Encrypto: Connect Protect Communicate

Mrs. Karishma Urkude[1], Hemant Nimje[2], Hemant Bopche[3], Om Sable[4],
Om Shahare[5], Gaurav Mudliar[6], Mayur Panghate[7]
[1,2,3,4,5.6.7]*G.H. Raisoni College of Engineering and Management, Nagpur*
doi.org/10.64643/IJIRT209761

***Abstract*— Private messaging platforms are used by the people every single day, as some of them keep the trace of conversation, On maybe on server, backup or on the device. At the same moment the chat hides every information completely can still be misused for threats and harm of some plans. Therefore, this paper presents ENCRYPTO, A web-based chatroom made for only two people where the conversation exists only for as long as the room exists. Messages are encrypted in the browser using AES-GCM with 256-bit key, and the room key is given from one user to another user through an ECDH P-256 Exchange, so server only relays unreadable ciphertext.**

Rooms are kept in server memory only and deleted when the both the server leaves the chatroom. To deal with the misuse, every message in analyzed by the sender's browser by a rule-based threat detection engine before it is encrypted. Normal messages are not saved in the database but the messages that engine flags that are likely to be risky are done to saved, for example a" I have a knife", is saved with category, room code, username and IP Addresses in access SQLite table. ENCRYPTO is a prototype and not a replacement for mature messengers, but it shows that privacy and accountability not have to be an all or nothing choice.

***Index Terms*— end-to-end encryption, ephemeral chat, AES-GCM, ECDH, client-side moderation, forensic logging, Socket.IO.**

I. INTRODUCTION

A private conversation is often taken for granted. But certainly, we expect it to stay between us. Though, many chat services store messages on their servers, keeping backups or leave history on device for years. Even when the chat is encrypted the text are often stay in some kind of form in the general resources. For people who can want it to say it once and make it remember in other mind also having the backup of that information is just human mind for sharing information from some sources to be reliable to be connect with on user to another user and data is vanished.

The idea of ENCRYPTO is where two people enter a chatroom, exchange information and leave. Once both of user have left the system is made for collecting as little as possible explanation of the information exchange.

There is the catch, this platform is attractive to people who want to plan harmful activities creates a old tension between privacy and safety in order to conclude a major plan. Most plan public discussion have two choices either the system reads everything or nothing at all.

ENCRYPTO tries and middle way as it flags or check the illicit language before the message is encrypted. If the text looks dangerous or causing a order to cause any type of harm is that only text message is kept as a evidence in the database to understand the incident basis.

This contributions of this works:

An encryption design that uses AES-GCM-256 for messages and ECDH P-256 for passing the room secret between users, without the server ever seeing the key.

A rules-based threat detection system works on text before encryption and can handle simple obfuscation such as "kill" or "k.i.l.l".

A selective logging mechanism that stores only flagged messages, protects them at the rest, limits how many are kept, and restricts who can read.

The key objectives of this research include:

To develop a web-based platform that allows only two participants at a time. The server relays only cipher to encrypt every message in the browser.

The room code works only for discovery and that the encryption key is separate and random. To vanish it's live data from server memory when both the user leaves the chat. To detect threatening or illegal

language in a message before it is encrypted and grade it. To store only flagged or detected information messages, with context such usernames, IP address and time where to protect the stores data.
To restrict access stored incident to an administration.

II. RELATED WORK

Several known messengers already offer strong encryption. Signal uses the signal protocol, where the double ratchet with pre keys to give forward secrecy and post-compromise security; Marlinspike and Perrin and by Cohn-Gordon and colleagues (2017).
WhatsApp adopted the same protocol for its end-to-end encryption chats.
Telegram used server-side encryption for ordinary chats and offers end-to-end encryption chats only in its client Element support end-to-end encrypted rooms through the Olm and megolm protocols and federal allows servers as to conclude the long-lived history and multiple device use of it.
For durable conversations these systems are built. Keeping history where it supports many devices and handles offline delivery.
ENCRYPTO has opposite choice where having no history of the shared information is valid with no accounts. It values where the shared information is remembers by the human mind not the user device for the privacy exchange information.

III. SYSTEM ARCHITECTURE

Client layer(browser): built with React 19 and Vite. The encryption code, the key pairs threat detection engine. Everything privately shares information here.
Cryptography modules: crypto.js handles the room secret and the AES-GCM encryption of room messages and the transfer of the room secret.

Layer	Technology
Front End	React 19
Build Tool	Vite
Back End	Node.js
Web Framework	Express
Real Time Messaging	Socket.IO
Message Encryption	WebCrypto API AES-GCM-256
Key Exchange	WebCrypto API, ECDH P-256
Hashing	SHA-256
Threat Detection	Custom rule-based engine (JavaScript, regex)
Room Storage	In-memory JavaScript Map
Forensic Database	SQLite
At-rest encryption	AES-256-GCM (Node crypto)

Table1.1Technologies Used

Moderation modules: moderation.js where the text is normalizes and applies the severity rules of it.
Transport layer: Socket.IO runs the events and contributes such as create-room, join-room, send-message, send-private-message, send-room-key.
Server layer (Node.js and Express): server.js acts as a relay and validator. It checks request rates, message length, base64ciphertext, key format, room codes, usernames and then forwards the data. It decrypts nothing.
In-memory room store: a JavaScript Map holds the active rooms where both the user will have conversation. it disappears when the room closes or the server restarts.
Forensic storage: Database.js writes flagged incidents into an SQLite table named incidents, where a limit on the number of stored rows with optional AES-256-GCM encryption of the message field.
Configuration: Values such as database path, forensic key, admin token, allowed client address, port.

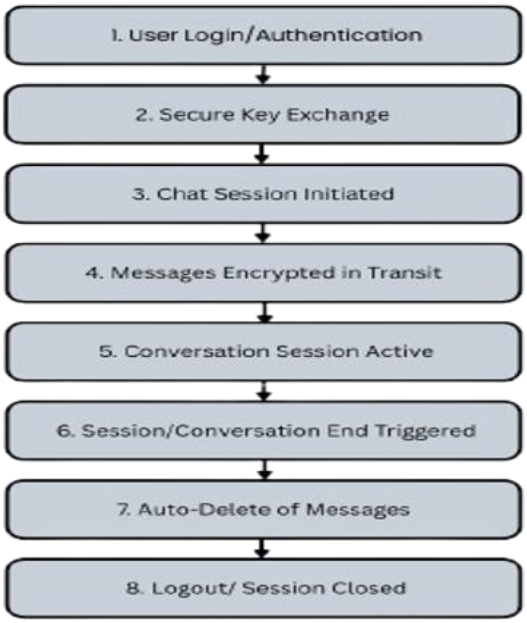

Fig: -Workflow of the ENCRYPTO

IV. PROBLEM STATEMENT:

People who prefer to have short secret conversation have two options currently. Mainstream messengers keep their messages in database or in other form as backup if any worst scenarios happen, As the conversation doesn't really disappear. Anonymous or fully hidden tools, whereas, the operator give no way at all to react when someone uses the chat to threaten. The problem addresses by this project stated as how a chatroom can be built by us for two users where the ordinary conversation is encrypted and solemnly doesn't store and removes the conversation once the room is left by the users also detecting the dangerous or threatening text and keep proof of them without storing them.

This problem has several parts:

1. The key must not be known to the server.
2. When the room is closed, it must disappear completely.
3. While typing detection must be fast enough.
4. The evidences, once kept, casual access must be protected.

V. FUNCTIONAL MODULES

Room management: Creates room with four characters, having only limit of two
person supports optional password protection with the hashed password, a list the open rooms, A knock and approve flow lets the host accept or decline a joiner with wrongs pass key.

Encryption module: Generates secret room and AED GCM - 256 encryption and decryption is done.

Key exchange module: Generates ECDH P- 256 encryption key pairs is shared between two people.

Threat detection module: Normalizes the text and assign a level of risk and score to the flagged message in case of direct threat, threat context, violence, weapon, attack planning, harassment etc. is done in the chatroom.

Validation and rate-limiting module: checks input format, message size and request frequency on server reducing abuse.

VI. METHODOLOGY

Requirement study: Systems had to guarantee what we firstly decided: only two users, messages are encrypted, vanish text after leaving the room, threatening text detection and minimal database storage for it. 2. Technology selection: React and Vite for real-time messages exchange also chosen for a fast client, Express and Socket.IO. 3. Room design: Rooms for only two people real-time communication as it concludes in server memory to exist. A four code is to find a room and has no role in encryption.4. key management: ECDH P-256 key pair is created by its own browser when the page loads. The 32-byte secret key generated the room. When someone joins, the secret is encrypted with the key derived from the two user's browser's ECDH exchange and deliver through the server.5. Message encryption: Every message with AES GCM-256 is encrypted with the random 12 byte. so identical sentences produce different cipher texts.6. Threat detection design: The text is normalised where standardised unicode.7. selective logging: The flagged result, the plain text is attached to a forensic message and the server saves it with category, reason, score. room code, usernames, IP address and timestamp. 8. cleanup: When users leave or disconnect, the server room is deleted, and the browser remove the secret message from the memory.

VII. RESULTS AND DISCUSSION

The prototype was tested locally and on servers from two separate devices. Observed behaviour are followed: Rooms could be created and joined using the four-character code. At third user trying to enter a full room was refused. After second user joined therefore, automatically the key transfer is completed and sand both can sending messages on the platform so where to conduct the form of the people to communicate privately it is well balance for it. When the user is done with conversation the normal conversation is not found in the database. The threat engine was checked with sample inputs hello - normal message, not stored: None bomb - A single sensitive word: Low Kill you - sensitive word: HIGH Every incident record include the room, usernames, IP addresses, time, score, category of it world data with internal logic and ensures scalability for future system is cost effective, user friendly, and capable of smart waste management solutions.

VIII. LIMITATIONS AND FUTURE SCOPE:

Limitation:
A modified client can bypass it and the server doesn't re-check the text.2. There is no public key infrastructure, so a malicious server can replace the keys to bypass.3. Detection is ruled based and can miss new phrasing or flag a none sensitive text.4. The tests were small and manual, and no large-scale evaluation is done.

Future Scope:1. QR method for verifying keys between users.2. Rate limiting backend by Redis so that it restarts and starts working across the servers.3. A report option to justify and clarify the flagged message 4. A transparent visualization over the improving accuracy on the flagged system by machine learning methods without adding the hidden surveillance.

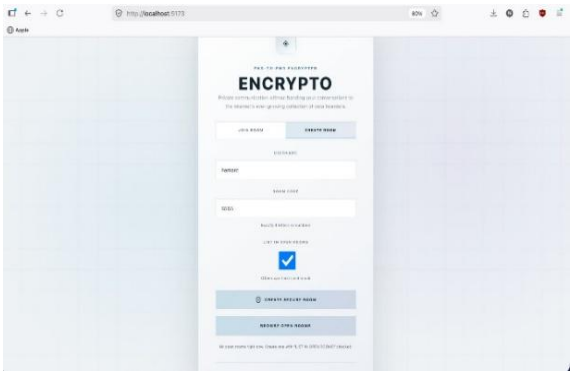

Fig1: Create room by User 1

User 1 specify a display name and a unique 4-character code. A checkbox is present there so that the publicly discover the room broadcasting.

Fig 2: -Room joined by the user 2

User specify a display name of user 2 and 4-character code so that the user 1 and user 2 can join together to chat secretly and privately.

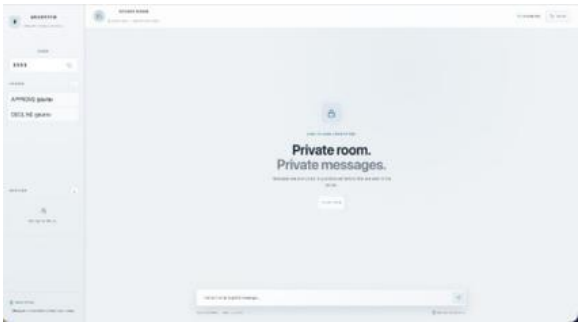

Fig3: Enter the chatroom

After writing the same code by both the users they are now ready for chat in a private room with end-to-end encryption for their messages.

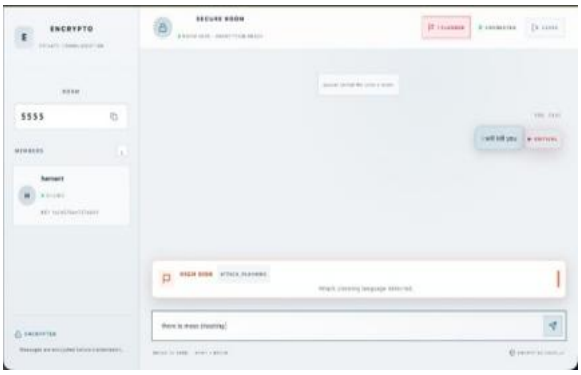

Fig4: High Risk text detected

As any suspicious messages are done by any of the user these messages will be flagged and will get save in the databases.

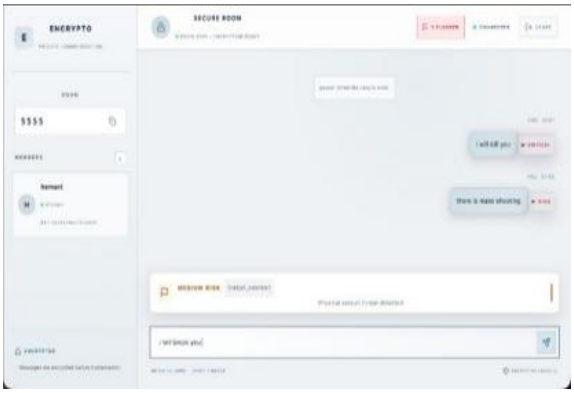

Fig5: Medium risk text detected

This is the medium risk type of suspicious message means it is a type of medium type of danger which can be not so dangerous.

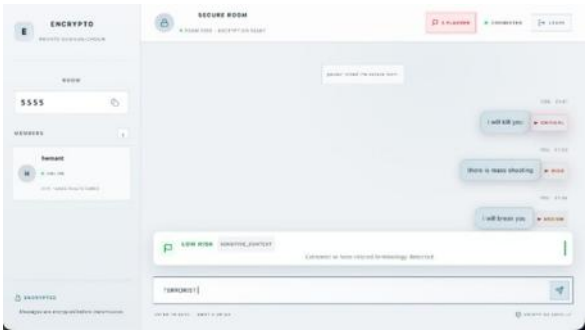

Fig6: Low risk text detected

This is the low-risk type of suspicious message means the person is just trying threat someone in a low-risk type manner.

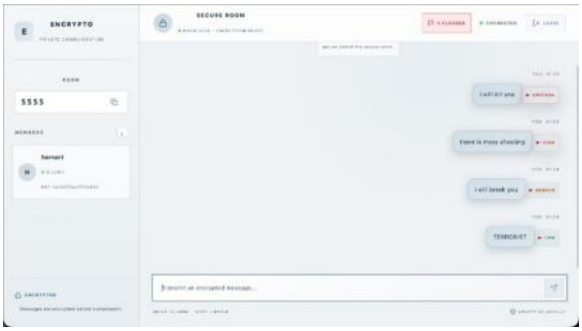

Fig7: All the types of threat

As the one more threat message is added above i.e critical message threat which is more dangerous than any of the above.

IX. CONCLUSION

ENCRYPTO strives that the private, secure chat between two users must be encrypted and vanish the text, when the user leaves the server and weather reacts the dangerous messages for further investigation of the case incase normal conversation is not saved after the particularly around checking is done from the client-side for the vital regards and key trust, demonstrating the practical middle ground secrecy and total surveillance, and it gives a base on which stronger safeguards can be built.

REFERENCES

[1] Fette, I., and Melnikov, A. (2011). The WebSocket protocol. RFC 6455. Internet Engineering Task Force (IETF).

[2] Banker, K., Bakkum, P., Verch, S., Garrett, D., and Hawkins, T. (2016). MongoDB in Action: With Deterministic Real-Time Messaging Architectures. Manning Publications.

[3] Pimentel, V., and Nickerson, B. G. (2012). Communicating and coordinating software agents using WebSocket's. IEEE Internet Computing, 16(4), 45–53.

[4] Tilkov, S., and Vinoski, S. (2010). Node.js: Using JavaScript to build high-performance network programs. IEEE Internet Computing, 14(6), 80–83.

[5] Grigorik, I. (2013). High-Performance Browser Networking: What Every Web Developer Should Know About Networking and Web Performance. O'Reilly Media.

[6] Marlinspike, M., and Signal Foundation. (2016). The Double Ratchet Algorithm and Ephemeral Key Exchange. Signal Cryptographic Specifications.

[7] Unger, N., Dechand, S., Bonneau, J., Fahl, S., Perl, H., Goldberg, I., and Smith, M. (2015). SoK: Secure messaging. In Proceedings of the IEEE Symposium on Security and Privacy (S&P) (pp. 232–249). IEEE.

[8] Ermoshina, K., Musiani, F., and Halpin, H. (2016). End-to-end encrypted messaging protocols: An overview. Internet Science, 9934, 244–254. Springer.

[9] Cohn-Gordon, K., Cremers, C., Garratt, L., Millican, K., and Milner, K. (2020). On the in-security of ephemeral messaging applications. Journal of Computer Security, 28(6), 633–662.

[10] Baccarini, M., and Read, B. (2023). Forensic analysis of ephemeral messaging applications: Disappearing messages or evidential data. Forensic Science International: Digital Investigation, 46, Article 301590.

[11] Davidson, T., Bhattacharya, D., and Weber, I. (2017). Automated hate speech detection and the problem of offensive language. Proceedings of the International AAAI Conference on Web and Social Media, 11(1), 512–515.

[12] Fortuna, P., and Nunes, S. (2018). A survey on automatic detection of hate speech in text. ACM Computing Surveys, 51(4), 1–30.

[13] Gillespie, T. (2018). Custodians of the Internet: Platforms, Content Moderation, and the Hidden Decisions That Shape Social Media. Yale University Press.

[14] Gorwa, R., Binns, R., and Katzenbach, C. (2020). Algorithmic content moderation: Technical and political challenges in automated speech governance. Big Data & Society, 7(1), Article 2053951719897945.

[15] Nobata, C., Tetreault, J., Thomas, A., Yashkov, Y., and Chang, Y. (2016). Abusive language detection in online user content. In Proceedings of the 25th International Conference on World Wide Web (WWW) (pp. 145–153).

[16] Sanfilippo, S. (2020). Redis Memory Optimization and Ephemeral State Management. Redis Labs Documentation Series.

[17] Srikantaiah, S., Singhal, V., and Kandemir, M. (2011). Adaptive memory tiering and garbage collection in event-driven asynchronous runtimes. ACM SIGPLAN Notices, 46(8), 115–126.

[18] Kleppmann, M. (2017). Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media.

[19] Voigt, P., and von dem Bussche, A. (2017). The EU General Data Protection Regulation (GDPR): A Practical Guide. Springer International Publishing.

[20] Schwartz, P. M., and Solove, D. J. (2014). Reconciling personal data and information privacy law under data minimization doctrines. California Law Review, 102(4), 877–916.