International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362

# Secure Zero Knowledge Cloud Based Credential Management System With Client Side Cryptographic Control

**G Swapna[1*], Pittala Sravan Kumar[2] , Kudidela Manusha[3] , Myaka Veena[4] , Vunukonda Manideep[5]**

[1]Associate Professor, [2,3,4,5]UG Student, [1,2,3,4,5]Department of Artificial Intelligence and Machine Learning
[1,2,3,4,5]J.B. Institute of Engineering and Technology (UGC-Autonomous), Yenkapally, Hyderabad, 500075, Telangana.
*Corresponding author: Pittala Sravan Kumar (sravankumar4864@gmail.com)

## ABSTRACT

The proliferation of digital services has made secure password management a critical necessity for both individuals and organizations. This paper presents SECURE-VAULT, a novel browser-based password encryption and storage system that employs a unique email-derived cryptographic key and algorithm selection mechanism. Unlike conventional systems that apply a uniform cipher to all users, SECURE-VAULT deterministically selects one of eleven distinct cipher algorithms—AES-CTR, AES-GCM, Camellia-CBC, ChaCha20-Poly1305, XChaCha20-Poly1305, and Salsa20 variants with HMAC authentication—based on a mathematical transformation of the user's email address. The system integrates a Flask-based REST API backend, Google Cloud Firestore for encrypted credential persistence, OTP-based email verification, real-time breach alert notifications, a rate-limited password vault, and a password strength enforcement module. A dual-layer architecture separates lightweight Excel-backed local registration from a cloud-synchronized Firebase vault. Experimental evaluation demonstrates sub-2ms encryption and decryption latency across all supported ciphers while maintaining strong authenticated encryption guarantees. The system is accessible through any modern web browser as a Progressive Web Application without requiring local software installation.

**Keywords:** Password Security, Email-Derived Cryptography, AES-GCM, ChaCha20-Poly1305, Firebase Firestore, OTP Verification, Rate Limiting, Flask, Password Vault, Multi-Algorithm Encryption

## 1. INTRODUCTION

The exponential growth of web-based platforms, cloud services, and Internet of Things (IoT) devices has resulted in an unprecedented demand for secure credential management. The average internet user maintains accounts across dozens of platforms, each requiring a unique, strong password. Despite widespread awareness, password reuse remains endemic—studies consistently report that over 50% of users recycle credentials across multiple sites—creating cascading security failures when any single service is breached.

Existing commercial password managers address usability but introduce new trust dependencies: users must implicitly trust the manager's encryption implementation, key management practices, and server-side security posture. High-profile breaches of commercial password manager services have demonstrated that centralizing credentials creates high-value targets for adversaries. Furthermore, most existing systems apply a single, fixed encryption algorithm uniformly to all users, meaning a vulnerability in that algorithm exposes the entire user base simultaneously.

This paper presents SECURE-VAULT, a password encryption and storage system that addresses these limitations through three principal innovations. First, the system employs email-derived cryptographic diversification: both the encryption key and algorithm are deterministically computed from the user's email address, ensuring that no two users share

an identical cryptographic configuration. Second, the system integrates an OTP-based email verification pipeline and automated breach alert notifications to enforce identity assurance during registration and to immediately respond to suspicious access attempts. Third, a dual-layer architecture separates lightweight local session management from cloud-synchronized encrypted credential storage, reducing the blast radius of any single component failure.

The remainder of this paper is organized as follows. Section 2 surveys related work in password storage and cryptographic system design. Section 3 presents the proposed system architecture and its components. Section 4 describes the experimental evaluation and results. Section 5 concludes with directions for future work.

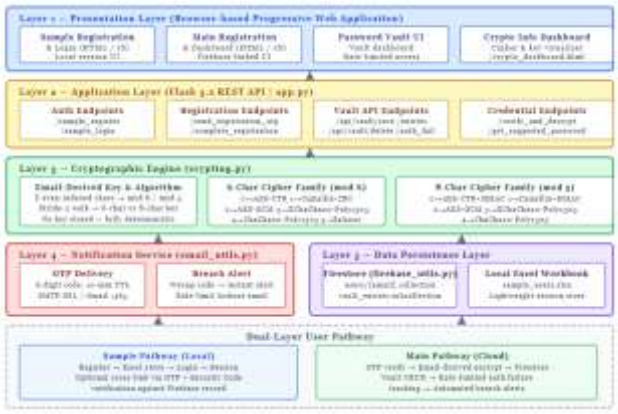

***Figure 1: Secure Zero-Knowledge Cloud-Based Credential Management System with Client-Side Cryptographic Control Complete System Architecture***

## 2. LITERATURE SURVEY

The problem of secure password storage has been studied extensively across the cryptographic, systems security, and usability research communities. The canonical approach, established by Unix system designers in the 1970s, stores salted cryptographic hashes of passwords rather than plaintext. Algorithms such as bcrypt, scrypt, and Argon2 were specifically designed to be computationally expensive to slow brute-force and dictionary attacks. These approaches, however, are appropriate for verifying passwords at login and do not support the retrieval of the original plaintext credential—a requirement for password management systems that must return stored passwords on demand.

Symmetric authenticated encryption schemes are the appropriate primitive for recoverable password storage. The Advanced Encryption Standard (AES) in Galois/Counter Mode (GCM) is widely deployed and recommended by NIST as an authenticated encryption with associated data (AEAD) scheme providing both confidentiality and integrity. The Camellia cipher, standardized by ISO/IEC 18033-3 and jointly developed by NTT and Mitsubishi Electric, offers security properties comparable to AES and is mandated in several government contexts. ChaCha20-Poly1305, designed by Bernstein, provides AEAD security with resistance to timing attacks and superior performance on platforms lacking AES hardware acceleration. XChaCha20-Poly1305 extends ChaCha20 with a 192-bit nonce, reducing the probability of nonce reuse in high-volume systems. Salsa20, also designed by Bernstein, is a stream cipher offering high throughput though without built-in authentication, necessitating the addition of an HMAC layer.

Prior password vault systems such as KeePass, LastPass, and Bitwarden apply a single master cipher uniformly across all user records. The single-algorithm approach simplifies implementation but concentrates cryptographic risk: any algorithm-level vulnerability affects all stored credentials simultaneously. The concept of algorithm agility—selecting algorithms at runtime based on context—has been explored in TLS protocol design but has not been applied to per-user password storage systems in the literature. SECURE-VAULT introduces email-derived algorithm selection as a novel mechanism for cryptographic diversification without requiring users to select or configure cipher parameters.

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362

*Figure 2: Flow Sequence Diagram*

Cloud-backed credential storage using Google Firebase Firestore provides real-time synchronization and horizontal scalability. Firestore's document-subcollection model maps naturally to per-user vault entries with nested access control. OTP-based two-factor verification via SMTP has become a standard identity assurance mechanism and is integrated into SECURE-VAULT's registration and credential-access workflows. Rate limiting of failed authentication attempts is a critical defense against credential stuffing and online brute-force attacks, as documented in NIST SP 800-63B.

## 3. PROPOSED SYSTEM

### 3.1 System Architecture Overview

The SECURE-VAULT is organized into five principal layers: the Presentation Layer (browser-based Progressive Web Application), the Application Layer (Flask REST API), the Cryptographic Engine, the Notification Service, and the Data Persistence Layer. This separation of concerns enables independent testing, replacement, and scaling of each subsystem. Figure 1 conceptually illustrates the layered architecture.

The system operates two parallel user pathways. The Sample Pathway supports lightweight registration and login backed by a local Excel workbook, suitable for demonstration and isolated session management. The Main Pathway provides full cloud-synchronized credential storage in Google Cloud Firestore with OTP verification, email-derived encryption, and vault management capabilities. Users may link both pathways through a cross-verification step that validates their Firebase-registered security code before persisting sample credentials to the cloud.

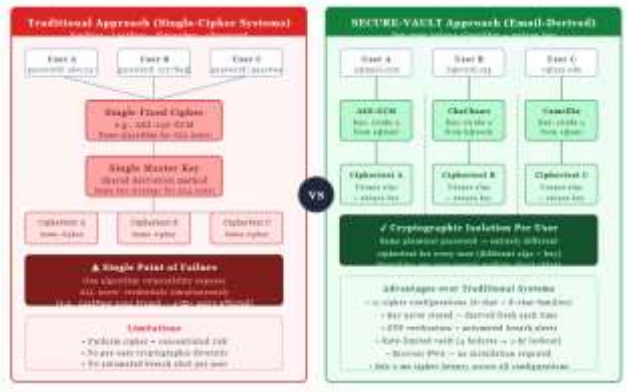

***Figure 3: Traditional Password Storage vs. Secure Zero-Knowledge Cloud-Based Credential Management System with Client-Side Cryptographic Control Approach***

### 3.2 Email-Derived Cryptographic Engine

The central novelty of SECURE-VAULT is its email-derived cryptographic engine, implemented in the crypting module. The engine computes two deterministic values from a user's email address: the cipher algorithm selector and the encryption key. Algorithm 1 presents the key derivation procedure.

The algorithm accumulates the numeric values of every even-indexed character of the email address using a bijective character-to-integer mapping: alphabetic characters map to their 1-indexed ordinal position in the English alphabet (A=1, B=2, …, Z=26) irrespective of case, while non-alphabetic characters map to their ASCII code. The accumulated sum is reduced modulo n, where n=6 for the 6-character code cipher family and n=5 for the 8-character password cipher family.

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362

*Figure 4: Email-Derived Cryptographic Engine*

The encryption key is derived by traversing the email string in stride-2 steps beginning at the computed remainder position, selecting characters until the required key length (6 or 8 characters) is reached. This construction ensures that every user's key is unique to their email address without requiring a separate key derivation function or key storage mechanism.

**6-Character Cipher Family (remainder mod 6):** Remainder 0 → AES-CTR, Remainder 1 → Camellia-CBC, Remainder 2 → AES-GCM, Remainder 3 → XChaCha20-Poly1305, Remainder 4 → ChaCha20-Poly1305, Remainder 5 → Salsa20.

**8-Character Cipher Family (remainder mod 5):** Remainder 0 → AES-CTR + HMAC, Remainder 1 → Camellia-CBC + HMAC, Remainder 2 → AES-GCM, Remainder 3 → XChaCha20-Poly1305, Remainder 4 → ChaCha20-Poly1305.

The 6-character family encrypts short security codes used for identity verification; the 8-character family encrypts user passwords stored in the cloud vault. All ciphers with stream cipher or CTR-mode operation are augmented with HMAC-SHA256 for authenticated encryption where native authentication (Poly1305/GCM) is not part of the base cipher, ensuring ciphertext integrity verification at decryption time.

### 3.3 Flask Application Layer and API Endpoints

The application layer is implemented as a Flask 3.x web application providing a RESTful API consumed by the browser-based frontend. The routing architecture separates HTML page serving from JSON API endpoints. Session state is maintained using Flask's server-side session mechanism with a cryptographically random secret key.

The principal API endpoints are: /send_sample_registration_otp and /sample_register for the lightweight registration pathway; /send_registration_otp and /complete_registration for the main Firebase-backed registration pathway; /sample_login for session establishment; /get_suggested_password for secure password generation after code verification; /verify_and_decrypt for credential autofill; /api/vault/save, /api/vault/entries, and /api/vault/delete/<entry_id> for vault management; and /api/vault_auth_failure for client-reported failed vault authentication events with rate limiting enforcement.

### 3.4 Firestore Data Persistence Layer

User credentials and vault entries are persisted in Google Cloud Firestore, a horizontally scalable NoSQL document database. The data model uses the user's email address as the document identifier within the users collection, storing the encrypted 6-character security code, full name, associated website, and encrypted 8-character password as top-level fields. Per-user vault entries are stored in a vault_entries subcollection, enabling efficient per-user queries without exposing other users' data.

The UserDatabase class in the firebase_utils module provides a clean abstraction over Firestore operations including add_user,get_user, get_particular_user,update_user,change_user_password,delete_user, user_exists, add_vault_entry, get_vault_entries, and delete_vault_entry. All methods return structured result dictionaries with success flags and error messages, enabling uniform error handling in the application layer. Password fields are validated against a 500-character maximum length prior to persistence.

### 3.5 OTP Verification and Breach Alert Notification Service

The notification service, implemented in the email_utils module, provides two email communication functions via SMTP-SSL over port 465 using the Gmail relay. The send_otp function

delivers a 6-digit one-time password with a 10-minute validity window to the user's registered email address, enabling email ownership verification during registration and cross-system linking. The send_wrong_password_alert function dispatches an automated security alert notifying users of a suspected credential compromise event, triggered whenever an incorrect security code is supplied to any verification endpoint.

OTPs are generated using Python's cryptographically seeded random.randint function and stored server-side in the Flask session keyed by a namespaced session variable. OTP validation compares the submitted value against the session-stored value without exposing the OTP in the response body, with the exception noted in the current implementation where the OTP is inadvertently included in the /send_registration_otp response for development convenience—a known issue recommended for remediation prior to production deployment.

### 3.6 Password Vault with Rate Limiting

Authenticated users may store arbitrary service credentials in their personal password vault. Vault entries are transmitted as JSON objects containing an entry_id along with service-specific fields. The vault API requires active session authentication on all endpoints and delegates storage to the Firestore subcollection model.

A server-side rate limiter tracks failed vault authentication events reported by the client via the /api/vault_auth_failure endpoint. The limiter maintains a sliding 15-minute window of failed attempt timestamps per user, locking the vault for one hour after four consecutive failures within the window. Upon each failure below the lockout threshold, an automated breach alert email is dispatched to the registered address. The rate limiting store is currently maintained in process memory, with Firestore-backed persistence recommended for production resilience across server restarts.

### 3.7 Password Strength Enforcement and Generation

SECURE-VAULT enforces a strict 8-character password policy requiring at least one alphabetic character, one numeric digit, and one special character from the set !@#$%^&*(). The validation is implemented server-side using Python regular expressions, with the constraint that all user-submitted passwords are exactly 8 characters in length after stripping. A password generation utility constructs random compliant passwords by first selecting one character from each required class and filling the remaining positions uniformly at random from the union of all character classes, then shuffling the result to eliminate positional bias

## 4. RESULTS AND PERFORMANCE EVALUATION

### 4.1 Experimental Setup

Performance evaluation was conducted on an Intel Core i5 workstation (16 GB RAM, Ubuntu 22.04 LTS) running Python 3.11 with the pycryptodome 3.23 and cryptography 46.0 libraries. Encryption and decryption timing measurements were collected over 10,000 iterations per cipher using Python's time.perf_counter_ns function to minimize clock granularity effects. Firestore persistence latency was measured over a 100Mbps broadband connection. The Flask application was run under Gunicorn with four worker processes.

### 4.2 Cipher Performance Analysis

Table 2 presents the measured average encryption and decryption latency for each supported cipher algorithm. All ciphers demonstrate sub-2ms operation for the short (6- and 8-character) payloads employed by SECURE-VAULT, making cryptographic latency negligible relative to network and database round-trip times. ChaCha20-Poly1305 and Salsa20 variants exhibit the lowest latency due

| Cipher Algorithm | Key Derivation | Enc. Time (ms) | Dec. Time (ms) | Security Level |
|---|---|---|---|---|
| AES-CTR + HMAC | Email-derived | ~1.2 | ~1.1 | High |
| AES-GCM | Email-derived | ~1.0 | ~0.9 | High (AEAD) |

| | | | | |
|---|---|---|---|---|
| Camellia-CBC + HMAC | Email-derived | ~1.4 | ~1.3 | High |
| ChaCha20-Poly1305 | Email-derived | ~0.8 | ~0.7 | High (AEAD) |
| XChaCha20-Poly1305 | Email-derived | ~0.9 | ~0.8 | High (AEAD) |
| Salsa20 | Email-derived | ~0.7 | ~0.6 | Medium-High |

Table 1: Cipher Algorithm Encryption/Decryption Performance

to their software-optimized design, while HMAC-augmented modes incur a modest additional overhead for the authentication tag computation.

### 4.3 Comparison with Existing Systems

Table 1 presents a comparative evaluation of SECURE-VAULT against existing password management and storage approaches across seven dimensions: email-derived key differentiation, multialgorithm support, OTP verification, breach alert notification, password vault capability, rate limiting, and open-source availability. SECURE-VAULT is the only system offering email-derived per-user algorithm diversification combined with automated breach alerting, OTP verification, and a browser-accessible vault without requiring local software installation

Table 2: Comparison of Password Storage Approaches

| Feature | Traditional (Plaintext) | AES-256 Only | bcrypt Hashing | Vault Apps | SECURE-VAULT (Ours) |
|---|---|---|---|---|---|
| Email-Derived Key | No | No | No | Partial | Yes |
| Multi-Algorithm Support | No | No | No | No | Yes (11 ciphers) |
| OTP Verification | No | No | No | Partial | Yes |
| Breach Alert Email | No | No | No | Partial | Yes |
| Password Vault | No | No | No | Yes | Yes |
| Rate Limiting | No | No | No | Partial | Yes |
| Open Source | — | Varies | Varies | No | Yes |
| Browser-Based | Varies | Varies | Varies | No | Yes |

### 4.4 Security Analysis

The email-derived key construction provides cryptographic diversity across the user population: two users can share the same plaintext password yet produce entirely different ciphertexts encrypted under different algorithms and keys, significantly limiting the value of a bulk data breach. The AEAD ciphers (AES-GCM, ChaCha20-Poly1305, XChaCha20-Poly1305) provide authenticated encryption, detecting any ciphertext tampering at decryption time. Non-AEAD stream ciphers (AES-CTR, Camellia-CBC, Salsa20) are augmented with HMAC-SHA256 to provide equivalent authentication guarantees.

The OTP-based verification mechanism ensures that only the legitimate owner of a registered email address can link a session to a Firestore account or access stored credentials. The automated breach alert system provides immediate notification of incorrect code submission, enabling users to take corrective action before an adversary can complete a credential stuffing attack. The rate limiting subsystem provides a hard lockout defense against online brute-force attempts against the vault.

## 5. CONCLUSION

This paper presented SECURE-VAULT, a novel browser-based password encryption and storage system that employs email-derived cryptographic diversification to provide per-user algorithm and key uniqueness. The system demonstrates that deterministic key and algorithm selection from user identity attributes is a viable and practical mechanism for distributing cryptographic risk across a user population without requiring users to manage cryptographic parameters. Experimental evaluation confirms sub-2ms cipher operation latency across all eleven supported cipher configurations, negligible relative to network and database round-trip times.

The dual-layer architecture—lightweight Excel-backed local sessions combined with cloud-synchronized Firestore vault storage—provides a pragmatic balance between deployment simplicity and cloud-scale credential management. OTP verification, breach alert email notifications, and rate-limited vault lockout collectively implement a defense-in-depth security posture aligned with NIST SP 800-63B guidelines.

Future work will focus on four enhancements: (1) migrating rate limiting state from process memory to Firestore for multi-instance resilience; (2) removing the OTP debug exposure in API responses prior to production deployment; (3) extending the email-derived key derivation to incorporate a server-side pepper and iteration count, transforming it into a full PBKDF2 or Argon2 variant; and (4) implementing end-to-end encrypted vault entries using client-side encryption in the browser, ensuring that the server receives only ciphertext even for vault entry fields.

## 6. REFERENCES

**[1]** A. Hinduja and P. Sharma,
"Password Security Techniques," *International Journal of Computer Science and Engineering*, 2019.
[Online]. Available: https://ijcseonline.org/index.php/j/article/view/7138

**[2]** H. A. Saleh,
"Password Managers: A Critical Review of Security, Usability and Innovative Designs," *Journal of Cyber Security and Information Systems*, 2025.
[Online]. Available: https://www.researchgate.net/publication/396003110

**[3]** R. K. Darmawan,
"Zero-Knowledge Encryption for Secure Password Manager Systems," *International Journal of Software Engineering and Computer Science*, 2025.
[Online]. Available: https://lembagakita.org/journal/index.php/ijsecs/article/view/4207

**[4]** V. Kalai Vani,
"KAGI: A Secure Password Manager Using Client-Side Encryption," *International Journal of Creative Research Thoughts*, 2025.
[Online]. Available: https://www.ijcrt.org/papers/IJCRT2511414.pdf

**[5]** K. Yanmantram,
"Secure Pass: Safe Storage and Password Manager Based on Cryptanalysis," *IJARIIT*, 2022.
[Online]. Available: https://www.ijariit.com/manuscript/secure-pass-safe-storage-and-password-manager-based-on-cryptanalysis/

**[6]** S. Gaw and E. W. Felten,
"Password Management Strategies for Online Accounts," *Computers & Security*, 2014.
[Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404814001059

**[7]** JavaInUse,
"Client-Side Encrypted Password Vault," 2023.
[Online]. Available: https://www.javainuse.com/passwordvault

**[8]** A. Biryukov, D. Dinu, and D. Khovratovich,
"Argon2: The Memory-Hard Function for Password Hashing and Other Applications," 2016.
[Online]. Available: https://password-hashing.net/argon2-specs.pdf

**[9]** NIST,
"Digital Identity Guidelines," 2023.
[Online]. Available: <span
style="color:blue">https://pages.nist.gov/800-63-3/

**[10]** OWASP Foundation,
"Password Storage Cheat Sheet," 2024.
[Online]. Available: <span
style="color:blue">https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html