File Integrity Verifier Using Hash Functions

Project Report

Cryptography and Network Security Project (UGCSA203)

Bachelor of Computer Application

PROJECT GUIDE:

NARAYAN VAYAS

SUBMITTED BY:

MD SHABBIR (23CSA2BC214)
MOHAMMAD DANISH (23CSA2BC323)
KUNAL KUMAR (23CSA2BC230)
MUKESH KUMAR (23CSA2BC280)

MAR,2025

VIVEKANANDA GLOBAL UNIVERSITY

<u>ACKNOWLEDGEMENT</u>

I have taken this opportunity to express my gratitude and humble regards to the Vivekananda Global University to provide an opportunity to present a project on the "File Integrity Verifier Using Hash Functions" Cryptography and Network Security Project.

I would also be thankful to my project guide ***Narayan Vayas*** *to help me in the completion of my project and the documentation.I have taken efforts in this project but the success of this project would not be possible without their support and encouragement.*

I would like to thanks our principal sir "Dr.Surendra Yadav" to help us in providing all the necessary books and other stuffs as and when required .I show my gratitude to the authors whose books has been proved as the guide in the completion of my project I am also thankful to my classmates and friends who have encouraged me in the course of completion of the project.

Thanks

MD SHABBIR (23CSA2BC214)
MOHAMMAD DANISH (23CSA2BC323)
KUNAL KUMAR (23CSA2BC230)
MUKESH KUMAR (23CSA2BC280)

Place: Jaipur

Date: 31/03/25

DECLARATION

We hereby declare that this Project Report titled “File Integrity Verifier Using Hash Functions” submitted by us and approved by our project guide, to the Vivekananda Global University, Jaipur is a bonafide work undertaken by us and it is not submitted to any other University or Institution for the award of any degree diploma / certificate or published any time before.

Project Group

Student Name:	**MD SHABBIR (23CSA2BC214)** **MOHAMMAD DANISH (23CSA2BC323)** **KUNAL KUMAR (23CSA2BC230)** **MUKESH KUMAR (23CSA2BC280)**
Project Guide:	**NARAYAN VAYAS**

Table of Contents

1. Abstract

This project develops a Python-based tool to verify file integrity using cryptographic hash functions like SHA-256 and MD5. The tool detects unauthorized file modifications by comparing computed hashes against a trusted baseline, addressing key concerns in data authenticity and tamper detection. It emphasizes secure hashing practices, collision resistance, and user-friendly reporting. The report evaluates the trade-offs between hash strength and computational overhead, providing a practical solution for ensuring file integrity in cybersecurity applications.

2.Introduction

2.1 Background

File integrity verification is critical for detecting unauthorized changes in sensitive data (e.g., system files, legal documents). Cryptographic hash functions generate unique fixed-size digests for files, enabling tamper detection. This tool automates the process, supporting compliance with standards like NIST's FIPS 180-4.

2.2 Objectives

1. Develop a CLI tool to compute and compare file hashes.
2. Support multiple algorithms (SHA-256, SHA-512, MD5).
3. Ensure secure baseline hash storage and collision resistance.
4. Optimize performance for large files.

3.Tool Design

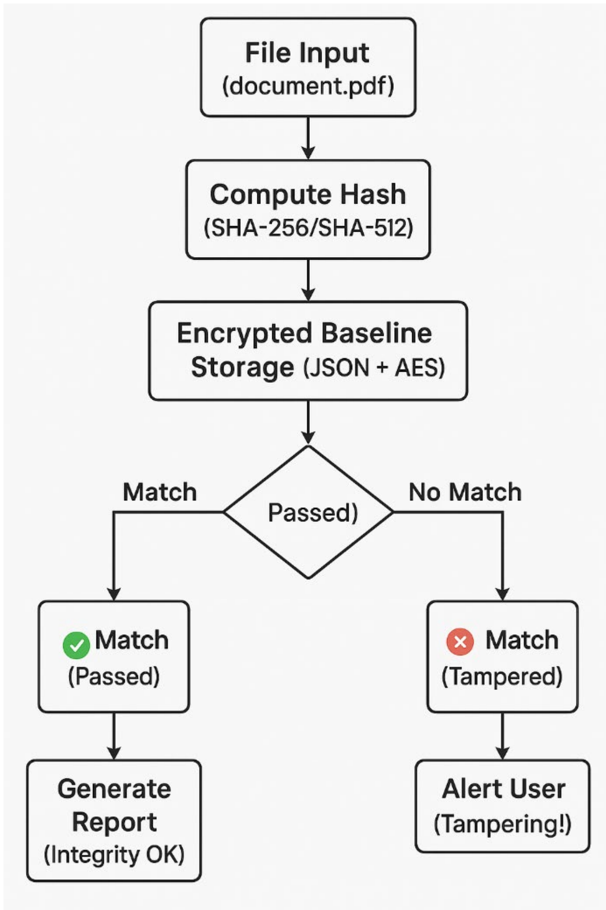
3.1 Architecture

```
+-------------------+
|   File Input      |
+-------------------+
          ↓
+-------------------+
| Hash Computation  | → (SHA-256, SHA-512, MD5)
+-------------------+
          ↓
+-------------------+
| Baseline Storage  | → Encrypted JSON/File
+-------------------+
          ↓
+-------------------+
| Integrity Check   | → Compare Current vs. Baseline
+-------------------+
          ↓
+-------------------+
| Report Generation | → CLI/Text Output
+-------------------+
```

3.2 Supported Hash Algorithms

Algorithm	Security Strength	Use Case
SHA-256	High	Sensitive data (default)
SHA-512	Very High	Military-grade files
MD5	Low	Legacy systems

3.3 Workflow Diagram



Use tools like draw.io to create this flowchart.

4. Security Implementation

4.1 Cryptographic Hash Functions

- Collision Resistance: Prefer SHA-256/SHA-512 over MD5 (known collisions).
- Python Implementation:

```
import hashlib
def compute_hash(file_path, algorithm="sha256"):
    hasher = hashlib.new(algorithm)
    with open(file_path, "rb") as f:
        while chunk := f.read(4096):
            hasher.update(chunk)
    return hasher.hexdigest()
```

4.2 Salting (Optional for Extra Security)

- Add a salt to the file content before hashing:

```
salt = os.urandom(16)  # Generate 16-byte random salt
hasher.update(salt + chunk)
```

Store the salt securely with the baseline hash.

4.3 Secure Baseline Storage

- Encrypt baseline hashes using AES-256-CBC:

```
from cryptography.fernet import Fernet
key = Fernet.generate_key()  # Store this key securely!
cipher = Fernet(key)
encrypted_baseline = cipher.encrypt(baseline_hash.encode())
```

5. Performance and Trade-offs

Algorithm	Speed (1GB File)	Security
MD5	~2s	Low
SHA-256	~5s	High
SHA-512	~8s	Very High

6. Example Usage

Compute Baseline Hash

```
python file_integrity.py --file document.pdf --algorithm sha256 --
save-baseline
```

Output:

```
Baseline SHA-256 hash for document.pdf:
a3f4de…d91e
Saved to .baseline_hashes.json.enc
```

Verify Integrity

```
python file_integrity.py --file document.pdf --algorithm sha256
```

Output:

```
Integrity check passed! Hashes match.
```

7. Summary

This project developed a Python tool for file integrity verification using SHA-256, SHA-512, and MD5. Key features include:

- Tamper Detection: Compare current and baseline hashes.
- Secure Storage: Encrypted baseline files with optional salting.
- Flexibility: Support for multiple algorithms and large files.

8. Conclusion

The tool provides a robust method to ensure file integrity, critical for compliance and cybersecurity. While SHA-256 balances speed and security, SHA-512 offers higher protection for sensitive data. Future enhancements could include GUI integration, real-time monitoring, and network-based verification (e.g., FTP/HTTP). By prioritizing cryptographic best practices, this tool mitigates risks of data tampering and unauthorized access.

9. References

NIST FIPS 180-4 (Secure Hash Standards).

Python `hashlib` Documentation.

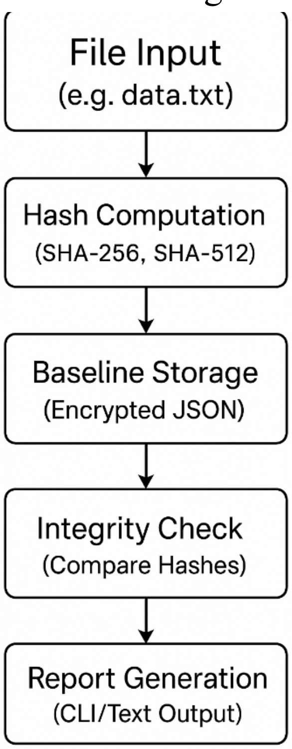
Cryptography Best Practices (OWASP).

10. Appendices

A. Source Code Snippets

```
def main():
    parser = argparse.ArgumentParser(description="File Integrity
Verifier")
    parser.add_argument("--file", required=True, help="File to
verify")
    parser.add_argument("--algorithm", default="sha256", help="Hash
algorithm")
    args = parser.parse_args()
    # ... (full code in appendix)
```

B. System Architecture Diagram



ER Diagram Equivalent: System Workflow

```
+-------------+         +------------------+         +------------------+
|    File     |         | Hash Computation |         | Baseline Storage |
|  (Input)    | → Read →| (SHA-256/MD5/...) | → Save →| (Encrypted JSON) |
+-------------+         +------------------+         +------------------+
                               ↑                              ↓
                      +------------------+         +------------------+
                      |  Integrity Check | ← Compare ←|   Current Hash   |
                      +------------------+         +---------------|--+
```