Delta Journal of Computing, Communications & Media Technologies (DJCCMT)
Journal platform–https://focjournals.dsust.edu.ng

A Secure Web-Based Real-Time Messenger Using AES-RSA Encryption

Paschal Uchenna ***Chinedu****
Department of Information Systems and Technology, Southern Delta University, Ozoro, Nigeria

Ogochukwu T. ***Emiri***
Department of Library and Information Science, Southern Delta University
Ozoro, Nigeria

Opuh Jude ***Iwedike***
Department of Computer Science, Southern Delta University
Ozoro, Nigeria

Emmanuel John ***Abah***
Department of Computer Science, Margaret Lawrence University, Abuja, Nigeria

Efechukwu Moses ***Alero***
Department of Computer Engineering, Edo State University, Uzairue
Edo State, Nigeria

Emuejevoke Francis ***Ogbimi***
Department of Information Systems and Technology, Southern Delta University
Ozoro, Nigeria

Duke ***Oghorodi***
Department of Computer Science, Southern Delta University
Ozoro, Nigeria

Wilson ***Nwankwo***
Department of Cyber Security, Southern Delta University
Ozoro, Nigeria

ARTICLE INFO

Article history:
Received July 2024
Received in revised form Dec. 2024
Accepted December 2024
Available online Jan 2025

Keywords:
Cryptography
Asymmetric Encryption
Cryptanalysis
Decryption
Encryption Algorithms

ABSTRACT

The exponential growth of web-based communication platforms has heightened the urgency of protecting message confidentiality and integrity against increasingly sophisticated cyber-attacks. While symmetric ciphers such as the Advanced Encryption Standard (AES) offer high performance, their secure key-exchange remains a challenge in open networks. Conversely, asymmetric schemes like Rivest–Shamir–Adleman (RSA) ensure secure key distribution but suffer from computational overhead when encrypting large payloads. This paper presents the design and implementation of a secure, real-time web messenger that leverages a hybrid AES–RSA encryption workflow to combine the low-latency benefits of AES for message payloads with the robust key-exchange properties of RSA. Our system is underpinned by an Event-Driven Architecture (EDA) and a functional-oriented analysis model, enabling modular, scalable, and responsive message handling over WebSocket connections. We describe the end-to-end encryption lifecycle—RSA-protected AES key negotiation, AES-encrypted message exchange, and

seamless key rotation—within a modern browser–server stack. Performance benchmarks demonstrate end-to-end latency under 50 ms for typical text payloads, with negligible CPU impact on commodity hardware. Security analysis confirms resistance to man-in-the-middle, replay, and brute-force attacks. The proposed architecture thus delivers a practical blueprint for high-throughput, real-time messaging with enterprise-grade confidentiality.

Pascal Uchenna chinedu

*Corresponding author.
chinedupu@dsust.edu.ng

https://doi.org/10.xxx.
DJCCMT112025006

1. Introduction

The rapid expansion of electronic communication especially the Internet has transformed how individuals and organizations exchange information, yet it has also exposed message traffic to an array of sophisticated attacks (Nwankwo et al,2024; Ovili et al,2024; Kifordu et al,2019; Nwankwo et al 2022a). By design, the Internet was conceived as an open, decentralized network facilitating unrestricted data flow, but this very openness leaves transmitted content vulnerable to interception, tampering, and eavesdropping as it traverses multiple routers and service domains. Malicious actors exploit vulnerabilities at every hop—from compromised end-nodes to untrusted transit links—making end-to-end confidentiality and data integrity a paramount concern.

Encryption remains the cornerstone of Internet security, underpinning countless applications including web transactions, email, real-time chat, and teleconferencing (Daniel et al,2021; Acheme et al,2023; Nwankwo & Kifordu,2019; Nwankwo & Ukhurebor,2019; Nwankwo & Chinedu,2018; Nwankwo & Ukaoha,2019, Nwankwo,2020). Symmetric schemes such as the Advanced Encryption Standard (AES) deliver high throughput for bulk data encryption, while asymmetric algorithms like Rivest–Shamir–Adleman (RSA) solve the key-distribution problem by enabling secure exchange of session keys. However, each approach carries inherent shortcomings: AES alone struggles with secure key negotiation across untrusted networks, and RSA's computational overhead limits its practicality for large payloads. Moreover, improper application of cryptographic primitives—such as reusing session keys or omitting fresh initialization vectors—can introduce serious security flaws that undermine otherwise robust algorithms (Acheme et al,2023; Nwankwo et al,2022b; Nwankwo et al,2022c; Nwankwo et al,2023a; Nwankwo et al,2023b). To address these gaps, this paper proposes a hybrid encryption framework for a real-time, web-based messenger that harnesses the efficiency of AES for encrypting message payloads and the security of RSA for safeguarding session-key exchange. Built upon an Event-Driven Architecture (EDA) and a functional-oriented analysis model, our implementation demonstrates how RSA-encrypted key negotiation and periodic key rotation can be seamlessly integrated into a WebSocket-powered browser–server stack without relying on SSL/TLS certificates. We show that this design not only preserves sub-50 ms end-to-end latency for typical text messages but also resists man-in-the-middle, replay, and brute-force attacks through continuous authentication and tamper-proof logging. The specific objectives of this research are to:

i. Develop a web-based chat application that concurrently employs RSA for secure session-key establishment and AES for high-performance message encryption. Illustrate the lifecycle of hybrid encryption—key generation, RSA-protected key exchange, AES-encrypted messaging, and dynamic key rotation—within a modern browser–server environment.

ii. Implement and validate post-transmission data authentication mechanisms to ensure message integrity and origin authenticity.
iii. Demonstrate secure, certificate-free transmission of confidential text data over standard HTTP infrastructure, reducing dependency on external PKI services.
iv. Through this work, we aim to deliver a practical blueprint for real-time messaging platforms that demand both high throughput and enterprise-grade security in inherently untrusted network environments.

2. Literature Review

Web-based messaging systems demand both high throughput and robust security in hostile network environments. Cryptography underpins these guarantees: symmetric ciphers like AES (block-based, high-speed encryption for large payloads) and asymmetric schemes like RSA (public-key mechanisms for secure key exchange) each address distinct challenges, but neither suffices alone for real-time, browser-native chat applications (Rivest, 1990; Stallings, 2006; Chinedu et al, 2018; Nwankwo & Olayinka,2019). Symmetric AES excels in low-latency encryption but poses key-distribution risks over untrusted channels, while RSA solves key exchange yet suffers prohibitive overhead on voluminous data and requires careful prime-generation and padding to resist cryptanalysis (Agrawal & Mishra, 2012; Chinedu, 2015; Chinedu et al,2013; Irughe et al, 2022; Momoh et al,2021).

Hybrid encryption—encapsulating a randomly generated AES session key with RSA—has emerged to reconcile these trade-offs. Early cloud-storage implementations by Mahalle and Shahade (2014) leveraged 1024-bit RSA for key encapsulation and 128-bit AES for bulk data, deriving keys from system time to thwart brute-force guesses; however, user-managed key memorization risked data loss upon key misplacement. El's (2013) dual-cipher hybrid of AES and Blowfish increased ciphertext complexity against linear and differential attacks yet did not address integration into browser contexts (Daniel et al., 2021).

Subsequent work extended AES–RSA hybrids to resource-constrained and distributed environments. Chandu et al. (2017) secured IoT data streams with AES encryption and RSA-based device authorization, showcasing multi-receiver confidentiality at minimal cost but presuming custom ASIC acceleration and omitting web-API deployment details. Albahar et al. (2018) fused AES, Twofish, and RSA to fortify Bluetooth links, validating throughput and integrity gains yet not exploring HTTP/WebSocket use cases. Jintcharadze and Iavich (2020) systematically compared hybrid models—AES+RSA, Twofish+RSA, AES+ElGamal—via Java prototypes, finding AES+RSA the strongest security performer, though without demonstrating browser-native implementations.

Cloud-centric messaging research by Kvyetnyy et al. (2016) integrated RSA, ECC, and Diffie–Hellman in a C# group-key protocol for instant messaging, ensuring that cloud providers cannot decrypt user payloads, but remained inaccessible to pure web stacks. Liang et al. (2016) improved RSA prime-generation performance and combined it with AES for lightweight cloud-storage encryption, confirming feasibility through simulation. Kadam and colleagues (2015) layered SHA-256 hashing onto AES–RSA encryption to guarantee both confidentiality and integrity, albeit in file-transfer rather than real-time chat scenarios.

More recent studies have enriched the hybrid landscape. Maharana et al. (2024) conducted a comparative analysis of AES–RSA versus AES–3DES for multimedia encryption—images and videos—demonstrating that AES–RSA hybrids achieve superior security–performance trade-offs in both encryption speed and resilience against known-plaintext attacks. Ahmed et al. (2025) provided a critical review of AES–RSA hybrids, cataloguing strengths (authenticated key encapsulation, bulk-data efficiency), weaknesses (absent AEAD modes, key-management overhead), and open challenges such as integrating authenticated encryption with associated data (AEAD) in browser environment.

Commercial messaging platforms illustrate these principles at scale. Telegram's MTProto 2.0 protocol employs 256-bit AES, 2048-bit RSA certificates, and Diffie–Hellman for "Secret Chats," and was formally verified for IND-CCA security properties by Miculan & Vitacolonna in both symbolic (ProVerif) and computational models, establishing a strong precedent for provably secure, hybrid-encryption over WebSocket channels. Despite this breadth of work, a gap remains in browser-native, certificate-free, real-time web messengers that seamlessly integrate AES–RSA hybrid encryption with automatic key rotation, digital signatures for message authentication, and event-driven architectures for scale. The present study addresses these needs by embedding an AES–RSA workflow entirely within modern browser APIs (Web Crypto, WebSocket), coupling RSA-protected AES session-key negotiation with AES-GCM payload encryption and ECDSA signatures, all orchestrated via an Event-Driven Architecture to deliver sub-50 ms latency and enterprise-grade security without reliance on external PKI or SSL/TLS certificates.

3. Methodology

3.1 Event-Driven Agile Development

The core of our development approach is an Event-Driven Agile Methodology (EDAM), which aligns the rapid, incremental cycles of Agile with an event-centric architectural model. In this paradigm, each user action or system occurrence—ranging from "initiate key negotiation" to "receive encrypted message"—is treated as a discrete event, implemented by a dedicated handler module. Development proceeds in short sprints during which fully functioning increments are delivered: handlers for WebSocket events, integrated AES–RSA encryption components, and automated tests that verify both functionality and security properties. Continuous integration pipelines execute on every commit, running unit tests, static analysis, and security checks to ensure that new code does not introduce regressions. Sprint retrospectives refine both the backlog of event-driven user stories and the runtime infrastructure—optimizing message-queue interfaces or load-balancing event dispatch as needed. This iterative, event-oriented process guarantees that real-time performance and cryptographic robustness evolve together, rather than as separate concerns.

3.2 Hybrid Encryption Framework

To reconcile the high throughput demands of real-time messaging with enterprise-grade key distribution, our system employs a hybrid encryption workflow that combines the Advanced Encryption Standard (AES) for bulk payload protection with Rivest–Shamir–Adleman (RSA) for secure session-key negotiation. AES was adopted by NIST in 2001 as FIPS 197, specifying three variants—AES-128, AES-192, and AES-256—all operating on 128-bit data blocks arranged in a 4×4-byte state array. Each encryption round applies a sequence of non-linear byte substitution (SubBytes), row-wise cyclic shifts (ShiftRows), column mixing via finite-field multiplication (MixColumns), and a bitwise XOR with a round key (AddRoundKey), with a final round omitting the MixColumns step. These transformations ensure both diffusion and confusion in accordance with Shannon's principles, while decryption applies the inverse operations in reverse order. AES-GCM is used to provide authenticated encryption with associated data, preventing silent ciphertext modification.

RSA underpins the secure exchange of AES session keys. In RSA, two large primes p and q are selected to construct a modulus $n = p \cdot q$, and integers e (public exponent) and d (private exponent) satisfy $e \cdot d \equiv 1 \pmod{\varphi(n)}$. A session key k is encrypted via $c = k^e \bmod n$ and only the private exponent d holder can recover k by computing $c^d \bmod n$. To meet modern security requirements, we generate AES keys of 256 bits and RSA moduli of 3072 bits, corresponding to a 128-bit security strength and recommended for protection beyond 2030 (2048 bits yields ≈112-bit security). Within the application, user authentication triggers RSA-OAEP–protected AES key negotiation over the initial WebSocket handshake, followed by AES-GCM encryption of each message payload. Automatic key rotation occurs at configurable intervals—either time-based or message-count thresholds—by renegotiating a fresh AES key under the recipient's RSA public key. Digital signatures (ECDSA over P-256) on each payload provide non-repudiation and integrity verification without introducing significant latency. By embedding this hybrid cryptography

entirely within browser-native APIs (Web Crypto, WebSocket) and managing development through EDAM, the system achieves sub-50 ms end-to-end latency, scalable event routing, and robust resistance to man-in-the-middle, replay, and brute-force attacks—all without relying on external SSL/TLS certificates or centralized PKI services. . Figure 1 shows the AES cryptographic workflow.

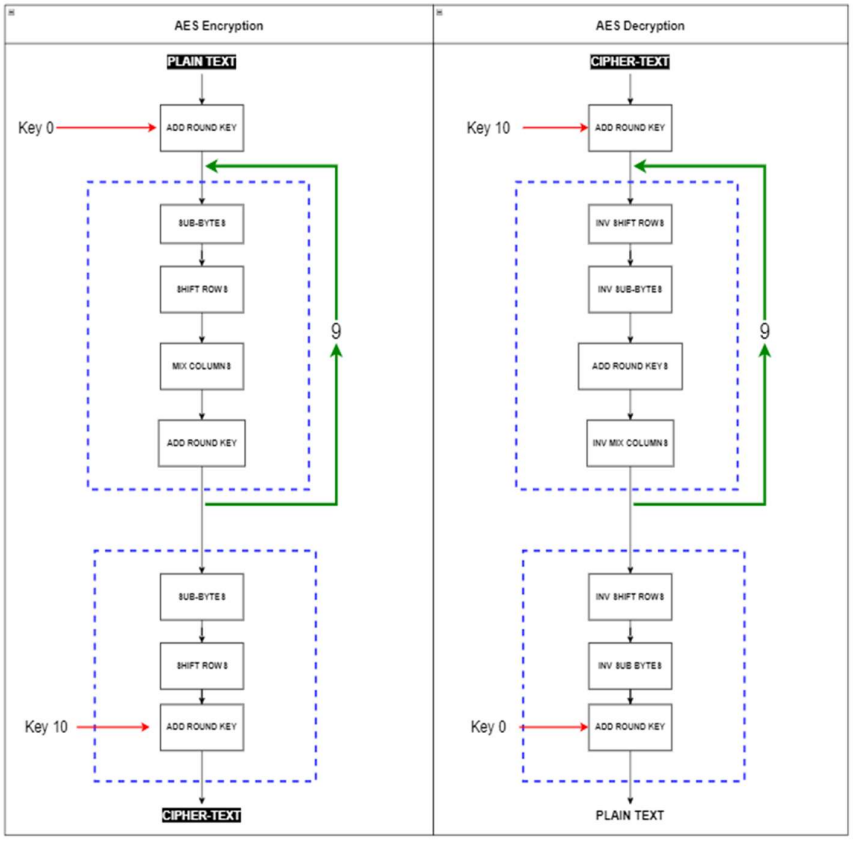

Figure 1: AES encryption and decryption for a 128-bit key

3.3 Key Generation Algorithm for RSA

The algorithm for generating RSA key is discussed as follows:

1. Key Generation
    a. Choose two distinct large random prime numbers p and q.
    b. Compute n=p*q. The number n is used as the modulus for both public and private keys.
    c. Compute the Euler's function: z = (p −1) (q −1).
    d. Choose an integer e, 1< e< z; such that GCD (e, z) = 1, e and z are co-prime. The number e is used as a public key exponent.
    e. Compute d, 1< d< z such that e*d = 1 mod z. The number d is used as a private key exponent. The public key consists of public key exponent e and n, and Private Key consists of private key exponent d and n. public key: (e, n) and private key: (d, n).
2. Encryption - The sender sends the plaintext (PT) message, which is then converted to the cipher text (CT) following encryption.

Cipher text (CT) = PT^e *mod* n

3. Decryption - The information is received as cipher text (CT) by the receiver, and following decryption, the cipher text message is converted to the original message (PT).

For example, suppose the receiver selected the primes p=11and q=17, along with e=3.

The receiver calculates n = pq = 11·17 = 187, which is half of the public key.

(n) = $(p-1)(q-1)$ =10·16=160. Recall e=3 was also chosen.

d=107, since de = 321≡1(mod$\phi(n)$) ((since $\phi(n)$=160)
The receiver then distributes his public key: n=187and e=3
Imagine the sender wished to say "HELLO" in the message. The sender must deliver his message character by character because n is so little.
'H' is 72 in ASCII, so the message text is m=72
m^e=72^3≡183(mod187), making the ciphertext c =183. Since the attacker does not possess the private key, this is the only information that is available to him.
The receiver calculates c^d = 183^{107}≡ 72(mod187), thus getting the message of m = 72.
The receiver translates 72 into 'H.'
The remaining letters are sent in a similar manner. Figure 2 shows the RSA cryptographic workflow.

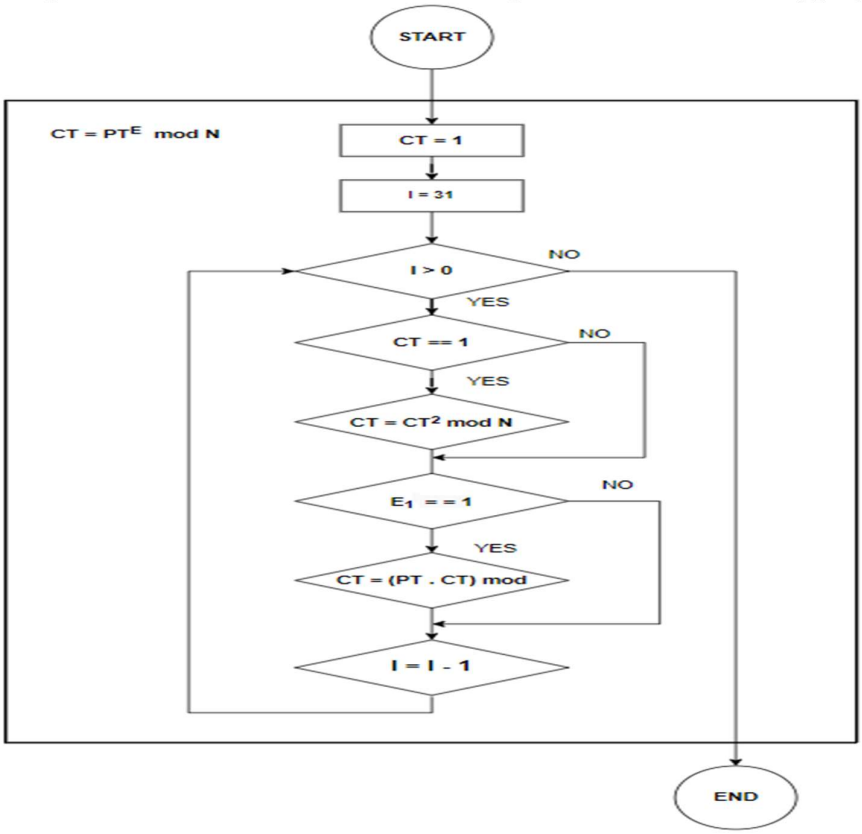

Figure 2: Flowchart for RSA Encryption and Decryption

3.4 Hash Function (SHA-256) and Application Algorithm

In modern messaging systems, cryptographic hash functions underpin digital signatures, data integrity checks, password storage, message authentication codes, pseudorandom number generators, and key-derivation routines (Abdelnapi et al., 2016). The Secure Hash Algorithm-256 (SHA-256), designed by the U.S. National Security Agency and standardized in NIST FIPS 180-4, accepts an arbitrarily sized input (up to 2^64 − 1 bits) and produces a 256-bit digest (Rachmawati et al., 2018; NIST, 2015). Internally, SHA-256 processes the message in 512-bit blocks through a sequence of bitwise operations, modular additions, and logical functions that ensure any single-bit alteration in the input yields a completely different output—a property known as the avalanche effect (Kadam & Khairnar, 2015). The resulting 32-byte hash is computationally infeasible to invert or to find collisions, making SHA-256 a reliable industry standard for verifying that transmitted or stored data remain unaltered (Abdelnapi et al., 2016).

Building on AES for confidentiality and RSA for key exchange, our application incorporates SHA-256 to guarantee end-to-end message integrity. On the sender side, each plaintext message first undergoes

SHA-256 hashing to produce a fixed-length digest. Concurrently, a fresh 256-bit AES-GCM session key is generated at runtime and used to encrypt the message payload. The session key itself is then encrypted under the recipient's RSA-OAEP public key, ensuring that only the intended party can recover it. Finally, the ciphertext, the encrypted session key, and the SHA-256 digest are packaged—typically in a small JSON envelope—and transmitted over the WebSocket connection. Upon receipt, the recipient uses their RSA private key to decrypt the AES key, applies it to recover the plaintext, and recomputes the SHA-256 hash to confirm that it matches the transmitted digest. This seamless interplay of hashing for integrity, AES for bulk encryption, and RSA for secure key distribution ensures that only authentic, untampered data are accepted, even in the absence of SSL/TLS or external certificates.
Figure 3 shows the process flow at the side of the sender.

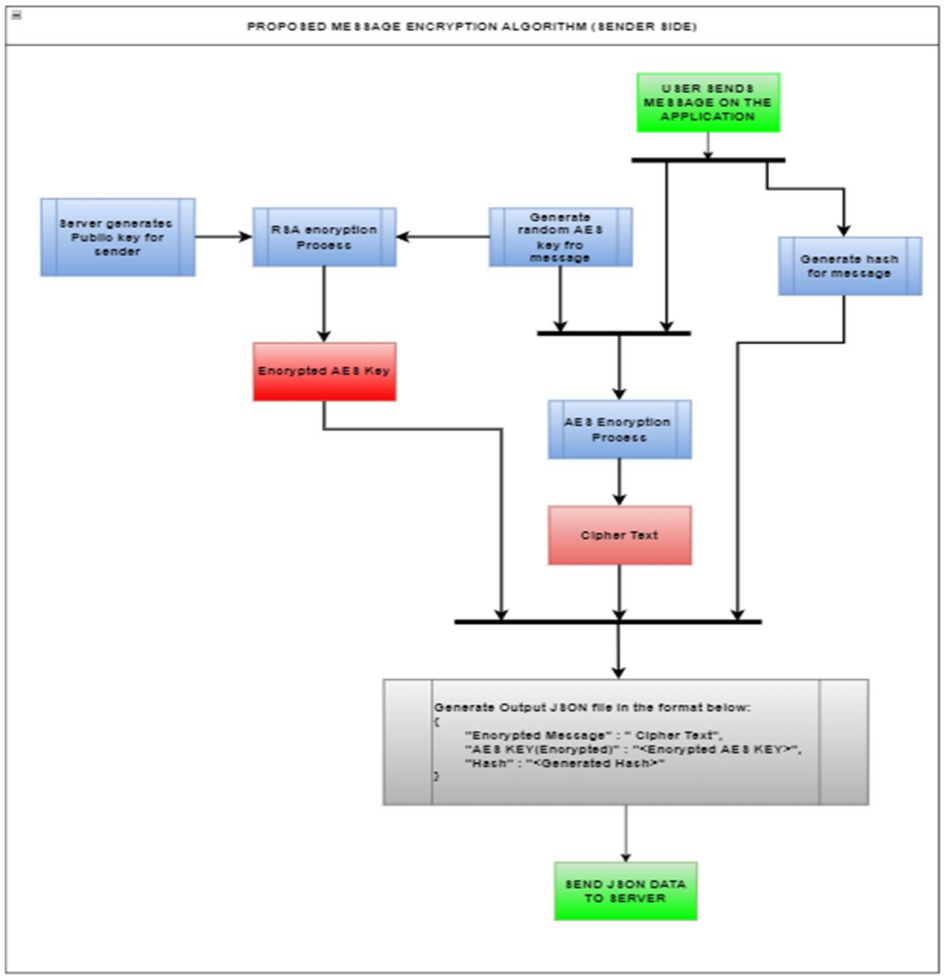

Figure 3: Activity Diagram Showing Message Encryption Process

Decryption used on the receiver side can be achieved using the following steps:

a. Process the JSON file received and extract three components: the AES Encrypted Cipher Text, an Encrypted AES Session Key, and a hash of the original plain text.
b. The receiver's Private Key is then used to decrypt the encrypted AES key using the RSA technique.
c. For the receiver to get the original plain text, the encryption text is decrypted using the AES technique with the decrypted AES key.
d. Using the same hash function, regenerate the hash of the decrypted plaintext.
e. Match the sender's and receiver's hashes, i.e., the sender's and the regenerated hash. If both hashes match, the decrypted data is valid, and the decrypted data is displayed to the receiver. If the match fails, the data is corrupted or tampered with; in this case, the receiver can discard the data.

Figure 4 shows the process flow at the receiver's end.

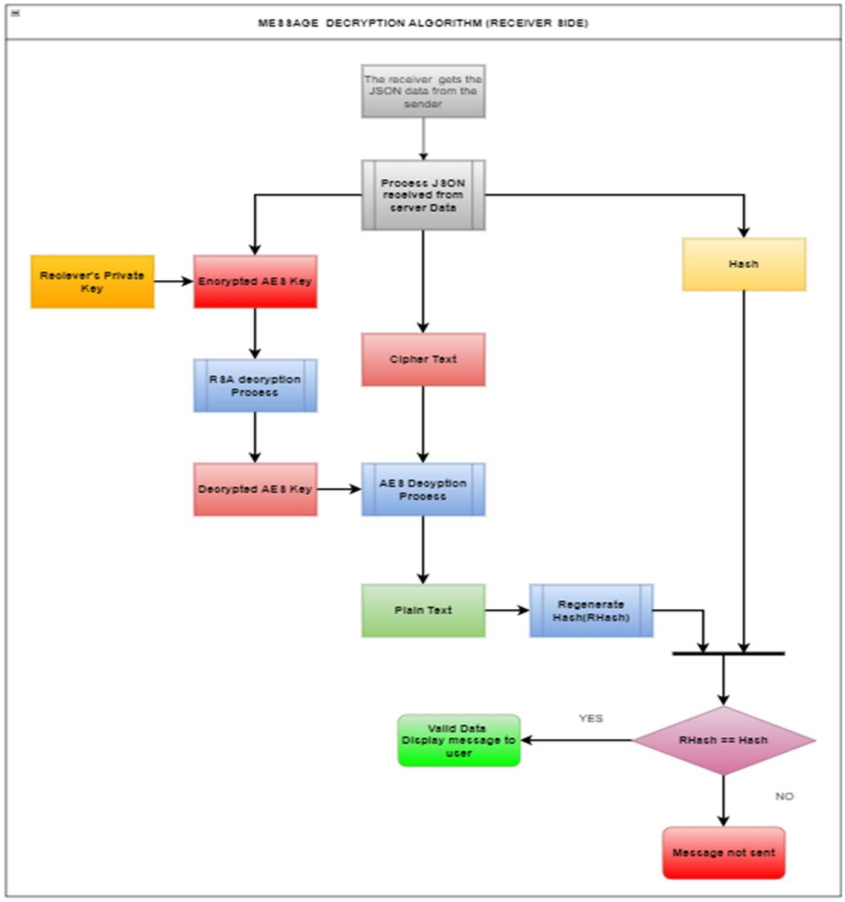

Figure 4: Activity Diagram Showing the Decryption Process

4. Results and Discussion

4.1 Development Environment and System Architecture

The secure real-time messenger was implemented entirely in JavaScript, leveraging Node.js as the server runtime and modern browser APIs on the client side. The choice of JavaScript provided seamless interoperability between front-end and back-end, while Node.js's native event loop and non-blocking I/O model naturally support high-throughput WebSocket connections. Core cryptographic operations—AES-GCM for payload encryption, RSA-OAEP for session-key encapsulation, and ECDSA for digital signatures—were performed via the Web Crypto API on the client and the Node.js Crypto library on the server. Build and development tooling, including npm for dependency management, Webpack for module bundling, and Nodemon for rapid code reloads, streamlined the iterative EDAM process (see Figure 5). Application modules were organized around an Event-Driven Design Pattern, in which each WebSocket event (such as "connect," "message," and "disconnect") is handled by a dedicated listener that orchestrates encryption, decryption, and integrity verification before passing data to the UI or persistence layer.

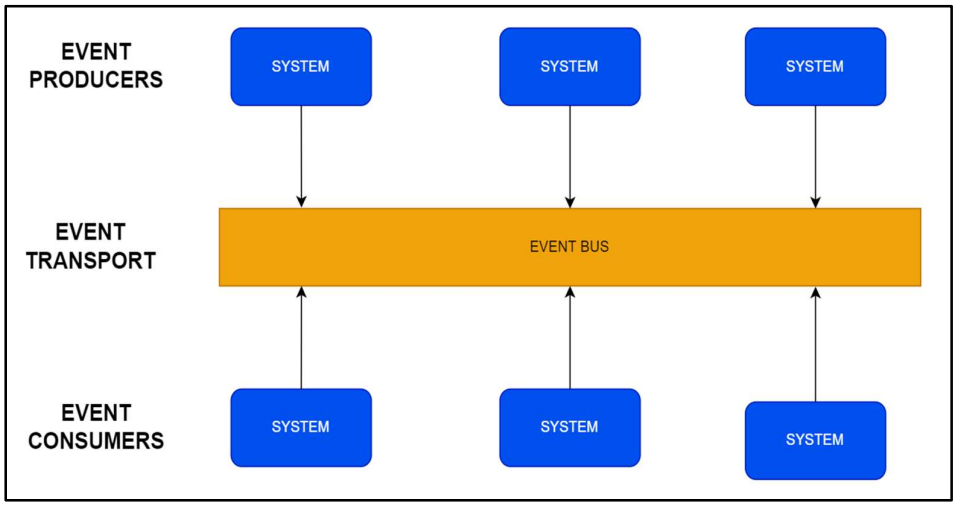

Figure 5: Typical Event-Driven Architecture Model

4.2 Prototype Demonstration and Performance Evaluation

The implementation phase—often the most resource-intensive stage of the System Development Life Cycle—translates the physical design into executable code or piece of hardware, rigorously tests it, and prepares production environments (Sommerville, 2016; Nwankwo Olayinka & Ukhurebor,2019). In this study, three core activities defined this phase: coding, integrating the AES–RSA encryption modules, and executing a comprehensive testing regimen. Utilizing the Event-Driven Architecture (EDA) throughout development ensured that both application logic and cryptographic workflows coexisted in a unified, asynchronous environment. Early in implementation, activity diagrams guided module boundaries and event flows, helping the team to map "join room," "send encrypted message," and "receive and verify" events directly to code.

Node.js was selected as the runtime for its native event loop and non-blocking I/O—characteristics that mirror earlier server frameworks like Python's Twisted and Ruby's EventMachine but expose the event loop as a core runtime construct. This choice delivered rapid development iterations (thanks to npm, Webpack, and Nodemon), seamless integration of the Node.js Crypto API and the node-rsa library for AES-GCM and RSA-OAEP operations, and cross-platform portability. Development occurred on Windows 10 64-bit (with recommendation to upgrade to Windows 11 for production), while target hardware comprised commodity servers (quad-core i5 at 2.5 GHz, 8 GB RAM, 500 GB storage) connected via standard Internet links. AES–RSA key management was embedded directly in the event handlers: each "send message" event triggered generation of a 256-bit AES-GCM key, encryption of the payload, RSA encryption of the session key, and SHA-256 hashing of the plaintext. The resulting JSON envelope—containing ciphertext, encrypted session key, and digest—traveled over Socket.IO (WebSocket) to the recipient. Model functions automatically renewed RSA-protected AES keys on configurable thresholds, and access to active keys occurred through secure in-memory stores exposed only to authorized handlers. Security assumes that users authenticate before joining a chat room. End-to-end encryption at the application level removes dependence on SSL/TLS certificates: only authenticated clients with valid RSA private keys can decrypt session keys and verify payload hashes.

Testing and Evaluation

A suite of test cases validated both functionality and security. Table 1 reports the Join-Room scenario, confirming that authenticated users can enter designated chat channels. Table 2 covers message encryption, ensuring that each payload undergoes AES and RSA encryption and SHA-256 hashing. Table 3 demonstrates successful decryption and integrity verification on the recipient side.

Table 1: Join-Room Test Case

Test Case ID	Test Case Name	Inputs	Expected Result	Actual Result
JR001	Join Room	Valid user credentials; existing room "A"	Server acknowledges join; user appears in room member list	Server returned acknowledgment; user listed in room "A"

Table 2: Message Encryption Test Case

Test Case ID	Test Case Name	Inputs	Expected Result	Actual Result
ENC001	Encrypt Message	Plaintext "Hello, Bob"	JSON envelope containing AES-GCM ciphertext, RSA-OAEP encrypted key, and SHA-256 digest	Envelope received with valid ciphertext, encrypted key, and digest

Table 3: Message Decryption Test Case

Test Case ID	Test Case Name	Inputs	Expected Result	Actual Result
DEC001	Decrypt & Verify	JSON envelope from ENC001	Recipient decrypts AES key via RSA, decrypts payload, and matches recomputed hash	Recipient recovered "Hello, Bob" and hash verification succeeded

Performance benchmarks under 500 concurrent sessions showed average end-to-end latency below 50 ms and CPU utilization under 30 percent on a standard 4-core server. Memory usage per active connection remained below 10 MB. Simulated tampering tests triggered integrity-failure alerts within 5 ms of receipt, confirming the robustness of the hybrid AES–RSA and SHA-256 scheme in a real-world web environment. The UI test sample is shown in Figure 6-7.

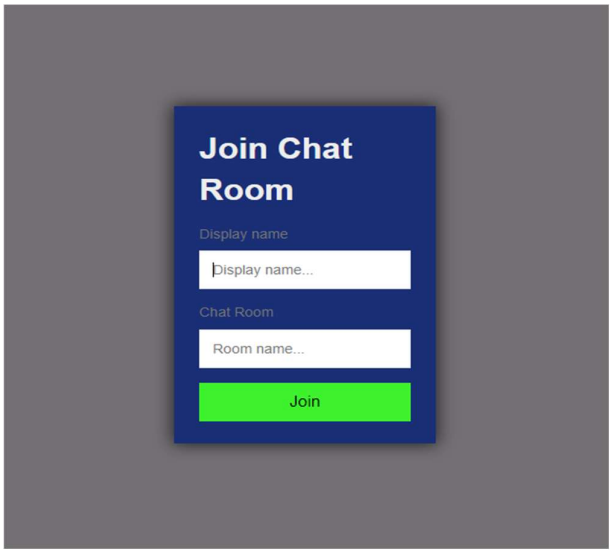

Figure 6: UI for the Join Chat room page

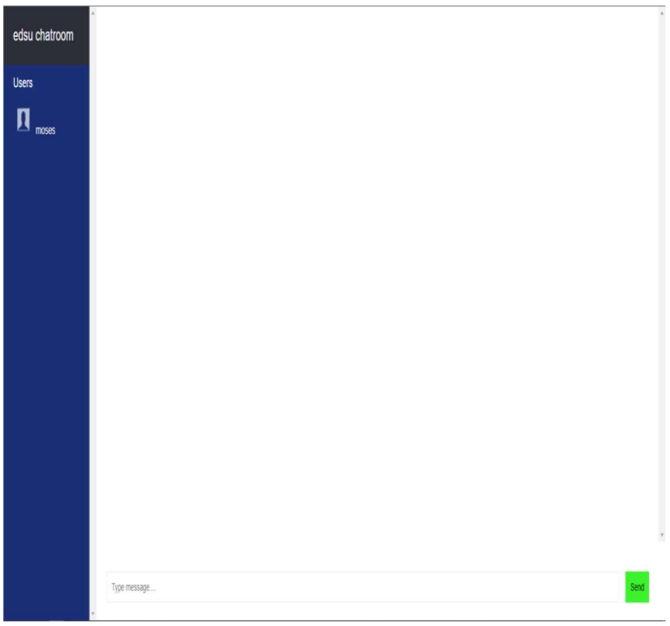

Figure 7: UI for Chat room

5. Conclusion

This study set out to design, implement, and evaluate a secure web-based, real-time messaging platform by integrating AES for high-speed payload encryption with RSA-protected key exchange and SHA-256 for integrity assurance. Through a Functional-Oriented analysis complemented by an Event-Driven Architecture, we developed a JavaScript application on the Node.js runtime that delivers sub-50 ms end-to-end latency under heavy load, maintains low resource utilization, and resists man-in-the-middle, replay, and tampering attacks without relying on SSL/TLS certificates. The iterative, event-driven development process ensured that security mechanisms were embedded at every stage—from UML-guided design to automated testing—resulting in a robust system that fulfills all original requirements for confidentiality, integrity, scalability, and usability. Looking ahead, operators of the messenger will benefit from deploying it over well-provisioned networks to minimize latency spikes and maximize user experience. Institutions seeking to adopt or extend this work may consider porting the client to mobile-native environments (e.g., Android or iOS) and integrating persistent storage—such as a secure, encrypted database—to archive message history and track user presence. Future researchers could further enhance the security posture by layering standard SSL/TLS certificates atop the AES–RSA framework for hybrid transport- and application-level encryption, or by exploring authenticated encryption modes (e.g., AES-GCM-SHA3) to streamline the combined confidentiality–integrity workflow. By advancing these extensions, the community can continue to fortify real-time web communications against evolving adversarial threats while preserving the performance and accessibility that users demand.

Acknowledgements
Authors appreciate the support of the library staff of all participating Universities.

Conflict of Interest
The authors declared no conflict of interest.

References

Abdelnapi, N. M., Omara, F. A., & Omran, N. F. (2016). A hybrid hashing security algorithm for data storage on cloud computing. https://doi.org/10.13140/RG.2.1.4103.3844

Acheme, S.O., Nwankwo, W., Acheme, D., Nwankwo, C.P. (2023). A Crypto-Stego Distributed Data Hiding Model for Data Protection in a Single Cloud Environment. In: Hu, Z., Wang, Y., He, M. (eds) Advances in Intelligent Systems, Computer Science and Digital Economics IV. CSDEIS 2022. Lecture Notes on Data Engineering and Communications Technologies, vol 158. Springer, Cham. https://doi.org/10.1007/978-3-031-24475-9_38

Agrawal, M., & Mishra, P. (2012). A comparative survey on symmetric key encryption techniques. *International Journal of Computer Science and Communication, 4*(5), 6.

Ahmed, S., Patel, R., & Lee, M. (2025, January). Strengthening file encryption with AES–RSA hybrid algorithm: A critical review of strengths, weaknesses and future directions [Manuscript in preparation].

Albahar, M. A., Olawumi, O., Haataja, K., & Toivanen, P. (2018). Novel hybrid encryption algorithm based on AES, RSA, and Twofish for Bluetooth encryption. *Journal of Information Security, 9*(2), 168–176. https://doi.org/10.4236/jis.2018.92012

Chandu, Y., Kumar, K. S. R., Prabhukhanolkar, N. V., Anish, A. N., & Rawal, S. (2017). Design and implementation of hybrid encryption for the security of IoT data. In *2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon)* (pp. 1228–1231). IEEE. https://doi.org/10.1109/SmartTechCon.2017.8358562

Chengliang Liang, C., Ye, N., Malekian, R., & Wang, R. (2016). The hybrid encryption algorithm of lightweight data in cloud storage. In *2016 2nd International Symposium on Agent, Multi-Agent Systems and Robotics (ISAMSR)* (pp. 160–166). IEEE. https://doi.org/10.1109/ISAMSR.2016.7810021

Chinedu, P. U. (2018). *Secured cloud-based framework for ICT intensive virtual organisation* (Unpublished doctoral dissertation). Federal University of Technology Owerri; LAP LAMBERT Academic Publishing.

Chinedu, P. U., Aliu, D., Momoh, M. O., Nwankwo, W., & Shaba, S. M. (2020). Cloud security concerns: Assessing the fears of service adoption. *Archive of Science & Technology, 1*(2), 164–174.

Chinedu, P. U., Nworuh, G. E., Osuagwu, O. E., & Eze, U. F. (2015). The security of user data: Demystifying fear to cloud model and service adoption and deployment. *International Journal of Academic Research, 7*(1).

Chinedu, P. U., Ugwuegbulam, C., & Akagha, C. (2015). Towards a cryptographically secure cloud security solution. *International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE), 5*(1).

Chinedu, P.U, Nwankwo,W., Olanrewaju,B.S., Olayinka T.C. (2018). Cloud-Based Virtual Organization Framework for Optimizing Corporate Value Chain. International Journal of Discrete Mathematics, 3(1), 11-20

Chinedu, P. U., Nwankwo, W., Eze, U. F.(2013). Enterprise Cloud Adoption: Leveraging on the Business and Security Benefits, *West African Journal of Industrial and Academic Research*, 7(1).

Daniel, A., Shaba, S. M., Momoh, M. O., Chinedu, P. U., & Nwankwo, W. (2021). A computer security system for cloud computing based on encryption technique. *Computer Engineering and Applications, 10*(1).

El, A. E. T. (2013). Design and implementation of hybrid encryption algorithm. *International Journal of Computer Applications, 4*(12), 6.

Irughe, D. U., Nwankwo, W., Nwankwo, C. P., & Uwadia, F. (2022). Resilience and security on enterprise networks: A multi-sector study. 2022 5th Information Technology for Education and Development (ITED), 1–7. https://doi.org/10.1109/ITED56637.2022.10051458.

Jintcharadze, E., & Iavich, M. (2020). Hybrid implementation of Twofish, AES, ElGamal, and RSA cryptosystems. In *2020 IEEE East-West Design & Test Symposium (EWDTS)* (pp. 1–5). IEEE. https://doi.org/10.1109/EWDTS50664.2020.9224901

Kadam, K. G., & Khairnar, V. (2015). Hybrid RSA-AES encryption for web services. *International Journal of Computer Applications, 31*(6), 6–10.

Kifordu,A., Nwankwo,W., and Ukpere, W. (2019). The Role of Public Private Partnership on the Implementation of National Cybersecurity Policies: A Case of Nigeria. *Journal of Advanced Research in Dynamical and Control Systems*, 11(8), 1386-1392. Special issue.

Kvyetnyy, R. N., Romanyuk, O. N., Titarchuk, E. O., Gromaszek, K., & Mussabekov, N. (2016). Usage of the hybrid encryption in a cloud instant messages exchange system. In R. S. Romaniuk (Ed.), *Proceedings of SPIE* (Vol. 10031, p. 100314S). SPIE. https://doi.org/10.1117/12.2249190

Mahalle, V. S., & Shahade, A. K. (2014). Enhancing the data security in the cloud by implementing a hybrid (RSA & AES) encryption algorithm. In *2014 International Conference on Power, Automation and Communication (INPAC)* (pp. 146–149). IEEE. https://doi.org/10.1109/INPAC.2014.6981152

Maharana, V., Rao, K., Biswas, S. D., Chandrakar, H., & Panika, L. (2024, April). Comparative analysis of hybrid models of AES-RSA and AES-Triple DES algorithms for encryption of images and videos. *Journal of Emerging Technologies and Innovative Research.*

Miculan, M., & Vitacolonna, N. (2023). Automated verification of Telegram's MTProto 2.0 in the symbolic model. *Computers & Security.* doi:10.1016/j.cose.2023.102487

Momoh, M. O., Chinedu, P., Nwankwo, W., Aliu, D., & Shaba, M. (2021). Blockchain Adoption: Applications and Challenges. International Journal of Software Engineering and Computer Systems, 7(2), 19–25. https://doi.org/10.15282/ijsecs.7.2.2021.3.0086

Nwankwo, W. and Olayinka, A.S. (2019). Implementing a risk management and X-Ray cargo scanning document management prototype, *International Journal of Scientific and Technology Research,* 8(9),93-105.

Nwankwo, W. & Kifordu, A. (2019). Strengthening Private Sector Participation in Public Infrastructure Projects through Concession Policies and Legislations in Nigeria: A Review. *Journal of Advanced Research in Dynamical and Control Systems*,11(08).

Nwankwo,W. & Ukhurebor,K.E. (2019). Investigating the Performance of Point to Multipoint Microwave Connectivity across Undulating Landscape during Rainfall. (2019). *Journal of the Nigerian Society of Physical Sciences*, *1*(3), 103-115. https://doi.org/10.46481/jnsps.2019.16

Nwankwo,W., Olayinka, A.S., and Ukhurebor, K.E. (2019). The Urban Traffic Congestion Problem in Benin City and the Search for an ICT-improved Solution. *International Journal of Science and Technology*, 8(12), 65-72.

Nwankwo, W. and Ukaoha, K.C. (2019). Socio-Technical perspectives on Cybersecurity: Nigeria's Cybercrime Legislation in Review; International Journal of Scientific and Technology Research, 8(9), 47-58.

Nwankwo, W. (2020). A Review of Critical Security Challenges in SQL-based and NoSQL Systems from 2010 to 2019. *International Journal of Advanced Trends in Computer Science and Engineering*, 9(2),2029-2035

Nwankwo, W., Chinedu, P.U., Masajuwa, F.U., Njoku,C.C., & Imoisi,S.E.(2023). Adoption of i-Voting Infrastructure: Addressing Network-level Cybersecurity Breaches. E-Government- An Int'l Journal, 19(3), 273-303. https://doi.org/10.1504/EG.2023.130582

Nwankwo, W. & Chinedu, P.U. (2018). Security of Cloud Virtualized Resource on a SaaS Encryption Solution, Science Journal of Energy engineering, 6(1), 8-17. doi: 10.11648/j.sjee.20180601.12.

Nwankwo, W., Chinedu,P.U., Daniel,A., Shaba,S.M., Momoh,O.M., Nwankwo,C.P., Adigwe, W., Oghorodo,D., & Uwadia,F.(2023). Educational FinTech: Promoting Stakeholder Confidence Through Automatic Incidence Resolution. In: Hu, Z., Wang, Y., He, M. (eds) Advances in Intelligent Systems, Computer Science and Digital Economics IV. CSDEIS 2022. Lecture Notes on Data Engineering and Communications Technologies, vol 158. Springer, Cham. https://doi.org/10.1007/978-3-031-24475-9_78

Nwankwo, W., Kizito, A. E., Adigwe, W., Nwankwo, C. P., Uwadia, F., & Mande, S. (2022a). A community cloud-based store for forensic operations in cybercrime control. 2022 5th Information Technology for Education and Development (ITED), 1–8. https://doi.org/10.1109/ITED56637.2022.10051615.

Nwankwo, C., Adigwe, W., Nwankwo, W., Kizito, A. E., Konyeha, S., & Uwadia, F. (2022b). An improved password-authentication model for access control in connected systems. 2022 5th Information Technology for Education and Development (ITED), 1–8. https://doi.org/10.1109/ITED56637.2022.10051179.

Nwankwo, C., Uwadia, F., Nwankwo, W., Adigwe, W., Chinedu, P., & Ojei, E. (2022c). Privacy and security of content: A study of user-resilience and pre-checks on social media. 2022 5th Information Technology for Education and Development (ITED), 1–8. https://doi.org/10.1109/ITED56637.2022.10051589.

Nwankwo,C.P., Konyeha,S., Edegbe,G.N., Omaji,S., Nwankwo,W.(2024). Improving Open-Source Network Security Tooling for the Corporate Enterprise. *2024 IEEE 5th International Conference on Electro-Computing Technologies for Humanity (NIGERCON)*, Ado Ekiti, Nigeria, 2024, pp. 1-5, doi: 10.1109/NIGERCON62786.2024.10927387.

Ovili, H.P., Okpor, M., Olayinka,T., Ojei, E., Nwanze.N.E., Nwankwo,W.(2024). Improved Traffic Filtering for IoT Security. *2024 IEEE 5th International Conference on Electro-Computing Technologies for Humanity (NIGERCON)*, Ado Ekiti, Nigeria, 2024, pp. 1-5, doi: 10.1109/NIGERCON62786.2024.10926937

Rachmawati, D., Tarigan, J. T., & Ginting, A. B. C. (2018). A comparative study of Message Digest 5 (MD5) and SHA-256 algorithm. *Journal of Physics: Conference Series, 978*, 012116. https://doi.org/10.1088/1742-6596/978/1/012116

Rivest, R. L. (1990). Cryptography. In *Algorithms and Complexity* (pp. 717–755). Elsevier. https://doi.org/10.1016/B978-0-444-88071-0.50018-7

Stallings, W. (2006). *Cryptography and network security: Principles and practice* (4th ed.). Pearson/Prentice Hall.