Secure Network Messenger Using SSH for Improved Anonymity

Dr.Neela Madheswari
dept. Of Cyber Sceurity
Mahendra Engineering College
Namakkal,India

Aloysius Rosario K
dept. Of Cyber Security
Mahendra Engineering College
Namakkal,India

Arunkumar M
dept. Of Cyber Security
Mahendra Engineering College
Namakkal, India

Aasif Hameed S
dept.of Cyber Security
Mahendra Engineering College
Namakkal,India

Anbarasu P
dept.of Cyber Security
Mahendra Engineering College
Namakkal,India

***Abstract*—Secure communication systems are essential in modern digital environments where privacy and confidentiality are critical. This paper presents a secure end to end network messenger that utilizes SSH based authentication and RSA key exchange for secure identity verification. After authentication, symmetric encryption algorithms including AES, TripleDES, RC4, Blowfish, and ChaCha20 are implemented for message confidentiality. The system also benchmarks the average encryption and decryption time for each algorithm to evaluate performance tradeoffs. Experimental results demonstrate that modern algorithms such as AES and ChaCha20 provide better performance and security compared to legacy algorithms. The proposed system ensures secure communication while allowing comparative analysis of symmetric encryption efficiency.**

Keywords—Secure messaging, SSH authentication, RSA, AES, Symmetric encryption, Encryption benchmarking, Network security

I. Introduction

Secure communication over networks is a fundamental requirement in distributed systems. Traditional messaging systems often rely on centralized encryption mechanisms that may expose vulnerabilities. End to end encryption ensures that only communicating users can access the content of messages.

This paper proposes a secure network messenger built using SSH based authentication through the Paramiko library. RSA is used for secure key exchange, and multiple symmetric encryption algorithms are implemented to encrypt message payloads. The system also evaluates encryption performance by measuring average encryption and decryption times for different symmetric algorithms.

The primary goal of this work is to analyze security strength and computational performance of commonly used symmetric encryption algorithms within a real time messaging system.

II. System Architecture

The proposed system consists of three major components:

1. Authentication and Session Management Module
2. Cryptographic Processing Module
3. Secure Communication Module

The server authenticates users using SSH password authentication. After successful authentication, public keys are exchanged between clients for secure RSA encryption of symmetric session keys. Messages are then encrypted using the selected symmetric algorithm before transmission.

III. Cryptographic Design

A. Asymmetric Encryption

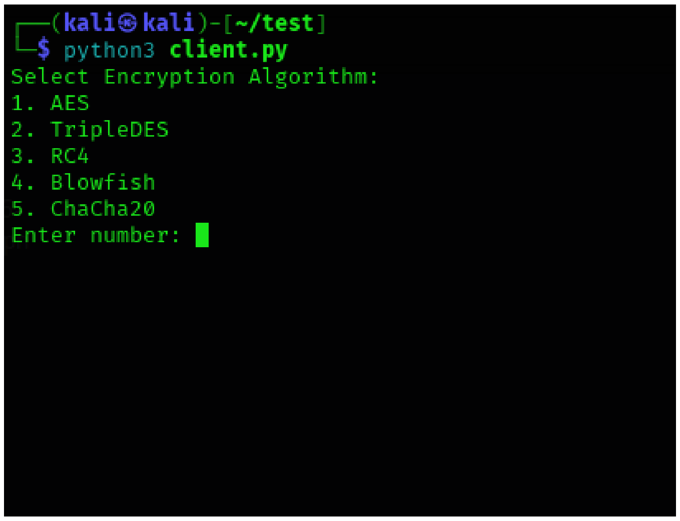
RSA with 2048 bit keys is used for secure key exchange. Each client generates a public private key pair. The public key is shared via the server, while the private key remains confidential.

B. Symmetric Encryption Algorithms

The system supports the following symmetric algorithms:

1. AES
2. Triple DES
3. RC4
4. BlowFish
5. ChaCha20

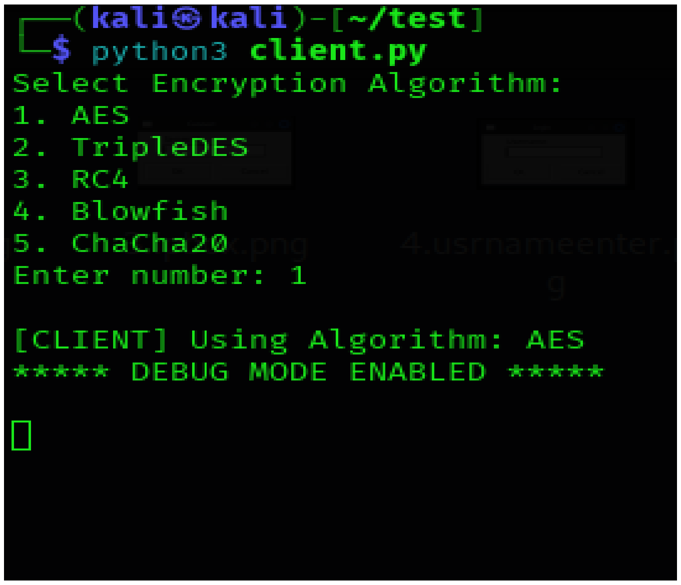
AES and ChaCha20 are modern secure algorithms. TripleDES and Blowfish are legacy algorithms included for benchmarking comparison. RC4 is included for performance comparison though it is considered insecure for modern systems.



```
┌──(kali㉿kali)-[~/test]
└─$ python3 client.py
Select Encryption Algorithm:
1. AES
2. TripleDES
3. RC4
4. Blowfish
5. ChaCha20
Enter number: 
```

Fig.1. Algorithm Selection Options

We can enter the alloted integer to select an algorithm to use for the communication



```
┌──(kali㉿kali)-[~/test]
└─$ python3 client.py
Select Encryption Algorithm:
1. AES
2. TripleDES
3. RC4
4. Blowfish
5. ChaCha20
Enter number: 1

[CLIENT] Using Algorithm: AES
***** DEBUG MODE ENABLED *****
```

Fig.2. After Selecting the Desired Algorithm

Messages are encrypted using Cipher Feedback mode where applicable. For ChaCha20, stream cipher encryption is applied.

IV. User Authentication And Key Generation

When a client starts, the following steps are performed:

1. The user provides credentials (username and password).

2. The client generates an RSA public–private key pair.

3. The public key is sent to the server and stored for future communication.

4. The private key remains securely stored on the client side.

The server maintains a mapping between usernames and their corresponding public keys. This enables secure key exchange between users.

A. Hybrid Encryption Mechanism

The messaging process uses a hybrid encryption model consisting of the following steps:

1. Symmetric Key Generation
For every message, a random symmetric session key is generated.
2. Message Encryption
The plaintext message is encrypted using the selected symmetric algorithm (e.g., AES, ChaCha20, or Blowfish). Symmetric encryption is chosen for its computational efficiency.
3. Key Encryption Using RSA
The generated symmetric session key is encrypted using the receiver's RSA public key. This ensures that only the intended recipient can decrypt the session key.
4. Secure Transmission
The encrypted message and encrypted session key are transmitted together to the server, which forwards them to the intended recipient.
5. Decryption Process
Upon receiving the message the recipient decrypts the symmetric session key using their RSA private key.The decrypted symmetric key is then used to decrypt the actual message.This approach ensures confidentiality while maintaining performance efficiency.

B. Abbreviations and Acronyms

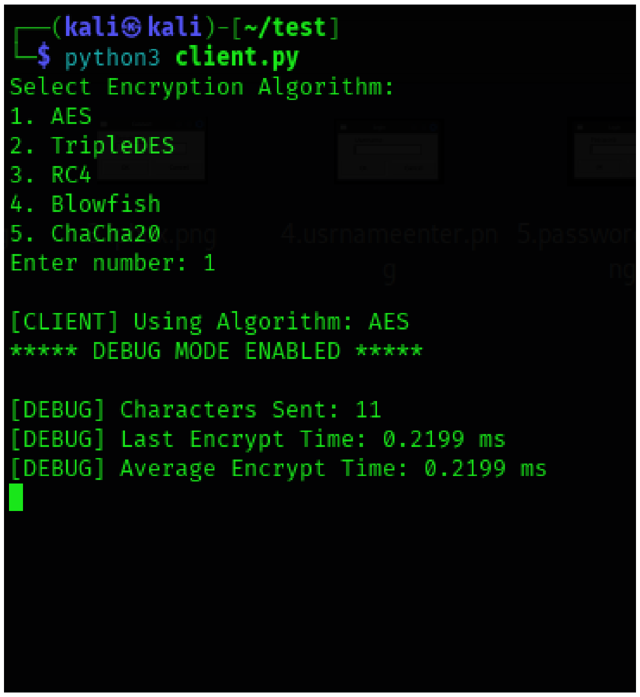
The following abbreviations and acronyms are used throughout this paper:

1. E2EE: End-to-End Encrypted
2. SSH : Secure Shell
3. RSA : Rivest-Shamir-Aldman public key cryptographic algorithm
4. AES : Advance Encryption Algorithm
5. CBC : Cyber Block Chaining
6. GUI : Graphical User Interface
7. TCP : Transmission Control Protocol
8. DES : Data Encryption Standard

All acronyms are defined at their first occurrence in the text to maintain clarity and consistency with IEEE formatting standards.

C. Units

- When calculating encryption time and decryption time, the standard unit used is **seconds (s)**, which is the SI unit of time. However, because cryptographic operations usually complete very quickly, the measurements are often expressed in smaller units such as **milliseconds (ms)** or **microseconds (µs)**. One millisecond is equal to 10^{-3} seconds, and one microsecond is equal to 10^{-6} seconds. For lightweight symmetric algorithms like AES, ChaCha20, Blowfish, or Twofish, encryption and decryption times are typically measured in milliseconds or even microseconds, especially when processing small messages.

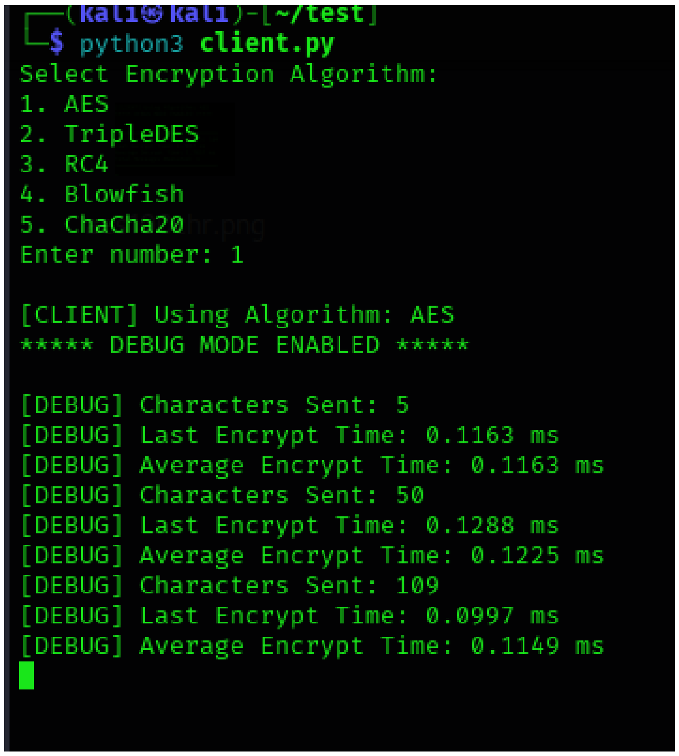


```
┌──(kali㉿kali)-[~/test]
└─$ python3 client.py
Select Encryption Algorithm:
1. AES
2. TripleDES
3. RC4
4. Blowfish
5. ChaCha20
Enter number: 1

[CLIENT] Using Algorithm: AES
***** DEBUG MODE ENABLED *****

[DEBUG] Characters Sent: 11
[DEBUG] Last Encrypt Time: 0.2199 ms
[DEBUG] Average Encrypt Time: 0.2199 ms
```

Fig.3. Encryption data

- In practical benchmarking , time is measured using high-resolution timers provided by programming languages. For example, in Python, functions such as time.perf_counter() or time.time() return values inseconds as floating-point numbers. Even though the base unit returned is seconds, the result is usuallymultiplied by 1000 to convert it into milliseconds for clearer comparison.

- For accurate benchmarking, especially in cryptographic performance analysis, multiple runs are performed and the **average time** is calculated. This helps reduce measurement noise caused by system load, CPU scheduling, and background processes. The final reported value is typically the mean encryption or decryption time per message, expressed in seconds, milliseconds, or microseconds depending on the scale of the results.



```
┌──(kali㉿kali)-[~/test]
└─$ python3 client.py
Select Encryption Algorithm:
1. AES
2. TripleDES
3. RC4
4. Blowfish
5. ChaCha20
Enter number: 1

[CLIENT] Using Algorithm: AES
***** DEBUG MODE ENABLED *****

[DEBUG] Characters Sent: 5
[DEBUG] Last Encrypt Time: 0.1163 ms
[DEBUG] Average Encrypt Time: 0.1163 ms
[DEBUG] Characters Sent: 50
[DEBUG] Last Encrypt Time: 0.1288 ms
[DEBUG] Average Encrypt Time: 0.1225 ms
[DEBUG] Characters Sent: 109
[DEBUG] Last Encrypt Time: 0.0997 ms
[DEBUG] Average Encrypt Time: 0.1149 ms
```

Fig. 4. Time units are mentioned in ms.

D. Equations

The equations are an exception to the prescribed specifications of this template. You will need to determine whether or not your equation should be typed using either the Times New Roman or the Symbol font (please no other font). To create multileveled equations, it may be necessary to treat the equation as a graphic and insert it into the text after your paper is styled.

Number equations consecutively. Equation numbers, within parentheses, are to position flush right, as in (1), using a right tab stop. To make your equations more compact, you may use the solidus (/), the exp function, or appropriate exponents. Italicize Roman symbols for quantities and variables, but not Greek symbols. Use a long dash rather than a hyphen for a minus sign. Punctuate equations with commas or periods when they are part of a sentence, as in:

1. Encryption Time:

$$Tenc=tend-tstart$$

Where:

· tstartt_{start}tstart = Time before encryption

· tendt_{end}tend = Time after encryption

2. Decryption Time:

$$Tdec=tend-tstart$$

3. Average Encryption Time:

If encryption is performed n times:

Tavg= n∑i=1nTi/n

Where:

· Ti = Time taken for each encryption

· n = Total number of run

Measuring both encryption and decryption time ensures that the proposed system not only maintains strong security through end-to-end encryption but also achieves acceptable performance levels suitable for real-time messaging applications.

E. Graphical User Interface

The proposed secure messaging system features a structured and user-centric graphical user interface (GUI) developed using the Tkinter framework in Python. The interface is designed to provide a modern messaging experience while seamlessly integrating end-to-end encryption mechanisms in the background. The overall layout follows a dual-panel architecture, ensuring clarity, ease of navigation, and efficient interaction.

The left panel of the interface is dedicated to contact management. It dynamically displays the list of authenticated users whose public keys have been successfully exchanged through the secure server. Users can select a contact from this panel to initiate or continue an encrypted conversation. This design enables organized communication and mimics widely adopted messaging application structures, improving user familiarity and reducing the learning curve.

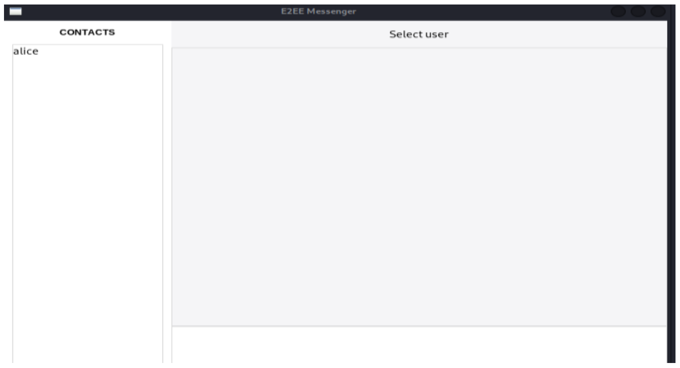


Fig.5. Default Graphical User Interface

The right panel functions as the primary chat window. Messages are displayed in a conversational format with visual differentiation between sent and received messages. Outgoing messages are right-aligned with a distinct background color, while incoming messages are left-aligned with a contrasting style. This alignment strategy enhances readability and provides clear visual feedback regarding message direction. A header label at the top of the chat window indicates the currently selected secure contact, confirming the active encrypted session.

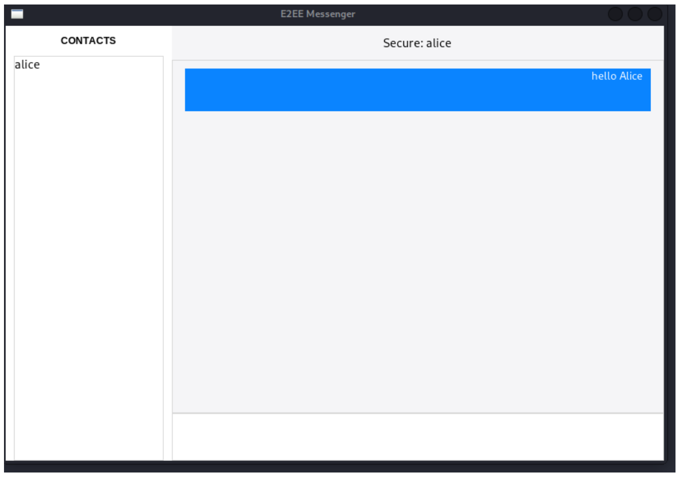


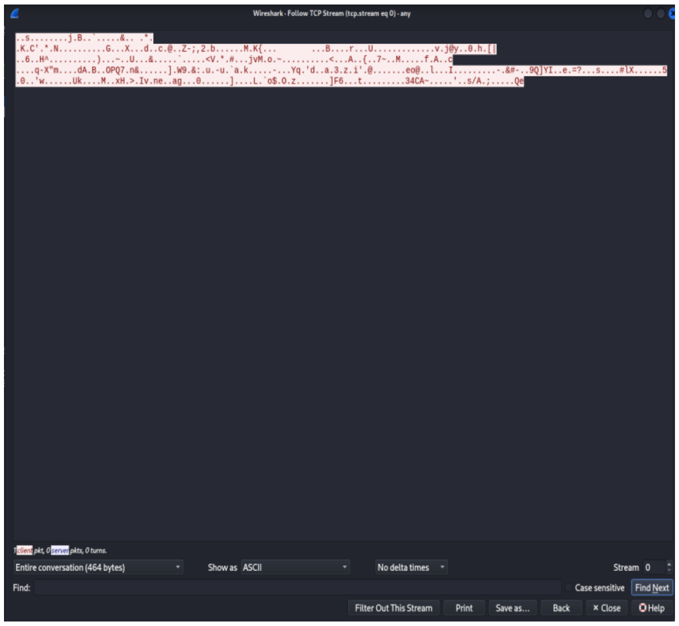
Fig.6.Graphical User Interface While Active Messaging

The bottom section of the interface contains a multi-line text input field and a send button. The multi-line input area allows users to compose detailed messages, and upon transmission, the message undergoes encryption before being sent over the network. Importantly, cryptographic debugging information such as encryption time, decryption time, and character statistics is logged exclusively in the terminal environment. This separation ensures that performance analysis does not interfere with the clean and professional appearance of the messaging interface.

Overall, the GUI design successfully integrates security, usability, and performance monitoring while maintaining a clear and intuitive user experience.

F. Network-Level Security Validation

To validate the security of the proposed end-to-end encrypted messenger at the network level, packet inspection was performed using Wireshark during live communication between two authenticated clients. Traffic was captured on the server communication port while multiple encrypted messages were exchanged. The captured packets were analyzed in both summary and detailed payload views. The results showed that all transmitted data appeared as encrypted SSH protocol segments, with no readable plaintext information such as usernames, message content, or cryptographic keys exposed in the network stream.


Wireshark · Follow TCP Stream (tcp.stream eq 0) · any

```
..s........j.B..`.....&.. .*.
.K.C'.*.N..........G...X...d..c.@..Z-;,2.b......M.K{...        ...B....r...U.............v.j@y..0.h.[|
..6..H^...........)...~..U...&.....`.....<V.*.#...jvM.o.~..........<...A..{..7~..M.....f.A..c
....q-X"m....dA.B..OPQ7.n&......].W9.&:.u.-u.`a.k.....-...Yq.'d..a.3.z.i'.@.......eo@..l...I.........-.&#-..9Q]YI..e.=?...s....#lX......5
.0..'w......Uk....M..xH.>.Iv.ne..ag...0......]....L.`o$.O.z.......]F6...t.........34CA~.....'..s/A.;.....Qe
```

1 client pkt, 0 server pkts, 0 turns.

Entire conversation (464 bytes) — Show as: ASCII — No delta times — Stream 0

Fig.6.Wire Shark Inspection

Further inspection of the packet payload confirmed that the transmitted data consisted entirely of ciphertext generated by the Secure Shell (SSH) transport layer combined with the application-level encryption mechanism implemented in the system. Even under deep packet inspection, the message contents remained unintelligible. This demonstrates that the proposed architecture provides strong protection against packet sniffing, eavesdropping, and unauthorized interception. The results confirm that both transport-layer encryption and application-layer end-to-end encryption work together to ensure comprehensive network-level security.

G. *Tables*

The following table presents the experimentally measured average encryption and decryption times for different symmetric cryptographic algorithms implemented in the proposed secure messaging system. The benchmarking was conducted using two message sizes, namely 50 characters and 100 characters, to evaluate the computational efficiency and scalability of each algorithm. The results are reported in milliseconds (ms), representing the average time required to complete the respective cryptographic operation across multiple iterations. This comparison enables a clear performance evaluation of AES, Triple DES, RC4, Blowfish, and ChaCha20 under identical system conditions.

TABLE I. ENCRYPTION/DECRYPTION TIME BENCHMARK

S.no	**Average Encryption Time**		
	Table column subhead	***50 chars***	***100chars***
1	AES	0.1288ms	0.0997ms
2	Triple DES	0.0852ms	0.1233ms
3	RC4	0.1235ms	0.1262ms
4	BlowFish	0.1768ms	0.1805ms
5	ChaCha20	0.1393ms	0.1633ms
S.no	**Average Decryption Time**		
	Table column subhead	***50 chars***	***100chars***
1	AES	2.3237ms	1.7857ms
2	Triple DES	1.9717ms	2.1457ms
3	RC4	1.8650ms	1.9699ms
4	BlowFish	1.8540ms	1.7701ms
5	ChaCha20	2.6888ms	3.7317ms

Based on the experimental results obtained from the benchmarking analysis, a comparative evaluation of five symmetric encryption algorithms—AES, Triple DES, RC4, Blowfish, and ChaCha20—was performed using average encryption and decryption times for 50-character and 100-character message sizes. The results indicate that Triple DES demonstrated the lowest average encryption time for 50-character inputs (0.0852 ms), while AES achieved a slightly lower encryption time for 100-character inputs (0.0997 ms). However, when decryption performance is considered, Blowfish and RC4 exhibited comparatively lower average decryption times across both message sizes, with Blowfish showing the most consistent performance.

Although Triple DES performs efficiently in encryption for smaller inputs, its decryption time is relatively higher compared to RC4 and Blowfish. AES, being a widely accepted industry standard, demonstrates balanced performance with stable encryption and decryption timings, making it suitable for secure real-world applications. ChaCha20, while secure and modern, showed comparatively higher decryption times in the conducted tests.

Considering both security strength and computational performance, AES emerges as the most optimal and practically suitable algorithm for secure messaging systems, as it offers strong cryptographic security with competitive processing times. However, if the primary focus is purely on computational speed in this specific experimental setup, Blowfish and RC4 show slightly faster decryption performance. From a practical and security-oriented perspective, AES remains the best overall choice for deployment in secure end-to-end encrypted communication systems.

CONCLUSION AND FUTURE SCOPE

This paper presented the design and implementation of a secure end-to-end encrypted messenger system built using Secure Shell (SSH) for transport-layer protection and hybrid cryptography for application-layer security. The system integrates RSA for secure key exchange and multiple symmetric encryption algorithms, including AES, TripleDES, RC4, Blowfish, and ChaCha20, for message confidentiality.

Performance benchmarking was conducted to evaluate encryption and decryption times across different algorithms and message sizes. The experimental results demonstrate that AES provides a balanced combination of strong security and efficient performance, while other algorithms exhibit varying trade-offs between speed and computational cost. Network-level validation using packet analysis further confirmed that all transmitted data remains encrypted and resistant to interception. Overall, the proposed system successfully achieves secure, efficient, and scalable encrypted communication.

Future enhancements can focus on improving both security robustness and system scalability. Advanced features such as Perfect Forward Secrecy (PFS), digital signatures for message authentication, and key rotation mechanisms can be integrated to strengthen cryptographic resilience. The implementation of secure group messaging, file transfer encryption, and multimedia message protection would expand system capabilities. Additionally, migrating from a desktop-based architecture to a cross-platform or mobile-based deployment would improve usability and accessibility. Performance optimization through hardware-accelerated cryptography and large-scale stress testing can further enhance real-world applicability. These improvements would transform the current prototype into a production-grade secure communication platform suitable for broader deployment

Acknowledgment

The authors would like to thank the faculty members of the Department of CyberSecurity for their valuable guidance and technical support throughout the development of this project. Special thanks are extended to the project supervisor for continuous encouragement and constructive feedback.

References

[1] A Comprehensive End-to-End Solution for Web Security with Cryptography, Multi-Factor Authentication, and Secure Communication Kanderi Johith Kumar1 , Kandlapalli Aravind Sai2 , Adapala Dharshan Reddy3 , Chinthamreddy Pranay Daiwik Reddy4 , Shinu M Rajagopal* Dept. of Computer Science and Engineering, Amrita School of Computing, Bengaluru,
Amrita Vishwa Vidyapeetham, India
[2] V. Meyers, D. Gnad and M. Tahoori, ”Active and Passive Physical
Attacks on Neural Network Accelerators,” in IEEE Design & Test, vol.
40, no. 5, pp. 70-85, Oct. 2023, doi: 10.1109/MDAT.2023.3253603.
[3] E. V. Osipova and N. G. Butakova, ”Development of a Secure Messen
ger Based on the ECMQV Algorithm, Immune to Man-in-the-Middle
Attacks,” 2024 Conference of Young Researchers in Electrical and
Electronic Engineering (ElCon), Saint Petersburg, Russian Federation,
2024, pp. 264-267, doi: 10.1109/ElCon61730.2024.10468485.
[4] Rojali, Z. E. Rasjid and J. C. Matthew, ”Implementation of Rail
Fence Cipher and Myszkowski Algorithms and Secure Hash Algorithm
(SHA-256) for Security and Detecting Digital Image Originality,” 2022
International Conference on Informatics, Multimedia, Cyber and Infor
mation System (ICIMCIS), Jakarta, Indonesia, 2022, pp. 207-212, doi:
10.1109/ICIMCIS56303.2022.10017975.
[5] R. Shaw and S. Parveen, ”Literature Review on Packet Sniffing: Es
sential for Cybersecurity & Network Security,” 2024 5th International
Conference on Intelligent Communication Technologies and Virtual
Mobile Networks (ICICV), Tirunelveli, India, 2024, pp. 715-719, doi:
10.1109/ICICV62344.2024.00119.
[6] A. B. M. Sultan, S. Mehmood and H. Zahid, ”Man in the Middle
Attack Detection for MQTT based IoT devices using different Machine
Learning Algorithms,” 2022 2nd International Conference on Artificial Intelligence (ICAI), Islamabad, Pakistan, 2022, pp. 118-121, doi:
10.1109/ICAI55435.2022.9773590.
[7] M. L. Ali, S. Ismat, K. Thakur, A. Kamruzzaman, Z. Lue and
H. N. Thakur, ”Network Packet Sniffing and Defense,” 2023 IEEE
13th Annual Computing and Communication Workshop and Con
ference (CCWC), Las Vegas, NV, USA, 2023, pp. 0499-0503, doi:
10.1109/CCWC57344.2023.10099148.
[8] K. Zhang, C. Keliris, T. Parisini, B. Jiang and M. M. Polycarpou,
”Passive Attack Detection for a Class of Stealthy Intermittent Integrity
Attacks,” in IEEE/CAA Journal of Automatica Sinica, vol. 10, no. 4,
pp. 898-915, April 2023, doi: 10.1109/JAS.2023.123177.
[9] Y. Vasiliu, T. Okhrimenko, A. Fesenko and S. Dorozhynskyy, ”Passive
Eavesdropping Attack of Several Intruders on Deterministic Protocol
with Pairs of Entangled Qubits,” 2021 11th IEEE International Confer
ence on Intelligent Data Acquisition and Advanced Computing Systems:
Technology and Applications (IDAACS), Cracow, Poland, 2021, pp.
1068-1072, doi: 10.1109/IDAACS53288.2021.9660851.
[10] Z. Wu et al., ”Proactive Eavesdropping Performance for Integrated

Satellite–Terrestrial Relay Networks,” in IEEE Open Journal of the
Communications Society, vol. 4, pp. 2985-2999, 2023, doi: 10.1109/OJ
COMS.2023.3326340.
[11] G. Hu, J. Si, Y. Cai and N. Al-Dhahir, ”Proactive Eavesdropping
via Jamming Over Multiple Suspicious Links With Wireless-Powered
Monitor,” in IEEE Signal Processing Letters, vol. 29, pp. 354-358, 2022,
doi: 10.1109/LSP.2021.3132585.
[12] S. Liu, Y. Li and Z. Jin, ”Research on Enhanced AES Algorithm Based
on Key Operations,” 2023 IEEE 5th International Conference on Civil
Aviation Safety and Information Technology (ICCASIT), Dali, China,
2023, pp. 318-322, doi: 10.1109/ICCASIT58768.2023.10351719.
[13] A.M. Aburbeian and M. Fernandez-Veiga, ”Secure Internet Financial ´
Transactions: A Framework Integrating Multi-Factor Authentication
and Machine Learning,” AI 2024, vol. 5, pp. 177–194, 2024, doi:
10.3390/ai5010010.
[14] A. Kumar Verma, M. Rakhra, A. Bhattacherjee, A. Maan, T. Sarkar,
and V. Kumar Pandey, ”A Suggested Model for Using Multi
Factor Authentication Framework in Cloud Computing for SME,”
2024 International Conference on Cybernation and Computation (CY
BERCOM), Phagwara, India, 2024, pp. 1-6, doi: 10.1109/CYBER
COM63168.2024.10582681.
[15] D. D. Kumar, J. D. Mukharzee, C. V. D. Reddy, and S. M. Rajagopal,
”Safe and Secure Communication Using SSL/TLS,” 2024 International
Conference on Emerging Smart Computing and Informatics (ESCI),
Pune, India, 2024, pp. 1-6, doi: 10.1109/ESCI51234.2024.10580678.
[16] F. Bozkurt, M. Kara, M. A. Aydın, and H. H. Balik, ”Exploring the Vul
nerabilities and Countermeasures of SSL/TLS Protocols in Secure Data
Transmission Over Computer Networks,” 2023 12th IEEE International
Conference on Intelligent Data Acquisition and Advanced Computing
Systems: Technology and Applications, Dortmund, Germany, 2023, pp.SSS
1-6, doi: 10.1109/IDAACS52323.2023.10348784.