Proceedings of the 10th World Congress on Electrical Engineering and Computer Systems and Sciences (EECSS'24)
Barcelona, Spain - August 19 - 21, 2024
Paper No. CIST 140
DOI: 10.11159/cist24.140

Secure Chat Application with an End-to-End Encryption

Harrison Carranza[1], Miguel Bustamante[2], Aparicio Carranza[2], Suryaprakash Muniganti[3]
[1]Bronx Community College
2155 University Avenue, Bronx, NY, USA, 10453
harrison.carranza@bcc.cuny.edu
[2]Vaughn College of Aeronautics and Technology
86-01 23rd Avenue, Flushing, NY, USA, 11369
miguel.bustamante@vaughn.edu, aparicio.carranza@vaughn.edu
[3]Marist College
3399 North Rd, Poughkeepsie, NY, USA, 12601

Abstract **-** In today's world the significance of online communication has increased dramatically leading to the need, for strong encryption. This project focuses on creating a chat application that incorporates end to end encryption to enhance data security. The encryption technique used combines RSA (Rivest, Shamir, Adleman) for exchange and AES (Advance Encryption Standard) for message encryption striking a balance between security and performance. A crucial aspect of the development process was ensuring uninterrupted encrypted chats without any delays. The user interface of the application was designed using Pythons tkinter library making it user friendly and easy to navigate. Throughout the development phase extensive testing was conducted to identify and address any vulnerabilities. The encryption employed by the chat application has proven to be highly resilient, against security threats. We built this application using Python showcases how advanced security measures can be seamlessly integrated into real time chat platforms while maintaining an interface.

Keywords**:** Python, Cryptography, Encryption, RSA (Rivest, Shamir, Adleman), Advance Encryption Standard, tkinter

1. Introduction

In today's age communication methods are evolving at a pace. Instant messaging has become a part of our interaction thanks to its real time communication and user focused features. Whether it's through media platforms or professional collaboration tools, like Slack or Microsoft Teams or traditional text messaging on smartphones these methods have made it easier for us to connect instantly and effortlessly regardless of distance [1]. Given the landscape, the immediate and pressing need is for a chat application where user privacy is not an afterthought but a foundational pillar. This project aims to develop a Secure Chat Application that uses end-to-end encryption as its backbone. Such an application will ensure that only the sender and the intended recipient can read the content of their conversation, making it resistant to potential eavesdroppers, be it hackers, corporations, or any other unauthorized entity.

The rise of chat applications not only reflects the need for instant interaction but also emphasizes a pressing concern for security in these communications. Secure chat applications, such as the one highlighted in this project, prioritize user privacy by implementing end-to-end encryption. By ensuring that only the communicating users can read the messages, these platforms aim to reduce vulnerabilities and potential external threats. In this project, we present a robust chat application that integrates both client and server modules, facilitating secure public and private chat functionalities [2]. By leveraging the RSA for key exchange and the AES for message encryption, our approach ensures that all communications, whether broadcast or direct, are secured.

We could have achieved this project's functionality through a console application; however, for simplicity and enhanced user interaction, we employed the tkinter framework to create a user interface. By doing so, we ensured that alongside the robust cryptographic functions, we are presented with an interface that is both familiar and user-friendly. This design choice aligns with best practices in secure software design and strikes a balance between usability and security. The chat program emphasizes data persistence by allowing encrypted chat histories to be saved and retrieved, offering users both convenience and an additional layer of security [3].

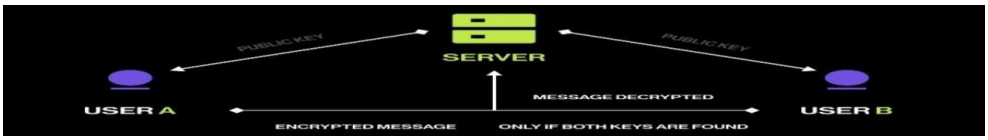

Fig. 1: End to End Encryption

Fig. 1, shows a schematic of an encrypted communication system involving a server and two users, A and B. The server distributes public keys to both users [4]. User A sends an encrypted message to the server, which then decrypts it, but only if both the public and corresponding private keys are available. The decrypted message is then securely transmitted to User B. The process ensures that messages can only be read by the intended recipient.

We strived to preserve message integrity and user confidentiality by employing both RSA and AES encryption techniques. This commitment to security represents a crucial step in our design approach. By integrating these encryption methods, we provided dual layers of protection. RSA is for safeguarding during the key exchange and AES for ensuring the actual message encryption [5]. Through this dual approach, we effectively strengthened our application against a number of cyber threats, including man-in-the-middle attacks, and eavesdropping, data tampering [6].

The rest of the paper is organized as follows: Section I provided the motivation and background of the work. Section II and III present Literature review and Methodologies respectively. Our Application Testing, Discussions, Results and Future Scope are presented in Section IV and V. Finally, we concluded in section VI.

2. Literature Review

A number of researches have been carried out on the security mechanisms supporting digital communication platforms. The rising cases of cyber threats have drawn significant attention to the essentials of ensuring user privacy and data integrity in this digital age. Several scholars and industry professionals have highlighted the critical importance of end-to-end encryption in chat applications. While the concept of end-to-end encryption is not groundbreaking in itself, its detailed application and criticality in ensuring a secure channel for communication have been extensively discussed in the literature.

In the context of secure messaging, Randall (1997) highlighted the transition from traditional communication methods to digital platforms, underscoring the increasing need for enhanced security measures in chat applications. Their findings suggest that with the increase of cyber threats, there is an crucial need for applications to guarantee user data and confidentiality, thus advocating for advanced encryption methodologies like RSA and AES [7].

Building on this, Sabah et al. (2005) focused on the technical insights of implementing RSA in chat applications. They noted that while RSA is computationally intensive, its unparalleled security benefits in key exchanges, especially in chat applications, make it a worthwhile investment for developers. The combination of RSA for key exchanges and AES for actual message encryption provides a robust two-layered security system that can defeat most cyber-attacks [8].

Yet, the sector of encrypted chat applications is not without its challenges. Hughes (2007) identified potential vulnerabilities even in encrypted platforms, especially if the encryption keys are compromised. Their research underscores the importance of continually updating and refining encryption algorithms to stay ahead of potential threats [9]. This observation resonates with our project's approach, emphasizing the proactive enhancement of security measures.

From a user experience perspective, in their study, U. Tariq et al. (2023) argued that security should not come at the cost of usability. Applications must ensure an intuitive interface, even when employing complex encryption techniques in the background [10]. The use of platforms like tkinter, as noted in our project, aligns with this principle, offering users an accessible interface while maintaining tight security protocols.

While the sector of secure chat applications continues to evolve, the consensus in the literature underscores the critical importance of end-to-end encryption. Balancing advanced security measures with user-friendliness remains a pivotal challenge, but with advancements like the ones highlighted in our project, the future of secure digital communication appears promising.

3. Methodologies

3.1. Platform Selection and Environment Setup

In the early stages of our project, we recognized the importance of choosing the right development environment and platform. We settled on Python due to its versatility and rich library support, which allowed us to implement sophisticated cryptographic functions seamlessly. With Python's compatibility with encryption libraries, such as RSA and AES from the Crypto module, we ensured that our application is equipped with state-of-the-art security features.

The decision to employ Python also facilitated cross-platform compatibility, enhancing our application's accessibility across various operating systems. This choice contributed to efficient development workflows, as Python's extensive libraries and modules accelerated the creation and testing of cryptographic functions critical to our application's security framework.

3.2. Key Generation using RSA

One of the pivotal steps we took was the incorporation of the RSA algorithm, fundamental to the asymmetric key generation in our application. Upon initialization, our application automatically generates a public-private key pair using RSA. We designed it such that while the public key is available for secure communication initiation, the private key remains undisclosed, guaranteeing that encrypted messages are accessible solely by their intended recipients.

To ensure robust security, our implementation adheres to industry-standard protocols for key size and complexity. The keys are generated through a cryptographically secure process, minimizing vulnerability to brute-force attacks. Rigorous testing has confirmed the resilience of our key generation mechanism under various threat models, underscoring its reliability for secure communications.

Our RSA implementation utilizes a secure random number generator for key creation, ensuring unpredictability and resistance to cryptanalysis. We also incorporate best practices for entropy collection, which fortifies the cryptographic strength of the keys. This proactive approach to security significantly enhances the confidentiality and integrity of the user data transmitted across our platform.

3.3. Establishing Connection using Sockets

For the communication infrastructure, we leveraged Python's socket library, allowing our chat application to set up channels for users to communicate. Our design includes:

Server Setup: Our application stands ready for incoming connections, awaiting a client's attempt to connect. Once connected, it facilitates the exchange of RSA public keys, paving the way for subsequent encrypted communication.

Client Setup: The client, upon activation, seeks and connects to an available server, ensuring an exchange of RSA public keys.

This two-way mode ensures adaptability in communication initiation, granting any user the flexibility to act either as a server or client. The socket-based architecture not only offers scalability but also ensures minimal latency in message delivery. Our implementation of socket programming provides a stable and secure medium for transmitting encrypted data, with robust error-handling capabilities to maintain uninterrupted communication even in volatile network conditions.

3.4. Implementing AES for Message Encryption

With the establishment of a secure connection and the successful exchange of RSA keys, we employed the Advanced Encryption Standard (AES) to encrypt chat messages. Given that AES uses symmetric encryption, the same key encrypts and decrypts messages. To strengthen security, we implemented a feature where a randomly generated AES key gets encrypted with the partner's RSA public key before sharing.

Incorporating AES ensured the confidentiality of user messages, with each session key uniquely tied to an individual conversation. The AES keys were systematically encrypted using the RSA public key of the receiving party, thus safeguarding the key during transit. This dual-layer encryption approach provided a fortified defense against potential eavesdropping and unauthorized message decryption attempts.

The deployment of AES was carefully done to operate in Cipher Block Chaining (CBC) mode, which ensured that patterns were not discernible in ciphertext. This mode was selected for its feedback mechanism, which provided an additional layer of security. The AES keys, once exchanged, were handled with strict confidentiality protocols, ensuring they remained uncompromised throughout their lifecycle.

3.5. Integration of GUI using tkinter

Understanding the crucial role of user experience, we decided to use the tkinter library to construct an interactive GUI for our application. Our user interface features:

A comprehensive chat history window showcasing all messages.

An easily accessible message input field.

A strategically placed send button facilitating message transmission.

The GUI was developed with a focus on user-friendly interactions, ensuring straightforward access to all application features. Emphasis was placed on creating a responsive design that accommodates various screen sizes and resolutions. This approach resulted in an adaptable and efficient user interface, significantly enhancing the overall usability and accessibility of the chat application.

3.6. Message Transmission and Display

Every message a user sends undergoes AES encryption and then travels via the pre-established socket connection. When the recipient receives it, our application decrypts the message using the corresponding AES key, displaying it for the user in the chat interface. The transmission process was streamlined to ensure real-time delivery of messages with minimal delay. On the recipient's end, the swift decryption and display of messages in the chat interface were critical for a seamless communication experience. This efficient handling of messages, from encryption to display, underscored the application's reliability and effectiveness in secure messaging.

3.7. Persistent Chat History

One of our innovative features is the encrypted chat history storage. Every conversation gets encrypted and stored in a dedicated local file, named chat_history.txt. This design ensures that the chat history, even if, it is accessed by the third party, the contents cannot be read.

To further enhance privacy, the chat history file incorporates unique encryption for each user session. This approach not only protected the stored data from unauthorized access but also maintained its integrity in case of system compromises. The straightforward file naming convention facilitated easy retrieval and management of historical data, enhancing user convenience while upholding stringent security measures.

4. Steps

4.1. Running rsa_main_chat_without_encryption.py (Unencrypted Chat):

1. Navigate to the location of the script in your terminal or command prompt.
2. Run the script using the command: **python rsa_main_chat_without_encryption.py**.
3. When prompted, enter the IP address or use the default provided (e.g., **192.168.74.50**).
4. Enter the desired port number or use the default provided (e.g., **9999**).
5. Decide whether to host (server mode) by selecting **1** or to connect (client mode) by selecting **2**.
6. If there's an error like "[WinError 10049] The requested address is not valid in its context," it might mean the IP address you entered isn't correctly configured on your system or isn't reachable.

4.2. Running the GUI Version of the Chat

After successfully initializing the server and client, two GUI windows appear, named "SimpleChat". One represents "You" (host or server) and the other "Partner" (client or the other user).

Type a message in the "You" window and click on "Send". The message will appear in the "Partner" window.

4.3. Analyzing Traffic with Wireshark

1. Open Wireshark.
2. Select the interface **Adapter for loopback traffic capture**. This allows you to capture local traffic between applications running on the same machine.
3. Start the packet capture.
4. Interact with your chat application to send and receive messages.
5. In Wireshark, you should see the TCP packets corresponding to your chat application's traffic.
6. Inspect these packets to see the data being transferred. For the **rsa_main_chat_without_encryption.py** script, this data will be in plain text and readable. When using **rsa_main_chat.py** (which has RSA encryption implemented), the payload should appear as encrypted format.

4.4 Running rsa_main_chat.py (Encrypted Chat)

1. Navigate to the location of the script in your terminal or command prompt.
2. Run the script using the command: **python rsa_main_chat.py**.
3. Follow the same steps as before for IP, port, and mode selection.
4. Communicate using the chat, and this time, the data sent between the server and client should be encrypted. If you capture this in Wireshark, it won't be easily readable as plain text.

5. Results and Discussion

As a part of our project objective, we carried out a deep analysis of the Secure Chat application that we designed. Through this section, we provide an in-depth discussion of our results, giving insights into the empirical findings and potential implications of our chat solution.

5.1. Testing the Unencrypted Chat

To begin with, we simulated an environment where two instances of the chat application communicated without our proposed encryption. Using Wireshark, we intercepted the data packets being transferred between the server and the client. Below, Fig. 2, shows a command-line interface of a chat application running in a Windows environment. The screen displays an unencrypted chat session where users are able to exchange messages, visible as plain text entries.

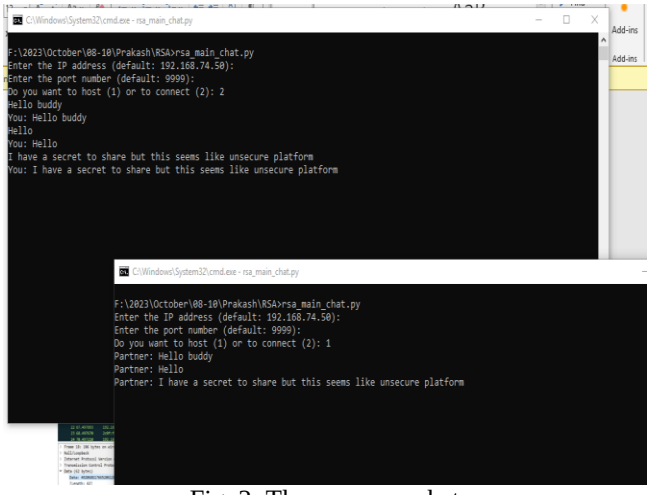

Fig. 2: The unsecure chat

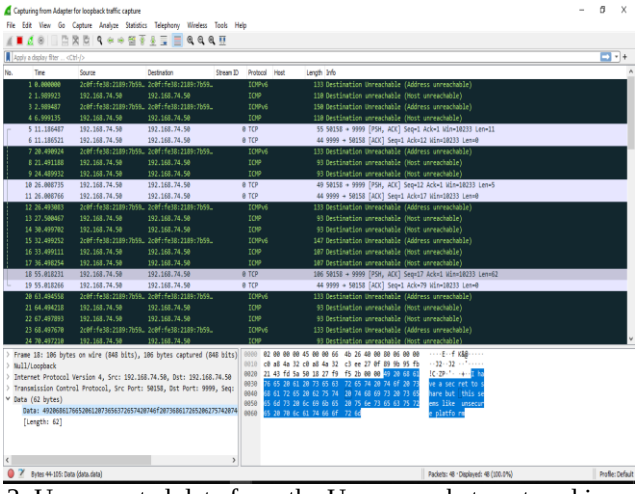

Fig. 3: Unencrypted data from the Unsecure chat captured in Wireshark

Fig. 3, displays a Wireshark packet analysis, capturing network traffic from the chat application shown in Fig. 2. It reveals the content of the messages sent over the network unencrypted, as evidenced by the readable excerpt from the data packet. As anticipated, in the absence of encryption, Wireshark easily captured and displayed the chat messages in plain text. This observation highlighted the vulnerabilities of an unencrypted communication system and underscored the significance of implementing robust encryption mechanisms.

5.2. Implementation of End-to-End Encryption

After observing the vulnerabilities in the unencrypted chat, we proceeded to evaluate the efficiency of our RSA and AES encrypted chat mechanism. Fig. 4 shown below represents the secure encrypted chat box which is implemented through the cryptographic mechanism.

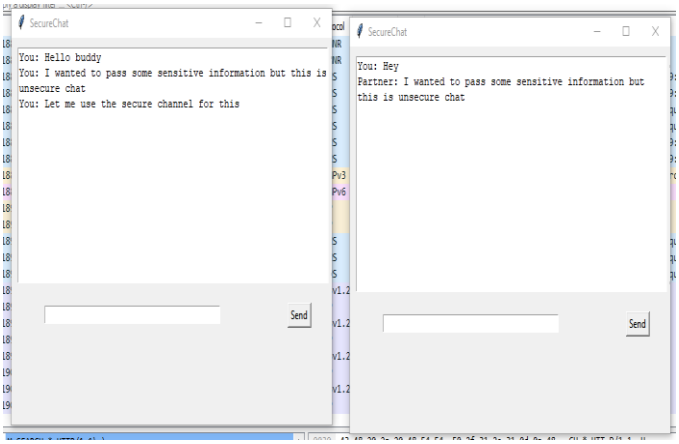

Fig. 4: Sending messages through an encrypted chat.

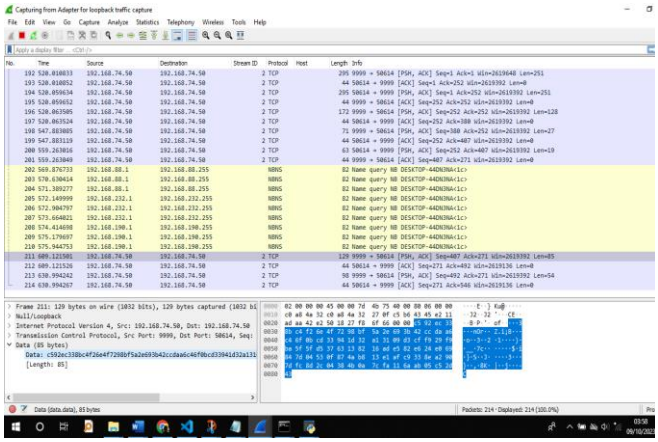

Fig. 5: The data captured by Wireshark is encrypted

Fig. 5, represents a snapshot of Wireshark captures network traffic from the secure chat application, displaying an obscured view of the data, indicative of successful encryption. The hex values suggest the data is no longer in plain text, verifying the effectiveness of the implemented end-to-end encryption.

When the end-to-end encryption was activated, Wireshark could no longer decipher the actual contents of the chat messages. Instead of plain text messages, Wireshark now captured encrypted data packets, showcasing the efficiency of our encryption methodologies.

5.3. Persistent Chat History & Security

One of our distinctive features was encrypting the chat history.

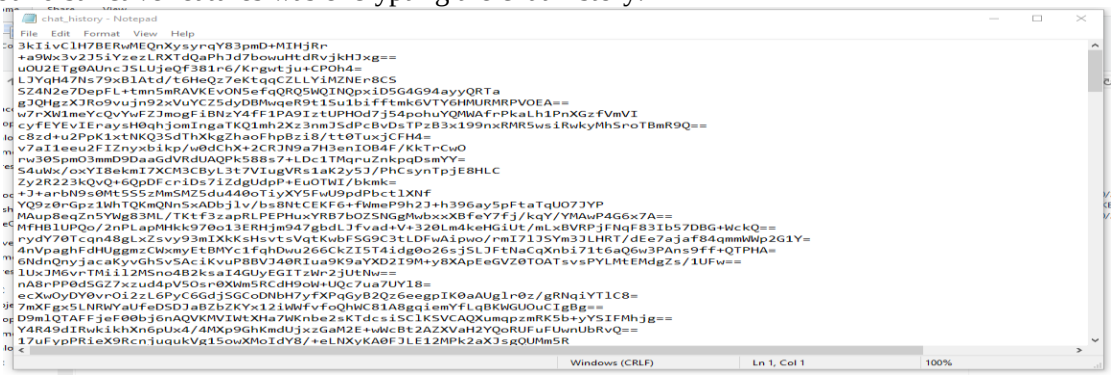

Fig. 6: Encrypted Chat History

Fig. 6, shows an image displaying the contents of a 'chat_history.txt' file opened in a Notepad application. The text shown consists of a long string of encrypted data, with no discernible words or phrases, only a series of alphanumeric characters and symbols that reflect the application of a secure encryption algorithm to the chat history. This ensures that without the corresponding decryption key, the information is unreadable, safeguarding user conversations against unauthorized access. When we accessed the **chat_history.txt** file, the messages were encrypted, ensuring an added layer of security. Even if unauthorized users accessed this file, the content remains unintelligible without the proper decryption key.

5.4. Flexibility of Server-Client Model

Our design permits any user to act as a server or client, a feature we believe would enhance flexibility and adaptability.

```
121
122     def run(self):
123         ip, port = self.get_ip_port()
124         choice = input("Do you want to host (1) or to connect (2): ")
125         if choice == "1":
126             self.setup_server(ip, port)
127         elif choice == "2":
128             self.setup_client(ip, port)
129         else:
130             print("Invalid Choice")
131             exit()
132         threading.Thread(target=self.sending_messages).start()
133         threading.Thread(target=self.receiving_messages).start()
134
```

Fig. 7: Python Encrypted Code

During testing, as shown in Fig. 7, this feature has proved beneficial, especially in scenarios where specific devices had restrictions preventing them from acting as servers. The ability to switch roles ensured uninterrupted communication.

5.5. Error Handling and Resilience

Exceptions are errors that Python reports when syntactically correct code executes, and an exceptional situation occurs. Normally, though, if you don't take the necessary precautions, an exception will cause your application to break. Sometimes, this is the desired behavior, but in other cases, we want to prevent and control problems such as these. Throughout our tests, we intentionally induced faults like abrupt client disconnections and invalid inputs to evaluate the system's robustness.

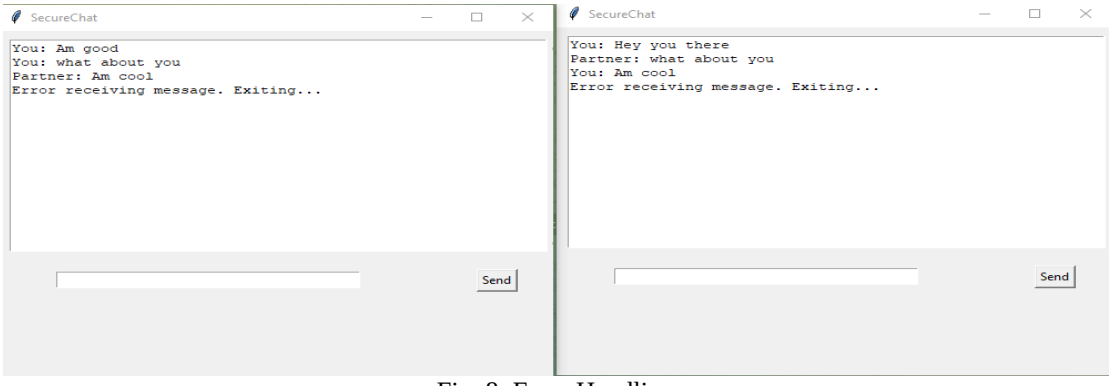

Fig. 8: Error Handling

In Fig. 8, the application demonstrated resilience, effectively handling most errors. It provided relevant feedback to users, such as "Partner has disconnected" or "Error receiving message. Exiting…", ensuring they were always informed about the chat's status.

6. Conclusion

The constant evolution of the digital sector underscores the urgency to prioritize data security, especially in real-time communication platforms. With the development of our project's application, we aimed to strike a balance between ensuring robust security and encrypting the user history messages. The combination of RSA for key exchanges and AES for message encryption has distinctly strengthened the security layer of our chat mechanism. However, as is the case with most technological projects, there is always an avenue for growth and enhancement. At present, our focus has been majorly centralized around text-based communication. Drawing inspiration from the transformation of modern chat platforms, which began as text-only platforms but eventually upgraded into multimedia communication hubs, we acknowledge the potential for further advancements in our application.

References

[1] Chatterjee, N., Chakraborty, S., Decosta, A., & Nath, A. (2018). Real-time Communication Application Based on Android Using Google Firebase. *International Journal of Advance Research in Computer Science and Management Studies*, 6(4). [Online] Available: www.ijarcsms.com.

[2] Sowah, R. A., Ofoli, A. R., Krakani, S. N., & Fiawoo, S. Y. (2017). Hardware design and web-based communication modules of a real-time multisensor fire detection and notification system using fuzzy logic. *IEEE Transactions on Industrial Applications*, 53(1), 559-566. https://doi.org/10.1109/TIA.2016.2613075.

[3] Anglano, C., Canonico, M., & Guazzone, M. (2016). Forensic analysis of the ChatSecure instant messaging application on android smartphones. *Digital Investigation*, 19, 44-59. https://doi.org/10.1016/j.diin.2016.10.001.

[4] Hegde, S., & Shah, S. (2015). A SURVEY ON THE LATEST WEB TECHNOLOGIES. [Online] Available: www.ijtra.com.

[5] G. Rovira Sánchez, "Implementation of a chat application for developers," 2017.

[6] A. Kumar and A. Singh, "Research paper on Group chatting Application." [Online]. Available: https://www.researchgate.net/publication/360483603. Accessed: 2023.

[7] Randall, N. (1997). Epilogue: The Soul of the Internet. In *The soul of internet: net gods, netizens, and the wiring of the world* (pp. 345-358). London: Computer Press.

[8] N. Sabah, J. M. Kadhim, and B. N. Dhannoon, "Developing an End-to-End Secure Chat Application," Nov. 2017. [Online]. Available: https://www.researchgate.net/publication/322509087_Developing_an_End-to-End_Secure_Chat_Application.

[9] J. Hughes, "LONG-TERM SECURITY VULNERABILITIES OF ENCRYPTED DATA," 2007.

[10] U. Tariq, I. Ahmed, A. K. Bashir, and K. Shaukat, "A Critical Cybersecurity Analysis and Future Research Directions for the Internet of Things: A Comprehensive Review," Sensors, vol. 23, no. 8, p. 4117, 2023. [Online]. Available: https://doi.org/10.3390/s23084117.