IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 3, July 2026
Impact Factor: 8.2

AES Vault - Secure Database Encryption Application

Likitha S[1], Prof. Thejaswini M N[2], Monisha L S[3]
Student, Department of Computer Applications[1,3]
Assistant Professor, Department of Computer Applications[3]
Maharaja Institute of Technology, Mysore, Mandya, Karnataka

Abstract: *Confidential information stored in application databases is frequently protected only by perimeter-level controls such as login credentials, firewalls, and network security, leaving the underlying data readable in plain text if a database is ever compromised. This paper presents AES Vault, a web-based application that removes this exposure by encrypting every record at the application layer before it is written to the database, so that the stored content is never available in readable form even to someone who gains direct access to the data store. The system is implemented in Python using the Streamlit framework and MongoDB as its NoSQL data store. Confidential text entries and PDF documents are encrypted using AES in Galois/Counter Mode with a 256-bit key, and the encryption key for every record is derived from a master password through the PBKDF2-HMAC-SHA256 key-derivation function using a randomly generated salt and a unique nonce, ensuring that identical plaintext never produces identical ciphertext twice. User credentials are independently protected using bcrypt password hashing. Access to the system is divided between two roles: an Admin, who alone may decrypt, view, search, and delete stored records, and a User, who may only submit new encrypted records. Every encryption, decryption, and deletion action is written to an audit log that records the acting user, the operation performed, the affected record, and a timestamp. Beyond the core encryption workflow, the prototype was extended with three additional capabilities: a title-based search facility, a record-deletion function restricted to administrators, and a personal profile dashboard that summarizes each user's own activity using bar and pie charts. The completed system was validated through unit, integration, security, negative, and user-acceptance testing across fifteen functional test cases, all of which produced the expected outcome, confirming that the encryption, authentication, access-control, and audit mechanisms operate correctly together as an integrated whole*

Keywords: AES-GCM Encryption; PBKDF2 Key Derivation; bcrypt Password Hashing; Role-Based Access Control; Application-Level Encryption; MongoDB; Streamlit; Audit Logging; Secure Data Storage

I. INTRODUCTION

The growth of digital systems has changed the way organizations in healthcare, banking, education, government, and research create, store, and manage information, and a large share of this information is now held in centralized digital databases. Because this information is often sensitive, protecting it from unauthorized access has become a persistent concern. Conventional safeguards such as login passwords, network firewalls, and transport-level encryption reduce the likelihood of an intrusion, but none of them prevent the data itself from being read in plain form once an attacker manages to reach the database directly.

AES Vault was built to close this specific gap. Rather than relying only on measures that guard the perimeter of a system, AES Vault encrypts confidential information at the point of storage, so that every record kept in the database is unreadable without the correct decryption key. The application was developed in Python using the Streamlit framework for its interface, and it uses AES in Galois/Counter Mode with a 256-bit key as its core encryption algorithm, a choice made

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 3, July 2026
Impact Factor: 8.2

because AES-GCM combines strong confidentiality with built-in integrity verification. Two categories of user are supported: an Admin, who has full authority to encrypt, decrypt, search, view, and delete stored records, and a User, who may add encrypted records but cannot view, search, or remove them. A personal profile dashboard was also added so that every user, regardless of role, can review a summary of their own activity, including counts of the records they have encrypted, decrypted, or deleted. In total, three capabilities were introduced beyond the original encryption workflow during this phase of development: a search function, a record-deletion function, and the user profile page.

The need for stronger data protection has existed since databases first began linking information across networks, and the conventional perimeter-security model, which assumes that guarding access to a database is sufficient to keep its contents safe, has repeatedly proven inadequate against phishing attempts and network intrusions. Once such an intrusion succeeds, any information stored without encryption becomes immediately readable. Application-level encryption addresses this weakness directly by transforming the data before it ever reaches the database, so that even a successful intrusion yields only ciphertext. AES-GCM, standardized with guidance from NIST, was selected for this purpose because it is well suited to protecting sensitive information of exactly this kind.

A. Contributions

Design and implementation of a three-layer application architecture that separates the Streamlit-based presentation layer from the Python business-logic layer and the MongoDB data layer, so that encryption, authentication, and storage concerns remain independently modifiable.

An encryption scheme combining AES-GCM 256-bit encryption with a PBKDF2-HMAC-SHA256 key-derivation function that uses a unique, randomly generated 16-byte salt and 12-byte nonce for every record, so that identical plaintext content never produces identical ciphertext across records.

A bcrypt-based authentication mechanism that hashes and salts every user password before it is stored, preventing plaintext credential exposure even if the underlying user collection is accessed directly.

A role-based access-control model that restricts decryption, viewing, searching, and deletion of records to Admin accounts, while allowing User accounts to contribute encrypted records without being able to read or remove them.

An audit-logging mechanism that records the acting user, the action performed, the affected record, and a timestamp for every encryption, decryption, and deletion event, supporting traceability of all sensitive operations.

Three additional features layered on top of the core vault functionality: a title-based search facility, an Admin-only record-deletion capability, and a per-user profile dashboard presenting activity statistics through bar and pie charts.

Taken together, these contributions show that a functioning, end-to-end encrypted data-storage prototype can be built from freely available, open-source components without requiring specialized infrastructure, while still meeting the core security expectations that organizations place on systems handling confidential information.

II. LITERATURE SURVEY AND EXISTING SYSTEM

A. Review of Related Work

The design of AES Vault draws on several independent strands of prior research. Work comparing the 128-, 192-, and 256-bit variants of AES for cloud-storage security found that AES-256 offers the strongest balance between protective strength and computational performance, and specifically highlighted the importance of a properly managed nonce in preventing exposure of the underlying algorithm, a finding that directly shaped the encryption design used in AES Vault. Separately, research into role-based access control in web applications demonstrated that assigning permissions to roles rather than to individual users reduces the likelihood of accidental misconfiguration, which informed the decision to separate Admin and User permissions in AES Vault rather than granting permissions on a per-account basis.

On the subject of credential protection, a comparative study of password-hashing schemes, including MD5, SHA-1, SHA-256, and bcrypt, concluded that a hashing algorithm must be deliberately expensive to compute in order to resist brute-force attacks, which supports the choice of bcrypt for password storage in the application's authentication module. Complementing this, research on authenticated encryption established that AES-GCM offers stronger guarantees for data at rest than modes such as CTR or ECB, because it verifies data integrity in addition to providing confidentiality, and this

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 3, July 2026
Impact Factor: 8.2

finding is the basis for selecting AES-GCM as the encryption standard used throughout AES Vault. A widely referenced treatment of applied cryptography further recommended the use of a unique salt together with a high iteration count, on the order of 100,000 or more, when deriving encryption keys from passwords through PBKDF2, a recommendation reflected directly in the salt size and iteration count used by the application's key-derivation function.

Beyond encryption itself, established guidance on audit logging in secure information systems specifies that a trustworthy log must record the identity of the acting user, the type of event, the object affected, and a timestamp, requirements that are mirrored in the structure of the audit-log collection implemented in AES Vault. Documentation and practitioner literature on MongoDB's document-oriented storage model and on Python-based web frameworks such as Streamlit informed the choice of these two technologies for the data layer and presentation layer respectively. Finally, studies examining data privacy obligations in healthcare information systems and insider-threat mitigation through access control both reinforce the value of pairing strict access restrictions with continuous activity logging, a combination that is central to the security design adopted in this project.

B. Existing Systems and Their Limitations

Most database-oriented applications that advertise stronger security than a conventional database still rely primarily on access-control checks at login. Once a user is authenticated, such systems typically allow that user to view every record in the database, meaning the protection stops at the point of entry rather than extending to the stored data itself. Standalone file-encryption utilities offer an alternative by encrypting individual files with strong algorithms, but they operate outside of, and cannot be integrated into, a web application or its underlying database, which makes them unsuitable for continuous, in-application use. Encrypting an entire database at the storage-engine level is technically possible, but it demands considerably more infrastructure and ongoing maintenance than most small deployments can justify.

Across these existing approaches, several recurring shortcomings can be identified: the absence of application-level, field-by-field encryption; the absence of any audit trail describing who accessed or altered which record and when; weak or entirely absent password-hashing practices; no built-in ability to search or manage encrypted content without first decrypting it in bulk; and interfaces that assume a level of technical expertise beyond that of an average user.

C. Research Gap

None of the reviewed tools combine application-level, per-record encryption with strict role-based restrictions on decryption and deletion, a persistent audit trail of sensitive operations, and a usable interface for managing encrypted content, within a single lightweight application. AES Vault is intended to close this gap by bringing these elements together: AES-GCM encryption applied before storage, PBKDF2-based key derivation, bcrypt password protection, a two-role access model, and comprehensive audit logging, supplemented with practical usability features such as search, deletion, and a personal activity profile.

III. PROPOSED SYSTEM

AES Vault is proposed as a Streamlit-based web application that resolves the limitations identified in the existing systems reviewed above. Every piece of confidential information a user submits, whether typed text or an uploaded PDF document, is encrypted using AES-GCM with a 256-bit key before it is written to the MongoDB database, so that the stored record never exists in the database as readable content. Role-based access control restricts decryption, viewing, searching, and deletion of records exclusively to Admin accounts, while User accounts are limited to submitting new encrypted records. All user passwords are protected using bcrypt hashing rather than being stored, or even temporarily held, as plain text. An audit-log collection records every encryption, decryption, and deletion event together with the acting user, the action type, the affected record, and a timestamp, giving administrators a complete trail of sensitive activity. The system further includes a title-based search facility, an Admin-only deletion capability, and a personal profile page that presents each user's own activity statistics, all wrapped in a consistent pink-gradient, glass-morphism visual theme.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 3, July 2026
Impact Factor: 8.2

A. Advantages of the Proposed Approach

Protects stored content directly, using AES-GCM 256-bit encryption, rather than relying solely on access control at the perimeter of the system.

Derives a unique encryption key for every record through PBKDF2-HMAC-SHA256 with a freshly generated salt, which defeats precomputed and brute-force attacks against the key derivation step.

Protects user credentials with bcrypt hashing, so that stored passwords cannot be reversed even if the user collection is exposed.

Restricts decryption, viewing, and deletion strictly to Admin accounts, preventing ordinary users from accessing or removing content they did not have explicit authority over.

Logs every encryption, decryption, and deletion action with a timestamp, user identity, and action type, giving administrators full traceability and accountability for sensitive operations.

Adds practical usability features, a title-based search facility, an Admin deletion capability, and a per-user activity dashboard, that make the vault convenient to operate rather than purely security-focused.

IV. METHODOLOGY

AES Vault was developed as an iterative extension of an existing baseline application, with functional and non-functional requirements identified first, followed by system and security design, and finally a module-by-module implementation and testing cycle.

A. Requirement Analysis

Functional requirements covering account creation, authentication, text and PDF record encryption, key derivation, Admin-only search, decryption and deletion, system-wide activity logging, dashboard statistics, and the profile page were identified alongside non-functional requirements addressing security, performance, usability, reliability, and scalability. On the performance side, targets were set for text-record encryption to complete in under 500 milliseconds, PDF encryption for files under 5 MB to complete in under 3 seconds, and dashboard charts to render within 2 seconds, while security requirements specified that no sensitive data, whether record content or user passwords, should ever be stored as plain text.

B. Security and Encryption Design

The encryption approach was designed around AES-GCM with a 256-bit key, chosen for combining confidentiality with authenticated integrity checking. For every encryption operation, a 16-byte salt and a 12-byte nonce are generated at random, and the encryption key itself is derived from the user's master password using the PBKDF2-HMAC-SHA256 function with 100,000 iterations, in line with recommended practice for password-based key derivation. Passwords used for account authentication are handled separately through bcrypt hashing, which applies its own internal salting so that identical passwords never produce identical hash values.

C. Database Design Approach

MongoDB was selected as the data layer because its document-oriented, schema-flexible model suits the variable structure of encrypted text and PDF records. Three collections were designed: vault_users for account credentials and roles, vault_records for the encrypted content itself, and vault_logs for the audit trail of encryption, decryption, and deletion events.

D. Development Process

Development proceeded module by module. The authentication module (auth.py) was implemented and tested first, followed by the encryption module (encryption.py), the database-access module (database.py), and a set of shared utility functions (utils.py). The Streamlit application (app.py) was then built to tie these modules into a complete presentation layer, after which the three additional features, search, delete, and the profile page, were layered on top of the working baseline. Each module was exercised individually before being integrated with the rest of the application, and the complete system was finally verified through the structured testing process described in Section VII.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 3, July 2026
Impact Factor: 8.2

V. SYSTEM ARCHITECTURE AND DESIGN

AES Vault follows a layered architecture that separates presentation, business logic, and data storage into three distinct layers, so that each layer can be maintained or extended independently of the others.

The presentation layer is implemented in app.py using Streamlit, and is responsible for rendering the landing page, login page, dashboard, add-text page, PDF-upload page, records view, audit-log view, and profile page, while also applying the application's pink-gradient CSS styling and managing page state through Streamlit's session mechanism. The business-logic layer consists of three Python modules: auth.py, which handles user registration and login, including password validation and bcrypt hashing; encryption.py, which implements the AES-GCM encryption and decryption routines together with key derivation; and utils.py, which provides supporting utility functions such as base64 image conversion and file-size formatting. The data layer is implemented through database.py, which connects to a MongoDB instance and exposes the three collections described below, and which also implements the log_vault_action function used to write entries to the audit-log collection.

The vault_users collection stores each account's auto-generated identifier, a unique username, the bcrypt-hashed password, and a role value of either admin or user. The vault_records collection stores an auto-generated identifier, a type field indicating whether the record is text or a PDF, a title for text entries or an original filename for PDF entries, and a data or file object holding the ciphertext, salt, and nonce in hexadecimal form. The vault_logs collection stores an auto-generated identifier, the username of the acting user, the action performed (ENCRYPT, DECRYPT, or DELETE), the title or filename of the affected record, and the timestamp at which the action took place.

Two actors interact with the system: the User, who may register, log in, and submit encrypted text or PDF records, and view only their own profile statistics; and the Admin, who additionally has the authority to search, decrypt, view, and delete any record, and to review the complete audit log. This separation keeps a regular user's interaction limited to contributing data, while giving the administrator full oversight of the vault's contents and activity.

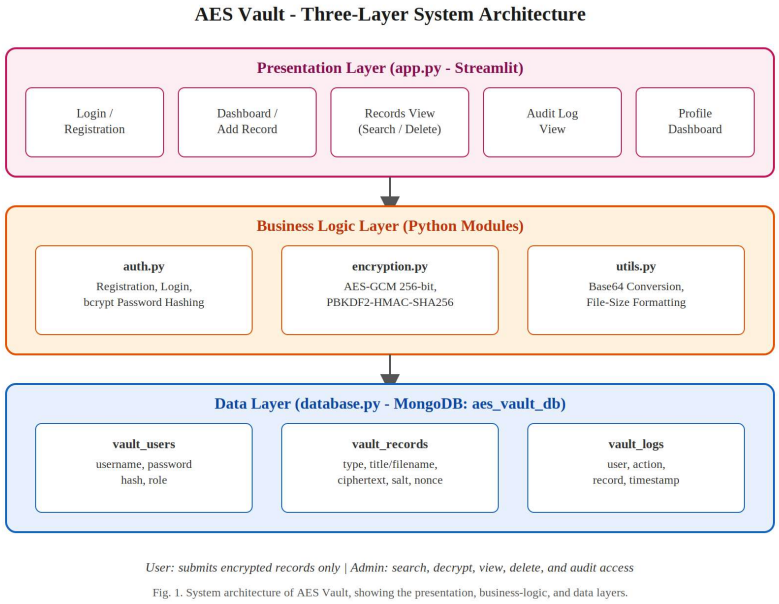

Fig. 1. System architecture of AES Vault, showing the presentation, business-logic, and data layers.

The security design follows a defense-in-depth principle built from five independent mechanisms operating together. Authentication, the first layer, ensures that only a registered and verified user can interact with the system, with passwords stored exclusively as bcrypt hashes. Authorization, the second layer, restricts sensitive operations, decryption, viewing, and deletion, to the Admin role alone. Encryption, the third layer, guarantees that even an attacker with direct database

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 3, July 2026
Impact Factor: 8.2

access sees only ciphertext, since AES-GCM protects every stored record and cannot be reversed without the correct key. Integrity verification, the fourth layer, uses AES-GCM's built-in authentication tag to detect any tampering with stored ciphertext and to reject decryption when the tag does not match. Audit logging, the fifth layer, records every sensitive operation in the vault_logs collection, enabling administrators to monitor activity and investigate anomalies after the fact.

VI. IMPLEMENTATION

A. Environment and Technology Stack

The application was implemented in Python 3.12 using Streamlit as the web-application framework for its frontend, with MongoDB Community Edition serving as the backing NoSQL database. The cryptography library was used to implement AES-GCM encryption and PBKDF2 key derivation, while the bcrypt library handled password hashing. Pandas was used to organize activity data for display, and Plotly Express was used to render the bar and pie charts that appear on the dashboard and profile pages.

B. Authentication Module Implementation

The auth.py module hashes each new password with bcrypt at registration time and stores only the resulting hash in the vault_users collection. During login, bcrypt's comparison function checks the submitted password against the stored hash without ever exposing the hash itself, and because bcrypt salts each hash internally, identical passwords produce different stored values across accounts, which defeats precomputed rainbow-table attacks. Registration additionally enforces a minimum password length of six characters, rejecting shorter passwords with an explicit error.

C. Encryption Module Implementation

The encryption.py module provides three functions. derive_vault_key accepts a master password and a salt and produces a 256-bit key using PBKDF2-HMAC-SHA256 with 100,000 iterations. secure_encrypt accepts the plaintext, generates a random 16-byte salt and 12-byte nonce using os.urandom, derives the corresponding key, encrypts the data with AES-GCM, and returns a dictionary containing the ciphertext, salt, and nonce encoded as hexadecimal strings for storage in MongoDB. secure_decrypt reverses this process: it reads the stored salt, nonce, and ciphertext, re-derives the key from the supplied master password, and decrypts the data, raising an Invalid Tag error, and refusing to return any output, whenever the ciphertext has been altered.

D. Database Module Implementation

The database.py module connects to a MongoDB instance (the aes_vault_db database on localhost:27017) and exposes references to the vault_users, vault_records, and vault_logs collections for use throughout the application. Its log_vault_action function accepts a username, an action type, and a record identifier, and writes a corresponding entry, together with a timestamp, into the vault_logs collection every time a sensitive operation occurs.

E. Search Feature Implementation

The records view was extended with a text-input field that filters the displayed records by title. The filtering itself is performed with a list comprehension over the records retrieved from vault_records, comparing the lowercased search term against the lowercased title or filename of each record, and the view reports that no matching records were found whenever the comparison yields an empty result.

F. Delete Feature Implementation

A delete control was added alongside the existing decrypt control on the records view. Selecting it invokes MongoDB's delete_one operation on the corresponding document in vault_records, after which a DELETE entry is written to vault_logs through log_vault_action, and the page is refreshed using Streamlit's rerun mechanism so that the updated record list is displayed immediately.

G. Profile Page Implementation

The profile page, reached through a new navigation-bar button, queries vault_logs to count how many encryption, decryption, and deletion actions the currently logged-in user has performed, and queries vault_records for the total number of stored records. It displays a role icon, the username, and a colored badge, an administrator badge in dark pink and a standard-user badge in a lighter pink, followed by four statistic cards and an account-information panel. The page

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 3, July 2026
Impact Factor: 8.2

also lists the user's five most recent actions in reverse chronological order, drawn from vault_logs, and shows an explanatory message instead when fewer than five actions exist.

H. User Interface Design

The interface was restyled around a pink-gradient theme, replacing an earlier blue color scheme, applied consistently to the navigation bar, buttons, input-field focus borders, and the dashboard background. A glass-morphism effect, visible as soft blur on cards and panels, was applied throughout, with dark pink used for headings and a pink left border used to distinguish individual audit-row entries. Success and error messages give users clear feedback after actions such as encrypting a record, deleting a record, or entering incorrect login credentials.

VII. RESULTS AND DISCUSSION

The completed application was validated through unit testing of individual functions in auth.py, encryption.py, database.py, and utils.py; integration testing to confirm that data encrypted through secure_encrypt could be correctly decrypted through secure_decrypt within the running application; security testing focused on confirming that plaintext is never stored, that AES-GCM's integrity check rejects tampered ciphertext, and that non-admin users cannot perform administrative actions; negative testing using invalid inputs such as short passwords or unauthorized decryption attempts; and a round of user-acceptance testing to confirm that the login, encryption, and dashboard workflows were intuitive to a new user.

Fifteen functional test cases were used to evaluate the system end to end, covering registration, password-length validation, duplicate-username handling, login and invalid-login handling, text and PDF encryption, title-based search, Admin decryption, unauthorized-decryption rejection, record deletion with audit-log creation, tampered-ciphertext rejection, profile-page display, and logout. All fifteen test cases produced their expected result. A representative subset of these cases is summarized below.

Test ID	Description	Expected Result	Outcome
TC01	User registration with a valid username and password	Account is created successfully	Pass
TC06	Encryption of a text record with a valid title and content	Record is encrypted and stored in the database	Pass
TC10	Decryption attempt by a non-admin user	Access-denied message is displayed	Pass
TC13	Decryption attempt on a tampered ciphertext value	Decryption fails with an integrity error	Pass

Functional inspection of the running application confirmed that the login page allows account access or new registration and includes a view-password option for user convenience. The Streamlit dashboard, reached after a successful login, presents navigation to text entry, PDF upload, record viewing, audit logs, and the profile page, alongside summary counts of total records, users, encryptions, and decryptions. The Admin dashboard additionally exposes user management, record search, decryption, and deletion controls. On the analytics view, a bar chart displays encryption and decryption activity while a doughnut chart shows the proportion of text versus PDF records, giving both users and administrators a quick visual summary of vault usage.

Overall, the validation process confirmed that records submitted through the application are consistently encrypted before storage, that only Admin accounts can successfully decrypt, search, or delete them, that tampering with stored ciphertext is reliably detected and rejected, and that the audit log, search, delete, and profile features all behave as designed. These outcomes indicate that the proposed architecture meets its intended goal of providing consent-free, transparent, application-level protection for confidential data without requiring specialized security expertise from the end user.

DOI: 10.48175/IJARSCT-37887

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 3, July 2026
Impact Factor: 8.2

VIII. ADVANTAGES AND LIMITATIONS

A. Advantages

Encrypts all confidential data at the application layer using AES-GCM 256-bit encryption before it reaches the database, so stored content remains protected even if the database itself is compromised.

Derives a unique key for every record through PBKDF2-HMAC-SHA256 with a fresh salt, guarding against brute-force and precomputation attacks on the key-derivation process.

Protects user credentials using bcrypt hashing, preventing plaintext password exposure even in the event of a database breach.

Enforces role-based access control that limits decryption, viewing, and deletion to Admin accounts, reducing the risk of accidental or malicious data exposure by ordinary users.

Maintains a complete audit trail of encryption, decryption, and deletion activity, supporting accountability and post-incident investigation.

Adds convenient, security-conscious usability features, a search function, a controlled deletion capability, and an individual activity dashboard, on top of the core encryption workflow.

B. Limitations

The current prototype carries several limitations, most of which are acknowledged directly in its own design scope. Application-level encryption and PBKDF2's 100,000-iteration key derivation introduce measurable computational overhead compared with an unencrypted system. The system currently supports only text entries and PDF documents, with no provision for other file types. Authentication relies on a single factor, username and password, without any additional verification step. There is no key-escrow mechanism, meaning a lost master password results in permanently inaccessible data, and the system does not include any automated detection of, or alerting for, suspicious activity beyond the passive audit log.

IX. FUTURE ENHANCEMENTS

Several directions have been identified for extending AES Vault beyond its current prototype scope. Two-factor authentication, using either a one-time password or an authenticator application, could be added to strengthen login security beyond a single password. The application could be hosted on a cloud platform such as AWS, Google Cloud, or Microsoft Azure, using MongoDB Atlas to enable remote database access. Support for additional file formats, including images, Word documents, and spreadsheets, would broaden the range of confidential content the vault can protect. Automated email notifications could be introduced to alert administrators when a record is deleted or decrypted. The existing search feature could be extended with filters for record type, upload date, and uploading user, making it easier to locate specific records as the vault grows. Finally, periodic key rotation could be introduced so that the master encryption key does not remain static indefinitely, reducing the risk associated with prolonged reliance on a single key.

X. CONCLUSION

AES Vault demonstrates that application-level encryption, combined with careful key management and strict access control, can meaningfully reduce the risk that a database breach exposes confidential information in readable form. Every record a user submits, whether typed text or an uploaded PDF, is encrypted using AES-GCM 256-bit encryption before it is stored, with the encryption key derived from the user's master password through PBKDF2-HMAC-SHA256 using a unique, randomly generated salt for each record, and a distinct nonce that prevents identical content from ever producing identical ciphertext. Because the encryption key itself is never stored alongside the data, an attacker who gains access to both the database and the application source code still cannot decrypt any stored record without the master password. User credentials are separately protected with bcrypt hashing, so that even the application never handles a plaintext password after registration. Role-based access control ensures that only Admin accounts can decrypt, view, or delete records, limiting the damage that a compromised User account could cause, while a complete audit trail in the vault_logs collection records every sensitive action for later review. The three features added during this phase of development, a

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 3, July 2026
Impact Factor: 8.2

title-based search facility, an Admin-only deletion capability, and a personal profile dashboard, extend the vault's usability without weakening its underlying security model.

All of the objectives set out at the beginning of this project were met: the resulting application is functional, secure, and validated through a structured testing process covering fifteen distinct test cases, each of which produced the expected outcome. The project shows that a working system built from open-source cryptographic libraries and a lightweight technology stack can provide meaningful, industry-aligned protection for confidential data, and it offers a practical foundation that can be extended with stronger authentication, broader file-format support, and additional operational safeguards as outlined in the future-enhancement directions above.

REFERENCES

[1] P. Karthikeyan and M. Suresh Kumar, "Design and Evaluation of AES for Data Security in Cloud Storage," International Journal of Computer Science and Information Security, vol. 14, no. 3, 2016.

[2] S. Chakraborty and I. Ray, "A Study on Role-Based Access Control in Web Applications," ACM Transactions on Information and System Security, vol. 9, no. 2, 2006.

[3] T. Malviya and R. Chandramouli, "Performance Analysis of Password Hashing Schemes," Journal of Information Security and Applications, vol. 28, 2016.

[4] R. Patel and H. Shah, "Authenticated Encryption Using AES-GCM for Secure Data Transmission," International Journal of Network Security and Its Applications, vol. 7, no. 4, 2015.

[5] C. Paar and J. Pelzl, Understanding Cryptography: A Textbook for Students and Practitioners, Springer, Berlin, Germany, 2010.

[6] A. Kent and L. Millett, Computers at Risk: Safe Computing in the Information Age, National Academies Press, Washington, DC, USA, 2002.

[7] K. Banker, P. Bakkum, and S. Verch, MongoDB in Action, 2nd ed., Manning Publications, Shelter Island, NY, USA, 2016.

[8] W. McKinney, Python for Data Analysis, 2nd ed., O'Reilly Media, Sebastopol, CA, USA, 2017.

[9] B. Cheng, S. Lee, and J. Liu, "Data Privacy and Encryption in Healthcare Information Systems," Journal of Medical Systems, vol. 36, no. 6, pp. 3586-3601, 2012.

[10] M. Collins, Data-Driven Security: Analysis, Visualization and Dashboards, Wiley, Indianapolis, IN, USA, 2014.

[11] Streamlit Inc., "Streamlit Documentation," 2024. [Online]. Available: https://docs.streamlit.io.

[12] MongoDB Inc., "MongoDB Manual," 2024. [Online]. Available: https://www.mongodb.com/docs.

[13] Python Software Foundation, "Python 3.12 Documentation," 2024. [Online]. Available: https://docs.python.org/3.12.

[14] J. Daemen and V. Rijmen, "Advanced Encryption Standard (AES) - A Review," IEEE Security & Privacy, 2001.

[15] D. Green and A. Smith, "Practical Cryptography in Web Applications," IEEE Internet Computing, 2018.