

File Integrity Verifier (FIV): A Cryptographic Hash-Based Approach for Digital File Integrity Verification

Jerald Benny¹

Available online at: www.xournals.com

Received 17th September 2026 | Revised 27th September 2026 | Accepted 01st October 2026

Abstract:

In the current digital era, ensuring the authenticity and integrity of files is a fundamental challenge in digital forensics and cybersecurity. This study presents the design and implementation of a File Integrity Verifier (FIV), a Python-based tool with an interactive graphical user interface (GUI) for verifying file integrity using multiple cryptographic hash algorithms. The tool generates both text and PDF-based reports containing hash values for individual files and can later verify the same files to detect any form of alteration or tampering. FIV employs popular hashing algorithms such as MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, BLAKE2, and SHA-3 variants. The GUI integrates light/dark themes for user convenience, while the generated reports are formatted in a readable tabular form with wrapped content for long hash strings. The experimental results confirm that any alteration in file content produces a completely different hash value, demonstrating the reliability of cryptographic hashing for integrity verification. This project emphasizes the importance of hash-based validation as an essential digital forensic process for ensuring evidence authenticity.

Keywords: Digital forensics, File integrity, Hash functions, Cryptographic verification, Python, GUI.

Author:

1. Business Analyst - Digital Forensic, Innefu Labs, Delhi, India.

Introduction

In the contemporary digital landscape, the integrity and authenticity of electronic data have become vital concerns for individuals, organizations, and especially forensic investigators. Every day, vast volumes of data are created, transmitted, and stored across multiple digital devices and networks. These files may include crucial documents, multimedia evidence, system logs, or legal records — all of which can be easily manipulated, intentionally or unintentionally. Such manipulations may go unnoticed unless a systematic mechanism exists to verify the originality of the digital evidence. Hence, file integrity verification plays a fundamental role in digital forensics, cybersecurity, and evidence management systems.

File integrity verification involves ensuring that a file's content has not been modified from its original version. The most reliable and widely accepted method to accomplish this is through cryptographic hash functions, which transform data into a fixed-length hash value that acts as a unique digital fingerprint for that file. If even a single byte of the file changes, the resulting hash value will differ significantly, thereby signalling potential tampering. This property known as the avalanche effect ensures the effectiveness of hash algorithms in integrity validation.

Hash functions such as MD5, SHA-1, SHA-2, SHA-3, and BLAKE2 are among the most recognized and standardized algorithms used for this purpose. They are computationally efficient and produce hash digests that are practically impossible to reverse or replicate intentionally without altering the original file. In digital forensic field, these hash values serve as digital signatures, ensuring that evidence presented in court remains unaltered from its original acquisition state.

Traditional integrity verification processes in digital forensics often rely on command-line tools or manual hashing, which can be cumbersome for investigators dealing with multiple files or directories. Moreover, the results generated in raw text formats can be difficult to interpret and present in official forensic documentation. This project bridges that gap by introducing a File Integrity Verifier (FIV), an automated, GUI-based tool designed to simplify,

visualize, and standardize the integrity verification process.

Developed entirely in Python, FIV integrates multiple hashing algorithms within a single interactive interface. It enables users to select files or entire folders, generate hash values using different cryptographic standards, and export well-structured reports in both text and PDF formats. The tool also includes a verification feature that allows investigators to compare newly generated hashes against previously saved manifests to detect any instances of tampering or data loss. By offering a clear and accessible interface with light and dark mode themes, FIV improves usability for both novice and expert forensic practitioners.

Furthermore, the tool emphasizes the importance of transparent and verifiable documentation in digital evidence handling. The generated PDF reports include formatted tables and file metadata, ensuring that forensic analysts can include these outputs directly in their official case documentation. Each report maintains visual consistency and legibility, highlighting mismatches and missing files with clear distinctions.

In essence, FIV provides an automated, accessible, and visually refined solution for ensuring digital file integrity. The tool reinforces the principle of trust through verification, enabling investigators, cybersecurity professionals, and researchers to authenticate files quickly, accurately, and confidently. By combining cryptographic rigor with a modern interface, the File Integrity Verifier sets a strong foundation for integrating automated integrity verification into everyday forensic workflows.

Methodology

The proposed File Integrity Verifier (FIV) tool was designed to automate and streamline the file integrity verification process using cryptographic hash functions. The methodology adopted for this system encompasses five major stages: input acquisition, hash generation, report generation, verification, and result visualization. Each stage is structured to ensure reliability, usability, and forensic soundness.

System Architecture Overview

The architecture of the FIV system is modular, ensuring that each functional component operates independently yet integrates seamlessly into the overall workflow. The primary modules include:

1. **User Interface Module:** Developed using Python's tkinter library, this module provides an interactive graphical interface that allows users to add files or folders, select hashing algorithms and initiate hashing or verification processes. It includes user prompts and real-time logs for transparency.
2. **Hashing Engine Module:** This module is responsible for computing cryptographic hash values using Python's built-in hashlib library. It supports multiple algorithms, including MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, BLAKE2 and SHA-3 variants. The hashing process reads file data in binary mode to ensure byte-level accuracy, producing consistent digests across different systems.
3. **Manifest Management Module:** Once hashes are computed, this module structures the output into a manifest, a dictionary that stores filenames, sizes, modification timestamps, and corresponding hash values. The manifest is exported in both TXT and PDF formats for accessibility and evidential presentation.
4. **Verification Module:** During verification, the system reads an existing manifest file and recomputes the hashes of the provided evidence files. It then performs a one-to-one comparison between the stored hash values and the newly generated ones to determine file integrity status categorizing them as OK, MISMATCH, or MISSING.

5. **Reporting Module:** Using the reportlab library, this module generates a formatted PDF report that includes the title, file metadata, hash value and results.

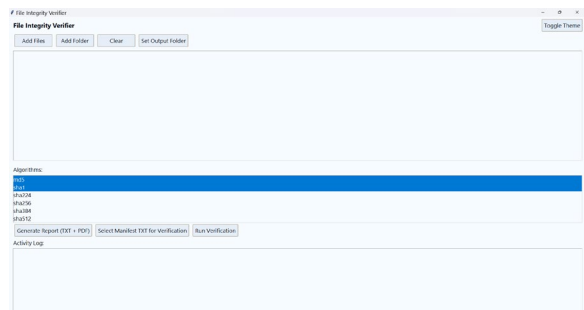


Figure 1: FIV Interface (Light mode)

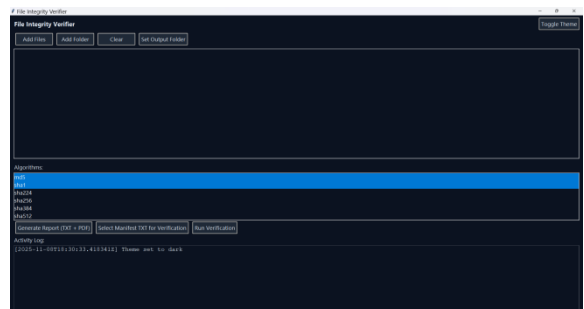


Figure 2: FIV Interface (Dark mode)

Workflow Description

The overall workflow of the FIV system can be divided into two major phases:

- i. Integrity Generation
- ii. Integrity Verification

Phase 1: Integrity Generation

1. **Input Selection:** The user selects one or more files or folders through the GUI.
2. **Algorithm Selection:** The user chooses from available hash functions (MD5, SHA-2, SHA-3, BLAKE2, etc.).

3. **Computation:** The selected algorithms compute the hash value for each file using Python's hashlib
4. **Manifest Creation:** The computed hashes, file sizes, and modification timestamps are stored in a manifest structure
5. **Report Generation:** The system exports results in TXT and PDF formats.

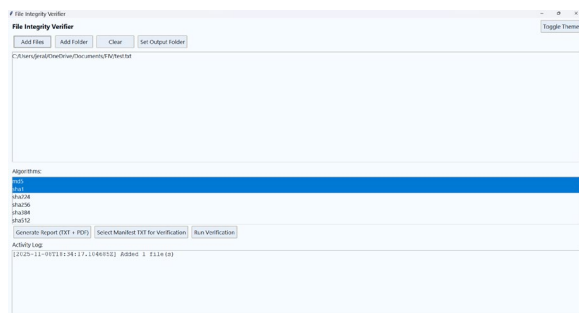
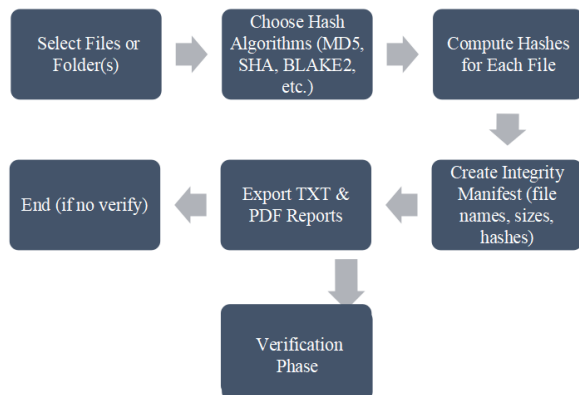


Figure 3 Selecting the file

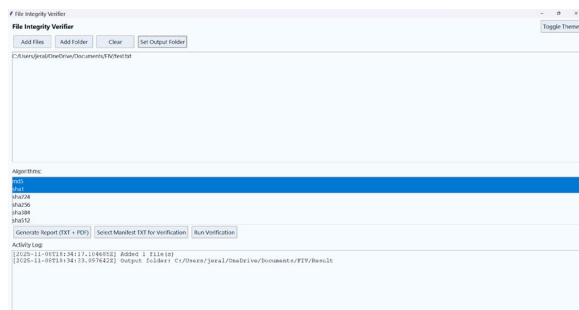


Figure 4 Setting the Output folder for saving the reports

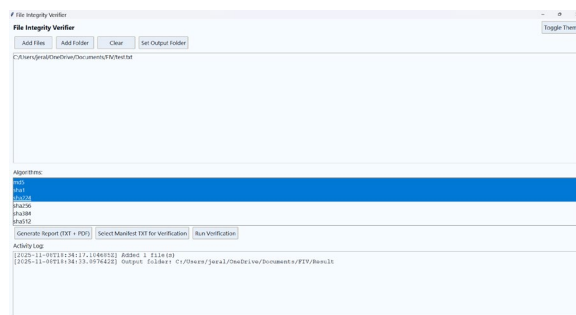


Figure 5. Select the required Hash Algorithms

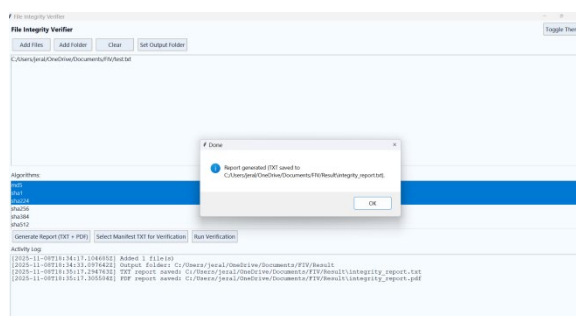


Figure 6. Generating Hash Report

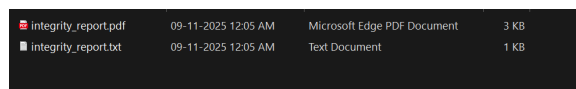


Figure 7. Integrity Report of the file in .txt & pdf format

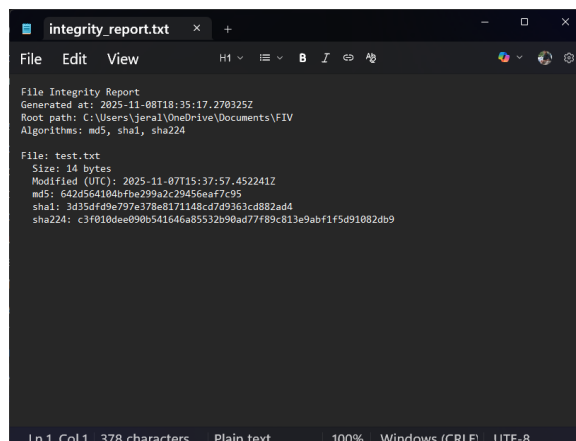


Figure 8. Integrity Report in .txt format

File Integrity Verifier (FIV)

Generated at: 2025-11-08T18:35:17.270325Z
 Root path: C:\Users\jurnal\OneDrive\Documents\FIV
 Algorithms: md5, sha1, sha224

File Name	test.txt
Size (bytes)	14
Modified (UTC)	2025-11-07T15:37:57.452241Z
md5	642d564104b2c29456eaf7c95
sha1	3d35df9e797e378e8171148cd7d9363cd982ad4
sha224	c3d10dee090b541646a8532b90ad7789c813e9abf115d91082db9

Figure 9. Integrity Report in .pdf format

Phase 2: Integrity Verification

1. Evidence Input: The user selects both the evidence files and the previously saved manifest.
2. Re-Hashing: Each evidence file is hashed using the same algorithms as the original manifest.
3. Comparison: The system compares the new hashes with the stored ones.
4. Result Classification: Files are labelled as OK, MISMATCH, or MISSING based on comparison outcomes.
5. Report: Results are displayed in the GUI's verification dialog and saved as new TXT/PDF reports.

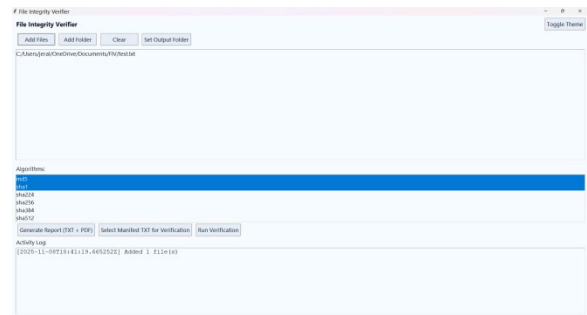
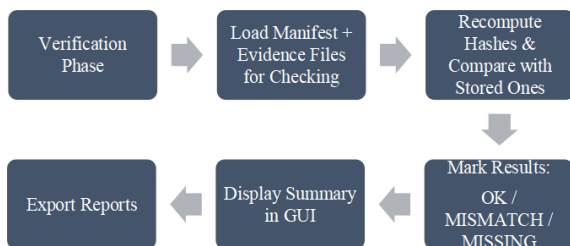


Figure 10. Adding the file for verification

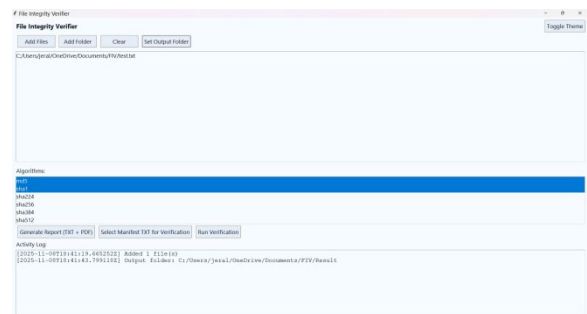


Figure 11. Setting the Output folder for saving report

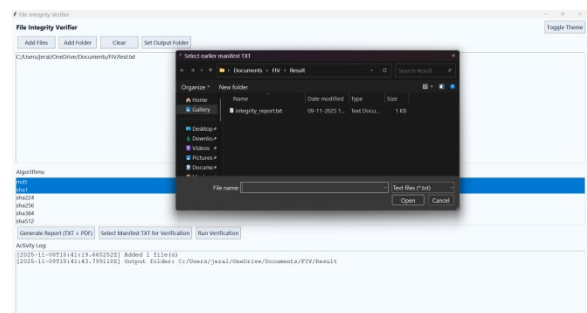


Figure 12. Selecting the manifest file (.txt) for verification

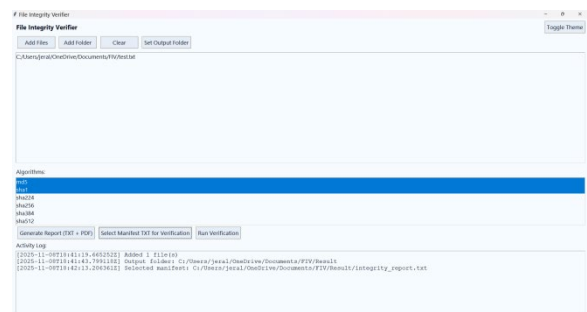


Figure 13. Manifest file (.txt) is loaded

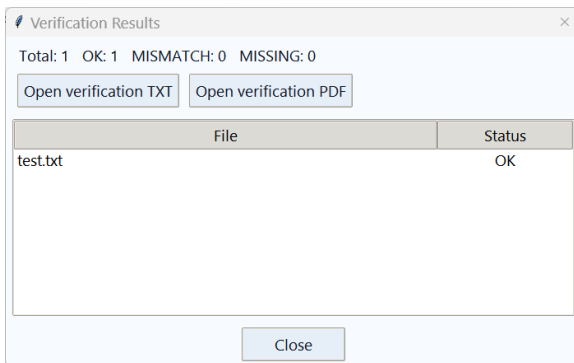


Figure 14. Verification Dialog Box of MATCHING

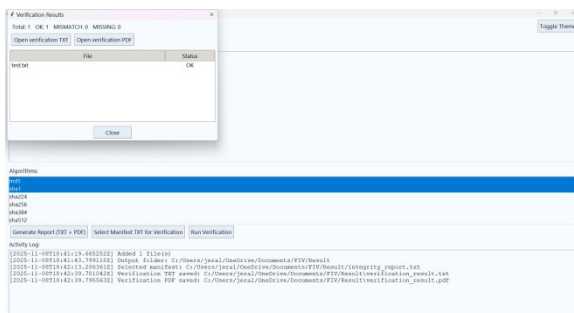


Figure 15. Verification Result Visualization of MATCHING

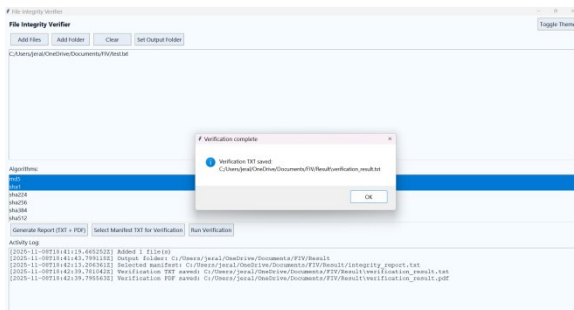


Figure 16. Verification Reports are saved in the location provided earlier

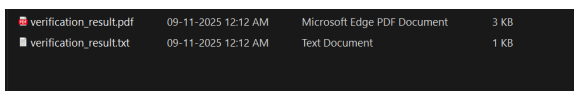


Figure 17. Verification Report of the file in .txt & pdf format

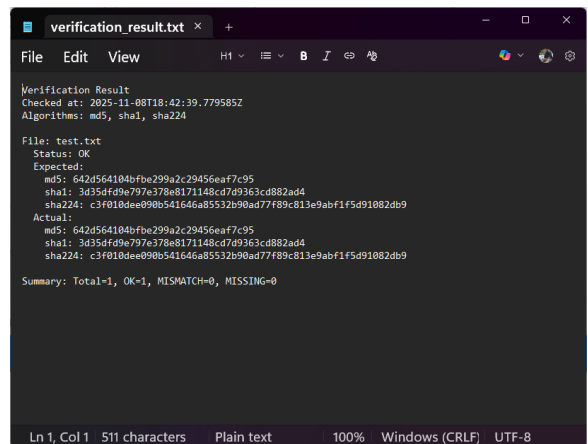


Figure 18. Verification Report in .txt format of MATCHING

Verification Report

Verification Result
Checked at: 2025-11-08T18:42:39.779585Z
Algorithms: md5, sha1, sha224

Field	Value
File Name	test.txt
Status	OK
md5 (expected)	642d564104bfbc299a2c29456eaf7c95
sha1 (expected)	3d35df9e797e378e8171148cd7d9363cd882ad4
sha224 (expected)	c3f010dee090b541646a85532b90ad77f89c813e9abf1f5d91082db9
md5 (actual)	642d564104bfbc299a2c29456eaf7c95
sha1 (actual)	3d35df9e797e378e8171148cd7d9363cd882ad4
sha224 (actual)	c3f010dee090b541646a85532b90ad77f89c813e9abf1f5d91082db9

Summary: Total=1, OK=1, MISMATCH=0, MISSING=0

Figure 19. Verification Report in .pdf format of MATCHING



Figure 20. Adding the file (tampered) for verification

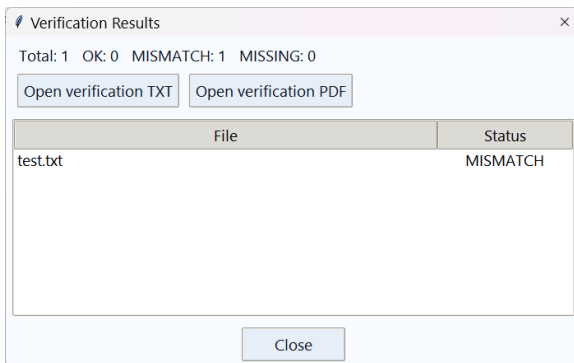


Figure 21.Verification Dialog Box of MISMATCHING

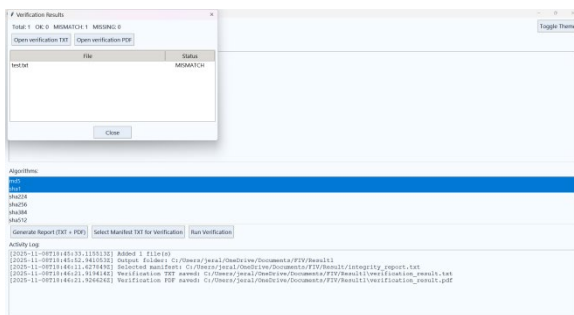


Figure 22.Verification Result Visualization of MISMATCHING

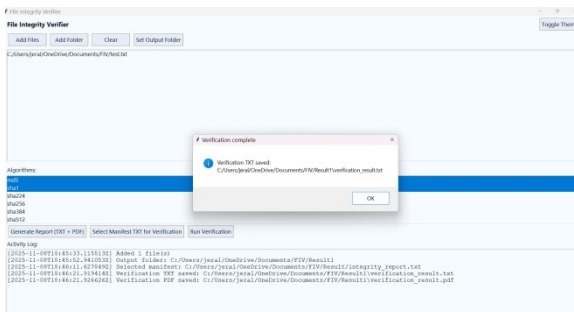


Figure 23.Verification Reports are saved in the location provided earlier

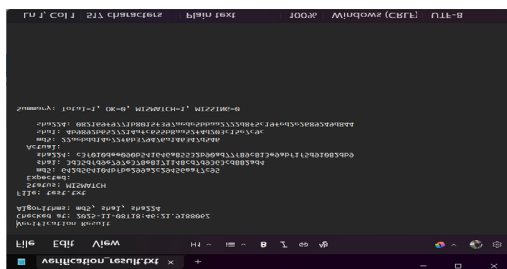


Figure 24.Verification Report in .txt format of MISMATCHING

Verification Report

Verification Result
Checked at: 2025-11-08T18:46:21.91806Z
Algorithms: md5, sha1, sha224

Field	Value
File Name	test.txt
Status	MISMATCH
md5 (expected)	6420564104bfe299a2c29456eaf7c95
sha1 (expected)	3d35df09e797e378e8171148cd7d9363cd882ad4
sha224 (expected)	c3f010dee090b541646a85532b90ad77f89c813e9abf1f5d91082db9
md5 (actual)	22aebdd14e726b379476a146347d546
sha1 (actual)	4b9892b6527214afc55b8aa52f4d203c15e7c9c
sha224 (actual)	082169f971b8015f397aede5bbaa2722d9f5c19fed2e2689249d844

Summary: Total=1, OK=0, MISMATCH=1, MISSING=0

Figure 25.Verification Report in .pdf format of MISMATCHING

GUI Design

The GUI offers a minimal, user-friendly interface featuring:

- File/Folder selection options
- Hash algorithm selection panel
- Activity log viewer
- Result dialog box summarizing total, OK, mismatched, and missing files
- Integrity Report and Verification Report (TXT and PDF)
- Light/Dark mode toggle

Report Generation

Reports are generated in two formats:

- Text (.txt): Plain textual summary for portability.
- PDF: Formatted tables with clear layout, and hash strings.

File Name	Original Status	Action Applied	Expected Result	Actual Tool Output
text.txt	Original	None	Match	OK
office.exe	Original	None	Match	OK
note.docx	Original	None	Match	OK
text.txt	Original	Modified	Mismatch	MISMATCH
image1.png	Original	Modified	Mismatch	MISMATCH
note.docx	Original	Modified	Mismatch	MISMATCH
image2.png	Original	Deleted	Missing File	MISSING
note.pdf	Original	Deleted	Missing File	MISSING

Table 1. Integrity & Verification Results

Conclusion

The development and implementation of the File Integrity Verifier (FIV) tool mark a significant advancement in simplifying and standardizing the process of digital file integrity verification for forensic and cybersecurity applications. Through the integration of multiple cryptographic hash functions, a user-friendly graphical interface,

and professional reporting capabilities, the tool successfully bridges the gap between technical integrity validation and practical forensic usability.

The experimental evaluation confirmed the tool's reliability and efficiency. FIV accurately identified altered, missing, and unmodified files through byte-level hash comparison, thereby validating the effectiveness of cryptographic functions such as MD5, SHA-2, SHA-3, and BLAKE2. The clear differentiation between OK, MISMATCH, and MISSING statuses reinforces the forensic value of this tool in detecting tampering or unauthorized modification of digital evidence. Furthermore, the automatically generated TXT and PDF reports support documentation requirements for forensic purposes.

From a forensic perspective, the FIV system ensures that evidence authenticity can be verified in a reproducible and legally defensible manner. Every generated report includes time stamps, file metadata, and cryptographic details, strengthening the chain of custody and compliance with digital evidence handling standards.

The File Integrity Verifier (FIV) successfully demonstrates the application of cryptographic hash functions in digital forensic validation. Its multi-algorithm support, clear reporting, and user-friendly GUI make it a practical forensic utility. By generating both TXT and PDF reports and supporting multi-algorithm verification, it bridges functionality with accessibility for forensic professionals.

References:

- Casey, Eoghan. *Digital Evidence and Computer Crime: Forensic Science, Computers, and the Internet*. 3rd ed., Academic Press, 2011.
- Casey, Eoghan. "Practical Approaches to Recovering Digital Evidence." *Forensic Science Review*, vol. 31, no. 1, 2019, pp. 45–64.
- Easttom, Chuck. *System Forensics, Investigation, and Response*. 4th ed., Jones & Bartlett Learning, 2023.
- Garfinkel, Simson L. "Digital forensics research: The next 10 years." *Digital Investigation*, vol. 7, 2010, pp. S64–S73. Elsevier.
- Kessler, Gary C., and John S. Haggerty. "An Overview of Hash Algorithms and Their Role in Digital Forensics." *Journal of Digital Forensics, Security and Law*, vol. 9, no. 2, 2014, pp. 1–14.
- Menezes, Alfred J., Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
- National Institute of Standards and Technology (NIST). *Secure Hash Standard (SHS): FIPS PUB 180-4*. U.S. Department of Commerce, 2015.
- Nelson, Bill, Amelia Phillips, and Christopher Steuart. *Guide to Computer Forensics and Investigations*. 7th ed., Cengage Learning, 2022.
- Nolan, Richard, et al. *Guide to Integrating Forensic Techniques into Incident Response*. Carnegie Mellon University, Software Engineering Institute, 2005.
- Russell, Deborah, and G. T. Gangemi Sr. *Computer Security Basics*. 2nd ed., O'Reilly Media, 2006.
- Schneier, Bruce. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. 2nd ed., Wiley, 1996.
- Stallings, William. *Cryptography and Network Security: Principles and Practice*. 8th ed., Pearson Education, 2023.
- Tobiasz, M., and P. Rutkowski. "Integrity Checking and Data Authentication in Digital Evidence." *Applied Computer Science*, vol. 16, no. 2, 2020, pp. 60–74.
- United States Department of Justice. *Electronic Crime Scene Investigation: A Guide for First Responders*. 2nd ed., Office of Justice Programs, 2008.
- Van Der Knijff, Ronald. "Hash-Based Evidence Authentication in Digital Forensics." *Forensic Science International: Digital Investigation*, vol. 22, 2017, pp. 38–49.
- Zimmermann, Paul. *An Overview of SHA-3 and Modern Cryptographic Hashing*. National Institute of Standards and Technology, 2021.
- Python Software Foundation. *hashlib — Secure Hash and Message Digest Algorithms*. *Python Documentation*, version 3.12, 2024, <https://docs.python.org/3/library/hashlib.html>.
- ReportLab Developers. *ReportLab User Guide: PDF Generation for Python*. ReportLab Inc., 2024, <https://www.reportlab.com/documentation/>.