

ChunkGuard: a user-space secure file storage system with chunk-level encryption and integrity protection

Biryomumeisho Joshua

Universal Technology and Management University (UTAMU)

Ocen Emmy Fred

Universal Technology and Management University (UTAMU)

Bbosa Denis

Mbarara University of Science and Technology

Fred Paul Mark Jjunju

`fpm.jjun.ju@utam.ac.ug`

Universal Technology and Management University (UTAMU)

Short Report

Keywords: ChunkGuard, secure file storage, data-at-rest security, chunk-level encryption, AES-CTR, HMAC-SHA256, PBKDF2, tamper detection

Posted Date: October 22nd, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11164566/v1>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

Abstract

Objective

Data stored in plaintext may become exposed when conventional storage protections are bypassed through device theft, direct storage access, or improperly sanitized media. This study evaluated ChunkGuard, a user-space secure file storage prototype integrating confidentiality, integrity verification, access control, and tamper detection. Files were divided into 1024-byte chunks protected using AES-128-CTR with per-chunk keys and initialization vectors, HMAC-SHA256 authentication, PBKDF2-HMAC-SHA1 user authentication, and SHA-256 metadata integrity checks.

Results

ChunkGuard passed all 30 functional and security-oriented test cases. For a 1 MB file, mean encryption, decryption, and integrity-verification times were 160.833 ms, 78.559 ms, and 16.734 ms, corresponding to throughputs of 6.2 MB/s and 12.7 MB/s for encryption and decryption. Chunking produced the theoretical minimum number of chunks across tested boundary conditions. Deliberate ciphertext modification was detected through HMAC verification, and unauthorized cross-user operations were rejected. Component profiling showed that chunk-file disk writes accounted for 78.7% of encryption time, compared with 17.2% for AES encryption, 3.0% for HMAC computation, and 1.0% for key and initialization-vector generation, identifying storage I/O as the performance bottleneck. These findings demonstrate feasible fine-grained authenticated storage while highlighting the need for improved chunk organization.

Introduction

Protection of data at rest is important because storage media can be copied, removed, lost, stolen, or accessed outside the normal application login path [1]. Mature systems such as BitLocker, eCryptfs, Linux fscrypt, and OpenZFS address this problem at volume, filesystem, or dataset level [2–5]. ChunkGuard explores a deliberately smaller design point: a user-space prototype in which file contents are divided into independently managed 1024-byte chunks so that encryption, integrity checks, ownership enforcement, and performance costs can be observed directly.

The study does not propose a new cryptographic primitive and does not position ChunkGuard as a replacement for production encrypted filesystems. Its objective is to evaluate whether a fine-grained chunk architecture can implement confidentiality-oriented storage protection and explicit tamper detection correctly within a constrained prototype, and to identify where the implementation spends its execution time. The design uses established cryptographic building blocks and evaluates the actual implementation through functional, security-oriented, boundary, and performance tests.

System design and evaluation

ChunkGuard was implemented as a user-space desktop application in Python using PyCryptodome for cryptographic operations. The system separates authentication, file operations, validation and chunk management, cryptographic helpers, and metadata management. Existing-file operations first perform an ownership check so that a caller who is not the recorded owner is rejected before chunk contents are processed.

Each plaintext file is divided into fixed 1024-byte chunks. AES-128 in Counter (CTR) mode provides confidentiality [6, 7]. A fresh 128-bit key and initialization vector are generated for each chunk on each write. HMAC-SHA256 is computed over each ciphertext chunk to detect modification [8], and SHA-256 digests are used for master-metadata consistency checks [9]. User passwords are processed with a random 32-byte salt and PBKDF2-HMAC-SHA1 using 20,000 iterations and a 32-byte output [10, 11]. The current prototype uses the same per-chunk secret for AES and HMAC; this is retained as an implementation characteristic and is treated as a security-hardening limitation rather than a recommended construction.

The file API supports creation, encryption, decryption, positional read and write, size queries, deletion, per-file integrity checking, and system-wide health checking. The implementation also includes real-file import, in-place encryption, and a recovery sidecar. These extended functions are described in Additional file 1 because they are not necessary to interpret the primary performance results.

Evaluation used four complementary test groups. First, a 30-case automated suite covered authentication and input validation, file-lifecycle operations, access-control enforcement, deliberate tampering, and master-list health checks. Second, chunking behavior was tested at and around 1024-byte boundaries from 1 to 5000 bytes. Third, encryption, decryption, and integrity-check timings were measured over files ranging from 16 bytes to 1 MB, with five repetitions per file size. Fourth, the 1 MB encryption path was profiled into key/initialization-vector generation, AES-CTR encryption, HMAC computation, and physical chunk-file writes. All reported values came from execution of the implemented functions rather than analytical estimates. Detailed parameters, test categories, and benchmark tables are provided in Additional file 1.

Results

All 30 automated functional and security-oriented tests passed. The test suite included 8 authentication and validation cases, 12 core file-lifecycle cases, 2 access-control cases, 3 tamper/integrity cases, and 5 system-health/master-list cases. Boundary testing produced the theoretical minimum number of chunks, $\lceil \text{size}/1024 \rceil$, at every tested size, including 1023, 1024, and 1025 bytes.

At 1 MB, mean encryption time was 160.833 ms, decryption time was 78.559 ms, and integrity verification required 16.734 ms. These measurements correspond to approximate throughputs of 6.2

MB/s for encryption and 12.7 MB/s for decryption. Representative results for smaller files and the complete benchmark description are supplied in Supplementary information.

Component profiling changed the interpretation of the performance bottleneck. For the 1 MB workload, physical chunk-file writes consumed 145.17 ms, or 78.7% of the profiled encryption-path time. AES-128-CTR accounted for 31.78 ms (17.2%), HMAC-SHA256 for 5.61 ms (3.0%), and key/initialization-vector generation for 1.91 ms (1.0%) for details see Table 1. Thus, in this implementation, creating and writing many small chunk files cost substantially more than the cryptographic operations themselves.

Security-oriented tests produced the expected behavior within the stated threat model. Flipping a single ciphertext byte caused HMAC verification to fail rather than returning silently corrupted plaintext. A second authenticated user attempting to query or decrypt a file owned by another user received an access-denied response. These tests demonstrate implemented behavior under specific scenarios; they are not a formal security proof.

Table 1
Summary of primary ChunkGuard evaluation findings

Evaluation	Measure	Result
Functional/security tests	Automated test suite	30/30 passed (100%)
1 MB performance	Mean encryption time	160.833 ms
1 MB performance	Mean decryption time	78.559 ms
1 MB performance	Mean integrity-verification time	16.734 ms
1 MB performance	Approximate throughput	6.2 MB/s encrypt; 12.7 MB/s decrypt
Component profile	Chunk-file disk writes	145.17 ms (78.7%)
Component profile	AES-128-CTR	31.78 ms (17.2%)
Component profile	HMAC-SHA256	5.61 ms (3.0%)
Component profile	Key/IV generation	1.91 ms (1.0%)
<i>Note: Component profiling was an instrumented profiling run; its summed component time should not be treated as the same timing measurement as the separate end-to-end mean encryption benchmark.</i>		

Interpretation and context

The main contribution is a measured systems observation rather than a claim of cryptographic novelty (Table 1). ChunkGuard shows that a user-space application can combine chunk-level encryption, explicit integrity checking, and ownership enforcement in a reproducible small-scale prototype. The most informative empirical result is that storage organization, not cipher execution, dominated the measured

encryption path. This points directly to an engineering hypothesis for the next version: preserve logical chunk-level verification while consolidating physical storage into indexed containers or larger objects.

The result is consistent with the design choices of mature encrypted-storage systems, which generally operate at coarser physical granularity and use established key hierarchies. Linux fscrypt divides file contents into independently encrypted data units and notes that unnecessarily small data units reduce performance [3]. OpenZFS uses authenticated-encryption suites at dataset level, while BitLocker protects whole volumes [4, 5]. These systems solve broader deployment problems than ChunkGuard and are not performance comparators for the present experiment.

A production-oriented successor should also modernize the cryptographic composition. AES-CTR is a confidentiality mode and does not itself authenticate ciphertext [7]. The prototype adds HMAC separately, whereas authenticated-encryption modes such as AES-GCM integrate confidentiality and authentication in one construction [12]. Future experiments should compare a properly key-separated encrypt-then-MAC design with an authenticated-encryption alternative, while measuring throughput, metadata overhead, storage amplification, and random-access performance under the same workloads.

Limitations

The study has several limitations. The maximum benchmarked file size was 1 MB, and performance was measured in a single development environment; therefore, the reported timings should not be generalized to large files, concurrent workloads, network storage, cloud-object storage, or other hardware. The metadata store is unencrypted and contains chunk-related key material, which makes the prototype unsuitable for production deployment when an attacker can read application metadata. The same per-chunk secret is used for AES-CTR and HMAC rather than deriving separate encryption and authentication keys. The recovery sidecar also exposes sensitive recovery material and requires redesign before practical use.

The application is not concurrency-safe, system-wide integrity checking scales linearly with the number of files, and positional reads currently process a full chunk chain rather than directly seeking to the required chunk. The study evaluated scenario-based tampering and access-control behavior but did not perform formal security analysis, adversarial fuzzing, crash-consistency testing, replay/rollback testing, or interoperability testing against an independent implementation. These limitations define the next evaluation priorities and constrain the conclusions to the implemented prototype and tested workloads.

Abbreviations

AES: Advanced Encryption Standard; CTR: Counter mode; HMAC: Hash-based Message Authentication Code; PBKDF2: Password-Based Key Derivation Function 2; SHA: Secure Hash Algorithm; IV: Initialization vector; I/O: Input/output; GUI: Graphical user interface.

Declarations

Supplementary Information

Detailed threat model and architecture, cryptographic parameters, implementation details, functional test-suite summary, representative benchmark results, component-level profiling, boundary validation, and security-validation scenarios.

Ethics approval and consent to participate

Not applicable. This study did not involve human participants, human data, human tissue, or animals; it reports the design and performance evaluation of a software system.

Consent for publication

Not applicable. This manuscript does not contain any individual person's data in any form.

Availability of data and materials

The datasets and source code generated and/or analysed during the study are available from the corresponding author on reasonable request.

Competing interests

The authors declare that they have no competing interests.

Funding

Not applicable.

Authors' contributions

BJ, OEF, FPMJ and DB contributed equally to the study. All authors read and approved the final manuscript.

Acknowledgements

Not applicable.

References

1. Scarfone KA, Souppaya MP, Sexton M. Guide to storage encryption technologies for end user devices. NIST Special Publication 800 – 111. Gaithersburg: National Institute of Standards and Technology; 2007. 10.6028/NIST.SP.800-111.
2. Halcrow MA. eCryptfs: an enterprise-class encrypted filesystem for Linux. In: Proceedings of the Linux Symposium. Ottawa; 2005. pp. 201–218.
3. Linux Kernel Documentation. Filesystem-level encryption (fscrypt). <https://docs.kernel.org/filesystems/fscrypt.html>. Accessed 24 Sep 2026.
4. Microsoft. BitLocker overview. <https://learn.microsoft.com/en-us/windows/security/operating-system-security/data-protection/bitlocker/>. Accessed 24 Sep 2026.
5. OpenZFS. zfsprops(7): encryption property. <https://openzfs.github.io/openzfs-docs/man/master/7/zfsprops.7.html>. Accessed 24 Sep 2026.
6. National Institute of Standards and Technology. Advanced Encryption Standard (AES). FIPS PUB 197-upd1. 2023. 10.6028/NIST.FIPS.197-upd1
7. Dworkin M. Recommendation for block cipher modes of operation: methods and techniques. NIST Special Publication 800-38A. 2001. 10.6028/NIST.SP.800-38A
8. National Institute of Standards and Technology. The Keyed-Hash Message Authentication Code (HMAC). FIPS PUB 198-1. 2008. 10.6028/NIST.FIPS.198-1
9. National Institute of Standards and Technology. Secure Hash Standard (SHS). FIPS PUB 180-4. 2015. 10.6028/NIST.FIPS.180-4
10. Moriarty K, Kaliski B, Rusch A. PKCS #5: password-based cryptography specification version 2.1. RFC 8018. 2017. 10.17487/RFC8018
11. Turan MS, Barker EB, Burr WE, Chen L. Recommendation for password-based key derivation: part 1: storage applications. NIST Special Publication 800 – 132. 2010. 10.6028/NIST.SP.800-132.
12. Dworkin M. Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC. NIST Special Publication 800-38D. 2007. 10.6028/NIST.SP.800-38D

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- [ChunkGuardBMCRResearchNotesSupplementaryInformationFINAL.docx](#)