Journal Publication of International Research for Engineering and Management (JOIREM)
Volume: 04 Issue: 4 | April-2026
ISSN (O) 3107-6696

SecureVault: A Privacy-Centric Encrypted Storage Application for Android

[1]ADARSH RAJ, [2]A. KRUTHICK ROSHAN, [3]G. KARTHIK

[1,2,3]*Dept. of Computer Science & Engineering, Hindustan University, Chennai*
23cu0310419@student.hindustanuniv.ac.in, 23cu0310385@student.hindustanuniv.ac.in, 23cu0310424@student.hindustanuniv.ac.in

---

Abstract - Smartphones have increasingly become the primary platforms for storing sensitive personal and professional data. However, many existing storage solutions do not provide sufficient privacy protection. Cloud-based services offer convenience, but they introduce risks such as unauthorized access and exposure to third-party data handling. Likewise, several on-device vault applications rely on weak or non-standard encryption methods, making them susceptible to compromise using common forensic tools. To address these challenges, this work presents Secure Vault, an Android application designed with strong cryptographic foundations and layered authentication mechanisms. The application secures all stored data, including images, videos, audio, and documents, using AES-256 encryption. In addition, it supports both PIN-based and biometric authentication for enhanced access control. By operating entirely on-device without reliance on cloud services, SecureVault minimizes potential security vulnerabilities and significantly reduces the overall attack surface. Our evaluation on real Android hardware showed that the system achieves strong security guarantees while keeping encryption overhead low enough for practical daily use. We also outline planned extensions, including optional encrypted cloud backup, AI-driven anomaly detection, and intrusion lockout mechanisms.

Keywords—AES-256 encryption, Android security, secure local storage, biometric authentication, data privacy, mobile application security.

I. INTRODUCTION

Over the past decade, smartphones have evolved from basic communication devices into comprehensive platforms for storing sensitive personal and professional data. Users increasingly rely on these devices to manage medical records, financial documents, private media, and confidential business information. However, such data is often stored on devices that remain vulnerable to loss, theft, or unauthorized access, raising significant concerns regarding data security and privacy. This shift has made mobile data security a genuinely serious concern, not just a theoretical one [1], [2].

Despite these advancements, many existing storage solutions still fail to meet comprehensive security and privacy requirements. Cloud storage services such as Google Drive and Dropbox remain widely adopted due to their convenience; however, they centralize user data on third-party infrastructure. As a result, service providers may retain access to encryption keys, while sensitive metadata can remain exposed despite claims of end-to-end encryption [3]. On the other hand, locally-based vault apps often protect data with simple password gates, without any serious cryptographic backing. Single-factor authentication, especially with weak or reused passwords, offers minimal protection against brute-force or shoulder-surfing attacks [4].Single-factor authentication, particularly when based on weak or reused passwords, provides insufficient protection against threats such as brute-force and shoulder-surfing attacks [4].

To address these limitations, this work presents Secure Vault, a system designed to enhance data security through a combination of strong encryption and layered authentication. The proposed approach integrates AES-256 encryption at the file level with a two-factor authentication mechanism that combines PIN-based and biometric verification. All data is stored locally on the device using Android's native Keystore infrastructure, eliminating reliance on external servers. The system is designed to achieve a balance between robust security and practical usability for non-technical users.

The main contributions of this work are as follows:

1. A multi-layer authentication framework that combines AES-256 encryption with PIN-based and biometric verification.
2. A fully offline, privacy-preserving storage solution that operates without cloud dependency.
3. Performance benchmarks for encryption and decryption across representative file types on real Android hardware.

4. A side-by-side comparison of Secure Vault with current vault apps and cloud solutions across key security dimensions

The rest of this paper is structured as follows. Section II reviews related work. Section III introduces the proposed system. Section IV covers the architecture. Section V explains the methodology and implementation choices. Section VI presents our results. Section VII highlights use cases. Sections VIII and IX conclude with future directions.

II. LITERATURE SURVEY

A. Cloud-Based Storage Solutions

Cloud platforms offer several advantages, including cross-device synchronization, large storage capacity, and convenient file sharing capabilities. But the trade-offs are significant. In most setups, encryption keys are held by the provider, not the user. This means data can be accessed by the provider, or exposed in the event of a breach. Even services that claim end-to-end encryption may still leak metadata and access patterns to third parties [3].

B. Basic Local Vault Applications

Many vault apps on the Play Store offer little more than password-gated folders. Files are often stored in formats that remain extractable using standard forensic tools, and the encryption—if present—tends to use proprietary or outdated ciphers. Relying on a single password with no rate-limiting or lockout mechanism makes these apps vulnerable to dictionary attacks [4].

C. Advanced Encrypted Storage Systems

More sophisticated systems do apply standard ciphers like AES, but they often require network connectivity for key management, or introduce enough computational overhead to make them impractical on mid-range devices [1]. Usability tends to suffer, which leads to low adoption even when the underlying security is sound.

D. Summary of Limitations

Across these categories, the common failure modes are:

- Lack of standardized and well-audited encryption mechanisms in consumer-grade vault applications.
- Excessive reliance on single-factor authentication, often without proper lockout or rate-limiting policies.
- Dependence on internet connectivity, limiting usability in offline or low-connectivity environments.
- Centralized architectures that introduce a single point of failure and increase overall risk.
- Poor usability, which may encourage users to adopt less secure workarounds.

Secure Vault was designed with each of these failure modes in mind, aiming for a decentralized, on-device approach that does not sacrifice usability for security [1], [4].

III. PROPOSED SYSTEM

Secure Vault is a native Android application designed to provide users with effective control over their sensitive data, without requiring technical expertise or continuous internet connectivity.

A. System Objectives

The core goals guiding the design were:

- Protect all stored data, including images, videos, documents, and audio, using strong and standardized encryption.
- Ensure that access to vault contents is restricted to properly authenticated users.
- Support full offline operation—no data should leave the device.
- Present a clean, intuitive interface that encourages correct security behavior rather than working around it.

B. Key Features

The system includes the following capabilities

1. Implementation of AES-256 encryption for all imported files prior to storage.
2. Two-factor authentication mechanism combining numeric PIN and biometric verification (fingerprint or facial recognition)..
3. Support for commonly used file formats, including JPEG, MP4, PDF, and MP3.
4. Secure local storage within an application-specific directory, isolated from system-level media indexing.
5. A user-friendly, card-based interface that supports intuitive actions such as file addition, viewing, and deletion.

C. System Overview

At a high level, user interaction flows through four logical layers: the UI layer passes authentication requests to the

Authentication Layer; once the user is verified, file operations go to the Encryption Layer for AES processing; encrypted output is then written by the Storage Layer to the app's private directory. Android's Keystore system underpins both authentication and encryption key handling throughout (see Fig. 1).

[Fig. 1: SecureVault System Architecture]

IV. System Architecture

The system architecture adopts a modular, layered design that improves testability, extensibility, and overall clarity, particularly from a security perspective.

A. Architectural Components

1. User Interface Layer: Implements screens for login, registration, vault browsing, file import, and settings using Material Design components.
2. Authentication Layer: Handles PIN verification and biometric authentication via Android's Biometric Prompt API [5]. PIN credentials are securely stored as salted hashes derived using a key derivation function, providing resistance against offline brute-force attacks.
3. Encryption Layer: The system employs AES-256 encryption in GCM or CBC mode [1], with a unique initialization vector (IV) generated for each file to ensure strong data confidentiality. The encryption key is derived from user credentials and stored in Android Keystore [6].
4. Storage Layer: Manages encrypted file storage in the app's private directory. File metadata is held in a local SQLite database; actual file content is stored as encrypted binary blobs.

B. System Workflow

Upon launching the application, the user is authenticated using either a PIN or biometric verification. The Authentication Layer validates the provided credentials and, upon successful verification, grants access to the secure vault. All files within the vault are stored in encrypted form. When a file is accessed, it is decrypted in memory and rendered through an appropriate viewer without being written back to persistent storage in plaintext form. During file import, the system reads the selected file, encrypts it using AES-256, generates a unique initialization vector (IV), and stores the resulting ciphertext within a secure application directory (see Fig. 2).

[Fig. 2: Data Flow Diagram for Secure Vault Operations]

V. Methodology

A. Cryptographic Model

All files imported into Secure Vault are encrypted using AES-256 prior to storage. Let K denote the secret encryption key, IV the unique initialization vector generated per file, P the plaintext data, and C the resulting ciphertext. The encryption and decryption processes are defined as follows:

C = AES_Enc(K, IV, P) P = AES_Dec(K, IV, C)

B. Key Management

The encryption key is not stored directly. Instead, it is derived from the user's PIN using the PBKDF2 key derivation function, combined with a randomly generated salt and a sufficiently high iteration count. Let K denote the derived encryption key. The process can be expressed as:

K = KDF(PIN, salt, iterations)

This slows down brute-force attempts considerably [2]. The derived key is then wrapped and stored in Android Keystore, which uses hardware-backed security where the device supports it [6].

C. Authentication Workflow

Upon application launch, the system first determines whether biometric authentication hardware is available and properly enrolled. If available, the BiometricPrompt interface is presented, and a successful biometric match grants immediate access to the vault. In cases where biometric authentication is unavailable or declined by the user, the system falls back to PIN-based verification. The entered PIN is processed using the stored salt and compared against the previously stored hash to validate authenticity. Access is granted only on a match [5]. This two-path approach balances both convenience and robustness.

D. Implementation Details

Secure Vault is implemented in Kotlin using Android Studio as the development environment. The system leverages AndroidX libraries for user interface design and lifecycle management, the Biometric Prompt API for biometric authentication, the Android Keystore system for secure key management, and SQLite for storing file metadata. To ensure system reliability, unit tests were conducted across repeated encryption and decryption cycles to verify data integrity and

identify potential edge-case failures or data corruption scenarios.

VI. Results and Discussion

A. Experimental Setup

Secure Vault was evaluated on mid-range Android devices equipped with 4–6 GB of RAM and running Android 11 or later. The evaluation dataset consisted of commonly used file types, including images (1–5 MB), videos (10–50 MB), PDF documents (0.5–5 MB), and audio files (2–10 MB), representing typical personal data usage scenarios.

B. Performance Results

Table I shows measured encryption times for each file type.

TABLE I: Average Encryption Time

File Type	Avg. Size	Enc. Time (s)
Image	3 MB	0.5
Video	25 MB	1.2
Document	2 MB	0.4
Audio	5 MB	0.7

CPU utilization peaked at approximately 35–40% during intensive operations, while memory usage remained stable throughout execution. These observations indicate that the application can operate efficiently in the background on typical mid-range devices without causing noticeable degradation in overall system performance.

[Fig. 3: Encryption Pipeline for Imported Files]

A. Security Comparison

Table II places Secure Vault alongside typical cloud and basic vault solutions across several critical dimensions.

TABLE II: Feature Comparison with Existing Systems

The comparison makes it fairly clear where existing options fall short. Cloud-based applications often limit user control over encryption keys, while many basic vault solutions lack robust cryptographic protection. In contrast, Secure Vault provides AES-256 encryption, hardware-backed key storage, and multi-factor authentication, while maintaining full functionality in offline environments.

C. Usability and UI

The screen flow moves from a login screen to the main vault dashboard, from which users can access file import, encryption progress, the secure file viewer, and settings. Informal feedback from test users was generally positive—biometric login was singled out as a particularly welcome convenience feature that removed friction from daily use (see Fig. 4).

[Fig. 4: Screen Flow for the Secure Vault User Interface]

VII. Advantages and Applications

Secure Vault's main strengths are: strong encryption through AES-256 with per-file IVs; fully offline operation that keeps data off third-party servers; a simple interface that lowers the barrier to secure practices; and broad file format support covering photos, videos, audio, and documents.

Practical use cases include personal privacy protection for everyday users, secure document handling on employee-owned mobile devices in enterprise settings, and sensitive media management in contexts like healthcare, legal

services, and journalism where confidentiality requirements are strict.

VIII. Conclusion

This paper presented Secure Vault, an Android application that protects personal data using AES-256 encryption and a two-factor authentication model combining PIN and biometric login. By keeping all operations on-device and removing any dependency on cloud infrastructure, Secure Vault meaningfully improves user privacy without requiring technical knowledge. Benchmark results confirmed that strong cryptographic protection can be delivered with low enough overhead for practical use on commodity Android hardware.

IX. Future Work

Several directions are planed for future development. Future enhancements will explore the integration of lightweight machine learning models to detect anomalous access patterns in real time. Additional improvements include the development of an intrusion detection mechanism capable of locking the vault or securely wiping cryptographic keys after a configurable number of failed authentication attempts. Furthermore, an iOS implementation is planned to extend the proposed security framework to a cross- platform environment.

X. Acknowledgment

The authors would like to express their sincere gratitude to their supervisors and peers for their guidance and constructive feedback throughout the design and evaluation of Secure Vault. They also acknowledge Hindustan University for providing the necessary infrastructure and test devices to support this work.

XI. References

[1] National Institute of Standards and Technology (NIST), "Advanced Encryption Standard (AES)," FIPS Publication 197, 2001.

[2] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2nd ed. New York, NY, USA: Wiley, 1996.

[3] C. Tankard, "Cloud computing security: What's new?" *Network Security*, vol. 2009, no. 5, pp. 3–8, 2009.

[4] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in *Proc. 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2011, pp. 3–14.

[5] Google, "BiometricPrompt API," *Android Developers Documentation*. [Online]. Available: https://developer.android.com/ [Accessed: Mar. 20, 2026].

[6] Google, "Android Keystore system," *Android Developers Documentation*. [Online]. Available: https://developer.android.com/ [Accessed: Mar. 20, 2026].

* *Corresponding author: Tel: +918825668092, Email: 23cu0310385@student.hindustanuniv.ac.in*