

Stenographic File Integrity Checker

Name: Sahil Dabekar

Intern Id: 339

This is a File Integrity Checker using Steganography.

- **File Integrity Checker** → It tells you if a file has been changed (tampered, corrupted, or hacked).
- **Steganography** → It means “hiding secret data inside normal files (like images or audio)” so no one knows it’s there.
- **In this project:**
To build a lightweight file integrity verification tool that:

1. Generates cryptographic hashes (SHA256) of files.

What are Cryptographic Hashes?

A cryptographic hash is like a **digital fingerprint** of a file.

It takes any input file (big or small: text, PDF, image, video) and converts it into a fixed-length string (e.g., 64 characters for SHA256).

Even a tiny change in the file (adding a space, changing 1 letter, or editing 1 pixel) produces a completely different hash.

File: "hello"

Hash:

2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362
938b9824

File: "hello!" (just one character added)

Hash:

334d74cdb3ef87c7ec7d1e9e9f25e6cf6a9ad9ba9a6d6e0d495fac16
a2fefc2c

2. Embeds these hashes inside a **cover file** (image) using steganography.
3. Extracts hidden hashes later to verify if the original file was **safe** or **tampered**.

4. This ensures **confidential file integrity monitoring** while keeping the verification hidden from attackers.

- **Why do we do this? (Usefulness)**

1. Detect file tampering

- Suppose you have a report, code, or config file.
- An attacker modifies it secretly.
- By comparing with the hidden hash, you can detect this modification immediately.

2. Hidden verification (Steganography advantage)

- Normally, integrity tools save the hash in a .txt file.
- Hackers can easily find and replace that file too.
- But if we hide the hash inside an innocent image/audio, no one knows where the real verification code is.

3. Digital forensics C security

- Investigators can hide file authenticity information inside an image/audio and later prove if evidence is original.

4. Secure data distribution

- If you send files over the internet, you can also send a stego image separately.
- Receiver can check file integrity secretly.

- **Tools and Setup:**

1. Language: Python

2. Libraries:

- hashlib → to generate file hashes (SHA256/SHA3).
- Pillow → for image steganography (hide data inside PNG).
 pip install pillow
- wave (optional) → for audio steganography (hide data inside WAV).
- argparse → for CLI options (embed/extract/verify).

- **Code** (steg_integrity.py):

```

• import sys
• import hashlib
• from PIL import Image
•
• # -----Generate SHA256 Hash -----
• def generate_hash(file_path):
•     hasher = hashlib.sha256()
•     with open(file_path, "rb") as f:
•         while chunk := f.read(4096):
•             hasher.update(chunk)
•     return hasher.hexdigest()
•
• # -----Embed Hash into Image -----
• def embed_hash(cover_img, hash_str, stego_img):
•     img = Image.open(cover_img)
•     binary_hash = ''.join(format(ord(c), '08b') for c in hash_str)
•     pixels = list(img.getdata())
•
•     new_pixels = []
•     hash_index = 0
•
•     for pixel in pixels:
•         r, g, b = pixel[:3]
•         if hash_index < len(binary_hash):
•             r = (r & ~1) | int(binary_hash[hash_index]) # Put bit in Red
channel
•             hash_index += 1
•             new_pixels.append((r, g, b))
•
•     img.putdata(new_pixels)
•     img.save(stego_img)
•     print(f"[+] Hash embedded into {stego_img}")
•
• # -----Extract Hash from Image -----
• def extract_hash(stego_img, hash_len=64):
•     img = Image.open(stego_img)
•     pixels = list(img.getdata())
•
•     bits = ""
•     for pixel in pixels:
•         r, g, b = pixel[:3]
•         bits += str(r & 1)
•         if len(bits) >= hash_len * 8:
•             break
•
•     extracted = ""

```

```

•     for i in range(0, len(bits), 8):
•         extracted += chr(int(bits[i:i+8], 2))
•     return extracted
•
•
• # ----- Verify File Integrity -----
• def verify(file_path, stego_img):
•     current_hash = generate_hash(file_path)
•     hidden_hash = extract_hash(stego_img)
•
•     print(f"Hidden hash: {hidden_hash}")
•     print(f"Current hash: {current_hash}")
•
•     if current_hash == hidden_hash:
•         print("[SAFE] File is original")
•     else:
•         print("[ALERT] File has been modified")
•
• # ----- Main Program -----
• if __name__ == "__main__":
•     if len(sys.argv) < 2:
•         print("Usage: python steg_integrity.py <mode> [args]")
•         print("Modes: genhash, embed, extract, verify")
•         sys.exit(1)
•
•     mode = sys.argv[1]
•
•     if mode == "genhash":
•         print(generate_hash(sys.argv[2]))
•
•     elif mode == "embed":
•         file_to_protect = sys.argv[2]
•         cover_img = sys.argv[3]
•         stego_img = sys.argv[4]
•         hash_val = generate_hash(file_to_protect)
•         embed_hash(cover_img, hash_val, stego_img)
•
•     elif mode == "extract":
•         print(extract_hash(sys.argv[2]))
•
•     elif mode == "verify":
•         verify(sys.argv[2], sys.argv[3])
•
•

```

1. Add a report.pdf file in same folder (StegFileIntegrity)
2. Take any image which has extension .png save that image as cover.png

- **Run the code:**

1. Generate hash of a file:
Make file fingerprint.

python steg_integrity.py genhash report.pdf

```
PS C:\Users\Lenovo\Desktop\harshali\digisurkhsha\StegFileIntegrity> python steg_integrity.py genhash report.pdf
e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855
```

2. Embed hash into image:
Hide it inside an image.

python steg_integrity.py embed report.pdf cover.png stego.png

```
PS C:\Users\Lenovo\Desktop\harshali\digisurkhsha\StegFileIntegrity> python steg_integrity.py embed report.pdf cover.png stego.png
[+] Hash embedded into stego.png
```

3. Extract hidden hash:
Pull it back

python steg_integrity.py extract stego.png

```
PS C:\Users\Lenovo\Desktop\harshali\digisurkhsha\StegFileIntegrity> python steg_integrity.py extract stego.png
e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855
```

4. Verify integrity:
Check if file is unchanged or tampered
It compares hidden hash (from stego image) vs current hash of report.pdf.

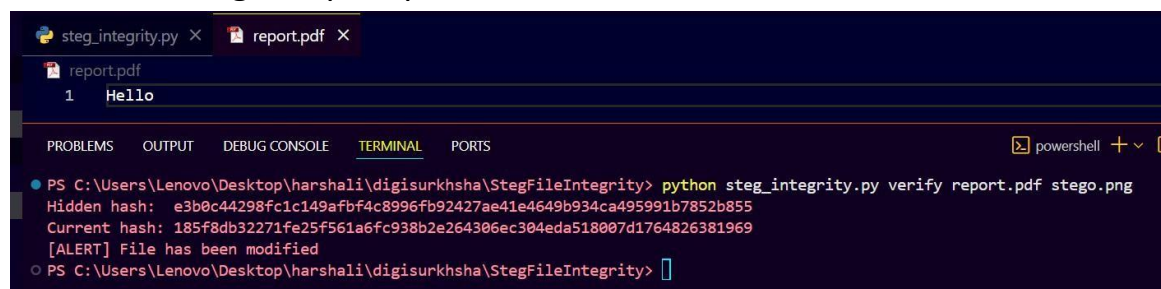
python steg_integrity.py verify report.pdf stego.png

Case 1: If file is safe =>

```
PS C:\Users\Lenovo\Desktop\harshali\digisurkhsha\StegFileIntegrity> python steg_integrity.py verify report.pdf stego.png
Hidden hash: e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855
Current hash: e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855
[SAFE] File is original
PS C:\Users\Lenovo\Desktop\harshali\digisurkhsha\StegFileIntegrity>
```

Case2: If file was modified =>

Add something in report.pdf file



```
steg_integrity.py x report.pdf x
report.pdf
1 Hello

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell + v
PS C:\Users\Lenovo\Desktop\harshali\digisurkhsha\StegFileIntegrity> python steg_integrity.py verify report.pdf stego.png
Hidden hash: e3b0c44298fc1c149afb4c8996fb92427ae41e4649b934ca495991b7852b855
Current hash: 185f8db32271fe25f561a6fc938b2e264306ec304eda518007d1764826381969
[ALERT] File has been modified
PS C:\Users\Lenovo\Desktop\harshali\digisurkhsha\StegFileIntegrity>
```

- **Observations:**

1. Normal (original) file → System marked it [SAFE].
2. Modified file → System detected mismatch and raised [ALERT].
3. Steganography successfully hid file fingerprints inside an image without visible changes.

- **Conclusion:**

1. The tool successfully generated hashes, embedded them into images, and verified file integrity.
2. It can differentiate between safe and tampered files.
3. Demonstrates a practical approach to hidden file integrity monitoring using Python.