# 🗂️ Stenographic File Integrity Checker – Proof of Concept (PoC) Report Internship Information

- **Intern Name:** Anoop Shivadas
- **Internship Organization:** Digisuraksha Foundation
- **Intern ID:** 129

---

## 1️⃣ Project Overview

### 🎯 Objective

The purpose of this project was to create a **File Integrity Monitoring Tool** that leverages **steganography** for secure storage of cryptographic hashes. Instead of saving hashes in plain text, they are embedded into a **cover image** (PNG format) using the **Least Significant Bit (LSB)** technique.

This ensures hidden but verifiable integrity data for files, protecting against tampering and unauthorized changes.

### 🔑 Key Features

- Generate secure cryptographic hashes (SHA-256 & SHA3-256).
- Embed one or more file hashes inside a PNG image using LSB steganography.
- Extract and display stored hashes from a stego image.
- Verify the integrity of files by comparing their current hash with the embedded values.

---

## 2️⃣ PoC Steps

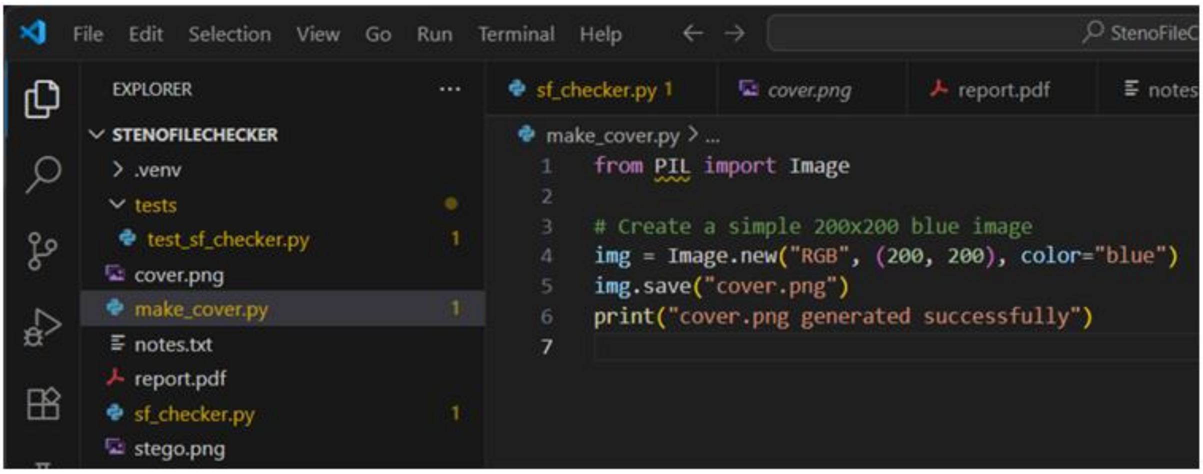
**Step 1: Setup Environment**

- Install **Python 3.x** and set up in **VS Code**.
- Install Pillow (Python Imaging Library fork) for image processing:
- pip install pillow
- Open the project folder **StenoFileChecker** in VS Code.

---

**Step 2: Create Test Files**

- Generate dummy target files such as report.pdf and notes.txt.
- Prepare a cover PNG image (cover.png) to embed the hash values.




---

**Step 3: Embed Hashes into Cover Image**

- Compute cryptographic hashes (SHA256/SHA3-256) of the target files.
- Pack these into a **JSON payload** with identifiers:
  - Magic bytes
  - Algorithm ID
  - Payload length
- Convert payload to a bitstream and hide inside RGB channel LSBs of the PNG cover image.
- Save the modified file as a **stego image**.


---

**Step 4: Extract Hashes from Stego Image**

- Read the embedded bits from the PNG image.
- Reconstruct the payload into its original JSON structure.
- Display all recovered file hashes.

---

**Step 5: Verify File Integrity**

- Recalculate current hashes of the monitored files.
- Compare with extracted hashes.
- Display output as:
  - ✅ **OK** if all files are unchanged.
  - ❌ **MISMATCH** if discrepancies are found.

---

## 3️⃣ Limitations

- Embedding capacity is **limited to image size** (~3 bits per pixel).
- LSB method is **sensitive to compression or editing** of the stego image.
- Filenames stored in JSON can cause confusion if **path differences** or **collisions** occur.

---

## 4️⃣ Possible Improvements

- Apply **compression & encryption** (AES-GCM) for additional confidentiality.
- Add **error-correcting codes** (Reed-Solomon) to improve tolerance.
- Extend the approach to **audio files (WAV)** or multiple redundant cover images.
- Develop a **graphical interface (Tkinter/PyQt)** for usability and batch operations.

---

## 5️⃣ PoC Deliverables

| File | Purpose |
|---|---|
| **sf_checker.py** | Main tool for embedding, extraction & verification |
| **make_dummy_files.py** | Script for creating test files & cover image |
| **tests/test_sf_checker.py** | Unit tests using Pytest |
| **REPORT.md** | Documentation with steps & references |

## ✅ Conclusion

This PoC successfully demonstrated how **steganography combined with cryptographic hashing** can be applied to **file integrity verification**. By embedding file hashes into a cover image, the solution ensures that integrity data remains **hidden yet accessible for validation**.

Although lightweight, the system lays the groundwork for more **robust file monitoring solutions** that could include encryption, redundancy, and GUI support in future iterations.