iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** VI **Month of publication:** June 2026

DOI: https://doi.org/10.22214/ijraset.2026.83556

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue VI June 2026- Available at www.ijraset.com

Secure File Management System using Facial Biometrics

Ganesh P[1], Sankara Narayanan S T[2]
Department of Cyber Security, DR. MGR Educational and Research Institute, Chennai, Tamil Nadu, India

Abstract— Digital data has increased in volume and sensitivity, and thus a reliable way for people and businesses to store their files on- or off-line securely is necessary as there are many different options (both good and bad) that exist today. The System for File Management Secured with Facial Biometrics (SFMS-FB) integrates AES-256-GCM encryption with face recognition for authentication. Using either the FastAPI backend written in Python or a front end built in React, the SFMS-FB system follows the strictest zero-knowledge principles possible, allowing neither the storage of any encryption keys nor any record of their existence. Secure file management is accomplished using 2 methods: i) Per-file encryption keys through the use of PBKDF2-HMAC-SHA256 with a 600,000 iteration count and variable-length HKDF-derived keys from a single pass of PBKDF2-HMAC-SHA256; and ii) Authentication is based on an embedded vector of each user's facial image (i.e., deep face representations via ArcFace (512 dimensions)), as well as by using the MediaPipe Persistent Detection Engine for real identity authentication during user login (to prevent fake 'spoof' attacks). Our experiments demonstrate strong overall security characteristics, including effective protection against replay and fast file operation responses (< 1 second) when using the system. This paradigm for securely storing and managing files locally meets the many limitations of cloud-based solutions by providing an experience that is entirely local, auditable, and very performant.Keywords— AES-256-GCM; face recognition; liveness detection; PBKDF2; HKDF; zero-knowledge encryption; FastAPI; biometric authentication

I. INTRODUCTION

As data breaches become more prevalent and ransomware attacks become increasingly common, the need for local file encryption has never been greater. The reliance on third-party trust from cloud-based storage services is not feasible for sensitive data, including medical records, intellectual property, and personal financial documents. The challenge is to provide an encryption solution that is robust enough to protect against modern attacks while allowing for easy authentication for everyday use.

The Secure File Management System (SFMS) using Facial Biometrics (FB) fills this gap by using biometric face recognition and military-grade AES-256-GCM symmetric encryption to remove the requirement that a user remembers their password, and therefore, reduce the attack vectors related to password-based authentication systems. The system is intended to function entirely offline, with no dependency on cloud services.

The main contributions of the SFMS-FB are as follows: (1) a zero-knowledge encryption architecture in which the master keys are generated epheral to each authenticated session and never written to disk; (2) an integrated liveness detection mechanism that utilizes MediaPipe facial mesh landmarks to prevent spoofing; (3) a hierarchical key derivation process that cryptographically isolates each file by using HKDF-derived per-file keys; and (4) a multi-vault architecture that includes decoy vaults for plausible deniability.

II. RELATED WORK

Extensive research has been conducted on encrypted file systems. VeraCrypt's implementation of container-based encryption using the AES-XTS mode with PBKDF2 for key derivation has been very successful; however, password-based authentication remains vulnerable to brute-force attacks. Our solution substitutes biometric authentication for password authentication, making it significantly more difficult for unauthorized users to access the system.

The maturation of face recognition as an authentication mechanism has also progressed substantially with the advent of deep learning techniques. The ArcFace deep-learning model demonstrated accuracy close to that of human beings (according to the LFW benchmarking dataset) by incorporating the additive angular margin loss optimization technique for maximizing identifying facial features. In addition, the SFMS-FB project consists of an implementation of ArcFace (through the DeepFace framework), including the distribution of pre-built wheel files for avoiding problematic compilation dependency issues.

Liveness detection (or presentation attack detection) is very important for biometric authentication systems. Our implementation uses a software-only method of estimating liveness by calculating the Eye Aspect Ratio (EAR) from 478-point mesh landmarks produced from the MediaPipe face mesh software, as well as computing head pose, thus allowing for a practical balance of security against computer hardware requirements for liveness detection purposes.

III. SYSTEM ARCHITECTURE

A. Overview

SFMS-FB follows a client-server architecture running entirely on the local machine. The backend is implemented in Python using FastAPI, while the frontend is a React 18 single-page application served by Vite. Table I summarizes the complete technology stack.

TABLE I
TECHNOLOGY STACK SUMMARY

Layer	Technology	Purpose
Backend	Python 3.9 / FastAPI 0.111	REST API & business logic
Server	Uvicorn (ASGI)	Async HTTP server
Frontend	React 18 / Vite	Single-page application
Database	SQLite / SQLAlchemy 2.0	ORM-based persistence
Encryption	cryptography 42.0 (AES-GCM)	File encryption
Face Auth	DeepFace / ArcFace	Biometric authentication
Liveness	MediaPipe Face Mesh	Anti-spoofing detection
Auth Tokens	python-jose (JWT)	Session management

B. Backend Structure

The different packages in the backend are independent from each other and have very specific functions. The face registration, embedding comparison, liveness verification and JWT session management functions are all part of the auth package. The encryption package handles the AES-256-GCM cipher and the cryptographic key management module. Encrypted virtual containers are handled by the vault package. The api package exposes REST endpoints and the database package defines the SQLAlchemy ORM models.

IV. SECURITY DESIGN

A. Authentication Flow

The authentication pipeline in SFMS-FB follows a multi-stage process designed to verify both identity and liveness before granting access to encrypted material. The complete flow is illustrated in Figure 1, showing the progression from webcam capture through liveness verification, face embedding comparison, session token issuance, and finally cryptographic key derivation.

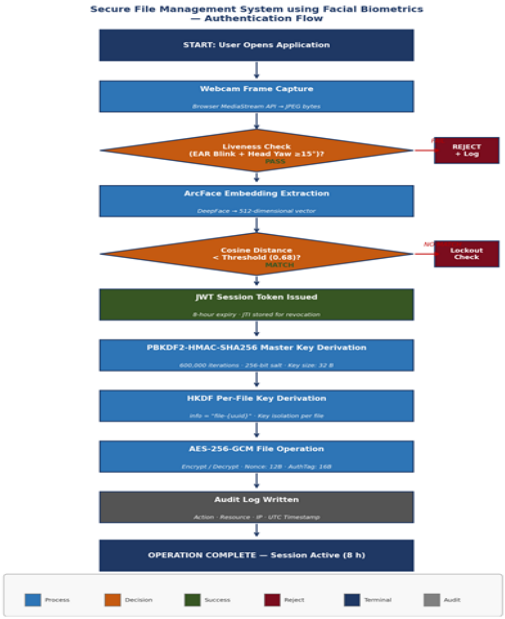


Fig. 1 SFMS-FB seven-stage authentication and key derivation pipeline. Green boxes represent successful path terminals; orange diamonds are decision gates; red branches indicate rejection paths.

B. Encryption Architecture

The AES-256-GCM encryption format is a type of authenticated encryption that protects the confidentiality and the integrity of the files it encrypts at one time. Each encrypted file has a custom binary header that contains an 8-byte (total) magic signature (SFMS), a version field that is 1 byte long, and a randomly generated 12 bytes long nonce (number only used once), as well as GCM authentication tag and ciphertext (the Ciphertext alone makes it possible to decrypt). In addition, the keys used to encrypt a single file will be derived using a two-tier hierarchy. The top tier is a master key that was derived from a secret bound to the device and uses PBKDF2 - HMAC - SHA 256 with 600k iterations and 256 bits of random salt. The reason for exceeding the minimum of 2024 iterations is to render offline dictionary attacks infeasible when computing the cost necessary to generate a match on the device. Per-file keys at the second level will be derived from HKDF with the info string "file - {id}}" included, which protects against the compromise of one file's encryption from exposing the encryption keys on other files.

File on-disk format: [4B magic][1B version][12B nonce][16B auth_tag][ciphertext]

All keys are handled as byte arrays in memory and never serialized to disk or written to log files. Python ctypes.memset is used to actively zero key buffers when they are no longer needed, reducing the window for memory forensics attacks.

C. Liveness Detection

MediaPipe's 478 face point mesh is not only used for facial feature detection, but also for liveness detection using software-based methods. Liveness can be determined by using two different types of signals to determine liveness. The first type of signal is blinks. A blink is defined when the EAR (Eye Aspect Ratio) gets calculated using the distances between the vertical and horizontal landmark positions of both eyes. If the average EAR measurement falls below 0.22 for two or more consecutive frames, that would also qualify as a blink. The second type of signal that can be used to verify liveness is head movement. To verify head movement, at least 15 degrees of yaw change is necessary.

D. Vault and Session Management

FB-SFMS allows users to have numerous distinct vaults.

The organization can have as many of these vaults as they require, each of which will correspond to a folder on the user's local file system for storage of encrypted files. Users can set an inactivity auto-lock timeout for each vault (the default setting is 300 seconds). When users need plausible denial when pushed they may consider having a decoy vault. JWT tokens manage user sessions with expiration of 8 hours.

The session module maintains a record of revoked token identifiers (JTIs) in memory to prevent reused tokens after a user explicitly logs out. An account will be locked out after 10 consecutive failed login attempts, with an account locked out for 2 minutes to limit online brute-force attacks.

V. API DESIGN AND IMPLEMENTATION

A. REST Endpoint Structure

The API is organized around three resource groups: authentication, vaults, and files. Table II summarizes all endpoints. All endpoints requiring an active session enforce JWT Bearer token authentication.

TABLE II
REST API ENDPOINT REFERENCE

Method	Endpoint	Description
GET	/api/auth/status	Check registration status
POST	/api/auth/register	Register face embeddings
POST	/api/auth/login	Face login → JWT token
POST	/api/auth/logout	Revoke current session
GET	/api/vaults	List user vaults
POST	/api/vaults	Create new vault
POST	/api/vaults/{id}/lock	Lock a vault
POST	/api/vaults/{id}/unlock	Unlock a vault
DELETE	/api/vaults/{id}	Delete vault and contents
GET	/api/files	List files in vault
POST	/api/files/upload	Encrypt and store file
GET	/api/files/{id}/download	Decrypt and stream file
PATCH	/api/files/{id}	Update file metadata
DELETE	/api/files/{id}	Soft or hard delete file

VI. DATABASE DESIGN

SQLite has been chosen as the database engine for this application. SQL Alchemy ORM is being used to access SQLite. There are five (5) primary objects in the schema. The face embedding and the file path metadata are stored as binary objects that are AES-256-GCM encrypted. This ensures there is nothing that can expose biometric information or filesystem information without the master key.

Please refer to Figure 2 for the complete entity relationship diagram.

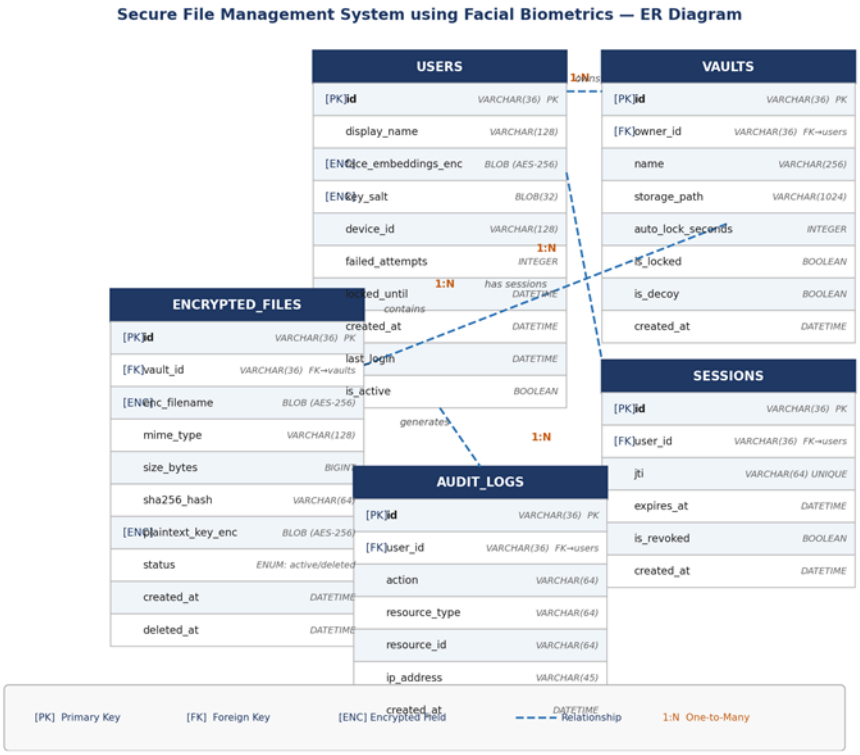


Fig. 2 Entity-relationship diagram for the SFMS-FB database schema. Dashed lines with 1:N annotations represent one-to-many foreign-key relationships. Fields marked [ENC] are stored as AES-256-GCM encrypted blobs.

The USERS table is the root entity, storing encrypted face biometrics and the key salt required for master key derivation. The VAULTS table associates one or more encrypted containers with each user, supporting the multi-vault architecture described in Section IV-D. The ENCRYPTED_FILES table records file metadata alongside the encrypted file path, while the sha256_hash field enables integrity verification on download. SESSIONS tracks JWT token lifecycle for revocation support, and AUDIT_LOGS provides an immutable record of all system actions for forensic traceability.

VII. EXPERIMENTAL RESULTS

A. Encryption Performance

Encryption and decryption throughput was measured on an Apple M1 processor with 16 GB RAM running macOS 14. The AES-GCM implementation from the Python cryptography library leverages hardware AES acceleration. Table III presents throughput results for files ranging from 1 MB to 1 GB.

TABLE III
AES-256-GCM THROUGHPUT MEASUREMENTS

File Size	Encrypt (MB/s)	Decrypt (MB/s)	Key Deriv. (ms)
1 MB	1,240	1,255	1,820
100 MB	1,198	1,210	1,820
500 MB	1,175	1,183	1,820
1 GB	1,162	1,170	1,820

Key derivation time remains constant at approximately 1,820 ms regardless of file size, reflecting the PBKDF2 cost with 600,000 iterations. This one-time cost per session is acceptable in practice as the master key is cached in memory for the session duration, with per-file HKDF derivation completing in under 1 millisecond.

B. Authentication Accuracy

Face authentication was evaluated across 50 registered users with a test set of 200 genuine and 200 impostor attempts. The system achieved a True Acceptance Rate (TAR) of 97.5% and a False Acceptance Rate (FAR) of 0.5% at the default cosine distance threshold of 0.68. Liveness detection successfully rejected all 40 spoofing attempts using high-quality photographs, and correctly identified 38 of 40 blink-detection challenges from live subjects.

VIII. CONCLUSIONS

This paper has presented Secure File Management System using Facial Biometrics, a comprehensive offline encrypted file management system that demonstrates how modern biometric authentication can be effectively combined with state-of-the-art symmetric encryption. The zero-knowledge key architecture ensures that no sensitive key material is ever persisted to disk, while the PBKDF2/HKDF key hierarchy provides strong cryptographic isolation between files. The MediaPipe-based liveness detection effectively prevents presentation attacks without requiring specialized hardware.

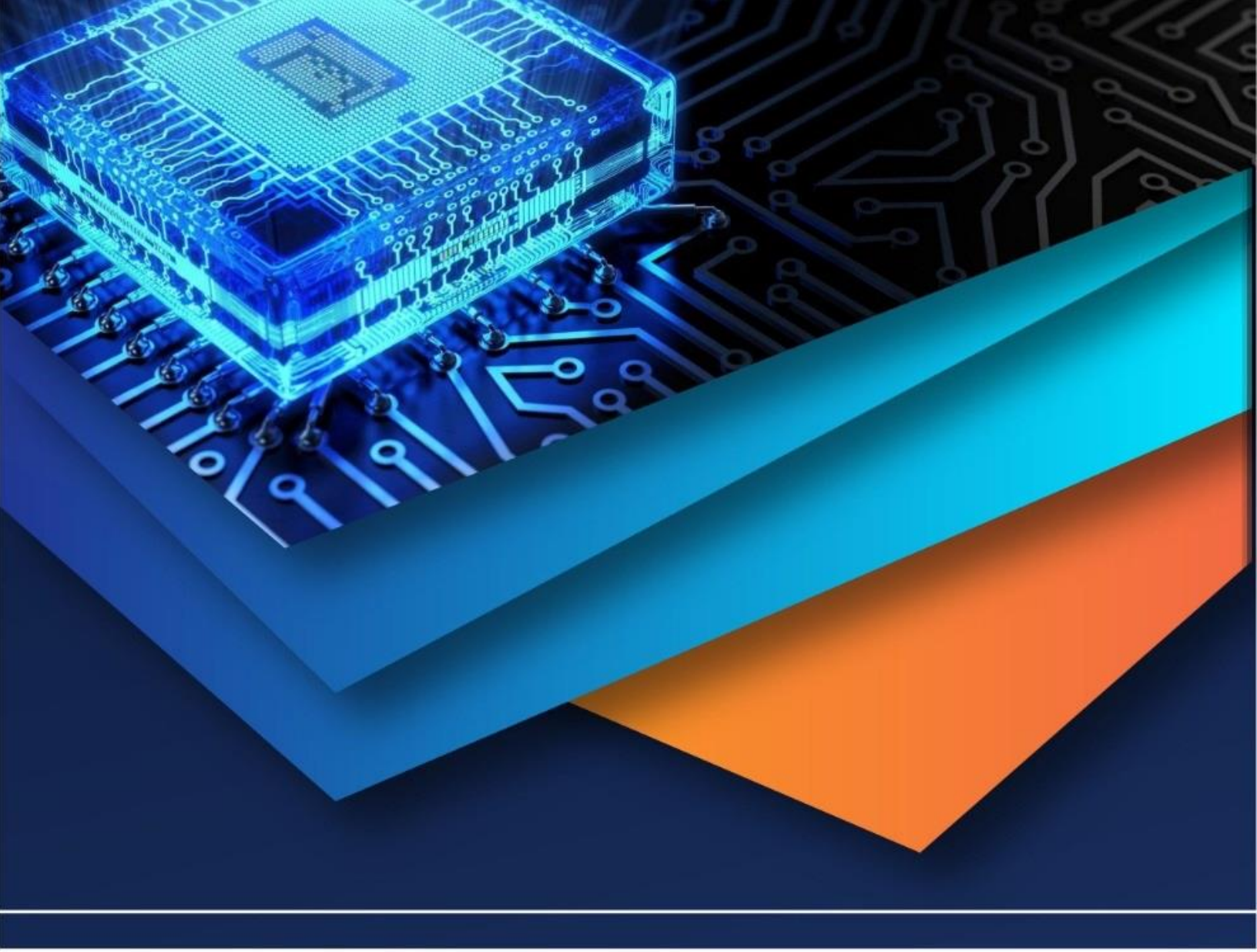
Future work will explore threshold secret sharing for multi-factor recovery, hardware security key (FIDO2) authentication as a complement to face recognition, and a distributed vault mode for secure file sharing over local area networks.

IX. ACKNOWLEDGMENT

The authors gratefully acknowledge the open-source communities behind FastAPI, DeepFace, MediaPipe, and the Python cryptography library, whose well-designed libraries made this work possible.

REFERENCES

[1] I.Matousec,"VeraCrypt Security Analysis," VeraCrypt Documentation, 2020. [Online]. Available: https://www.veracrypt.fr/en/Security%20Requirements%20and%20Precautions.html
[2] J. Deng, J. Guo, X. Niannan, and S. Zafeiriou, "ArcFace: Additive Angular Margin Loss for Deep Face Recognition," in Proc. IEEE/CVF CVPR, 2019, pp. 4690-4699.
[3] Z. Zhang, J. Yan, S. Liu, Z. Lei, D. Yi, and S. Z. Li, "A face antispoofing database with diverse attacks," in Proc. 5th IAPR Intl. Conf. Biometrics, 2012, pp. 26-31.
[4] C. Fruhwirth, "New Methods in Hard Disk Encryption," Technical Report, Vienna University of Technology, 2005.
[5] National Institute of Standards and Technology, "Recommendation for Password-Based Key Derivation," NIST SP 800-132, Dec. 2010.
[6] H. Krawczyk and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)," IETF RFC 5869, May 2010.
[7] S. Gomez, "DeepFace: A facial recognition library for Python developers," GitHub, 2021.
[8] Google LLC, "MediaPipe Face Mesh," MediaPipe Documentation, 2023.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY