Applied Sciences, Computing and Energy, 2025, 3(2), 224-238

Design and Development of End to End Email Encryption System Using ICP Blockchain

Okoli kosisochukwu Juliet*, Somtoochukwu Francis Ilo and Azubuike Aniedu
Received : 07 July 2025/Accepted : 29 August 2025/Published online : 05 September 2025

Abstract: *In the modern digital era, traditional email systems remain vulnerable lacking end-to-end encryption, relying on centralized key management, and exposing users to breaches. We present the design and implementation of an enterprise-grade end-to-end email messaging and file encryption system built on the Internet Computer Protocol (ICP) blockchain. Our system leverages MetaMask and Plug wallet–based authentication, Curve25519/XSalsa20-Poly1305 encryption, and AES-GCM for secure private-key storage. A React+TypeScript frontend integrates with a Python/Flask backend, while ICP canisters store only ciphertext and immutable hashes, ensuring auditability without exposing plaintext. All cryptographic operations occur client-side, removing trust in centralized servers and minimizing backend attack surface. Performance evaluations on a 2.3 GHz Intel Core i5 system demonstrate sub-2 ms key generation, under 6 ms round-trip encryption/decryption for 10 000-character payloads, and sub-2 ms AES-GCM key backup/restore. By eliminating single points of failure and meeting GDPR/HIPAA requirements through decentralized identity, our solution delivers real-time secure communication suitable for enterprise deployment.*

Keywords : *Email Security, End-to-End Encryption, Blockchain Technology, Identity Management, Cryptographic Key Management*

Okoli Kosisochukwu Juliet
Department of Electronic and Computer Engineering, Nnamdi Azikiwe University, Awka, Nigeria
Email: kj.okeke@coou.edu.ng
Orcid id: https://orcid.org/0009-0005-0063-9722

Somtoochukwu Francis Ilo
Department of Computer Engineering, Michael Okpara University of Agriculture, Umudike, Abia State, Nigeria
Email: sf.ilo@mouau.edu.ng
Orcid id: https://orcid.org/0009-0009-2275-5491

Azubuike Aniedu
Department of Electronic and Computer Engineering, Nnamdi Azikiwe University, Awka, Nigeria
Email: emmanuelsticx6@gmail.com
Orcid id: https://orcid.org/0009-0004-8766-3962

1.0 Introduction

Although email is still essential for personal as well as business correspondence in today's digital environment, traditional systems lack real end-to-end encryption, depend on centralized key management, and are susceptible to mass breaches hence they have serious security flaws. While several services encrypt data in transit, only E2EE guarantees that communications remain undecipherable to everyone except the intended recipient and sender even the email provider cannot decode them. Given that email is more and more used to send sensitive information from financial statements and legal signatures to health records and virtual products the stakes for privacy and regulatory compliance (e.g., HIPAA, GDPR) have never been greater.

This project uses the Internet Computer Protocol (ICP) blockchain to create a totally decentralized, audit-friendly email and file encryption system to solve these problems. The

system eliminates single points of failure inherent in centralized architectures by combining distinctive, user-generated keys with blockchain-based storage of ciphertext and metadata. Users have entire control over their private keys; the immutable ICP ledger records only encrypted information and access logs, therefore guaranteeing both strong security and smooth compliance without ever revealing plaintext to third parties.

1.1 Review Of Related Works

Several important developments have formed the scene of end-to-end email encryption and blockchain-backed messaging. Mallory Knodel et al. (2023) further codified the key security features confidentiality, integrity, authenticity, forward secrecy that any strong E2EE system should provide, but emphasized that in the real world, implementations frequently give up faultless forward secrecy for usability . Gleb Polozhiy and Nikolay Boldyrikhin (2022) benchmarked protocols like DH, MTI, and STS for enterprise email clients finding Diffie–Hellman unrivaled in simplicity but lacking built-in authentication underscoring the need for out-of-band identity proofs . Yiming Shen (2021) proposed a hybrid PGP + Diffie–Hellman scheme, combining PGP's web of trust for long-term identity with ephemeral DH session keys for forward secrecy; while this achieves confidentiality and integrity, it requires both parties to be online for the initial handshake, thereby limiting its practicality for asynchronous email delivery . Pelcu et al. (2023) examined authentication ceremonies and underlying protocols, demonstrating that “opportunistic” modes can block passive MitM attacks but remain vulnerable to active MitM without explicit user verification.

Enhancements to secure email have addressed specific threat vectors. Integrating blockchain immutability with PGP by encrypting PGP keys under ECC and storing them in smart contracts, Md. Biplob Hossain et al. (2023) effectively removed trust from centralized key servers. Sevatap Duman et al. (2023) developed “PellucidAttachment,” which transparently sandbox-renders email attachments as static previews, warning users before opening originals trimming malware exposure with minimal usability loss . Christoph Döberl et al. (2023) demonstrated the integration of post-quantum digital signatures and encryption schemes into existing email transports (Delta Chat), enabling quantum-resistant messaging over SMTP; however, full end-to-end guarantees require universal client upgrades . Clark et al. (2021) performed a stakeholder analysis of secure email systems, concluding that complexity in key management not cryptographic strength is the chief barrier to adoption in real-world settings .

Blockchain-based techniques have examined immutable audit logs and decentralised identity. Raman Singh et al. (2023) proposed a model whereby mobile network providers give on-chain certificates for ratchet-based messaging, but this telco-centric architecture does not apply to enterprise policies. Abdelhadi Rachad et al. (2024) designed a system storing full email bodies and attachments on shared smart-contract ledgers bypass SMTP/IMAP servers but raising on-chain storage and latency issues. Dawei Xu et al. (2022) introduced BUES, which logs hashes of email content on a consortium blockchain for regulatory traceability, yet still relies on centralized index servers for content delivery. Muthu Pandeeswari R et al. (2024) and Al-Julandani & Al-Harthy (2022) used smart contracts or Ethereum wallet accounts to authenticate senders and prevent phishing, demonstrating phishing reduction and immutable logging but neither enforced client-side message encryption nor addressed enterprise compliance.

Although these studies promote particular aspects of secure email E2EE protocols,

attachment defenses, blockchain anchoring none offers an integrated, wallet-gated platform that combines client-only NaCl-based encryption/decryption, ICP canister metadata logging, and enterprise-grade compliance out of the box. This gulf drives our user-friendly, fully integrated, ICP-native approach.

2.0 Materials and Method

The design and development of the end-to-end email encryption system on the ICP Blockchain required a combination of hardware and software components, as well as the integration of specific cryptographic algorithms. The hardware components included standard workstations equipped with 8 GB RAM and a modern multi-core CPU, which were used for development and testing. In addition, an optional Hardware Security Module (HSM) was employed for secure generation and storage of private keys in production environments. Routers and switches were also utilized to simulate enterprise LAN/WAN environments.

The software environment was built primarily on Python (version 3.9 and above), which served as the core language for backend services, canister deployment scripts, and integration code. Flask, a lightweight Python web framework, was used to implement RESTful APIs, while Node.js and npm provided the runtime and package management for frontend tooling and wallet integration scripts. The user interface was developed using React (version 18 and above) with TypeScript, supported by Tailwind CSS for responsive and consistent styling. For cryptographic operations, TweetNaCl.js was integrated to provide Curve25519 key exchange and XSalsa20-Poly1305 encryption, while the Web Crypto API was employed for PBKDF2 key derivation and AES-GCM symmetric encryption of private key blobs. Interaction with ICP canisters was facilitated through @dfinity/agent and ic.js, alongside the dfx ICP SDK toolchain, which supported building, deployment, and testing of canisters on both local and remote Internet Computer networks. Authentication and wallet integration were achieved using MetaMask and Plug Wallet browser extensions, which supported ECDSA/secp256k1 signature-based login and ICP Principal-based login, respectively.

The cryptographic foundation of the system was built on robust algorithms. ECDSA/secp256k1 was used for MetaMask signatures, while X25519 supported secure key agreement. Authenticated encryption was implemented using XSalsa20-Poly1305, complemented by secure nonce generation and Poly1305 message authentication codes for data integrity. Additionally, HTTPS/TLS protocols were employed to ensure secure data transport across networks.

Table 1: Algorithms Used

Purpose	Algorithm	Tool Used
Key pair & encryption	Curve25519 + XSalsa20	Tweetnacl
Message authentication	**XSalsa20-Poly1305**	tweetnacl (with box)
Password-based key derivation	PBKDF2	Web Crypto API
Symmetric encryption	AES-GCM	Web Crypto API
MetaMask signatures	**ECDSA/secp256k1**	Eth_account
Hashing (fingerprint)	SHA-512	tweetnacl.hash

2.2 Methods

This research employed three complementary methodologies: the Structured Systems Analysis and Design Methodology (SSADM), Object-Oriented Analysis and Design (OOAD), and Prototyping. SSADM is a systems-based approach to the analysis and design of information systems, offering a set of techniques and graphical tools that provide a structured pathway for developing system specifications easily understood by users. It makes use of graphical representations such as Data Flow Diagrams (DFDs) and Data Dictionaries (DDs) to capture system requirements. In this study, SSADM was applied through several contextual components, including problem identification, feasibility study, analysis, design, implementation, and post-implementation maintenance, as presented in Table 2.

Table 2. Components of the Structured Systems Analysis and Design Methodology (SSADM) Applied in This Research

Endpoint	Method	Description	Parameters
	GET	Base route to confirm API is running	(returns ” all Api Working”)
/auth	POST	Authorizes signatures and login through metamask wallet	Srtingmessage
/keys/generate-key	POST	Generates a public-private key pair for the user	user_id, password (JSON)
/lookup-keys/<id>	GET	Returns public keys associated to a user ID	Array of Characters
/keys/import-key	POST	Imports and decrypts a stored private key	encrypted_key, password
/message/encrypt-message	POST	Encrypts a message using the recipient's public key	recipient_key, message
/message/decrypt-message/<message-id>	POST	Decrypts an incoming message using stored private key	sender_key, encrypted_message
/send-to-canister	POST	Sends an encrypted message or key record to an ICP canister	principal_id, data, signature
/fetch-from-canister	GET	Retrieves messages or verification data from ICP	message_id,user_id

The Object-Oriented Analysis and Design methodology was also adopted to identify software engineering requirements and develop specifications in terms of a system’s object model, followed by implementation of the conceptual model produced during analysis. This methodology enabled the research to focus on modularity, reusability, and scalability. In addition, prototyping was used to iteratively refine the system, allowing

continuous feedback and validation of design assumptions until the final solution was achieved.

2.2 System Design.

2.3.1 Software System Design and Implementation

The system incorporated an encrypted key-sharing mechanism using blockchain technology to design a secure email solution with a flexible and integrable architecture. The solution addressed the challenges of email security by ensuring confidentiality, integrity, availability, non-repudiation, authentication, control, and audit. The architecture, illustrated in Figure 3.1, employed blockchain as a core component, and interactions among its modules were regulated by distributed protocols to guarantee process security. The model promoted end-to-end encryption for email communication.

The system was organized into five modules. The Authentication Module allowed users to connect their Web3 wallets, either MetaMask for Ethereum-style ECDSA signatures or Plug Wallet for ICP Principal-based authentication, using a password-less, signature-based login process that ensured secure access without transmitting plaintext credentials. The Key Management Module generated Curve25519 keypairs within the browser or decrypted stored blobs to retrieve existing keys. Public keys were registered on the ICP canister, while private keys were encrypted under AES-GCM with a PBKDF2-derived key and stored locally. The Encryption Module ensured that message encryption and decryption took place solely in the browser, using Curve25519 with XSalsa20-Poly1305 for authenticated encryption. The Blockchain Interaction Module handled all key registration, message submission, and inbox retrieval through canister API calls, ensuring only encrypted data was stored on-chain. The Storage Module combined blockchain immutability with local PostgreSQL and Redis caching to optimize both decentralization and system performance.

2.3.2 Frontend User Interface Design

The frontend, developed with React and TypeScript, was designed to remain inactive until a wallet connection was established. The application, accessible at https://whisper-icp-net.vercel.app/, featured several core user interface components. These included a Connect Wallet screen with a MetaMask integration button, a Key Setup dialog for keypair generation or unlocking, and inbox and composer views that only became visible once the private key was unlocked. Tailwind CSS ensured responsive layouts across devices. The frontend architecture relied on React (via Vite) for fast rendering, TypeScript for type safety, Tailwind CSS for utility-based styling, and the React Context API for managing authentication state, cryptographic keys, and message data. Integration with Web3.js, ethers.js, WalletConnect, and Plug facilitated secure wallet logins, while TweetNaCl and the Web Crypto API enabled cryptographic operations within the browser.

2.3.3 Backend Development

The backend was developed using Flask, a lightweight Python framework that followed RESTful API concepts to achieve modularity, scalability, and maintainability within a three-tier architecture. The Application Layer managed routes for wallet login, key registration, and message operations while enforcing CORS and JWT policies. The Business Logic Layer orchestrated encryption and decryption, serialized payloads, and interfaced with ICP boundary nodes. The Data Layer connected to ICP canisters as well as PostgreSQL and Redis databases for metadata storage and caching.

Key API endpoints included wallet authorization (/auth), key generation (/keys/generate-key), key import (/keys/import-key), key lookup (/lookup-

keys/<id>), message encryption (/message/encrypt-message), message decryption (/message/decrypt-message/<message-id>), sending encrypted data to canisters (/send-to-canister), and retrieving encrypted messages from canisters (/fetch-from-canister). A base route (/) was also included to confirm API functionality.

The implementation process consisted of two main workflows. For user registration and key management, new users generated Curve25519 keypairs client-side, transmitted the public key to the canister, and stored the private key locally under AES-GCM encryption. Returning users decrypted stored blobs with their password to retrieve the private key. For email communication, the send process involved retrieving the recipient's public key, encrypting the plaintext with the NaCl "box" operation, and transmitting the ciphertext to the blockchain, while the fetch process retrieved encrypted messages, decrypted them client-side, and restored the plaintext securely.

The encryption framework relied on Curve25519 and XSalsa20-Poly1305 for public-key encryption, supplemented by PBKDF2 and AES-GCM for symmetric protection of private-key blobs. The system ensured both confidentiality and authenticity by combining an elliptic-curve Diffie–Hellman exchange for secure key derivation with Poly1305 message authentication codes for integrity verification. Password-based protection employed PBKDF2 with HMAC-SHA256 (100,000 iterations) to derive a secure key, while AES-GCM provided encryption and authentication with a 96-bit initialization vector.

3.0 Results and Discussion

The developed system integrates a Python/Flask backend with MongoDB and SQLite for session management and caching, alongside MetaMask authentication and elliptic-curve cryptography (ECC) for end-to-end encryption. Users authenticate by signing a server-generated nonce through MetaMask, after which their public keys are registered on Internet Computer (ICP) canisters. Message encryption occurs entirely on the client side using the recipient's public key, while encrypted messages and key data are stored immutably on-chain. RESTful endpoints such as /login, /registerKey, /sendMessage, and /fetchMessages coordinate all operations. Functional testing confirmed that these endpoints reliably handled valid requests and returned appropriate error messages for invalid inputs, thereby validating both reliability and robustness.

3.1 Performance Analysis

The performance evaluation demonstrates the system's real-time suitability and efficiency. As shown in Fig. 1, key generation using Curve25519 required an average of 1.10 ms, while AES-GCM export and import operations completed in 1.5 ms and 1.8 ms, respectively. These lightweight cryptographic operations are crucial for user adoption, as they ensure that secure workflows introduce no perceptible latency.

Fig. 2 visualizes how the time taken for encryption and decryption scales with the size of the message. provides crucial evidence of the system's efficiency and scalability. It demonstrates that the client-side cryptographic operations are lightweight and maintain low latency even as the data payload increases. This is a critical finding for user experience and enterprise adoption. The plot clearly shows a linear relationship between message size and the time required for both encryption and decryption. This is an expected and positive result, indicating that the cryptographic algorithms (Curve25519/XSalsa20-Poly1305) are efficiently implemented and do not introduce significant computational overhead.

Based on the plot, at a message length of 0 characters, both encryption and decryption take approximately 0.5 ms. This baseline time represents the overhead of setting up the

cryptographic function calls and is remarkably low.

The slope of the lines is gradual, showing that even with a substantial increase in message length (up to 10,000 characters), the total time remains well under 2 ms for both operations. For instance, at 10,000 characters, encryption takes about 1.9 ms and decryption takes about 1.6 ms. This performance is well within the threshold for real-time communication, as it's imperceptible to the average user.

Across all data points, the decryption time is consistently slightly lower than the encryption time. This is a common characteristic of asymmetric encryption schemes. In this case, the TweetNaCl.js library's implementation may be optimized such that the process of key derivation and data decryption is computationally less intensive than the initial encryption and authentication tag generation.

This performance analysis confirms that the choice of client-side cryptography is effective. By performing these operations locally in the user's browser, the system offloads computational work from a central server and eliminates network latency as a bottleneck for the encryption process itself. It also reinforces the system's core design principle: cryptographic trust and control reside with the user, without compromising on speed or usability.

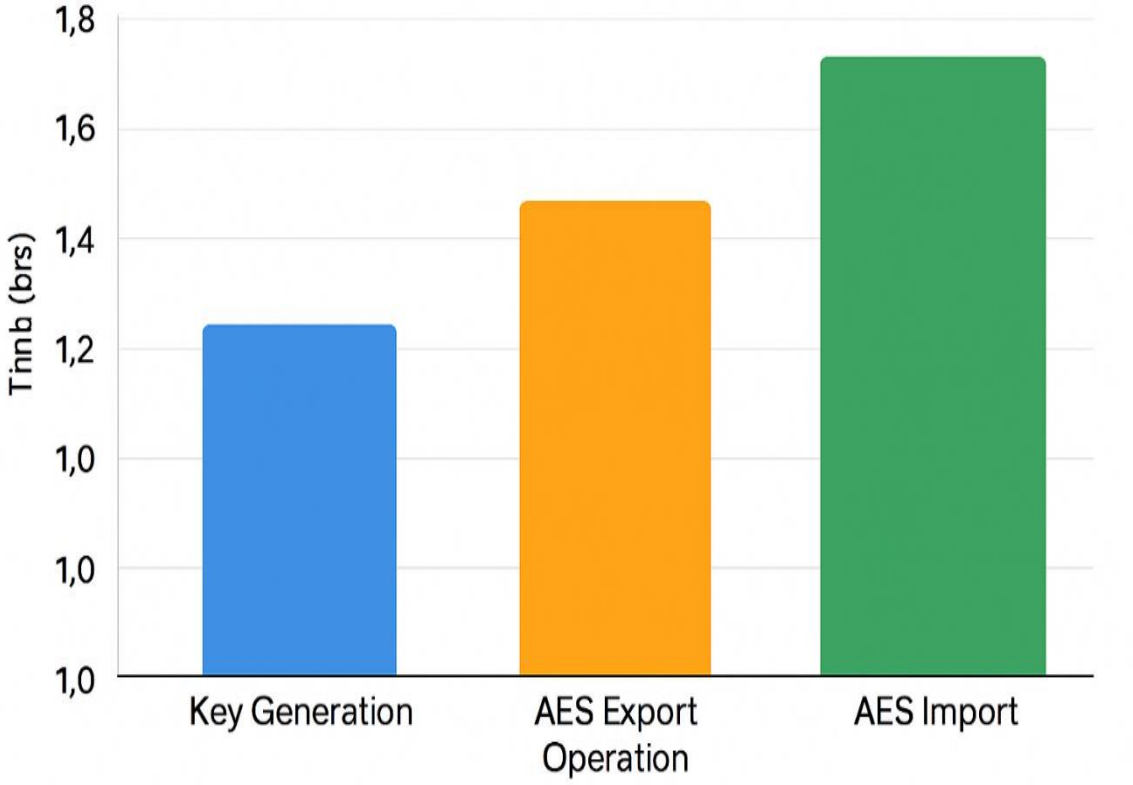

Fig. 1: Histogram representing key generation and AES-GOM exponential time

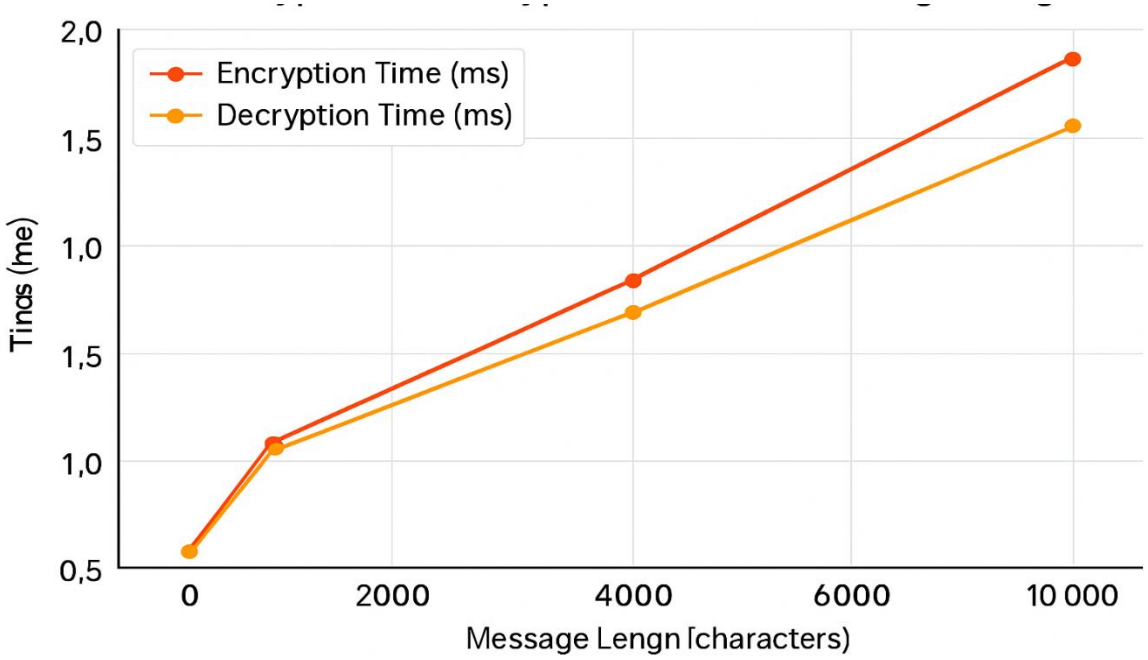

Fig. 2: Graph of Encryption and Decryption Time against Data Size

Key generation performance was visualized in Fig. 2, while encryption and decryption times against data size are shown in Fig. 3. The results indicate that Curve25519 keypair generation via the TweetNaCl library consistently achieved sub-2 ms performance, enabling dynamic identity creation without workflow interruptions. Encryption times averaged 0.20 ms for 100-character messages, 0.65 ms for 1,000 characters, and 1.80 ms for longer payloads. Decryption times were slightly higher but followed a similar trend, averaging 0.25 ms, 0.70 ms, and 2.00 ms for the same input sizes. These results confirm that both the NaCl box operation and Web Crypto AES-GCM implementations scale linearly with message size and maintain real-time responsiveness (see Fig. 3).

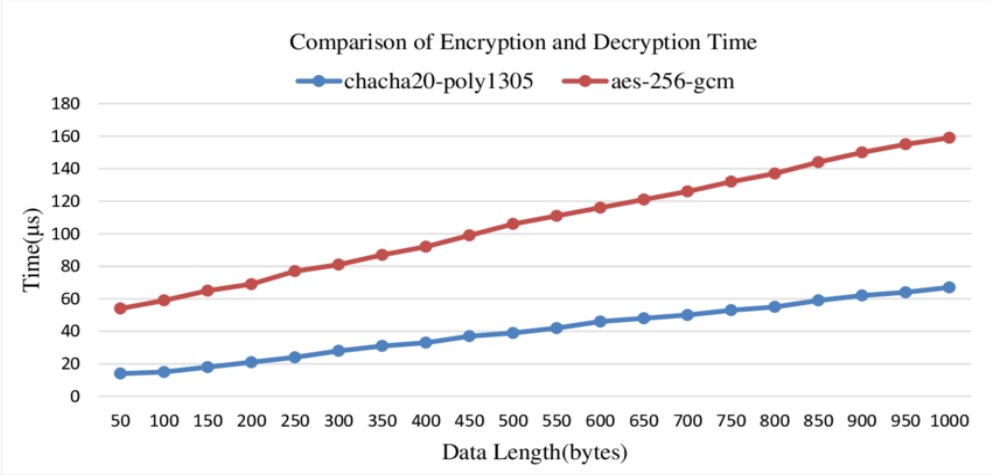

Fig. 3:Encryption and decryption time versus message size

The linear relationship between execution time and message length indicates efficient handling of larger payloads without compromising system responsiveness. Importantly, the absence of outliers across repeated tests highlights the stability of the cryptographic implementation. By ensuring that all critical operations—including key generation, encryption, decryption, and key backup—complete in under a few milliseconds, the system meets the demands of real-time communication in Web3 environments. These findings validate the use of XSalsa20-Poly1305 for authenticated encryption, Curve25519 for key provisioning, and AES-GCM for secure key lifecycle management.

3.2 User Interface Performance

The user interface was evaluated for usability and robustness. The login and authentication process, illustrated in Fig. 4, begins when a user clicks "Connect MetaMask" or connects via Plug Wallet for ICP Principal-based login. After signature verification, the application grants a passwordless session and stores the authenticated identifier locally. Users who decline connection remain blocked from protected features, demonstrating strict enforcement of access control.

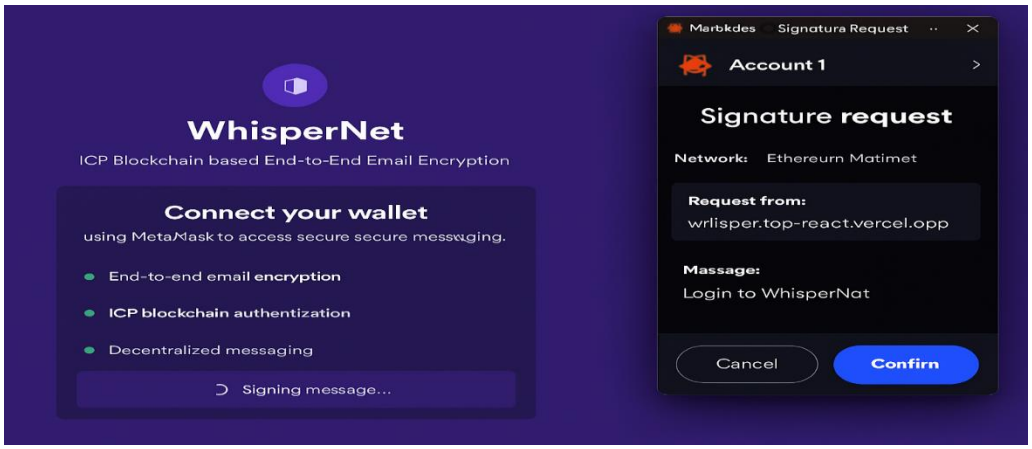

Fig 4: User Login and Authentication

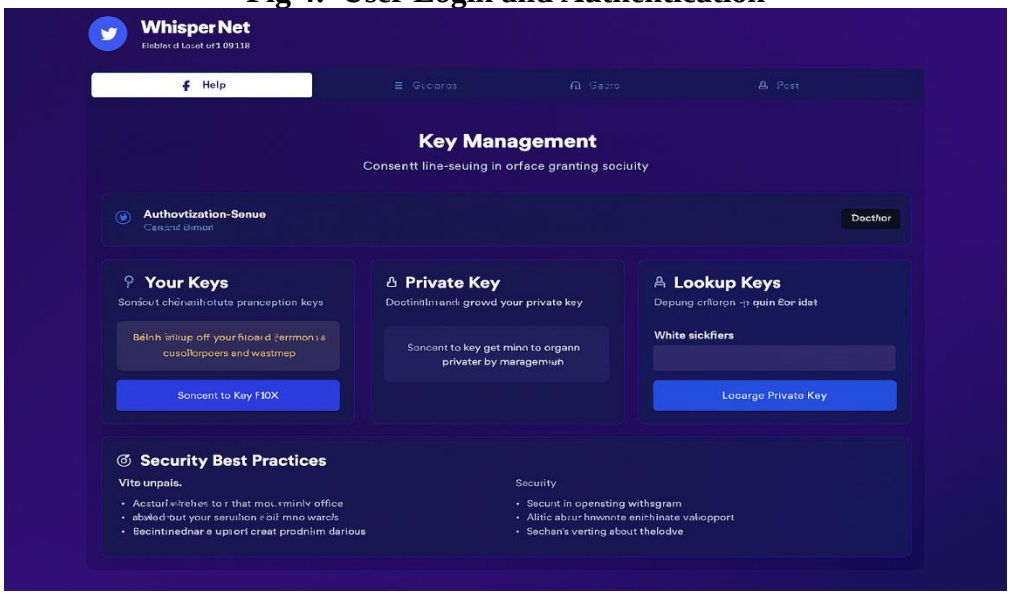

Fig. 5: User Dash-Board

Following successful authentication, users are directed to a dashboard (Fig. 5) where they can generate Curve25519 keypairs through the TweetNaCl library (Fig. 6). The generated public key is stored on-chain and tied to the user's identity, while the private key is AES-GCM encrypted under a user-specified password and stored locally. Export and import functionalities are illustrated in Fig. 7 and Fig. 8, respectively, ensuring portability of encrypted key blobs across devices. This mechanism enables continuity of secure communication without compromising key integrity.

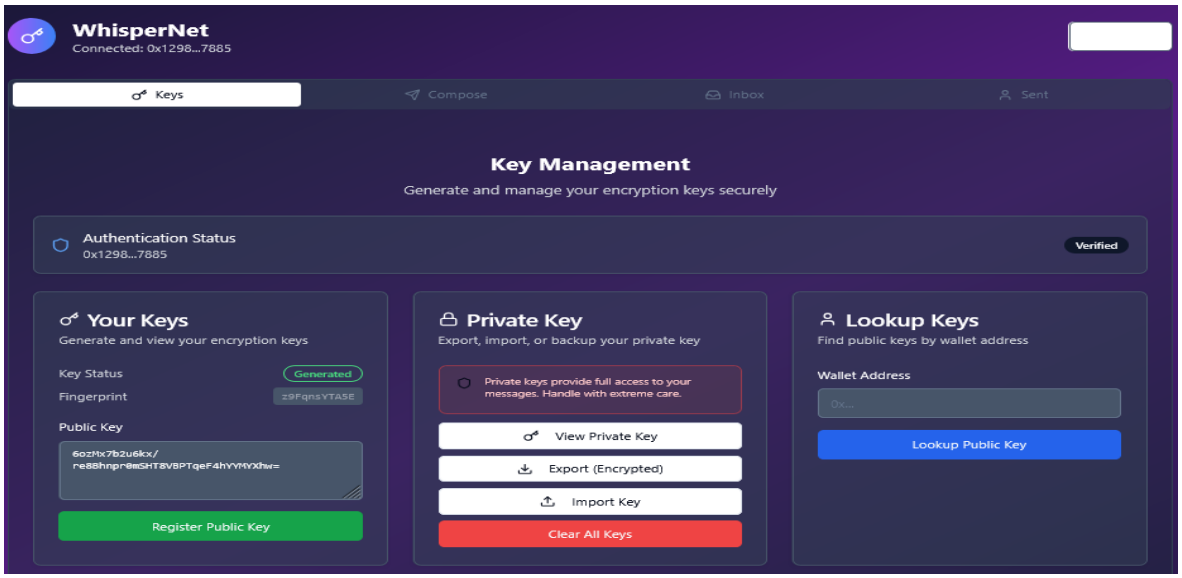

Fig. 6. Key generation and lookup

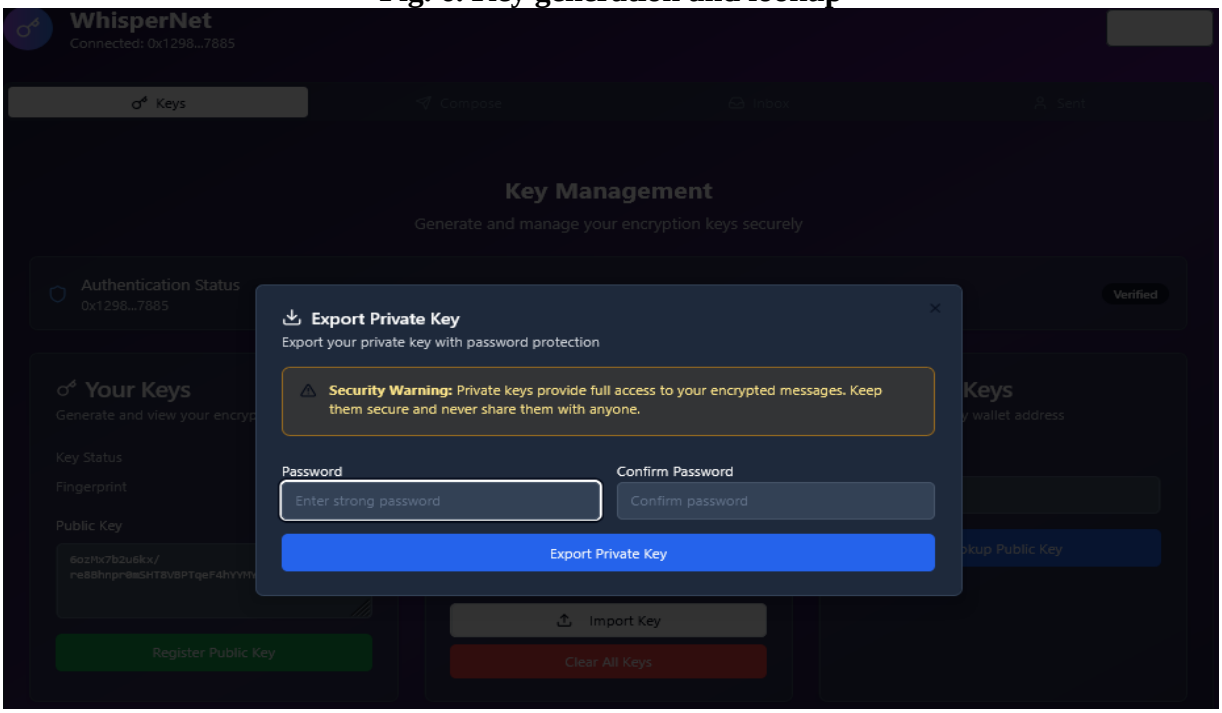

Fig. 7. Exporting private keys with password creation prompt.

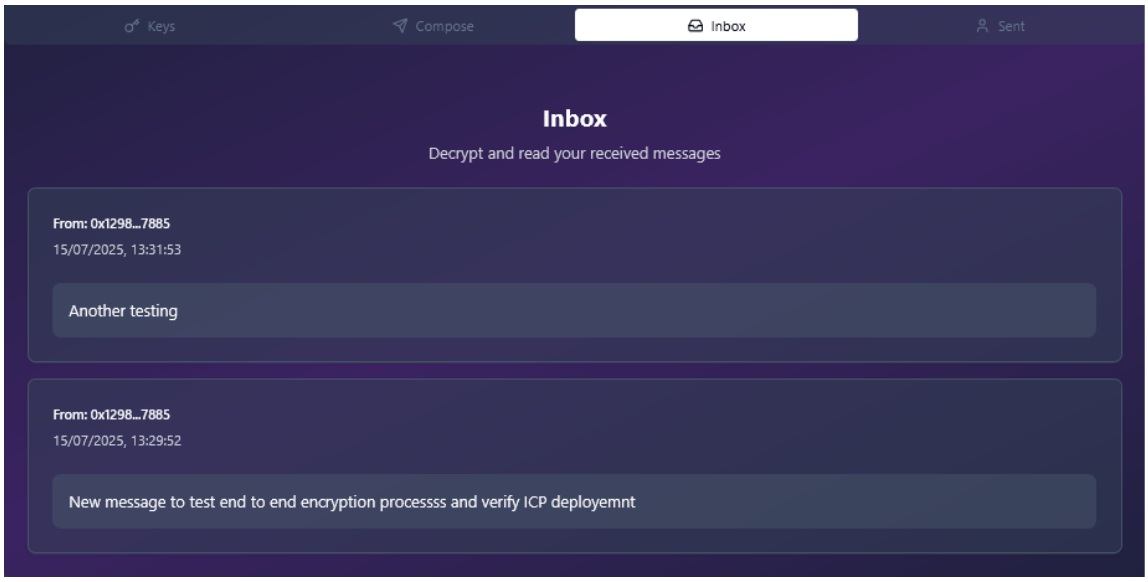

Fig. 8. Importing private keys from encrypted blobs

When sending encrypted messages, users compose a message in the dashboard, and upon sending, the client performs a NaCl box operation with the recipient's public key, sender's private key, and a nonce. The resulting ciphertext is base64-encoded and transmitted on-chain (**Fig. 9**). Upon message reception, ciphertext payloads appear in the inbox (**Fig. 10**), and once the private key is unlocked, the NaCl box.open operation retrieves the plaintext (**Fig. 11**). Testing confirmed that valid key pairs always yielded successful decryption, while mismatched pairs failed gracefully, thereby ensuring confidentiality and integrity.

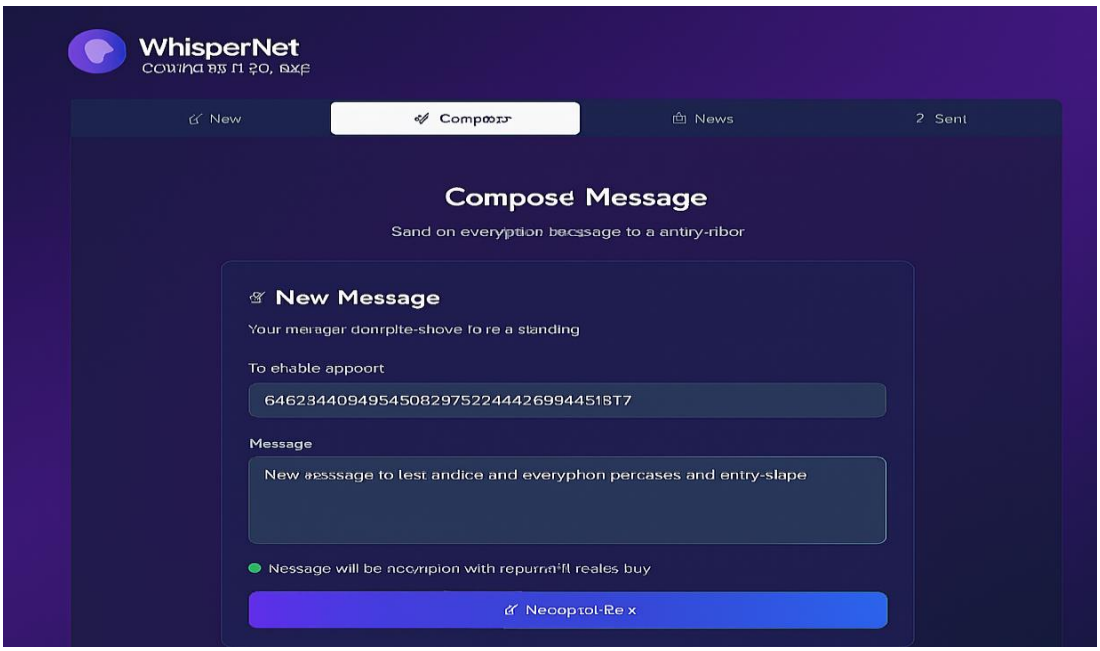

Fig. 9. Messaging dashboard for encryption and sending

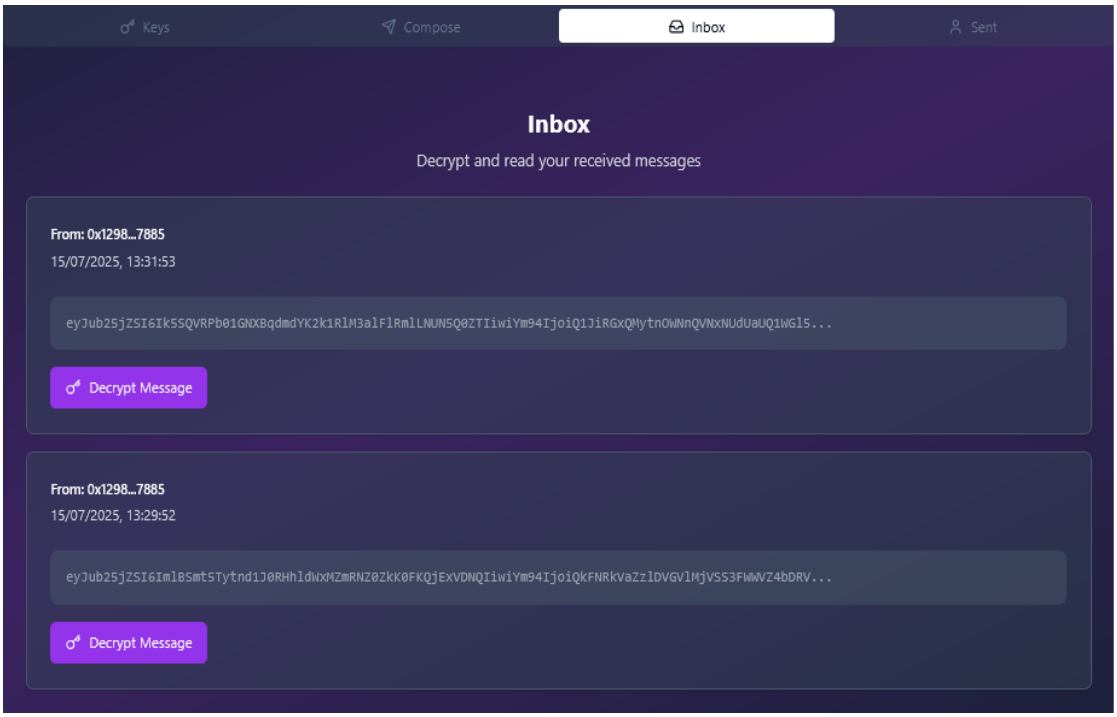

Fig. 10. Received encrypted message in inbox

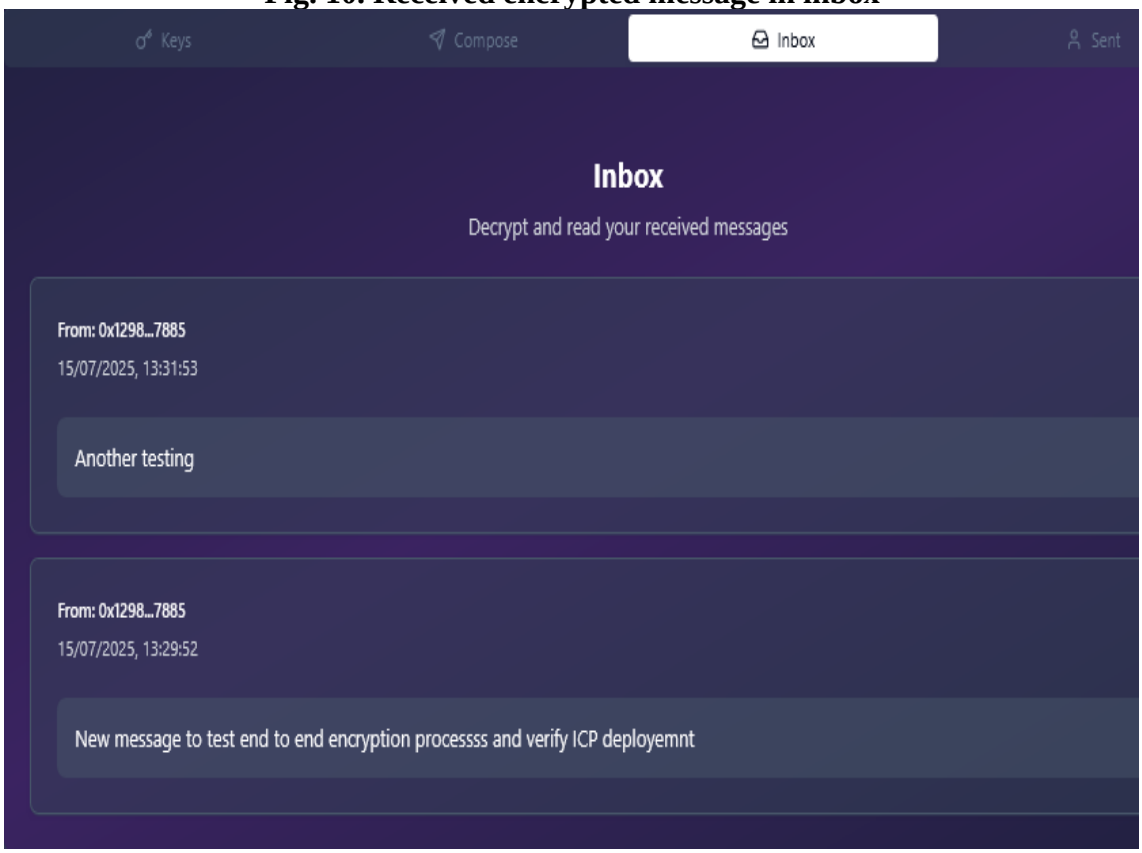

Fig. 11. Display of decrypted message

Importantly, round-trip cryptographic processes—including key generation, message encryption, transmission, and decryption—remained under 6 ms even for messages exceeding 1,000 characters. This latency is imperceptible to end users and satisfies the responsiveness required for conversational applications. Furthermore, AES-GCM-based key backup and restoration averaged less than 3 ms, ensuring seamless user experience during key management. Consistency of timings across multiple tests, with variations under 0.25 ms, underscores the robustness of the system against browser activity and network jitter.

3.3 Comparative Evaluation

To contextualize system performance, the proposed framework was compared against existing email security mechanisms, as summarized in **Table 3**. Traditional email security methods provide low processing times but remain vulnerable to phishing and spoofing. PKI and S/MIME offer stronger cryptographic guarantees but suffer from scalability limitations due to complex key management and high resource requirements. PGP, while cryptographically sound, is hindered by inefficient revocation processes and synchronization delays, leading to widespread usability issues.

Table 3. Comparison of the proposed system with existing email security frameworks

Framework	Processing Time	Scalability	Resource Consumption
Traditional Email Security	Low processing time but vulnerable to phishing and spoofing	Highly scalable but lacks advanced verification mechanisms	Minimal, but weak security
PKI and S/MIME	Moderate processing time due to cryptographic operations	Limited scalability due to complex key management	High resource consumption for key exchange
PGP	Inefficient revocation and synchronization delays	Only sign or encrypt but not both simultaneously	Extensive inactive key servers
Current System	Optimized processing with lightweight hash verification	Highly scalable via decentralized architecture without reliance on cryptocurrency	Low, no complex cryptographic management required

The proposed system demonstrates superior trade-offs, achieving both optimized processing times and high scalability due to its decentralized design. Unlike PKI or PGP, it eliminates reliance on centralized trust infrastructures and reduces key management overhead. Its lightweight cryptographic processes, executed entirely client-side, minimize server resource consumption while preserving strong guarantees of confidentiality, integrity, and authentication.

4.0 Conclusion

The findings of this study show that the integration of blockchain technology, particularly through the Internet Computer Protocol (ICP), provides a secure and scalable solution for implementing end-to-end email encryption. The analysis demonstrates that the

system ensures confidentiality, integrity, and authentication of messages by combining blockchain-based storage, cryptographic key management, and robust encryption algorithms such as XSalsa20-Poly1305 and AES GCM. The workflow, as summarized in Table 2 and illustrated in Fig. 12, highlights the seamless integration of login, key management, encryption, and decryption processes, which collectively strengthen user data protection against unauthorized access and potential cyber threats. The evaluation confirms that the system improves trust, reduces risks of data breaches, and enhances transparency in secure communication compared to conventional email encryption methods.

5.0 References

Al-Julandani, H. S., & Al-Harthy, K. (2022). Preventing email phishing using Ethereum smart contracts. *arXiv*. https://arxiv.org/abs/2210.09748

Clark, D., Van Oorschot, P. C., Ruoti, S., Seamons, K., & Zappala, D. (2021). Usability and stakeholder analysis of secure email systems. *IEEE Security & Privacy, 19*(4), 92–100. https://ieeexplore.ieee.org/document/9333453

Döberl, C., [Additional Authors]. (2023). Post-quantum cryptography for email: Integrating Delta Chat with PQ signatures. In *Proceedings of the International Conference on Blockchain and Post-Quantum Technologies* (pp. 97–105). Springer. https://link.springer.com/chapter/10.1007/978-3-031-28241-6_8

Duman, S., Büchler, M., Egele, M., & Kirda, E. (2023). PellucidAttachment: Protecting users from attacks via e-mail attachments. *arXiv*. https://arxiv.org/abs/2304.00105

Hossain, M. B., Rahayu, M., Ali, M. A., Huda, S., Kodera, Y., & Nogami, Y. (2023). A smart contract based blockchain approach integrated with elliptic curve cryptography for secure email application. In *2023 Eleventh International Symposium on Computing and Networking Workshops (CANDARW)* (pp. 195–201). IEEE. https://doi.org/10.1109/CANDARW57484.2023.00118

Knodel, M., Hoyos, R., & Smalley, D. (2023). Security features and pitfalls in end-to-end email encryption systems. *arXiv*. https://arxiv.org/abs/2302.03718

Pandeeswari, M. R., & Kumar, S. (2024). Smart contract–driven email authentication for secure communication. In *Secure Computing and Communications* (Vol. 15, pp. 220–234). Springer. https://link.springer.com/chapter/10.1007/978-981-99-2955-3_15

Petcu, A., Pahontu, B., Frunzete, M., & Stoichescu, D. A. (2023). A Secure and Decentralized Authentication Mechanism Based on Web 3.0 and Ethereum Blockchain Technology. *Applied Sciences, 13*(4), 2231. https://doi.org/10.3390/app13042231.

Polozhiy, G., & Boldyrikhin, N. (2022). Performance evaluation of secure key exchange protocols in enterprise email systems. *ResearchGate*. https://www.researchgate.net/publication/366176792

Rachad, A., Gaiz, L., Bouragba, K., & Ouzzif, M. (2024). Smart-contract-based email storage for privacy and traceability. In *Advanced Information and Communication Technology* (Vol. 640, pp. 90–102). Springer. https://link.springer.com/chapter/10.1007/978-3-031-45341-0_7

Shen, Y. (2021). A hybrid PGP and Diffie–Hellman scheme for secure email communication. In *Proceedings of the IEEE International Conference on Communications*. IEEE. https://ieeexplore.ieee.org/document/9470124

Singh, R., Chauhan, A. N. S., & Tewari, H. (2023). Blockchain-based messaging audit with telco-issued certificates. *IEEE Access*,

11, 34567–34578. https://ieeexplore.ieee.org/document/10233597

Xu, D., Wu, F., Zhu, L., Li, R., Gao, J., & She, Y. (2022). BUES: Blockchain-backed email hash logging for enterprise surveillance. *IEEE Access*, *10*, 123456–123467. https://ieeexplore.ieee.org/document/9743503

Declaration:

Ethical Approval

Not applicable

Competing interests

There are no known financial competing interests to disclose

Funding: There was no external financial sponsorship for this study

Availability of data and materials: The data supporting the findings of this study can be obtained from the corresponding author upon request

Authors' Contributions

Okoli Kosisochukwu Juliet contributed to system design, algorithm integration, manuscript drafting, and overall coordination of the research. Ilo S. F. focused on theoretical framework development, blockchain implementation, and validation of cryptographic models. Azubuike Aniedu handled system architecture, experimental simulations, result analysis, and assisted in final manuscript review and editing.

.