International Journal of Computational Research in Science and Technology
E-ISSN: 3108-0871, Volume 2 Issue 5
Sep-Oct- 2026, PP 01-6

File Integrity Verification Using Cryptographic Hashing: A Review Of Hash Functions, Keyed Hashes, Merkle Trees And Integrity Monitoring Systems

Patel Faheem Faruk[1]; Prof. J. M. Patade[2]; Prof. M. C. Kolse[3]; Prof. S. K. Said4

Department Of Artificial Intelligence And Data Science Jaihind College Of Engineering Pune, Maharashtra, India;

Abstract—Files on computers, servers and cloud storage can be altered by malware, intruders, faulty hardware or insiders, often without visible signs. File integrity verification detects such changes by computing a cryptographic fingerprint of each file and comparing it with a trusted reference. This paper reviews the cryptographic foundations and system designs for file integrity verification. It explains the properties of cryptographic hash functions, namely preimage, second-preimage and collision resistance, and examines the families in use, including the deprecated MD5 and SHA-1, which suffered practical collisions, the SHA-2 family and faster modern designs such as BLAKE2. Keyed-hash message authentication codes and digital signatures are reviewed as means of protecting stored hashes from attackers who could otherwise recompute them. Merkle hash trees for efficient verification of large file collections, file integrity monitoring tools, kernel-level integrity measurement and transparency logs are discussed. A five-layer architecture comprising file discovery, hashing, protected baseline storage, scheduled and event-driven verification, and alerting with response is proposed, together with research gaps, objectives, a comparison of hash algorithms and challenges such as legitimate updates, performance on large files and baseline tampering. The review concludes that strong hashes combined with protected baselines provide reliable tamper detection.

Index Terms—File integrity verification, cryptographic hash functions, SHA-256, HMAC, digital signatures, Merkle tree, integrity monitoring, tamper detection

I. Introduction

1.1 Motivation

When an attacker compromises a server, one of the first actions is often to modify system files, install backdoors or alter configuration so that access persists. Ransomware and malware likewise change files. Detecting these changes quickly is a basic security requirement and is mandated by many compliance standards.

A cryptographic hash function maps a file of any size to a short, fixed-length digest. If even one bit of the file changes, the digest changes unpredictably. Comparing current digests with stored reference values therefore reveals modifications. Kim and Spafford [1] designed Tripwire, an early file system integrity checker based on this idea, which influenced many later tools.

This paper reviews hash functions and the system designs used for file integrity verification, and proposes an integrity monitoring architecture.

1.2 Objectives of the Review

1. 1) To explain the security properties of cryptographic hash functions.

2. 2) To review hash algorithms and their current security status.

International Journal of Computational Research in
Science and Technology
E-ISSN: 3108-0871, Volume 2 Issue 5
Sep-Oct- 2026, PP 01-6

3. 3) To examine HMAC, digital signatures and Merkle trees for protecting integrity data.

4. 4) To propose a file integrity monitoring architecture.

5. 5) To identify challenges and best practices.

II. Overview of Key Concepts

2.1 Properties of Hash Functions

Rogaway and Shrimpton [2] formalized three security properties: preimage resistance (given a digest, it is hard to find an input), second-preimage resistance (given an input, it is hard to find another with the same digest) and collision resistance (it is hard to find any two inputs with the same digest). Fig. 1 organizes integrity mechanisms.

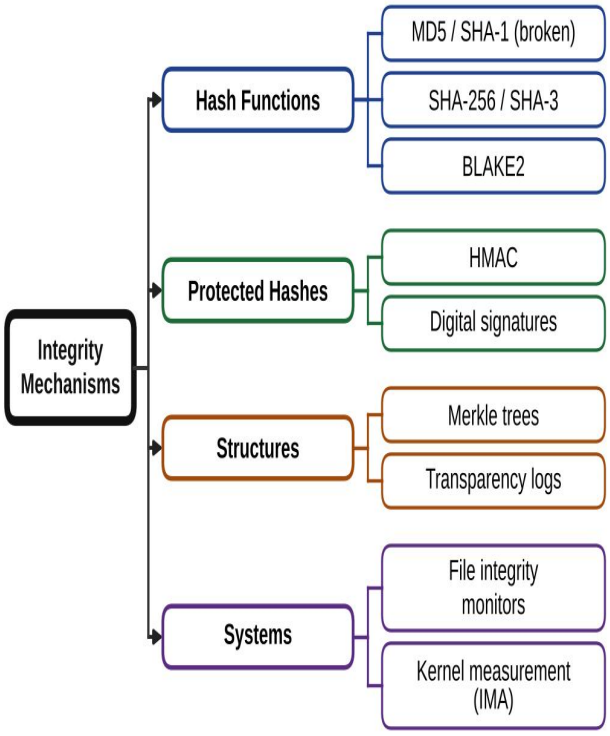


Fig. 1. Mechanisms for file integrity verification.

2.2 Hash Standards

The Secure Hash Standard [3] specifies SHA-1 and the SHA-2 family, and NIST guidance [4] describes which algorithms are approved for which applications.

III. Review of Existing Work

Wang and Yu [5] demonstrated practical collisions for MD5, and Stevens *et al.* [6] produced the first collision for full SHA-1, so neither should be relied on where collision resistance matters. SHA-256 remains secure, and Aumasson *et al.* [7] designed BLAKE2, which is faster than MD5 in software while providing modern security margins.

A plain hash can be recomputed by an attacker who modifies a file. Bellare *et al.* [8] introduced keyed hashing for message authentication, standardized as HMAC [9], so that only holders of a secret key can produce valid tags. Digital signatures [10] go further, allowing anyone to verify that a baseline was produced by an authorized party.

Merkle [11] proposed hash trees in which each parent node hashes its children, so the integrity of many files can be summarized by one root hash and individual files verified with short proofs. Certificate Transparency [12] uses Merkle trees to create tamper-evident public logs.

Kim and Spafford [1] described Tripwire's design, including a signed database of file hashes and attributes checked periodically. Sailer *et al.* [13] implemented an integrity measurement architecture in the Linux kernel that hashes files as they are loaded and records the values in a trusted platform module, enabling remote attestation. Table I summarizes these works.

Ref.	Author(s) / Year	Technique / Approach	Key Contribution	Limitation / Gap
[1]	Kim and Spafford, 1994	Tripwire integrity checker	Signed hash database for files	Periodic checks only
[11]	Merkle, 1987	Hash trees	Efficient verification of many items	Requires root protection
[8]	Bellare et al., 1996	Keyed hash functions (HMAC basis)	Provable message authentication	Key management needed
[2]	Rogaway and Shrimpton, 2004	Formal hash properties	Clarified security notions	Theoretical
[13]	Sailer et al., 2004	Kernel integrity measurement	Load-time hashing with TPM	Platform-specific
[5]	Wang and Yu, 2005	MD5 collision attack	Practical MD5 collisions	Algorithm-specific
[7]	Aumasson et al., 2013	BLAKE2 hash	Faster than MD5, secure	Less widely mandated
[6]	Stevens et al., 2017	Full SHA-1 collision	SHA-1 broken in practice	Algorithm-specific

TABLE I Summary of Literature Survey

IV. Identified Research Gaps

1. 1) Many tools still offer MD5 or SHA-1 options that users may select.
2. 2) Baselines stored on the same system can be modified by an attacker with administrative access.
3. 3) Distinguishing legitimate updates from malicious changes generates alert noise.
4. 4) Hashing very large files and storage systems frequently is expensive.
5. 5) Integrity verification in cloud and container environments is less mature.

International Journal of Computational Research in Science and Technology
E-ISSN: 3108-0871, Volume 2 Issue 5
Sep-Oct- 2026, PP 01-6

V. System Architecture

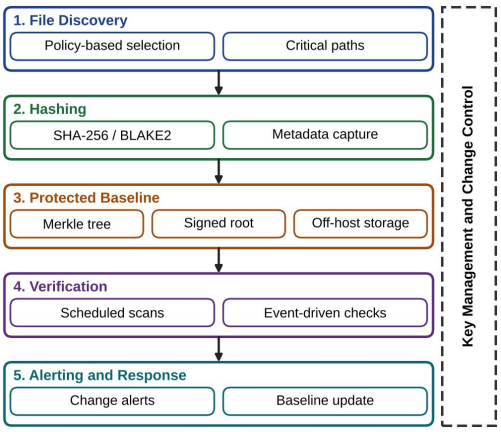


Fig. 2 shows the proposed architecture.

Fig. 2. Proposed file integrity monitoring architecture.

5.1 File Discovery Layer

Critical directories, configuration files, executables and data stores are selected according to policy.

5.2 Hashing Layer

Each file is hashed with SHA-256 or BLAKE2; metadata such as size, permissions and modification time is recorded.

5.3 Protected Baseline Layer

Hashes are organized in a Merkle tree; the root is signed with a private key or authenticated with HMAC and stored off-host or in write-once storage.

5.4 Verification Layer

Files are re-hashed on a schedule and on change events; current hashes are compared with the baseline and Merkle proofs pinpoint changed files.

5.5 Alerting and Response Layer

Unauthorized changes trigger alerts with file details; approved changes from change management update the baseline after re-signing.

VI. Implementation Methodology

Fig. 3 describes the workflow. For file F, the stored reference is

$$h = SHA\text{-}256(F) \quad (1)$$

and a keyed tag protecting the baseline is

International Journal of Computational Research in Science and Technology
E-ISSN: 3108-0871, Volume 2 Issue 5
Sep-Oct- 2026, PP 01-6

$$HMAC(K, m) = H((K \oplus opad) \| H((K \oplus ipad) \| m)) \quad (2)$$

In a Merkle tree, each internal node is H(left ‖ right), so a change in any file changes the root, and verification of one file requires only $\log_2 n$ hashes.

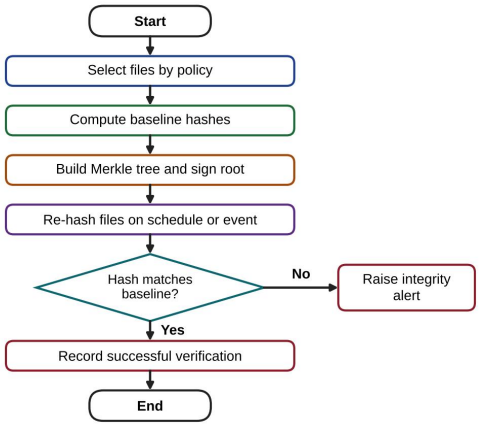


Fig. 3. File integrity verification workflow.

VII. Performance Comparison

Table II compares hash algorithms. MD5 and SHA-1 are broken for collision resistance, SHA-256 is the widely approved standard, SHA-3 offers a different design, and BLAKE2 provides high speed with strong security.

Algorithm	Digest Size	Collision Status	Recommendation
MD5	128-bit	Broken	Do not use
SHA-1	160-bit	Broken	Do not use
SHA-256	256-bit	Secure	Recommended
SHA-3-256	256-bit	Secure	Recommended
BLAKE2b	Up to 512-bit	Secure	Recommended (fast)

TABLE II Comparison of Hash Algorithms

VIII. Open Issues and Future Directions

Table III summarizes challenges. Future directions include integrity verification for containers and serverless functions, hardware-rooted attestation, transparency logs for software supply chains and post-quantum signatures for long-lived baselines.

Challenge	Mitigation
Baseline tampering	HMAC / signatures; off-host storage
Legitimate updates	Change management integration
Large files	Incremental and Merkle hashing
Weak algorithms	Enforce SHA-256 or stronger
Key compromise	Hardware key storage; rotation

TABLE III Challenges and Mitigation

IX. Conclusion

Cryptographic hashing provides a simple and powerful way to detect unauthorized changes to files. The reviewed literature shows that the choice of hash function matters, that baselines must be protected with HMAC or digital signatures, and that Merkle trees make verification efficient at scale. Combining these elements in a monitoring system with change management yields reliable tamper detection for personal, academic and organizational systems.

9.1 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. G. H. Kim and E. H. Spafford, “The design and implementation of Tripwire: A file system integrity checker,” in Proc. 2nd ACM Conf. Comput. Commun. Secur. (CCS), 1994, pp. 18–29.
2. P. Rogaway and T. Shrimpton, “Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance,” in Proc. Fast Softw. Encryption (FSE), 2004, pp. 371–388.
3. National Institute of Standards and Technology, “Secure Hash Standard (SHS),” FIPS PUB 180-4, 2015.
4. Q. Dang, “Recommendation for applications using approved hash algorithms,” NIST Special Publication 800-107 Rev. 1, 2012.
5. X. Wang and H. Yu, “How to break MD5 and other hash functions,” in Proc. EUROCRYPT, 2005, pp. 19–35.
6. 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.
7. J.-P. Aumasson, S. Neves, Z. Wilcox-O’Hearn, and C. Winnerlein, “BLAKE2: Simpler, smaller, fast as MD5,” in Proc. Appl. Cryptogr. Netw. Secur. (ACNS), 2013, pp. 119–135.
8. M. Bellare, R. Canetti, and H. Krawczyk, “Keying hash functions for message authentication,” in Proc. CRYPTO, 1996, pp. 1–15.
9. H. Krawczyk, M. Bellare, and R. Canetti, “HMAC: Keyed-hashing for message authentication,” IETF RFC 2104, 1997.
10. National Institute of Standards and Technology, “Digital Signature Standard (DSS),” FIPS PUB 186-5, 2023.
11. R. C. Merkle, “A digital signature based on a conventional encryption function,” in Proc. CRYPTO, 1987, pp. 369–378.
12. B. Laurie, A. Langley, and E. Kasper, “Certificate transparency,” IETF RFC 6962, 2013.
13. R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, “Design and implementation of a TCG-based integrity measurement architecture,” in Proc. 13th USENIX Security Symp., 2004.