Enhanced Encryption Model for Secured Communication Using Blockchain Technology

Onwuachu Uzochukwu Christian[1] and Ndueso, Etukudo Ekefre[2]

[1]Imo State University, Owerri, Imo State, Nigeria

[2]University of Port Harcourt, Port Harcourt, Rivers State, Nigeria.

DOI: https://doi.org/10.51584/IJRIAS.2026.11100085

Received: 21 October 2026; Accepted: 26 October 2026; Published: 13 November 2026

ABSTRACT

This study proposes an enhanced encryption model for secured communication using blockchain technology. The proposed system employs the Advanced Encryption Standard in Galois/Counter Mode (AES-GCM) for message confidentiality and authenticated encryption, public-key cryptography for protecting message encryption keys, digital signatures for sender authentication, and cryptographic hashes for message integrity verification. A blockchain-inspired, hash-linked audit ledger records message hashes and relevant transaction metadata without placing plaintext messages or secret keys on the ledger. The system is designed as a Python Flask web application with a database for user and message metadata management. The research methodology covers system analysis, cryptographic model design, implementation, functional testing, security testing, and performance evaluation. The proposed evaluation examines encryption and decryption latency, throughput, integrity verification, unauthorized-access resistance, and the overhead associated with audit recording. Experimental results will be generated through repeatable tests using documented datasets, fixed configurations, and clearly defined measurement procedures. The expected contribution is a practical communication prototype that integrates authenticated encryption with tamper-evident audit records while recognizing the limitations of a locally maintained ledger compared with a decentralized blockchain network.

Keywords: Blockchain technology, secure communication, AES-GCM, public-key cryptography, digital signatures, message integrity, Python Flask, encryption.

INTRODUCTION

The growing use of digital communication has increased the need for secure mechanisms that protect sensitive information against unauthorized access, message modification, impersonation, and other communication-related threats. Conventional communication systems often depend on centralized infrastructure and cryptographic mechanisms whose effectiveness relies on proper key management, secure implementation, and trust in the service provider. Blockchain technology offers a complementary approach through distributed recordkeeping, cryptographic linking, and tamper-evident transaction histories. However, blockchain alone does not guarantee message confidentiality and can expose sensitive information when private content is stored directly on a public ledger. Communication technology has become an essential component of modern society, supporting information exchange among individuals, businesses, educational institutions, government agencies, and financial organizations. Electronic communication systems enable rapid message transmission, remote collaboration, digital transactions, and the sharing of sensitive information. Despite these benefits, communication over digital networks faces security threats, including eavesdropping, unauthorized access, impersonation, data modification, and denial of service.

Encryption is one of the fundamental techniques used to protect information during storage and transmission. It transforms readable plaintext into ciphertext through a cryptographic algorithm and a key, making the original message inaccessible to parties who lack the required decryption capability. The Advanced Encryption Standard (AES), standardized by the National Institute of Standards and Technology (NIST), supports 128-bit, 192-bit, and 256-bit keys and is widely used to protect electronic information (NIST, 2023).

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue X October 2026

However, encryption alone does not address every communication security requirement. A secure communication system must also consider message integrity, sender authentication, recipient authorization, replay attacks, and cryptographic key management. Authenticated encryption modes such as AES-GCM provide confidentiality and integrity protection for encrypted messages, provided that the algorithm is correctly implemented and its nonce requirements are respected (Dworkin, 2007). Blockchain technology provides another mechanism for improving trust in digital systems. It organizes records into cryptographically linked blocks and supports a shared transaction history in distributed implementations. Alteration of a previously recorded block can invalidate the links that follow it, making unauthorized modifications detectable when the ledger is independently validated. Nevertheless, a blockchain does not automatically encrypt messages, authenticate every participant, or guarantee the truth of the information entered into it.

The integration of encryption and blockchain therefore presents an opportunity to develop a more comprehensive secure communication model. In the proposed system, message contents are encrypted before storage or transmission, while cryptographic hashes and selected metadata are recorded in a hash-linked audit ledger. Digital signatures are used to support sender verification, and a secure key-management mechanism protects the keys needed for message encryption and decryption. The project adopts Python Flask as the web application framework and SQLite as the prototype database. The proposed system is intended to demonstrate the interaction between cryptographic protection, message exchange, access control, and tamper-evident audit records. For an actual decentralized blockchain deployment, the local prototype would need to be extended with distributed consensus, independently operated nodes, transaction validation, and an appropriate trust model.

LITERATURE REVIEW

The reviewed studies provide evidence of the importance of cryptographic timestamping, digital signatures, privacy protection, and systematic security evaluation. Their major contributions and relevance to the present research are summarized below.

Author(s) and year	**Research focus**	**Major contribution**	**Relevance to this study**
Haber and Stornetta (1991)	Digital document timestamping	Cryptographic evidence of document history	Supports audit-record integrity
Fang et al. (2020)	Blockchain digital signatures	Reviews signatures for authentication and non-repudiation	Supports sender-signature verification
de Haro-Olmo et al. (2020)	Blockchain privacy and anonymisation	Reviews privacy risks and protection approaches	Supports minimizing sensitive ledger data
3WJunejo et al. (2021)	Blockchain privacy evaluation	Highlights challenges in evaluating privacy mechanisms	Supports measurable security evaluation
Boon and Mohd Banua (2024)	Blockchain security and privacy	Reviews applications and continuing challenges	Supports an integrated security design

MATERIALS AND METHODS

The proposed system integrates encryption and authentication with a cryptographically verifiable audit mechanism. It enables a sender to submit a message, encrypt it, and transmit the encrypted content to an authorized recipient. It also records appropriate integrity evidence and transaction metadata for later

verification. The major components include: User authentication module: Controls access to the application. Encryption module: Uses AES-256-GCM to encrypt message content. Key management module: Generates, protects, and securely distributes or wraps encryption keys. Digital signature module: Signs relevant message or transaction data and verifies signatures. Hashing module: Computes cryptographic digests using SHA-256. Audit module: Records and verifies transaction history using a hash-linked ledger or a decentralized blockchain. Database module: Stores encrypted messages, user information, and necessary transaction metadata. Recipient module: Verifies and decrypts messages for authorized recipients. The system is intended to protect the message itself while providing a mechanism for detecting unauthorized modifications to associated records.

System Architecture

The system architecture consists of the user interface, Flask application, cryptographic services, database, and audit ledger. The communication process follows this sequence: The sender logs into the application. The sender composes a message and selects the recipient. The application validates the input and generates or obtains a secure encryption key. The encryption module encrypts the message using AES-256-GCM. The signing module creates a digital signature over the specified message or transaction data. The application stores the ciphertext and required metadata. The audit module records the transaction hash and relevant metadata. The recipient requests the message. The system verifies the signature and checks the audit record. The system authorizes decryption and authenticates the ciphertext. The recipient receives the original message if all required checks succeed. The ledger should not contain plaintext messages, private signing keys, or secret encryption keys.

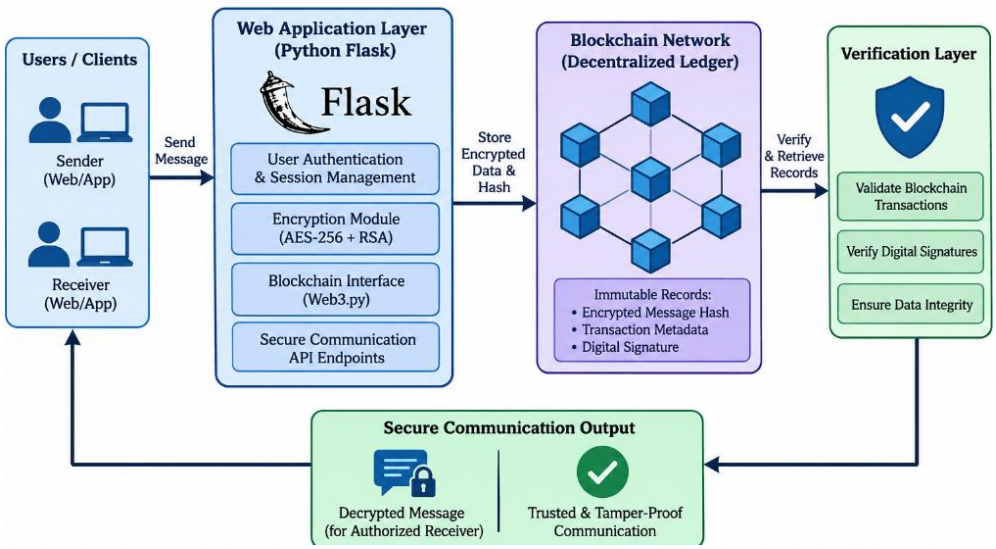


Figure 1: The System Architecture of the proposed system.

Figure 1 illustrates how the proposed Python Flask application integrates encryption, user authentication, blockchain-based audit recording, and verification to support secure communication. Users/Clients: The sender composes a message, while the recipient receives and verifies the protected message through a web interface. Web Application Layer (Python Flask): Manages user authentication, sessions, message processing, encryption functions, and secure API endpoints. Encryption Module: Uses AES-256-GCM for message confidentiality and integrity protection, alongside an appropriate public-key mechanism for protecting encryption keys. Digital signatures support sender verification. Blockchain Interface: Connects the application to the ledger, submits message hashes and relevant transaction metadata, and retrieves records for verification. Blockchain Network: Maintains transaction records and cryptographically linked blocks. In a decentralized deployment, network consensus and independently operated nodes help protect the shared ledger. Verification Layer: Validates the audit records, checks digital signatures, and detects discrepancies between the stored message and its recorded hash. Secure Communication Output: After authorization and successful verification, the intended recipient can decrypt and read the message.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue X October 2026

Cryptographic Algorithms Used.

AES-256-GCM is used to provide confidentiality and authenticated encryption. The algorithm uses a 256-bit key and produces ciphertext with an authentication tag. The encryption operation can be represented as:

$$(C, T) = \text{AES-GCM}_K(N, P, A)$$

Where: K = secret encryption key. N = unique nonce. P = plaintext message.

A = optional associated authenticated data. C = ciphertext. T = authentication tag.

The nonce must not be reused with the same key. During decryption, the application verifies the authentication tag before releasing the plaintext.

Secure Key Management

A secure key-management mechanism generates and protects the AES encryption key. For communication between different users, the key can be wrapped using an appropriate public-key encryption mechanism or delivered through a secure key-establishment protocol. The recipient must be able to obtain the correct key securely. Encryption alone is insufficient if keys are stored insecurely or made available to unauthorized users.

Digital Signature

Digital signatures are used to verify the authenticity and integrity of signed data. The sender signs the relevant message or transaction data using a private signing key, while the recipient verifies the signature using the corresponding public key. The system must protect private keys and associate public keys with users through a trusted registration or verification process.

SHA-256 Hashing

SHA-256 generates a fixed-length digest from message data or transaction records. A hash can be used to check whether data matches a previously recorded value. The hash function can be represented as:

$$H = \text{SHA256}(M)$$

Where M represents the selected data and H is its cryptographic digest. A hash is not encryption and cannot be used to recover the original message. Its security value depends on how the reference hash is stored and verified.

Blockchain or Hash-Linked Audit Ledger

The audit mechanism records transaction hashes and selected metadata to support traceability and tamper detection. For a hash-linked ledger, each record can include the hash of the previous record:

$$H_i = \text{SHA256}(D_i \parallel H_{i-1})$$

Where: H_i = hash of the current record. Di = current record data. $H_{\{i-1\}}$ = hash of the preceding record. $\parallel$ = concatenation.

This construction helps reveal record modification when verified against a trusted reference. However, a local hash-linked ledger is not equivalent to a decentralized blockchain. Decentralization requires a network of independently participating nodes and an appropriate consensus or validation mechanism.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue X October 2026

System Implementation Procedure

The implementation is carried out in the following stages.

Stage 1: Environment setup. Install Python and the required libraries, create a project directory, and configure the Flask application.

Stage 2: Database creation. Create the tables required for users, messages, and audit records.

Stage 3: User authentication. Implement registration, login, logout, password hashing, and access-control checks.

Stage 4: Encryption and decryption. Implement AES-256-GCM encryption, secure key handling, and authenticated decryption.

Stage 5: Digital signatures. Implement signature generation and verification using a suitable digital-signature algorithm.

Stage 6: Audit logging. Generate message or transaction hashes and append audit records. If a decentralized blockchain is required, implement the appropriate network interface and transaction-confirmation process.

Stage 7: Message transmission. Store the encrypted message and make it available to the intended recipient.

Stage 8: Verification and decryption. Verify the signature and audit evidence, then perform authenticated decryption for an authorized recipient.

Stage 9: Testing and evaluation. Execute functional, security, and performance tests and record the results.

Performance Testing

Performance evaluation measures the time and resources required to process messages.

The principal metrics are:

Encryption latency $T_e = t_{\text{end-encryption}} - t_{\text{start-encryption}}$

Decryption latency $T_d = t_{\text{end-decryption}} - t_{\text{start-decryption}}$

Throughput = [{Number of successfully processed messages}/{Elapsed time}]

Success Rate= [{{Passed functional tests}/{Total functional tests}} * 100]

Tests should be repeated for different message sizes under documented hardware and software conditions. The report should present observed measurements rather than assumed values.

Activity Diagram of the proposed system

It illustrates the step-by-step workflow of the proposed system, from message composition and encryption to blockchain recording, recipient verification, and message decryption. Start: The process begins when the sender initiates communication. Compose Message: The sender enters the message to be transmitted. Validate User Session and Input: The Flask application checks the user's session and validates the submitted message. Generate Encryption Key: The system generates a secure encryption key for AES-256-GCM encryption. Encrypt Message and Generate Digital Signature: The system encrypts the message and generates a digital signature to support sender authentication and message integrity

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue X October 2026

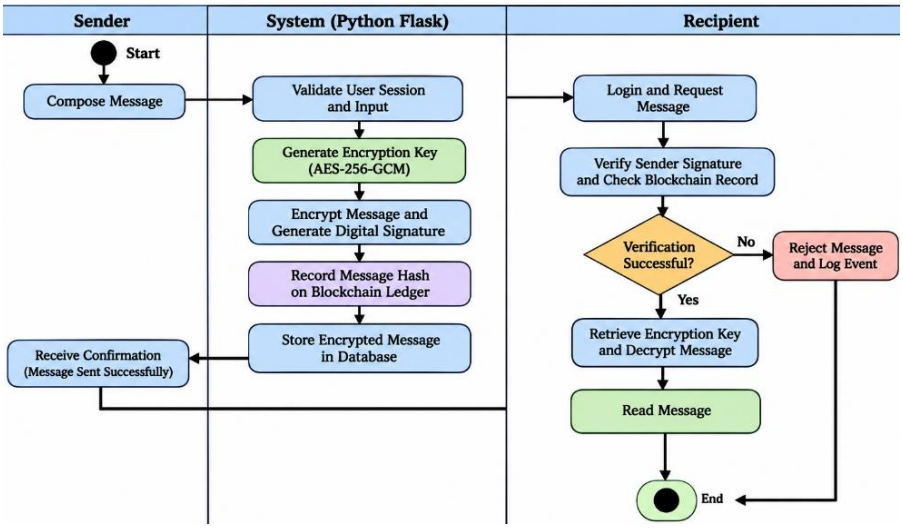


Figure 2: The activity diagram of the proposed system

Record Message Hash on Blockchain Ledger: A cryptographic hash and relevant transaction metadata are recorded on the ledger. The plaintext message and secret encryption key should not be stored on the blockchain. Store Encrypted Message: The encrypted message and required metadata are stored in the database. Receive Confirmation: The sender receives confirmation that the message has been submitted successfully. Login and Request Message: The recipient logs in and requests the encrypted message.

Verify Sender Signature and Check Blockchain Record: The system verifies the sender's digital signature and checks the corresponding ledger record. Verification Decision: If verification fails, the message is rejected and the event is logged. If verification succeeds, the system proceeds to decryption. Retrieve Encryption Key and Decrypt Message: The system securely retrieves or unwraps the encryption key for the authorized recipient and decrypts the message. Read Message: The recipient reads the original message, and the process ends.

The Sequence Diagram of the proposed system

The diagram shows the order in which the system components communicate. It helps in understaning where encryption, signature generation, database storage, audit recording, verification, and decryption occur. The diagram represents the intended workflow. Figure 3 illustrates the interaction between the sender, Flask web application, encryption module, digital signature service, blockchain ledger, SQLite database, and recipient during secure message transmission.

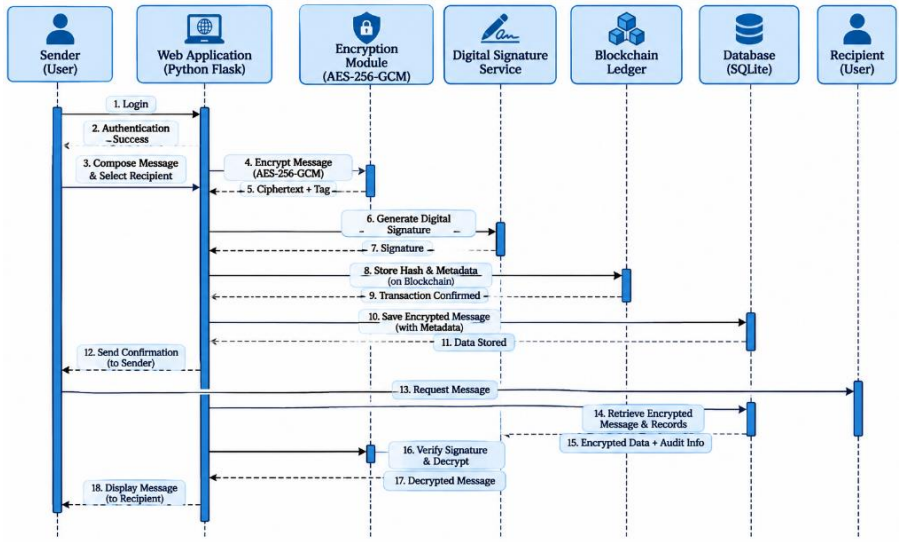


Figure 3. the Sequence Diagram of the proposed system

Login: The sender logs into the web application. Authentication: The application validates the sender's credentials. Compose Message: The sender enters a message and selects the intended recipient. Encrypt Message: The encryption module encrypts the message using AES-256-GCM. Return Ciphertext: The encryption module returns the ciphertext and authentication tag. Generate Digital Signature: The signature service signs the required message or transaction data. Return Signature: The digital signature is returned to the Flask application. Record Hash and Metadata: The application submits the message hash and selected transaction metadata to the audit ledger.

Transaction Confirmation: The ledger confirms the record according to its implementation. Save Encrypted Message: The application stores the ciphertext and associated metadata in SQLite. Storage Confirmation: The database confirms successful storage. Notify Sender: The application informs the sender that the message has been submitted. Request Message: The recipient requests the message.

Retrieve Encrypted Message: The application retrieves the ciphertext and relevant audit information.Return Stored Data: The database returns the encrypted message and associated records.Verify and Decrypt: The application verifies the signature and performs authenticated decryption for the authorized recipient. Return Decrypted Message: The original message is made available only if the required checks succeed. Display Message: The application presents the recovered message to the recipient.

EXPERIMENT AND RESULT

The system is designed to improve the confidentiality, integrity, authentication, and traceability of electronic communication through the integration of AES-256-GCM encryption, digital signatures, cryptographic hashing, and blockchain-based or hash-linked audit records.

The system is implemented using Python and the Flask web framework, with a database for storing encrypted messages and related transaction information. The evaluation focuses on the correctness of the encryption and decryption process, message integrity verification, digital-signature validation, audit-record verification, and system performance.

Table 1: Functional Testing Results

S/N	Test case	Expected result	Actual result
1	User login	Authorized user gains access	To be recorded
2	Message encryption	Plaintext is converted to ciphertext	To be recorded
3	Message transmission	Encrypted message is stored or transmitted	To be recorded
4	Message decryption	Authorized recipient recovers the original message	To be recorded
5	Digital-signature verification	Valid signature is accepted	To be recorded
6	Modified-message detection	Unauthorized modification is detected	To be recorded
7	Audit-ledger verification	Valid records pass integrity checks	To be recorded
8	Invalid-signature test	Message is rejected	To be recorded
9	Unauthorized-access test	Access is denied	To be recorded

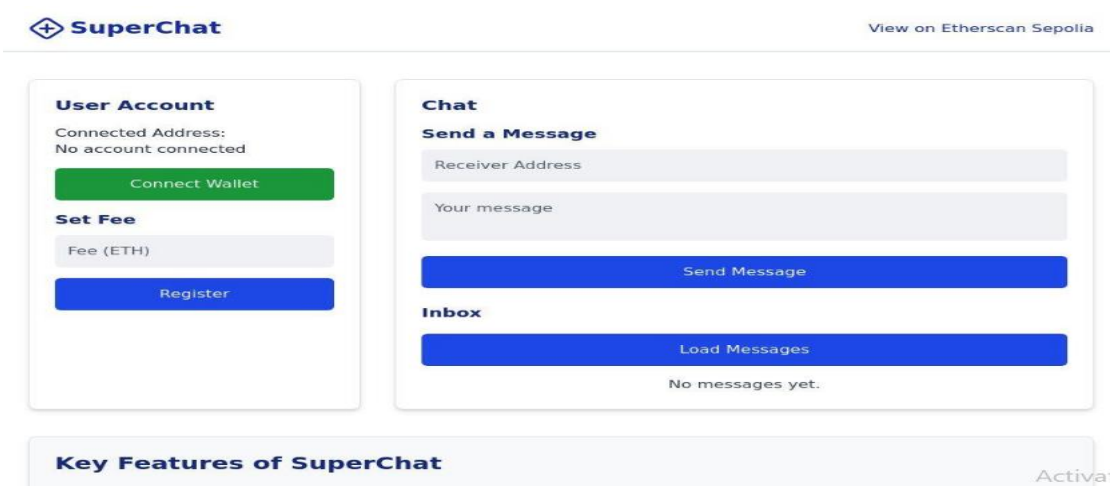


Figure 4: Secure Messaging Dashboard Output

The dashboard allows the sender to enter a message, select a recipient, encrypt the message, and submit it for secure transmission.

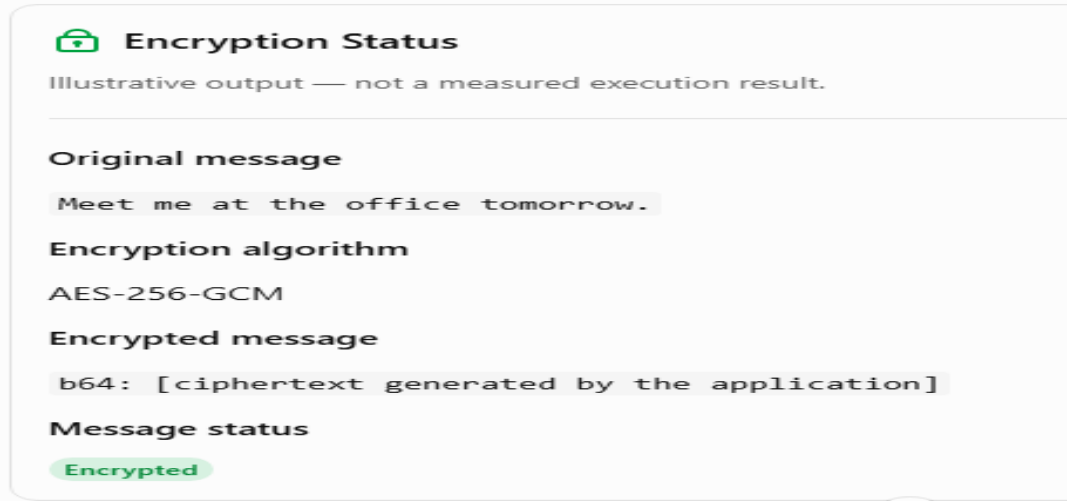


Figure 5: Message Encryption Output

The application converts the original message into ciphertext so that unauthorized users cannot read it without the appropriate key.

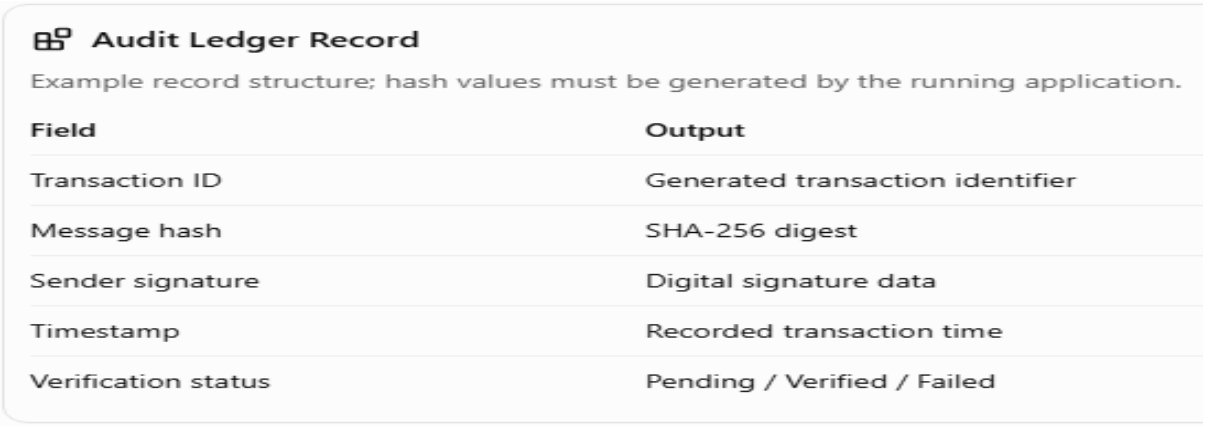


Figure 6: Blockchain Ledger Output

The ledger records evidence of the transaction and supports later integrity checks. In a local prototype, this should be described as a hash-linked audit ledger, not a decentralized blockchain.

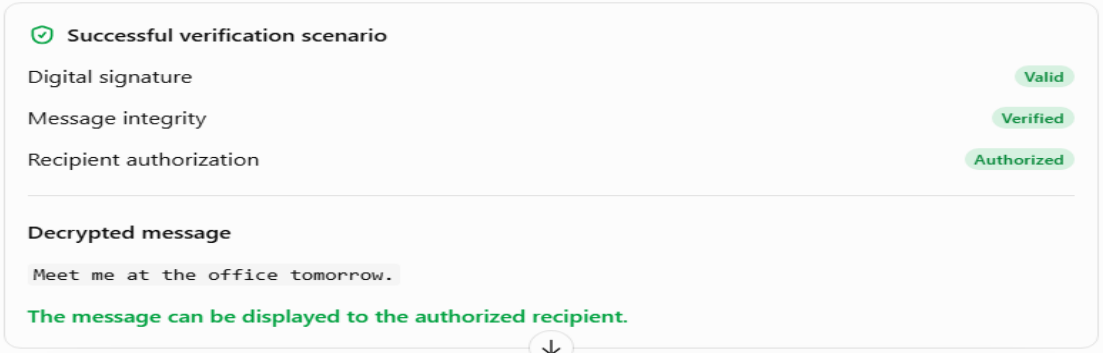


Figure 7: Message Verification and Decryption Output

If signature verification or authenticated decryption fails, the system should reject the message and record the failure.

RESULT DISCUSION

The proposed system uses the Advanced Encryption Standard (AES) with a 256-bit key in Galois/Counter Mode (GCM). AES-GCM provides confidentiality and authenticated encryption, enabling the system to detect unauthorized changes to the ciphertext or associated authenticated data. During encryption, the system accepts a plaintext message and encrypts it using a securely generated key and a unique nonce. The resulting ciphertext and authentication tag are stored or transmitted instead of the original plaintext. During decryption, the recipient's authorized application uses the appropriate key to recover the message. If the ciphertext, nonce, authentication tag, or authenticated data has been altered, authenticated decryption should fail. The expected outcome is that the original message is recovered when the correct key and unmodified encrypted data are supplied. This outcome must be confirmed through functional testing before it can be reported as an experimentally established result.

Digital signatures provide a mechanism for verifying the origin and integrity of a signed message or transaction. The sender signs the appropriate message data or its cryptographic digest using a private signing key, while the recipient verifies the signature using the corresponding public key. A valid signature indicates that the signed data has not changed since signing and that the signature corresponds to the relevant private key. It does not, by itself, prove the real-world identity of the sender; that depends on secure key registration and management. The system should be tested using both valid and invalid signatures. Valid signatures should pass verification, while signatures associated with modified data or an incorrect public key should fail. The results should be documented in the implementation's test records.

The proposed model uses cryptographic hashes to support the integrity and traceability of communication records. Each transaction can contain a message hash, transaction identifier, timestamp, sender information, and relevant verification metadata. When implemented with a decentralized blockchain, validated records can be replicated across independent nodes and protected by the network's consensus mechanism. However, when the prototype uses a local hash-linked ledger, the ledger only provides local tamper-evidence: an attacker with sufficient privileges may be able to rewrite the records and recompute the hashes. Therefore, the strength of the audit mechanism depends on the implementation. A local prototype should be described as a hash-linked audit ledger unless an actual decentralized blockchain network and its consensus mechanism have been implemented and tested. The evaluation should include modifying an existing record and checking whether the verification process detects the alteration. It should also examine whether changes to a record's hash or its previous-record reference are detected.

CONCLUSION

The proposed model uses AES-256-GCM authenticated encryption to protect message contents and detect unauthorized changes to encrypted data. Digital signatures provide a mechanism for verifying message authenticity and integrity, while SHA-256 hashing supports the generation of message fingerprints and the

verification of recorded data. The audit ledger provides a mechanism for identifying unauthorized modifications to communication records, provided that the ledger is correctly implemented and checked against a trusted reference. The system is designed around a Python Flask web application with an SQLite database for storing encrypted messages and associated metadata. This approach provides a practical foundation for implementing and testing secure communication within a web-based environment. It also demonstrates how established cryptographic techniques can be combined with blockchain-related audit mechanisms to strengthen communication security. However, the effectiveness of the proposed model depends on proper key management, secure user authentication, reliable public-key verification, unique nonce generation for AES-GCM, and appropriate protection of stored data. Furthermore, a locally maintained hash-linked ledger provides tamper evidence but does not offer the full decentralization and consensus properties of a distributed blockchain network. In conclusion, the project presents a structured approach to enhancing secure communication through the integration of modern encryption techniques and tamper-evident recordkeeping. It provides a foundation for further development and evaluation of secure messaging applications. Nevertheless, the actual performance and effectiveness of the implementation must be established through functional testing, security testing, and measured performance evaluation before quantitative claims can be made about its success.

REFERENCES

1. Barker, E. (2020). Recommendation for Key Management: Part 1 – General (NIST Special Publication 800-57 Part 1, Revision 5). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-57pt1r5
2. Dworkin, M. (2007). Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC (NIST Special Publication 800-38D). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-38D
3. National Institute of Standards and Technology. (2023). Advanced Encryption Standard (AES) (FIPS PUB 197, updated edition). U.S. Department of Commerce. https://doi.org/10.6028/NIST.FIPS.197-upd1
4. Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf
5. National Institute of Standards and Technology. (2020). Recommendation for Key Management: Part 1 General. NIST Computer Security Resource Center. https://csrc.nist.gov/pubs/sp/800/57/pt1/r5/final
6. Haber, S., & Stornetta, W. S. (1991). How to time-stamp a digital document. Journal of Cryptology, 3, 99–111. https://doi.org/10.1007/BF00196791
7. Merkle, R. C. (1988). A digital signature based on a conventional encryption function. In C. Pomerance (Ed.), Advances in cryptology—CRYPTO ’87 (pp. 369–378). Springer. https://doi.org/10.1007/3-540-48184-2_32
8. National Institute of Standards and Technology. (2015). Secure hash standard (SHS) (Federal Information Processing Standards Publication 180-4). U.S. Department of Commerce. https://doi.org/10.6028/NIST.FIPS.180-4
9. National Institute of Standards and Technology. (2020). Digital signature standard (DSS) (Federal Information Processing Standards Publication 186-4). U.S. Department of Commerce. https://doi.org/10.6028/NIST.FIPS.186-4
10. Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project. https://ethereum.github.io/yellowpaper/paper.pdf