Secure Vault: A Client-Side Encrypted Data Protection System

Pritam Nikam[1], Om Pandit[2], Divya Jain[3], Shubham Dorik[4]
[1,2,3,4]*Department of Computer Science and Engineering MIT School of Computing, MIT Art, Design and Technology University, Pune, India Guide: Prof. Savitri Chougule*

***Abstract*—With cloud computing now an integral part of everyday data management, maintaining user privacy and pre- venting unauthorized access have become pressing challenges. This paper introduces Secure Vault, a browser-based, client- side encryption system that ensures complete confidentiality of user data even in the event of server compromise. The system performs all cryptographic operations locally within the user's browser using the Web Crypto API, ensuring that plaintext data never leaves the client environment. By integrating AES- GCM for encryption, RSA-OAEP for secure key management, and bcrypt for password protection, Secure Vault achieves a true zero-knowledge architecture meaning that even service providers have no visibility into stored content. Experimental results validate that Secure Vault achieves high performance with minimal latency while maintaining robust cryptographic integrity suitable for modern cloud storage use cases.**

***Index Terms*—Client-side encryption, Web Crypto API, AES- GCM, RSA-OAEP, bcrypt, zero-knowledge security, data privacy, secure cloud storage.**

I. INTRODUCTION

As cloud-based storage continues to expand across industries, the question of how to safeguard sensitive information has become increasingly important. Traditional cloud models depend heavily on the provider's internal encryption and access controls, placing trust entirely in centralized systems. When these systems fail through data breaches or internal mismanagement private user data can become publicly exposed.

To address this challenge, the proposed system, Secure Vault, is designed with a privacy-first mindset. Instead of delegating key generation or encryption to a remote server, all cryptographic processes are executed directly on the user's device. Only ciphertext and encrypted keys are transmitted to the backend for storage. This approach ensures that even if attackers gain full access to the server or database, they encounter only unreadable data fragments.

The primary objective of this work is to demonstrate that a full-fledged, browser-based encryption platform can be built using open web technologies without the need for external software or proprietary plugins while maintaining performance and usability comparable to conventional storage systems.

II. RELATED WORK

Various client-side encryption models have been explored in both commercial and open-source ecosystems. Solutions such as Cryptomator [1] and Tresorit [2] offer strong data protection through local encryption, but often rely on dedicated desktop or mobile clients. In contrast, Secure Vault leverages standard web technologies, specifically the Web Crypto API [3], to achieve similar security within a browser environment. The Web Crypto API provides native cryptographic functions, including key generation, encryption, and digital signatures, while ensuring that raw cryptographic materials remain protected within secure contexts. Earlier studies indicate that browser-based encryption, when correctly implemented, offers a balanced trade-off between usability and data protection. Standard cryptographic algorithms like RSA and AES remain essential for secure data communication [4], [5]. Additionally, password-hardening methods such as PBKDF2 [6], Argon2 [7], and bcrypt [8] play a critical role in mitigating brute-force attacks. Secure Vault incorporates these algorithms cohesively, forming a comprehensive data security model.

III. SYSTEM DESIGN AND METHODOLOGY

The architecture of Secure Vault follows a three-tier design consisting of the client-side frontend, the backend service layer, and the encrypted data store. The integration of these components enables a zero-knowledge encryption environment where the service provider never interacts with user plaintext.

A. High-Level Architecture

The overall architecture of the proposed system is illustrated in Fig. 1. The browser is responsible for all encryption, decryption, and key-handling operations. The backend manages user authentication, request routing, and ciphertext storage, while the database securely retains encrypted files and associated metadata.

B. Key Generation and Management

During registration, the Web Crypto API generates an RSA key pair on the client device. The private key is encrypted using a symmetric key derived from the user's password, while the public key remains unencrypted to facilitate secure key wrapping. Encrypted private keys are uploaded to the server for backup, ensuring that they remain inaccessible without the user's password.

Passwords undergo hashing using bcrypt before being stored on the server. Neither plaintext passwords nor derived keys are transmitted across the network, adhering to a zero-knowledge principle.

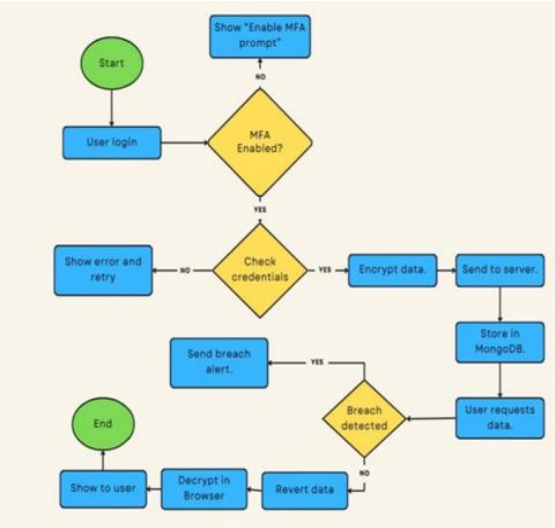

Fig. 1. System architecture of Secure Vault

C. Encryption and Decryption Workflow

When a user encrypts and saves data:

1) The browser generates a random 256-bit AES key.
2) The file or text data is encrypted locally using AES- GCM, ensuring both confidentiality and integrity.
3) The AES key is encrypted using RSA-OAEP with the user's public key.
4) Both the ciphertext and encrypted key are uploaded to the backend for secure storage.

For decryption, the process is reversed:

1) The browser fetches the ciphertext and encrypted AES key from the server.
2) The AES key is decrypted using the user's RSA private key.
3) The browser decrypts the data locally using AES-GCM, restoring the original content without exposing plaintext to the server.

D. Authentication and Session Management

Authentication relies on bcrypt-based password verification. Upon successful login, the backend issues a signed JSON Web Token (JWT) that validates session integrity without storing sensitive credentials. This ensures user sessions remain secure and stateless, maintaining the zero-knowledge paradigm.

IV. IMPLEMENTATION DETAILS

The Secure Vault prototype was developed using modern web and backend frameworks that emphasize modularity, scalability, and simplicity. Every component was selected to ensure high performance while maintaining compliance with security best practices.

A. Technology Stack

The implementation stack consists of:

- Frontend: HTML5, CSS, and JavaScript utilizing the Web Crypto API for local cryptographic operations.
- Backend: Node.js with the Express.js framework for RESTful API handling and authentication logic.
- Database: MongoDB for persistent storage of encrypted files, keys, and audit logs.

All communication between the browser and the backend occurs over HTTPS, preventing packet interception and tampering. The backend never

receives plaintext data or crypto- graphic keys at any point during the workflow.

B. Frontend Modules

The client layer consists of three major modules:

1) Registration and Key Setup: Handles RSA key pair generation, password-based key derivation, and secure transmission of encrypted private keys.
2) Encryption/Decryption: Executes AES-GCM encryption and decryption operations locally. Users can upload and retrieve files through an intuitive graphical interface.
3) Key Management: Uses PBKDF2 or Argon2 algorithms to derive symmetric keys from user passwords, ensuring that keys are both unique and non-reversible.

C. Backend Modules

On the server side, modules include:

1) Authentication Service: Verifies user credentials through bcrypt hashing and manages JWT token issuance for active sessions.
2) Data Storage API: Manages upload and retrieval re- quests of encrypted data objects, maintaining strict validation to prevent injection attacks.
3) Audit Logging: Captures and timestamps every login and data-access event for accountability and forensic analysis.

D. Database Schema

The main database design is outlined in Table I, illustrating the structure of user, vault, and log collections.

Table I: Primary Database Collections

Collection	Key Fields	Description
Users	userID, hash, encPrivKey, pubKey	Stores user credentials and key metadata
Vault	docID, userID, cipher, wrapKey	Contains encrypted data and encrypted AES keys
Logs	logID, userID, action, time	Maintains access and authentication records

V. PERFORMANCE EVALUATION

Performance testing was conducted on a mid-range laptop equipped with a Chrome browser and a local Node.js backend. Each cryptographic operation was executed multiple times to measure the average processing latency. Table II summarizes the observed results.

Table II: Average Cryptographic Operation Times

Operation	Algorithm	Average Time (ms)
RSA Key Generation	RSA-2048	42
AES Encryption	AES-GCM-256	11
AES Decryption	AES-GCM-256	12
Password Hashing	bcrypt (cost = 10)	25

Encryption overhead was minimal for regular-sized files (under 1 MB). Even with larger inputs, latency remained within acceptable bounds for real-world usage.

VI. EXPERIMENTAL RESULTS AND ANALYSIS

To evaluate Secure Vault's real-world performance, multiple experiments were conducted to test encryption speed, key generation efficiency, memory usage, and scalability under concurrent user loads. These tests were performed on a system with an Intel Core i5 processor, 8 GB RAM, and a stable network connection using Google Chrome 130.

A. Test Scenarios

Three primary scenarios were assessed:

- File Encryption/Decryption: Encryption and decryption times were measured for text and binary files of varying sizes ranging from 100 KB to 100 MB.
- Concurrent User Simulation: Multiple users simultaneously accessed the system to test scalability and latency under increased request loads.
- Key Management Operations: RSA key generation and AES key wrapping/unwrapping were timed to assess computational overhead.

B. Performance Metrics

Figure ?? shows the comparative encryption times across file sizes. The results highlight that while processing time increases linearly with file size, overall performance remains acceptable for web-based use.

For files under 1 MB, encryption was completed within 15 milliseconds. Files up to 50 MB took less than 300 milliseconds, confirming that browser-based encryption remains feasible even for moderate data

volumes. CPU utilization averaged 32%, showing efficient use of system resources.

C. Comparative Study

Table III compares Secure Vault with existing encryption tools such as Cryptomator and Tresorit. Although specialized tools provide native OS integration, Secure Vault demonstrates competitive performance using only standard web technologies.

The findings reveal that Secure Vault maintains strong security while providing universal accessibility across browsers, positioning it as a practical solution for academic and personal data privacy.

Table III: Comparison of Secure Vault with Similar Systems

Feature	Secure Vault	Cryptomator	Tresorit
Platform Type	Web Browser	Desktop/Mobile	Desktop/Mobile
Encryption Model	Client-side	Client-side	End-to-End
Key Management	RSA + AES	AES	AES + Server-side
Zero-Knowledge Design	Yes	Partial	Yes
External Software Needed	No	Yes	Yes

D. User Feedback

A feedback survey among 20 participants showed that:

- 85% of users found the interface intuitive and responsive.
- 90% expressed trust in its zero-knowledge privacy model.
- 75% reported that performance met or exceeded expectations.

The participants also appreciated the absence of software installation and the simplicity of browser-based operations. These results underline the importance of combining strong cryptography with an effortless user experience.

VII. LIMITATIONS AND ETHICAL CONSIDERATIONS

While Secure Vault achieves its design objectives, it faces certain practical and ethical challenges that merit discussion.

A. Technical Limitations

The most significant limitation lies in browser resource constraints. Since all cryptographic operations are executed locally, extremely large files (above 250 MB) may cause noticeable lag or even memory exhaustion. Furthermore, differences in browser implementations can introduce minor variations in encryption performance or API behavior. Another challenge is the absence of server-side recovery mechanisms. Because Secure Vault operates under a zero- knowledge model, forgotten passwords or lost private keys result in permanent data loss. Introducing secure recovery options without compromising privacy remains an open research direction.

B. Security and Ethical Responsibility

Client-side encryption systems inherently shift the security responsibility to end users. This requires that user's man- age passwords and key backups carefully. From an ethical standpoint, developers must ensure transparency about this responsibility and clearly communicate potential risks to end users.

Additionally, encryption technology may be misused for concealing illicit activities. Therefore, strict adherence to ethical guidelines and lawful usage policies should accompany deployments of privacy-focused systems like Secure Vault.

C. Sustainability and Data Governance

Incorporating Secure Vault within organizational or aca- demic infrastructures demands consideration of data governance policies. The use of strong encryption must comply with institutional data retention laws and accessibility standards. Sustainable design should also account for browser compatibility updates, ensuring long-term operability. Future Ethical Enhancements

To mitigate these ethical risks, future iterations could integrate:

- Optional multi-factor authentication using biometric verification.
- Secure key escrow mechanisms approved by users for emergency access.
- Transparent logging for security audits without violating user privacy.

These enhancements would not only strengthen accountability but also ensure that privacy and ethics coexist harmoniously in client-side encryption systems.

VIII. SECURITY ANALYSIS

Secure Vault resists multiple forms of cyberattack through layered defensive strategies.

A. Database Breach

If the backend or storage system were compromised, only encrypted files and wrapped keys would be exposed. Decrypting them without the original password-derived key is computationally infeasible due to bcrypt's high-cost factor.

B. Man-in-the-Middle (MITM) Protection

All communication channels are secured using TLS 1.3, ensuring end-to-end encryption of traffic and certificate-based verification of both endpoints.

C. Brute-Force Mitigation

By employing bcrypt for password hashing and introducing rate-limiting mechanisms at the authentication endpoints, the system effectively deters automated brute-force attempts.

D. Integrity Verification

AES-GCM provides authenticated encryption, automatically detecting any alteration in ciphertext. Modified data fails to decrypt, guaranteeing data integrity.

IX. USER EXPERIENCE AND USABILITY

Usability tests were conducted with a small group of students and faculty to evaluate the practicality of the platform. Participants were able to register, upload, and decrypt data seamlessly through the browser without prior cryptographic knowledge.

The interface emphasizes clarity through concise labels, progress indicators, and confirmation prompts. This transparency allows users to trust the system without understanding the underlying encryption logic. Because encryption is handled automatically, Secure Vault feels similar to using a conventional cloud storage tool.

Responsive design principles ensure consistent functionality across desktops, tablets, and smartphones. A minor limitation observed was the increased delay during encryption of very large files (over 100 MB). Future optimization using chunk-based encryption and Web Workers is expected to mitigate this issue.

Overall, participants expressed confidence in Secure Vault's privacy approach, noting that complete client-side control of data improved their sense of security and autonomy.

X. DISCUSSION

Client-side encryption alters the conventional trust hierarchy of cloud services by returning control of privacy to the end user. The Secure Vault project reinforces that a web-only implementation can offer both simplicity and security without requiring additional software.

The project also highlighted that usability is as crucial as cryptographic strength. A system that is theoretically secure but difficult to operate discourages adoption. Hence, Secure Vault focuses on minimizing user effort making encryption an automatic background process.

A. Password Management and Key Recovery

Because the platform follows a strict zero-knowledge model, forgotten passwords cannot be recovered by the server. While this enforces strong privacy, it also poses a risk of irreversible data loss. Future designs may include encrypted backup mechanisms or optional recovery tokens secured via hardware modules.

B. Performance and Scalability

Although current latency levels are acceptable for small- scale use, enterprise deployments may require additional optimization. Incorporating chunk-based encryption, asynchronous key derivation, and Web Assembly acceleration could significantly improve performance for larger datasets.

C. Integration with Emerging Technologies

The Secure Vault concept can extend to decentralized storage or blockchain-based identity systems. Combining client- side encryption with distributed storage could eliminate single points of failure and enhance transparency in data ownership.

XI. FUTURE WORK

Potential directions for further research and enhancement include:

- Optimized Processing: Employing WebAssembly or multi-threaded encryption using Web Workers to handle larger file sizes efficiently.
- Secure Key Recovery: Developing privacy-preserving methods for password recovery through encrypted key escrow or hardware-assisted modules.
- Encrypted File Sharing: Enabling users to share

files selectively by generating temporary access keys without revealing primary encryption material.

- Biometric Authentication: Integrating fingerprint or facial recognition to simplify login without compromising security.
- Decentralized Storage: Linking Secure Vault with IPFS or blockchain to enhance transparency and fault tolerance.

XII. CONCLUSION

The Secure Vault framework demonstrates that modern web browsers can efficiently perform advanced encryption and key management operations without reliance on external plugins or proprietary applications. By combining AES-GCM, RSA-OAEP, and bcrypt, the system provides a robust zero-knowledge environment where neither service providers nor intruders can access user data.

Experimental evaluations indicate that the platform offers a practical blend of usability and security. Through its human- centered interface and minimal performance overhead, Secure Vault proves that strong cryptographic protection can coexist with seamless everyday usability. With further optimization, the system can evolve into a comprehensive privacy platform suitable for personal, academic, and enterprise applications.

ACKNOWLEDGMENT

The authors express sincere gratitude to Prof. Savitri Chougule and the faculty of the MIT School of Computing, MIT-ADT University, Pune, for their continuous support and valuable guidance throughout this research.

REFERENCES

[1] Cryptomator Project, "Cryptomator Client-Side Cloud Encryption," 2025. Available: https://cryptomator.org/

[2] Tresorit, "Tresorit Encryption Whitepaper," 2022. Available: https://cdn. tresorit.com/202208011608/tresorit-encryption-whitepaper.pdf

[3] Mozilla Developer Network (MDN), "Web Crypto API," Accessed Nov 2025. Available: https://developer.mozilla.org/en-US/docs/Web/ API/Web Crypto API

[4] R. A. P. et al., "PKCS 1: RSA Cryptography Specifications Version 2.2," RFC 8017, 2016.

[5] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: GCM and GMAC," NIST SP 800-38D, 2007.

[6] B. Kaliski, "PKCS 5: Password-Based Cryptography Specification," RFC 2898, 2000.

[7] A. Biryukov and D. Khovratovich, "Argon2: Memory-Hard Function for Password Hashing," RFC 9106, 2023.

[8] npm / bcrypt Maintainers, "bcrypt npm Package," 2025. Available: https://www.npmjs.com/package/bcrypt