**Stenographic File Integrity Checker**

**1. Introduction**

Digital files are often shared, stored, and transmitted over untrusted channels. Ensuring the **integrity** of these files is critical, since even a small modification can compromise their meaning, security, or authenticity.

This project implements a **Stenographic File Integrity Checker (SFIC)**. The tool uses **cryptographic hashing** to compute a file’s fingerprint, then hides this fingerprint inside a **cover file** (such as a PNG image) using **steganography**. Later, the fingerprint can be extracted and compared to the current state of the file to detect any modifications.

By combining hashing and steganography, the solution provides both **integrity verification** and **covert storage** of verification data.

**2. Objectives**

* Generate cryptographic hashes (SHA-256 or SHA3-256) of target files.
* Embed these hashes inside a benign-looking cover image (PNG).
* Extract hidden hashes and compare with recomputed hashes to check integrity.
* Demonstrate detection of file tampering.

**3. Technical Approach**

**3.1 Hashing**

The tool uses Python’s **hashlib** library to compute secure cryptographic hashes:

* **SHA-256**: widely used, 256-bit digest.
* **SHA3-256**: newer standard, same digest length but different internal design.

The hash uniquely represents the file contents. Any modification to the file produces a completely different hash value.

**3.2 Payload Format**

The hash is stored inside a **structured payload** with metadata, ensuring robust extraction. The structure is:

* Magic header (SFI1)
* Version (1 byte)
* Algorithm code (1 = SHA-256, 2 = SHA3-256)
* Number of file entries (1 byte)
* For each file:
  + Filename length + filename (UTF-8)
  + Hash length + hash (binary)
* CRC32 checksum (4 bytes)

This format makes the hidden data **self-describing and verifiable**.

**3.3 Steganographic Embedding**

The payload is embedded inside a **cover PNG image** using the **Least Significant Bit (LSB)** method:

* Each payload bit replaces the last bit of an image pixel’s color channel (R, G, B).
* This ensures minimal visual change to the image.
* The image remains visually identical to humans, while carrying hidden integrity data.

PNG was chosen because it is **lossless**; lossy formats like JPEG would destroy hidden bits.

**3.4 Extraction & Verification**

1. The tool extracts LSBs from the stego image.
2. Bits are recombined into bytes, reconstructing the payload.
3. The payload is parsed (magic, version, algorithm, entries, checksum).
4. A fresh hash of each target file is computed.
5. Extracted hash and fresh hash are compared:
   * **MATCH** → file intact.
   * **MISMATCH** → file modified.

**4. Demonstration**

**Step 1: Embedding**

python -m sfic.cli embed --targets demo/report.pdf --cover demo/cover.png --out demo/stego.png --algo sha256

output:

Embedded payload into demo/stego.png

**Step 2: Verification (intact file)**

python -m sfic.cli verify --stego demo/stego.png --targets demo/report.pdf

Output:

Algorithm: sha256

report.pdf: MATCH

**Step 3: Verification (tampered file)**  
After modifying report.pdf slightly and re-running:

Algorithm: sha256

report.pdf: MISMATCH

This proves the tool detects integrity violations.

**5. Limitations**

* **Format**: Works only with PNG images. Other formats (JPEG, MP3) may corrupt hidden data.
* **Security**: If an attacker can regenerate the stego image with a new hash, integrity can be bypassed. Using an HMAC with a secret key would strengthen this.
* **Capacity**: Large payloads (many files or long names) require larger cover images.
* **Scope**: Current version is command-line only; no GUI yet.

**6. Improvements & Future Work**

* Support for **WAV audio files** using sample LSB embedding.
* Use of **HMAC** or encryption for stronger protection against re-embedding.
* **Graphical User Interface (GUI)** for user-friendly interaction.
* **Batch verification** for multiple files at once.
* Error-correcting codes to improve robustness against accidental corruption.

**7. Conclusion**

The Stenographic File Integrity Checker successfully combines cryptographic hashing with image steganography to provide a simple yet effective method of file integrity verification. The demonstration shows that the tool can embed a file’s fingerprint into a cover image and later detect even the smallest modification to the original file.

This project illustrates practical concepts in **cryptography, steganography, and software engineering**, and provides a foundation for further enhancements such as secure authentication and broader file format support.