CrypticWave: A zero-persistence ephemeral messaging system with client-side encryption*

Maryna Derkach[1,*,†], Danylo Matiuk[1,†], Inna Skarga-Bandurova[1,2,†] and Nataliya Zagorodna[1,†]

[1] Ternopil Ivan Puluj National Technical University, 56 Ruska str., 46025 Ternopil, Ukraine
[2] Oxford Brookes University, Headington Rd., Headington, Oxford, OX3 0BP, Oxford, UK

Abstract

This paper presents CrypticWave, a secure ephemeral messaging system that implements client-side authenticated encryption (AES-GCM), one-time message access, and volatile in-memory message storage. The encryption model ensures a high level of protection against tampering, as GCM provides built-in integrity verification. The system is deployed in a cloud environment using Docker containers, with a PostgreSQL database mounted on a RAM-based file system ensuring that all data is re-initialized after each restart, thereby enhancing user data protection and eliminating persistent traces. CrypticWave was tested under a threat model involving active adversaries with server access. Results show that proposed architecture significantly reduces metadata leakage and prevents message recovery after first access. System performance and usability were also evaluated through benchmarking and user testing. The findings support CrypticWave as a lightweight, privacy-preserving messaging solution suitable for sensitive information exchange in high-risk or surveillance-prone environments. The service is available at: https://www.crypticwave.tech/.

Keywords

zero-persistence, ephemeral messaging, client-side authenticated encryption, one-time message, in-memory storage

1. Introduction

Information technologies have significantly facilitated remote communication and data exchange, enhancing convenience and operational efficiency across numerous domains. However, the rapid increase in the transmission of confidential data through electronic channels poses substantial security challenges. This concern becomes particularly pronounced when dealing with sensitive data such as passwords, API keys, access tokens, banking details (account numbers, card numbers, IBAN, SWIFT), and confidential documents within finance, critical infrastructure, IoT, law, journalism, education, research, and personal communications [1]. Traditional digital communication tools, including email, messaging applications, and cloud storage, often fail to consistently meet the required security standards [2–4]. These methods usually store messages and data on centralised servers, increasing the likelihood of data breaches in the event of unauthorised access or cyberattacks [5]. Recent forensic research underscores this vulnerability, highlighting that even encrypted messaging platforms like WhatsApp, Signal, Telegram, Wickr, and Threema leave behind recoverable artefacts in memory. Such data remnants, including usernames, metadata, and occasionally message content, can be recovered through memory forensic techniques, particularly on desktop or web-based versions of these applications [6]. One way to address these vulnerabilities involves the use of one-time messaging services that automatically delete messages after viewing. Tools such as Privnote [7], OneTimeSecret [8], and One Time Chat [9] provide temporary, encrypted links or messages that self-destruct after initial access, thereby reducing the risk of unauthorised retrieval. Another notable tool, OnionShare, leverages onion routing technology

* CSDP'2025: Cyber Security and Data Protection, July 31, 2025, Lviv, Ukraine
* Corresponding author.
† These authors contributed equally.
m_derkach@tntu.edu.ua (M. Derkach); matiuk.danylo@icloud.com (D. Matiuk); iskarga-bandurova@brookes.ac.uk (I. Skarga-Bandurova); zagorodna_n@tntu.edu.u a (N. Zagorodna)
0000-0001-8977-2776 (M. Derkach); 0000-0001-5851-8433 (D. Matiuk); 0000-0003-3458-8730 (I. Skarga-Bandurova); 0000-0002-1808-835X (N. Zagorodna)

through the Tor network to ensure anonymous and secure file and message transfers [10]. These solutions have demonstrated effectiveness in protecting data within financial [11] and healthcare [12] sectors, as well as other sensitive areas. However, despite the existing advantages, most current solutions still have significant vulnerabilities. Most current one-time messaging solutions still temporarily store sensitive plaintext or encryption keys within server memory, which can be recovered using memory forensic methods. Even after viewing a message, traces of information or its metadata can remain in RAM, creating potential attack vectors, in particular through memory dumps, as confirmed by numerous studies [6]. Thus, an important question remains unresolved: how to create a service that not only instantly destroys messages after viewing, but also fundamentally prevents any traces of sensitive data from being left in the server memory, eliminating the risks of digital forensics and memory leaks? To address this query, this paper presents a basic architecture of the CrypticWave service for secure one-time encrypted messaging. CrypticWave implements the security by design principle, characterised by minimal data retention, absence of logging, and volatile storage using tmpfs. Crucially, our server architecture never stores encryption keys, but only encrypted data with the PostgreSQL database fully operating in RAM, thereby substantially mitigating forensic risks. Research in the field of optimizing cryptographic primitives, particularly in finding efficient bitsliced descriptions of S-boxes, demonstrates the potential to enhance performance and reduce resource consumption in client-side encryption [13]. Furthermore, the application of entropy-based methods for evaluating the strength of encryption algorithms enables a quantitative assessment of protection levels, which is especially critical for one-time messaging services [14]. Specifically, this paper makes the following contributions: (1) Design of a stateless, zero-persistence messaging system that ensures confidentiality using AES-GCM encryption performed entirely on the client side; (2) Implementation of volatile message storage utilising tmpfs and containerization to minimise potential memory leaks; (3) Evaluation of system performance and user trust through benchmarking and usability studies.

2. Methodology

As discussed above, current ephemeral messaging services often retain vulnerabilities due to temporary storage of encryption keys, server logs, or persistent databases structures. To address these challenges, CrypticWave implements a zero-persistence architecture characterized by: (1) Client-side AES-GCM encryption with enforced single-use message access; (2) Volatile message storage utilizing tmpfs combined with Docker containerization to ensure RAM-only data handling; (3) Immediate and automatic data deletion after first access, with no key information retained on the server; and (4) Absence of user registration or identity tracking, thereby eliminating any possibility of linking user activities.

2.1. Encryption model

Data encryption ensures that transmitted messages are rendered unreadable without a corresponding decryption key, thereby safeguarding information from malicious interception [15, 16]. The CrypticWave employs the Advanced Encryption Standard in Galois/Counter Mode (AES-GCM), an authenticated encryption mechanism recognized for its robust security guarantees and high efficiency (Table 1). AES-GCM provides simultaneous encryption and integrity verification through two main cryptographic operations, AES in Counter Mode (CTR) for encryption and GHASH Polynomial Authentication for verifying data integrity [17].

Table 1
Key components of AES-GCM algorithm

Component	Purpose
K (Key)	Encryption key (128, 192 or 256 bits)
IV (Nonce)	Unique initialization vector (96-bits)
AAD	Additional Authenticated Data (e.g., headers)
Plaintext	Unencrypted original message
Ciphertext	Encrypted message content
Authentication Tag (Tag)	128-bit hash ensuring data integrity and authenticity

AES-GCM operation begins by generating a 256-bit encryption key (K) and a 96-bit unique initialization vector (IV). Using these parameters, AES encryption in CTR mode encrypts the plaintext. Simultaneously, the GHASH function computes a polynomial-based authentication tag to check whether the data has been modified during transmission. This is done using the special GHASH hash function, which performs mathematical operations on the Galois field GF(2^{128}):

$$GHASH(H, AAD, Ciphertext) = X_{m+n+1}, \tag{1}$$

where $H = Ek(0^{128})$ denotes the hash key derived from encrypting 128 zero-bits using key; m, n represent the count of 128-bit blocks in *AAD* and *Ciphertext*, respectively, and X_i is computed as:

$$X_i = \sum_{j=1}^{i} S_j \cdot H^{i-j+1} = \begin{cases} 0, \text{for } i = 0, \\ (X_{i-1} S_i) \cdot H, \text{for } i = 1, \ldots, m+n+1, \end{cases} \tag{2}$$

Each plaintext block X_i undergoes XOR operations with CTR-generated encryption output, followed by multiplication by the hash key H. The result of the last iteration is used as the hash value GHASH.

Authenticated data (AAD) and ciphertext are individually padded to 128-bit multiples and combined into a single message S_i:

$$X_i = \begin{cases} AAD_i, \text{for } i = 1, \ldots, m-1, \\ AAD_m^* \| 0^{128-v}, \text{for } i = m, \\ Ciphertext_{i-m}, \text{for } i = m+1, \ldots, m+n-1, \\ Ciphertext_n^* \| 0^{128-u}, \text{for } i = m+n, \\ len(AAD) \| len(Ciphertext), \text{for } i = m+n+1, \end{cases} \tag{3}$$

where *len(AAD)*, *len(Ciphertext)* are 64-bit lengths of AAD and Ciphertext, respectively, v and u represent lengths of the final blocks of AAD and Ciphertext, respectively, || denotes the union of bit strings. Upon reception, the recipient, possessing the key K, decrypts the ciphertext and independently recalculates the authentication tag. If any data tampering occurs, even a single-bit change, the authentication verification fails, thus preventing unauthorized data modifications. This AES-GCM encryption model adopted by CrypticWave robustly secures data against interception, server-side vulnerabilities, and unauthorized alterations, providing strong cryptographic assurances of confidentiality and integrity.

2.2. System architecture

The CrypticWave architecture offers improvements over existing ephemeral messaging tools due to (1) fully client-side encryption, ensuring plaintext never reaches the server, (2) non-transmission

and non-storage of encryption keys on the server, (3) exclusive server-side storage of encrypted content without the ability to decrypt, (4) immediate and irreversible message deletion following a single access, (5) scalability and ease of deployment through Docker containerization, and (6) automated provisioning and management of SSL, databases, and web infrastructure. Figure 1 illustrates the system architecture, depicting the communication and data flow between components.

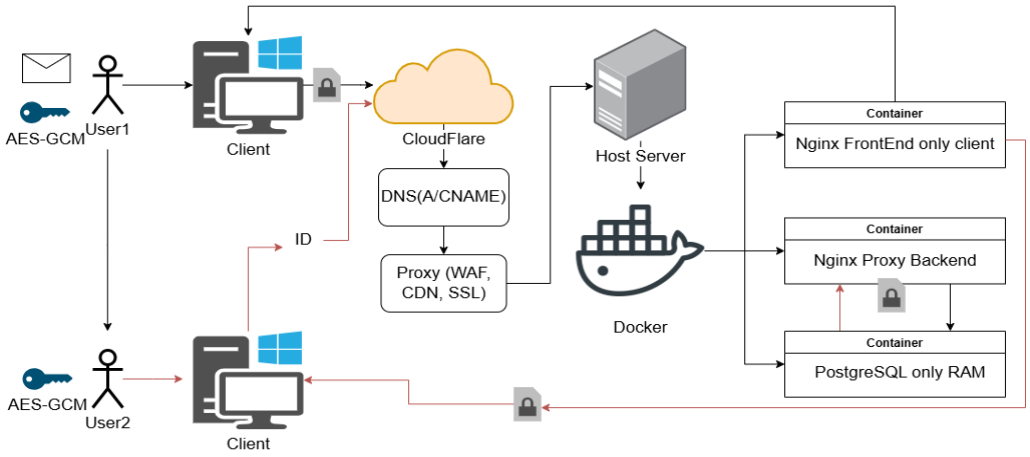

Figure 1: Base architecture of CrypticWave service

2.2.1. Operational workflow

The User1 (Sender) interacts with CrypticWave via a cross-platform front-end, inputting the message content. For each message, a unique message ID and AES encryption key are generated on the client-side. The message undergoes client-side AES-GCM encryption, producing encrypted data and an authentication tag. The encrypted message and tag are transmitted securely via the Web Crypto API to the server, generating a unique one-time-access link. User2 (Recipient) decrypts the message using the key embedded in the unique link. The client-side decryption restores plaintext and verifies the integrity via authentication tag. Immediately after successful access, the message is deleted from volatile RAM storage, ensuring zero recoverability.

2.2.2. Encryption and client-side logic

All encryption and decryption of messages happen directly on the user's device. This means that the server never sees the original message or the encryption key. When User1 creates a message, it is encrypted using a secure algorithm (AES-GCM) in the browser. After the message is encrypted, the system generates a unique link (ID) that includes the necessary information to access the message. This link is sent to User2, who can then open it and decrypt the message on their own device.

2.2.3. Network layer

CrypticWave integrates Cloudflare as a secure intermediary between users and the backend infrastructure, serving as both a reverse proxy and a comprehensive security layer. When users access the CrypticWave domain, their requests are first routed through Cloudflare's global edge network. This setup allows Cloudflare to handle DNS resolution and forward traffic to the appropriate backend services while hiding the true IP address of the host server. Cloudflare also manages SSL/TLS encryption by terminating HTTPS connections at the edge, reducing cryptographic load on the original server. To further protect the system, Cloudflare applies to a web application firewall (WAF), which detects and blocks malicious traffic. In the case of CrypticWave, the WAF is configured to filter out requests that contain SQL injection, cross-site scripting (XSS), or other common attack payloads. It also defends against brute-force attempts to

guess valid message links by rate-limiting access to sensitive endpoints. Dynamic API endpoints used to transmit and receive encrypted messages are excluded from caching to maintain message integrity and the one-time access guarantee. Access to the backend server is restricted to Cloudflare's infrastructure, preventing direct connections from the open internet. This security design reduces the attack surface and ensures that only filtered and validated traffic reaches the CrypticWave host environment.

2.2.4. Host server

The main backend of the system runs on a physical or virtual server. The different parts of the application are packaged in Docker containers, which are used to isolate individual components of the CrypticWave system ensuring consistent execution environments across different deployment platforms. Containerization also enhances fault isolation, enables microservice scaling, and simplifies orchestration through Docker Compose. The host server runs Nginx frontend only sends static files (like the webpage) to the user's browser; Nginx backend handles incoming and outgoing requests, acting as a bridge between the frontend and the database; PostgreSQL (in tmpfs) is a database, but it is stored only in the server's RAM (temporary memory). Nothing is saved on the hard disk, so once the server restarts or data is accessed and deleted, there's no way to recover it.

2.2.5. Technology stack and design considerations

Table 2 shows technology stack used to implement the CrypticWave development.

Table 2
Development technology stack

Function	Technology
Front-end	React + Vite
Front-end build	Vite
Styling	CSS Modules
Server	Node.js + Express
Database	PostgreSQL (in RAM via tmpfs)
HTTP-proxy	Nginx
Process management	PM2
SSL-certificates	Let's Encrypt + Certbot
Deployment	Docker + Docker Compose
Hosting	Proxmox container

The PostgreSQL database is deployed entirely in-memory using an RAM-based filesystem (tmpfs). This ensures that all encrypted messages reside only in volatile memory and are irretrievably lost upon access, server reboot, or container restart. This approach eliminates the risk of residual data being recovered through forensic analysis. While it limits the system's capacity to the available RAM and forfeits persistence across reboots, these are acceptable trade-offs in a design where message permanence is intentionally avoided. The full application stack is containerized using Docker and orchestrated with Docker Compose, enabling isolated, repeatable deployments across cloud infrastructure hosted on Proxmox. This architecture supports CrypticWave's core objectives: secure message lifecycle control, minimal metadata exposure, and efficient, stateless deployment. While its functionality overlaps with some Docker-native features, PM2 adds another layer of resilience and observability during development and staging without introducing significant complexity.

3. Results

The CrypticWave service was evaluated across three key dimensions: (i) security against post-compromise threats, (ii) system performance under realistic load, and (iii) usability and user trust. Where possible, we compare CrypticWave with two widely used ephemeral messaging tools: Privnote and OneTimeSecret. To the best of our knowledge, no prior academic work has published memory-dump analyses of ephemeral messaging systems such as Privnote or OneTimeSecret. At the same time, extensive memory-forensics research confirms that sensitive plaintext or encryption artifacts frequently remain in RAM post-deletion [18, 19]. Therefore, at this stage, we relied on data from literature and assumed that competing messaging tools may leave retrievable ciphertext, identifiers (IDs), or metadata (IP logs) in memory that could be recovered via forensic tools in post-compromise scenarios. For CrypticWave, we simulated a full server-compromise scenario where an attacker gains unrestricted access to the host system after a message is submitted but before it is accessed. The message payloads included synthetic sensitive data (e.g., API keys, passwords, session tokens). Using standard forensic tools such as volatility, strings, grep, and lsof, we conducted memory and disk inspections. No retrievable plaintext or associated metadata (e.g., sender IPs, message IDs, encryption keys) were found after tmpfs reset and process cleanup. Logs were also non-persistent and cleared upon restart, confirming CrypticWave's effective implementation of zero-persistence principles.

To evaluate system performance, we deployed CrypticWave in a controlled environment on a Proxmox virtual container with 2 vCPUs and 4 GB RAM. Stress tests were performed using work and custom Python clients simulating concurrent user behaviour under various load conditions (10, 50, and 100 concurrent users). Under varying load conditions, CrypticWave remained highly responsive. Cold response times, which include initial Docker container start-up, averaged 145 milliseconds, while warm response times stabilized around 48 milliseconds after boot. The system sustained a maximum throughput of approximately 620 messages per minute with 100 concurrent users, without degradation in performance. Steady-state memory usage remained around 208 MB, which includes the front-end, back-end, and the PostgreSQL database running entirely in RAM via tmpfs. This in-memory design significantly accelerated message access and deletion operations by eliminating disk I/O, and CPU usage did not exceed 11% even under peak load, confirming the system's suitability for lightweight, ephemeral messaging in real-time environments.

A usability study was conducted with 24 participants (12 technical and 12 non-technical users), split between an A/B comparison of CrypticWave vs. Privnote. Each participant completed a set of guided tasks (send, view, and delete a message), followed by a post-test survey. Table 3 presents comparison of usability metrics gathered from CrypticWave and Privnote.

Table 3
Comparative usability and trust scores between CrypticWave and Privnote

Parameter	CrypticWave	Privnote
System Usability Score (SUS)	84.6 / 100	76.2 / 100
Error rate (task failures)	0.21 per user	0.24 per user
Time to complete task	34.1 msec	37.6 msec
Reported trust (qualitative)	"High" (n=19)	"Medium" (n=13)

Participants particularly appreciated the one-time access feature, lack of registration, and clarity of encryption messages on the UI. However, a few users initially struggled to understand that the system does not retain messages at all, a usability challenge also noted in open-ended feedback.

Conclusions

This study presents CrypticWave, a stateless, zero-persistence encrypted messaging service designed around the principle of security by design. Through controlled experiments, we demonstrated that CrypticWave effectively prevents post-compromise data recovery, maintains stable performance under realistic user loads, and delivers a positive user experience with high perceived trustworthiness. These results validate the system's core architectural choices, particularly full client-side AES-GCM encryption, RAM-only storage using tmpfs, and the absence of user tracking or persistent logs. However, while CrypticWave eliminates several critical vulnerabilities found in traditional ephemeral messaging tools, some residual risks remain inherent to its architecture and operational environment. To further strengthen the security guarantees of CrypticWave, we plan to conduct controlled forensic analyses of self-hosted replicas of Privnote and OneTimeSecret and compare residual artefacts, metadata retention, and RAM persistence behaviour across platforms. Future iterations of CrypticWave will explore deploying a decentralized proxy network to reduce dependence on third-party infrastructure. We also plan to extend the bearer-link mechanism with optional multi-factor access, ephemeral DNS entries, and client-side passphrase layers to reduce the risk of token interception. A formal threat model and penetration testing campaign will be developed to evaluate CrypticWave against known attack vectors. Through these developments, we aim to establish CrypticWave not only as a practical tool for secure ephemeral messaging, but also as a reference model for zero-trust, zero-persistence communication architectures.

Declaration on Generative AI

Portions of this manuscript were prepared with the assistance of AI-based language tools, including OpenAI's ChatGPT and Grammarly, for tasks such as editing, grammar correction, and improving technical writing. All scientific content, architectural design, experimental methodology, and analysis were implemented, and validated by the authors. The AI was not used to generate data, conduct experiments, or formulate original research ideas. The authors used also Strike Plagiarism to search for possible plagiarism.

References

[1] O. Mishko, D. Matiuk, M. Derkach, Security of Remote IoT System Management by Integrating Firewall Configuration into Tunneled Traffic, Sci. J. TNTU, 115(3) (2024) 122–129.

[2] R. Chernenko, et al., Encryption Method for Systems with Limited Computing Resources, in: Cybersecurity Providing in Information and Telecommunication Systems, vol. 3288 (2022) 142–148.

[3] P. Petriv, I. Opirskyy, N. Mazur, Modern Technologies of Decentralized Databases, Authentication, and Authorization Methods, in: Cybersecurity Providing in Information and Telecommunication Systems II, vol. 3826, 2024, 60–71.

[4] Y. Shcheblanin, et al., Research of Authentication Methods in Mobile Applications, in: Cybersecurity Providing in Information and Telecommunication Systems Vol. 3421. (2023) 266–271.

[5] D. Tymoshchuk, O. Yasniy, M. Mytnyk, N. Zagorodna, V. Tymoshchuk, Detection and Classification of DDoS Flooding Attacks by Machine Learning Method, arXiv, 2024. doi:10.48550/arXiv.2412.18990

[6] A. R. Onik, J. Brown, C. Walker, I. Baggili, A Systematic Literature Review of Secure Instant Messaging Applications from a Digital Forensics Perspective, ACM Comput. Surv., 57(9) (2025). doi:10.1145/3727641

[7] K. Ermoshina, F. Musiani, Hiding from Whom? Threat Models and in-the-Making Encryption Technologies, Intermédialités / Intermediality, 32 (2018). doi:10.7202/1058473ar

[8] onetimesecret, GitHub—onetimesecret/onetimesecret: Keep Passwords and Other Sensitive Information out of Your Inboxes and Chat Logs (2025). https://github.com/onetimesecret/onetimesecret

[9] K. Kaczyński, M. Glet, One Time Chat—A Toy End-to-End Encrypted Web Messaging Service, in: Applied Cryptography and Network Security Workshops, ACNS 2024, Lecture Notes in Computer Science, vol. 14587, Springer, Cham, 2024, 136–151. doi:10.1007/978-3-031-61489-7_11

[10] D. Choudhary, The Onion Routing—The Good and the Bad, Symbiosis Institute of Computer Studies & Research (2018). doi:10.13140/RG.2.2.10181.09448

[11] R. Filchev, T. Dovramadjiev, R. Dimova, P. Parushev, Protection and Transfer of Financial Digital Data through Open Source Software, in: Intelligent Human Systems Integration (IHSI 2023): Integrating People and Intelligent Systems, AHFE Open Access, 69, 2023. doi:10.54941/ahfe1002845

[12] S. S. Lou, D. Lew, L. Xia, L. Baratta, E. Eiden, T. Kannampallil, Secure Messaging Use and Wrong-Patient Ordering Errors among Inpatient Clinicians, JAMA Netw. Open, 7(12) (2024) e2447797. doi:10.1001/jamanetworkopen.2024.47797

[13] Y. Opirskyy, O. Sovyn, O. Mykhailova, Heuristic Method of Finding Bitsliced-Description of Derivative Cryptographic S-Box, in: Proc. 2022 IEEE 16th Int. Conf. on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET), 2022, 104–109. doi:10.1109/TCSET55632.2022.9766883

[14] S. Yevseiev, S. Milevskyi, M. Melnyk, I. Opirskyy, M. Stakhiv, R. Stakhiv, Entropy Method for Assessing the Strength of Encryption Algorithms, in: Proc. 6th Int. Congr. on Human-Computer Interaction, Optimization and Robotic Applications (HORA-2024), 2024, 200165-1–200165-9.

[15] M. Derkach, O. Mishko, Using AES-256-CBC Encryption Algorithm to Store Autonomous Assistant Authentication Data, Scientific News of Dahl University, Electronic ed., 24 (2023).

[16] S. Dey, A. Ahmad, A. K. Chandravanshi, S. Das, A Secured Framework for Encrypted Messaging Service for Smart Device (Crypto-Message), in: Information Systems Design and Intelligent Applications, Advances in Intelligent Systems and Computing, 672, 2018, 415–424. doi:10.1007/978-981-10-7512-4_41

[17] S. Gueron, A New Interpretation for the GHASH Authenticator of AES-GCM, in: Cyber Security, Cryptology, and Machine Learning, CSCML 2023, Lecture Notes in Computer Science, 13914, 2023, 457–471. doi:10.1007/978-3-031-34671-2_30

[18] C. Maartmann-Moe, S. E. Thorkildsen, A. Årnes, The Persistence of Memory: Forensic Identification and Extraction of Cryptographic Keys, Digital Investigation, 6 (2009) 132–140.

[19] S. R. Davies, R. Macfarlane, W. J. Buchanan, Evaluation of Live Forensic Techniques in Ransomware Attack Mitigation, Forensic Science International: Digital Investigation, 33 (2020) 300979. doi:10.1016/j.fsidi.2020.300979