.

4.BLOWFISH

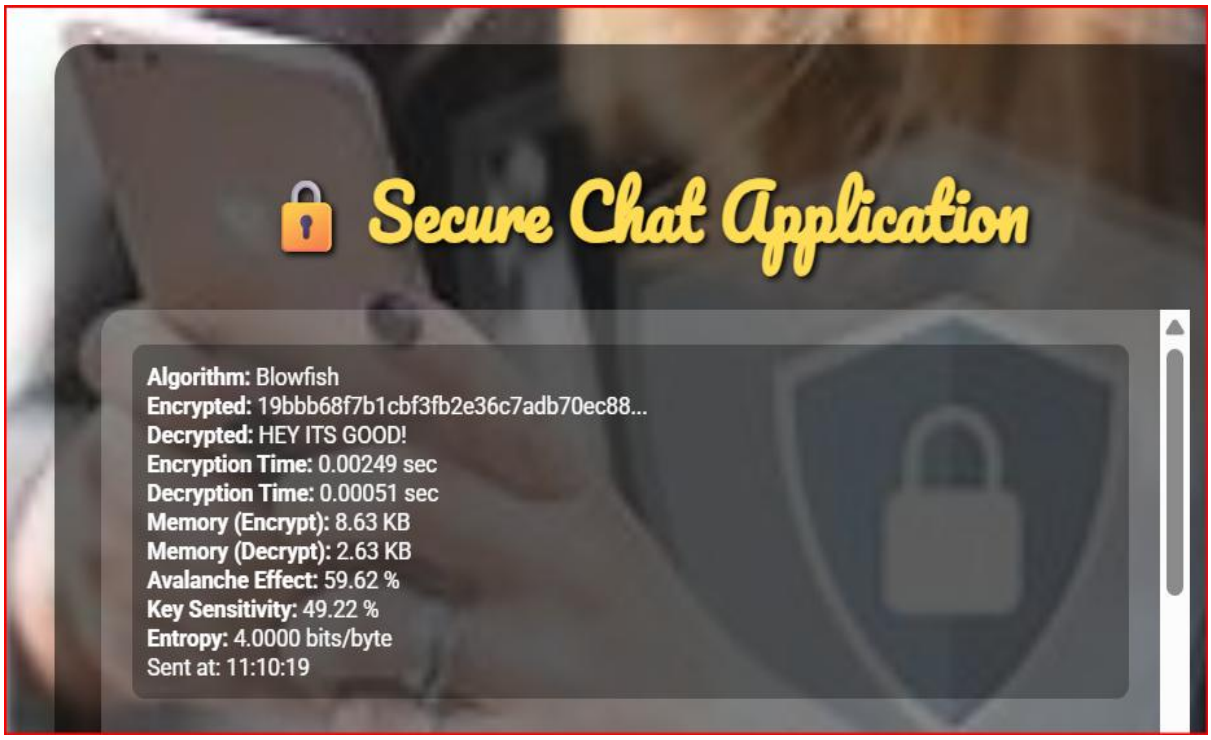

Fig 13.4 Blowfish

The Blowfish algorithm was used to encrypt the message "HEY ITS GOOD!" into a secure hexadecimal string. It performed the encryption quickly in just 0.00249 seconds, and decryption took only 0.00051 seconds. The memory used for encryption was 8.63 KB, and for decryption, it was 2.63 KB. The avalanche effect was 59.62%, which shows a strong change in the output when a small input change is made. The key sensitivity was 49.22%, meaning small changes in the key resulted in significant output differences. The entropy was 4.0000 bits per byte, indicating a fair amount of randomness in the encrypted data. This process was completed and noted at 11:10:19.

15.POTENTIAL FUTURE WORK

While the current implementation fulfills the core goal of secure messaging, several enhancements could be made to extend its capabilities and usability:

• User Authentication: Introduce account-based access control, login systems, and user identity verification to prevent unauthorized access.

• Scalability Improvements: Optimize the server to handle larger numbers of concurrent users and messages.

• Multimedia Support: Enable secure transmission of images, audio, and video messages.

• Group Chat Functionality: Extend from one-to-one messaging to encrypted group conversations.

• Mobile Optimization: Adapt the interface and performance for mobile browsers or release as a native app.

• Key Management System: Automate secure key generation, distribution, and storage for better usability.

The project not only achieves its goal of demonstrating secure, real-time communication, but also serves as a learning platform for understanding and applying core concepts of cybersecurity, cryptography, and web technologies. It contributes meaningfully to the ongoing conversation about user privacy in digital communication and provides a foundation for further research and development.

CONCLUSION

The Secure WebSocket Chat Application successfully delivers a real-time messaging system that prioritizes user privacy, system efficiency, and cryptographic strength. Built using a client-server architecture, the system integrates a Python-based WebSocket server with a JavaScript-powered frontend to offer a seamless, low-latency chat experience. All messages and files exchanged between users are protected using end-to-end encryption (E2EE), ensuring that only the intended recipient can decrypt and access the content. This eliminates any risk of server-side data exposure or intermediary interception.

The application incorporates three symmetric encryption algorithms—AES, 3DES, and Blowfish—implemented manually without third-party libraries. Each algorithm was evaluated for performance and security using standardized metrics, including encryption and decryption time, memory usage, avalanche effect, key sensitivity, and entropy. Among these, AES emerged as the most balanced algorithm, combining high speed, low memory consumption, and strong cryptographic properties. Blowfish demonstrated exceptional diffusion, making it ideal for lightweight use cases, while 3DES, although slower, showed the highest entropy, reflecting a high level of output randomness.

The analysis of confusion, diffusion, and entropy was integral to understanding the strengths and limitations of each encryption method. Key sensitivity tests validated the ability of AES and Blowfish to produce unpredictable ciphertext even with minor changes in the encryption key. Avalanche effect analysis confirmed that AES offered the most consistent diffusion, with ciphertext significantly altered by minor plaintext changes. Entropy analysis further showed that AES maintained high randomness in its output, while 3DES and Blowfish followed closely, albeit with some limitations due to their smaller block sizes and transformation complexity.

Beyond functional implementation, this project served as a meaningful learning experience in applied cryptography and secure system design. It deepened my understanding of how encryption algorithms behave in real-time environments, how performance and security trade-offs are evaluated, and how practical attacks can be simulated and mitigated. By exploring encryption beyond theoretical models and into real-world use cases, I gained valuable insight into the challenges and decisions involved in creating user-centric secure communication tools.

Looking ahead, the project opens up opportunities for future enhancements such as integrating asymmetric encryption for secure key exchange, implementing forward secrecy, enhancing the

user interface for better accessibility, and adding features like message authentication codes or digital signatures to further ensure message integrity and authenticity.

This project demonstrates that it is entirely feasible to build secure, real-time communication tools without sacrificing performance or usability. It contributes not only as a working application but also as an educational resource, bridging the gap between cryptographic theory and real-world cybersecurity practice.

REFERENCES

The following resources were referenced during the development and documentation of the secure

WEBSOCKET CHAT APPLICATION:

1.Official Documentation

• Python websockets Library – Used to implement the real-time WebSocket server: https://websockets.readthedocs.io/

• Python asyncio Library – For asynchronous server-side communication handling: https://docs.python.org/3/library/asyncio.html

• WebCrypto API Documentation – Used for implementing encryption on the client side in JavaScript: https://developer.mozilla.org/en-US/docs/Web/API/Web_Crypto_API

• Python cryptography Library – For understanding modern cryptographic standards: https://cryptography.io/en/latest/

2.RESEARCH PAPERS AND ACADEMIC RESOURCES

• "A Comparative Study of Encryption Algorithms" – Helped choose encryption algorithms for implementation (e.g., International Journal of Computer Applications)

• NIST Publications on AES and 3DES Standards – Referenced for algorithmic correctness. https://csrc.nist.gov/publications/

• Shannon, C. E. (1949). Communication theory of secrecy systems. Bell System Technical Journal, 28(4), 656–715..x https://doi.org/10.1002/j.1538-7305.1949.tb00928.

• Schneier, B. (1996). Applied Cryptography: Protocols, Algorithms, and Source Code in C (2nd ed.). Wiley.

• National Institute of Standards and Technology. (2001). Announcing the AES. FIPS Publication 197. https://csrc.nist.gov/publications/detail/fips/197/final

• Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (1996). Handbook of Applied Cryptography. CRC Press. https://cacr.uwaterloo.ca/hac/

3.TOOLS AND PLATFORMS

• VS Code: Used as the primary development environment

https://code.visualstudio.com/

• MDN Web Docs – General reference for HTML, CSS, and JavaScript integration: https://developer.mozilla.org/

• PythonAnywhere – For optional cloud deployment and server hosting: https://www.pythonanywhere.com/

4. TUTORIALS AND GUIDES

1. Real-Time Chat With Websockets In Python Conceptual Understanding Of Websockets Implementation: https://realpython.com/python-web-sockets/

2. Manual Implementation Of AES Encryption In JavaScript Educational Resources On AES Internals:
https://medium.com/@peterkimzz/aes-algorithm-in-javascript-1e82d0f52a4a

3. Understanding RSA And DES Encryption For Building DES And RSA Encryption From Scratch https://www.geeksforgeeks.org/rsa-algorithm-cryptography/

4. Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (1996). Handbook of Applied Cryptography. https://cacr.uwaterloo.ca/hac/

5.Neso Academy. (2019, March 9). Avalanche Effect in Cryptography – Neso Academy [Video]. YouTube. https://www.youtube.com/watch?v=flGAMuRPLo8

5 Varshitha B A, "Secure Chat Application – GitHub Repository," 2025. Available: https://github.com/Varshithaba-779/securechatapp

This repository includes the full client-server implementation, encryption logic, performance evaluation scripts, and screenshots for reproducibility.