Bhumkar Pooja Dnyaneshwar, 2026, 14:5
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology
An Open Access Journal

Secure File Encryption Systems Using AES-256-GCM And SHA-256: A Review Of Authenticated Encryption, Key Management And Integrity Verification

Bhumkar Pooja Dnyaneshwar[1] Prof. M. A. Auti[2] Prof. M. C. Kolse[3] Prof. S. K. Said[4]
Department of Artificial Intelligence and Data Science
Jaihind College of Engineering
Pune, Maharashtra, India

Abstract—Files containing academic records, financial statements, medical reports and business documents are routinely stored on laptops, removable drives and cloud folders, where they are exposed to theft, unauthorized access and silent tampering. This paper reviews the design of a secure file encryption system built on modern, standardized cryptography. The Advanced Encryption Standard with a 256-bit key in Galois/Counter Mode (AES-256-GCM) is examined as an authenticated encryption scheme that provides confidentiality and authenticity in a single operation, using a unique nonce for every encryption. The review explains why password-derived keys must be generated with a slow, salted key-derivation function such as PBKDF2 or Argon2, how the SHA-256 hash supports file integrity verification and identification, and why weaker primitives such as MD5 and SHA-1 should be avoided. A layered architecture covering the user interface, key management, cryptographic engine, integrity module and secure storage is proposed, together with an encrypted file format and a step-by-step encryption and decryption workflow. The paper also compares alternative ciphers and modes, discusses nonce misuse, key loss and post-quantum considerations, and identifies research gaps for practical tools. The review concludes that correctly combining authenticated encryption, strong key derivation and integrity checks yields a simple yet robust solution for protecting confidential files.

Index Terms—File encryption, AES-256-GCM, authenticated encryption, SHA-256, key derivation, Argon2, data confidentiality, data integrity

I. INTRODUCTION

Digital files have replaced paper for most personal, academic and professional records. Laptops are lost, USB drives are misplaced and cloud accounts are compromised, and in each case unencrypted files are immediately readable by whoever obtains them. Encryption converts such files into ciphertext that is useless without the correct key, so that a lost device does not automatically become a data breach.

Modern cryptography offers well-studied, standardized primitives for this task. The Advanced Encryption Standard (AES) [1], based on the Rijndael cipher [2], is the global standard block cipher, and Galois/Counter Mode (GCM) [3] turns it into an authenticated encryption scheme. The challenge for a practical system is less the choice of algorithm than its correct use: generating unique nonces, deriving strong keys from passwords, protecting metadata and handling errors safely.

This paper reviews these issues and proposes the architecture of a secure file encryption system using AES-256-GCM for encryption and authentication and SHA-256 [4] for integrity verification and file identification.

II. BACKGROUND AND FUNDAMENTAL CONCEPTS

1. Security Goals

A file encryption system should provide confidentiality (only key holders can read the file), integrity (any modification is detected) and authenticity (the ciphertext was produced by someone holding the key).

© 2026 Bhumkar Pooja Dnyaneshwar, This is an Open Access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly credited.

Fig. 1 classifies the cryptographic building blocks that address these goals.

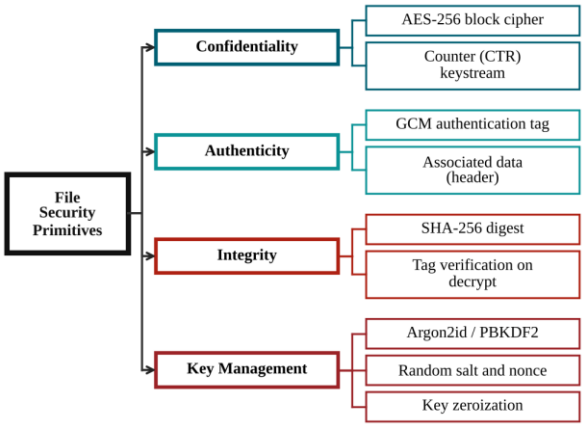


Fig. 1. Cryptographic building blocks of a secure file encryption system.

2. Authenticated Encryption

Bellare and Namprempre [5] showed that combining a cipher and a message authentication code correctly, in particular encrypting first and then authenticating, gives strong security, whereas careless combinations can fail. Authenticated Encryption with Associated Data (AEAD) modes such as GCM perform both steps in one primitive. In GCM, AES in counter mode produces a keystream that is XORed with the plaintext, and a polynomial hash over the ciphertext and associated data produces an authentication tag [6].

3. Hash Functions

A cryptographic hash such as SHA-256 maps any input to a fixed 256-bit digest; finding two inputs with the same digest is computationally infeasible. Hashes are useful for identifying files and verifying that a decrypted file matches the original, but on their own they do not prove who created a file, because anyone can recompute a hash.

III. LITERATURE SURVEY

Daemen and Rijmen [2] described the design rationale of Rijndael, including its substitution–permutation structure and resistance to linear and differential cryptanalysis, which led to its selection as AES. McGrew and Viega [6] introduced GCM and proved its security while demonstrating that it can be implemented efficiently in hardware and software; NIST later standardized it in SP 800-38D [3].

Sr. No.	Ref.	Year	Methodology	Findings	Research Gap
1	[5]	2000	Generic composition analysis	Encrypt-then-MAC proven secure	Theoretical; no file format
2	[2]	2002	Rijndael cipher design	Basis of AES; strong against known attacks	Mode of operation not covered
3	[6]	2004	Galois/Counter Mode	Efficient, provably secure AEAD	Catastrophic on nonce reuse
4	[11]	2005	Differential attack on MD5	Practical MD5 collisions	MD5 unsuitable for integrity
5	[7]	2016	Internet scan for GCM nonce reuse	Showed real-world forgery from repeated nonces	Studied TLS rather than files
6	[9]	2016	Argon2 memory-hard function	Resists GPU and ASIC guessing	Higher memory use on small devices
7	[8]	2017	PBKDF2 (PKCS #5 v2.1)	Salted, iterated password-based key derivation	Not memory-hard; GPU-friendly
8	[12]	2017	Full SHA-1 collision	Demonstrated SHA-1 is broken in practice	Motivates SHA-256 or SHA-3
9	[13]	2018	ChaCha20-Poly1305 AEAD	Fast software alternative to AES-GCM	Less hardware acceleration

TABLE I Summary of Literature Survey

The importance of correct nonce handling was demonstrated by Böck *et al.* [7], who scanned the internet and found servers that reused GCM nonces, allowing attackers to recover the authentication key and forge messages. This result shows that implementation discipline is as important as algorithm strength.

Because users usually remember passwords rather than random keys, key derivation is critical. PBKDF2, specified in PKCS #5 [8], repeatedly applies a pseudorandom function with a salt to slow down guessing. Argon2 [9], winner of the Password Hashing Competition, is additionally memory-hard, which makes large-scale guessing on GPUs far more expensive. HKDF [10] is used when a strong secret already exists and multiple sub-keys must be derived from it.

The history of hash functions shows why algorithm choice matters. Wang and Yu [11] found practical collisions in MD5, and Stevens *et al.* [12] produced the first collision for full SHA-1, so both are unsuitable for integrity protection. SHA-256 remains secure. As an alternative to AES on devices without hardware acceleration, ChaCha20-Poly1305 [13] offers comparable security with fast software performance. Looking ahead, NIST has standardized post-quantum key encapsulation in FIPS 203 [14], which will affect how file keys are shared. Table I summarizes these works.

IV. RESEARCH GAP

The review identifies the following gaps:

• Many student and open-source tools use outdated modes such as ECB or unauthenticated CBC, or reuse fixed nonces.

• Password-based tools often use fast hashes instead of salted, slow key-derivation functions.

• Encrypted file formats rarely authenticate metadata such as file name and version, enabling substitution attacks.

• Usability is often poor, so users fall back to unencrypted storage; key recovery is seldom considered.

• Few tools are designed with crypto-agility to migrate to post-quantum key exchange.

V. OBJECTIVES

The objectives of this review are:

- **O1:** To study AES-256-GCM and SHA-256 and the security properties they provide.
- **O2:** To review secure key-derivation and key-management practices for password-based encryption.
- **O3:** To propose a layered architecture and authenticated file format for a secure file encryption system.
- **O4:** To compare alternative ciphers, modes and hash functions.
- **O5:** To identify risks such as nonce reuse and key loss and recommend mitigations.

VI. PROPOSED SYSTEM ARCHITECTURE

The architecture in Fig. 2 separates responsibilities so that keys never leave the key-management layer in plain form and every output file is self-describing.

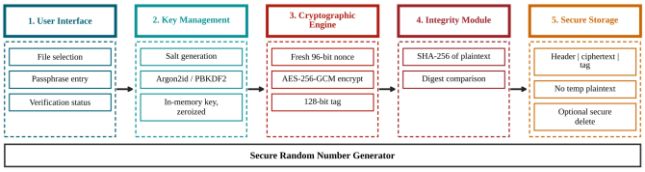


Fig. 2. Proposed layered architecture of the secure file encryption system.

1. User Interface Layer

A desktop or web interface lets users select files, enter a passphrase, and view the result of integrity verification. It never displays or logs keys.

2. Key Management Layer

A random 128-bit salt is generated, and the passphrase is converted to a 256-bit key using Argon2id or PBKDF2 with a high iteration count. Keys are held only in memory and are erased after use, following NIST key-management guidance [15].

3. Cryptographic Engine

For each file, a fresh 96-bit nonce is drawn from a cryptographically secure random number generator. AES-256-GCM encrypts the file and produces a 128-bit tag. The file header, containing a version number, salt, nonce and original file name, is passed as associated data so that it is authenticated but not encrypted.

4. Integrity Module

The SHA-256 digest of the original file is computed and stored inside the encrypted payload. After decryption, the digest is recomputed and compared, giving an additional end-to-end check and a convenient file fingerprint.

5. Secure Storage Layer

The output file is written as header, ciphertext and tag. Temporary plaintext files are avoided, and the original file can optionally be securely deleted after successful encryption.

VII. METHODOLOGY AND WORKING

The encryption workflow is shown in Fig. 3. The key is derived as

$$K = KDF(password, salt, cost\ parameters) \quad (1)$$

and GCM encryption with nonce N and associated data A produces

$$(C, T) = AES\text{-}GCM\text{-}Enc(K, N, P, A) \quad (2)$$

where P is the plaintext, C the ciphertext and T the authentication tag. During decryption, the tag is recomputed over A and C; if it does not match, the operation is aborted without releasing any plaintext. The integrity digest is

$$D = SHA\text{-}256(P) \quad (3)$$

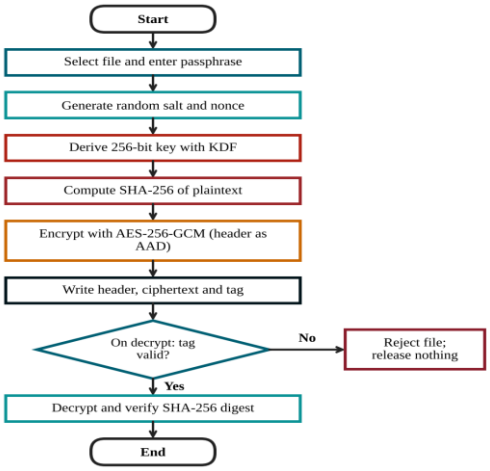


Fig. 3. Encryption and decryption workflow.

Because GCM security collapses if a nonce is ever repeated under the same key, the system derives a fresh key with a new salt for every file, which makes accidental nonce reuse practically impossible.

VIII. COMPARATIVE ANALYSIS

Table II compares common choices. AES-256-GCM provides confidentiality and authenticity with hardware acceleration on most processors; ChaCha20-Poly1305 is a strong alternative; CBC requires a separate MAC; and ECB is insecure because identical plaintext blocks produce identical ciphertext blocks.

Algorithm	Provides	Status
AES-256-GCM	Confidentiality + authenticity	Recommended
ChaCha20-Poly1305	Confidentiality + authenticity	Recommended
AES-CBC + HMAC	Both (if Encrypt-then-MAC)	Acceptable; complex
AES-ECB	Weak confidentiality	Insecure
SHA-256	Integrity digest	Recommended
MD5 / SHA-1	Broken collision resistance	Deprecated

TABLE II Comparison of Encryption Modes and Hash Functions

IX. CHALLENGES AND FUTURE SCOPE

Table III lists important challenges. Future work includes hybrid encryption that wraps file keys with public-key or post-quantum key encapsulation to support secure sharing, integration with hardware security modules and trusted platform modules, and streaming encryption for very large files with per-chunk authentication.

Risk	Mitigation
Nonce reuse	Fresh salt and key per file; CSPRNG nonces
Weak passwords	Argon2id, strength meter, passphrases
Key loss	Optional recovery key stored offline
Metadata tampering	Authenticate header as associated data
Quantum threat	Crypto-agile format; ML-KEM key wrapping

TABLE III Risks and Mitigation

X. CONCLUSION

Protecting files does not require new cryptography; it requires correct use of existing standards. The reviewed literature shows that AES-256-GCM provides efficient authenticated encryption, that salted and memory-hard key derivation is essential when keys come from passwords, and that SHA-256 remains a reliable tool for integrity verification while MD5 and SHA-1 should be retired. An architecture that authenticates headers, generates unique nonces, keeps keys in memory only and fails safely on tag mismatch provides a simple, reliable and practical solution for securing confidential files in personal, academic and organizational settings.

1. Acknowledgment

The authors express their sincere gratitude to the Department of Artificial Intelligence and Data Science, Jaihind College of Engineering, Pune, for providing the facilities, guidance and encouragement needed to carry out this work. The authors also thank the faculty members and colleagues whose valuable suggestions helped improve the quality of this review.

REFERENCES

1. National Institute of Standards and Technology, "Advanced Encryption Standard (AES)," FIPS PUB 197, 2001 (updated 2023).
2. J. Daemen and V. Rijmen, The Design of Rijndael: AES—The Advanced Encryption Standard. Berlin, Germany: Springer, 2002.
3. M. Dworkin, "Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC," NIST Special Publication 800-38D, 2007.
4. National Institute of Standards and Technology, "Secure Hash Standard (SHS)," FIPS PUB 180-4, 2015.
5. M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," in Proc. ASIACRYPT, 2000, pp. 531–545.
6. D. A. McGrew and J. Viega, "The security and performance of the Galois/Counter Mode (GCM) of operation," in Proc. INDOCRYPT, 2004, pp. 343–355.
7. H. Böck, A. Zauner, S. Devlin, J. Somorovsky, and P. Jovanovic, "Nonce-disrespecting adversaries: Practical forgery attacks on GCM in TLS," in Proc. 10th USENIX Workshop Offensive Technol. (WOOT), 2016.
8. K. Moriarty, B. Kaliski, and A. Rusch, "PKCS #5: Password-based cryptography specification version 2.1," IETF RFC 8018, 2017.
9. A. Biryukov, D. Dinu, and D. Khovratovich, "Argon2: New generation of memory-hard functions for password hashing and other applications," in Proc. IEEE Eur. Symp. Security Privacy (EuroS&P), 2016, pp. 292–302.
10. H. Krawczyk and P. Eronen, "HMAC-based extract-and-expand key derivation function (HKDF)," IETF RFC 5869, 2010.
11. X. Wang and H. Yu, "How to break MD5 and other hash functions," in Proc. EUROCRYPT, 2005, pp. 19–35.
12. M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, "The first collision for full SHA-1," in Proc. CRYPTO, 2017, pp. 570–596.
13. Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF protocols," IETF RFC 8439, 2018.
14. National Institute of Standards and Technology, "Module-Lattice-Based Key-Encapsulation Mechanism Standard," FIPS PUB 203, 2024.
15. E. Barker, "Recommendation for key management: Part 1—General," NIST Special Publication 800-57 Part 1 Rev. 5, 2020.