
Stenographic File Integrity Checker - PoC Report

Intern Name: Aditya Yashwant Borade
Internship Organization: Digisuraksha parhari foundation
Intern ID: 131

---

1. Project Overview

Objective:
To build a tool that monitors file integrity by embedding cryptographic hashes of target files into benign cover files (images) using **steganography**. This PoC focuses on PNG images using LSB (Least Significant Bit) embedding and supports **SHA256** and **SHA3-256** hashing algorithms.

Key Features:

- Generate cryptographic hashes of one or more files.
- Embed these hashes into a cover PNG image using LSB steganography.
- Extract hidden hashes from the stego image.
- Verify target files against stored hashes.

---

2. PoC Steps

Step 1: Setup Environment

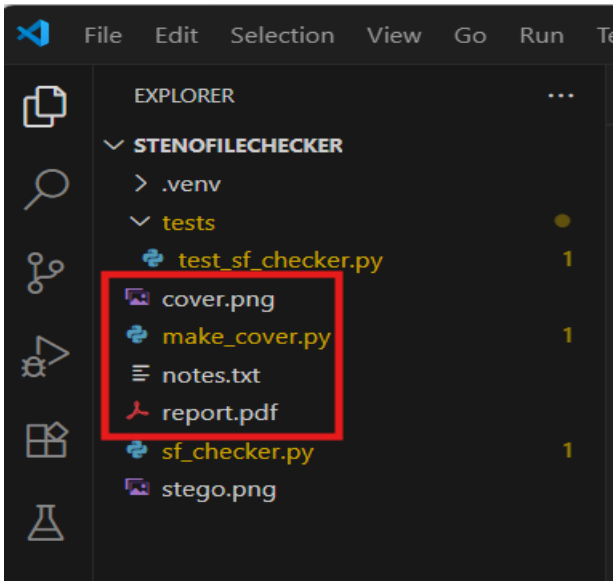
- Install Python 3.x and VS Code.
- Install required library: Pillow for image processing.

pip install pillow

- Open the project folder StenoFileChecker in VS Code.

Step 2: Generate Dummy Files

- Create dummy target files (report.pdf, notes.txt) and a cover image (cover.png) to test the PoC.



Step 3: Embed Hashes into Cover Image

- Compute hashes of target files using SHA256 or SHA3-256.
- Pack hashes into a JSON payload with a small header (magic bytes + algorithm ID + payload length).
- Convert payload into a bitstream and embed into the **LSB of RGB channels** of the cover PNG image.
- Save the resulting **stego image**.

```
PS D:\StenoFileChecker> python sf_checker.py embed --targets report.pdf notes.txt --cover cover.png --out stego.png --algo sha256
>>
```

Step 4: Extract Hashes from Stego Image

- Read the LSBs from the stego PNG image.
- Reconstruct bytes and parse the header + JSON payload.
- Display the stored file hashes.

Step 5: Verify File Integrity

- Compute current hashes of target files.
- Compare with the hashes extracted from the stego image.
- Output OK if files match or MISMATCH if any file differs.

---

3. Limitations

- Capacity depends on cover image size (3 bits per pixel).
- LSB embedding is fragile to image compression or editing.
- Filenames are stored verbatim in JSON; collisions or path differences may cause verification confusion.

---

4. Possible Improvements

- Compress and encrypt payload (e.g., AES-GCM) for confidentiality.
- Use error-correcting codes (Reed-Solomon) to tolerate minor LSB changes.
- Extend support for audio files (WAV) and multiple cover files for redundancy.
- Add a GUI (Tkinter/PyQt) for batch operations and visualization.

---

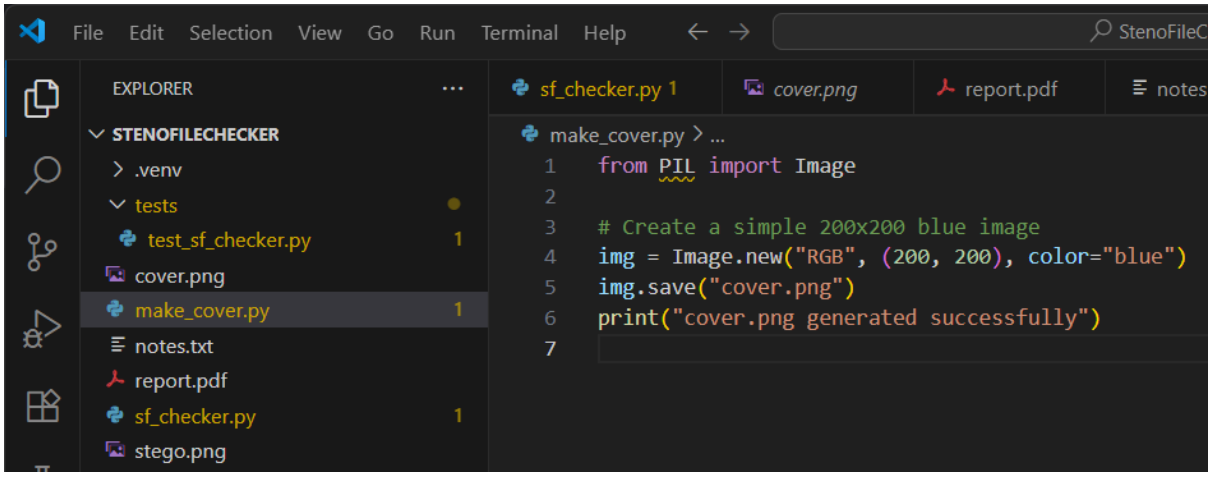
5. PoC Deliverables

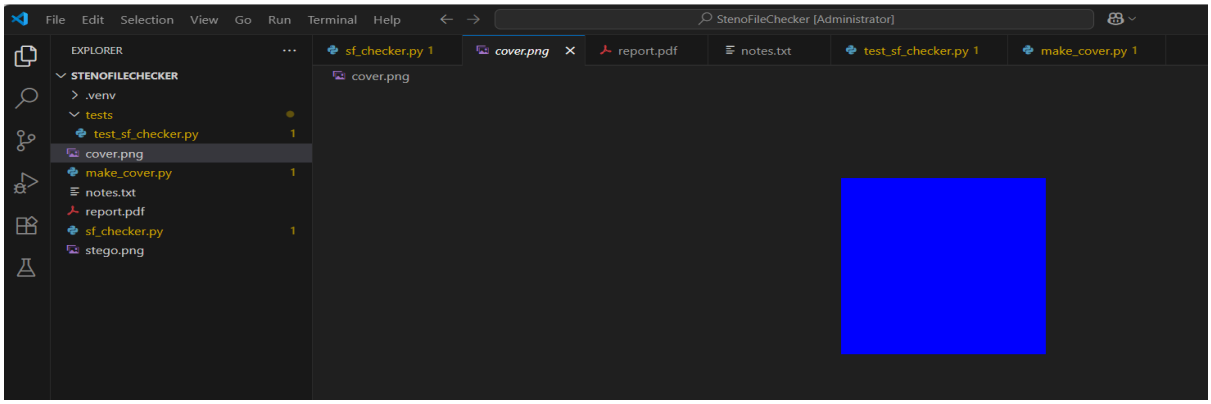
File	Purpose
sf_checker.py	Main PoC for embedding, extracting, and verifying hashes
make_dummy_files.py	Generates test files and cover image
tests/test_sf_checker.py	Pytest tests for validation
REPORT.md	Project report with steps and screenshots

---

6. Screenshots / Code Sections

- **Dummy files created:**





- **Embed hashes command:**

```
PS D:\StenoFileChecker> python sf_checker.py embed --targets report.pdf notes.txt --cover cover.png --out stego.png --algo sha256
>>
```

- **Extract hashes command:**

```
PS D:\StenoFileChecker> python sf_checker.py embed --targets report.pdf notes.txt --cover cover.png --out stego.png --algo sha256
>>
Embedded hashes for 2 file(s) into stego.png
Intern placeholders:  Aditya Borade Digisuraksha foundation 131
PS D:\StenoFileChecker>
```

- **tests/test_sf_checker.py :**

sf_checker.py 1 | report.pdf | notes.txt | report.pdf | test_sf_checker.py 1 | make_cover.py 1

tests > test_sf_checker.py > ...

```python
import os
import tempfile
from sf_checker import compute_hash, compute_hashes, pack_payload, unpack_payload, embed_into_png_lsb, extract_from_png_lsb
from PIL import Image
def make_temp_file(content: bytes) -> str:
    fd, path = tempfile.mkstemp()
    os.write(fd, content)
    os.close(fd)
    return path
def make_temp_png(width=16, height=16) -> str:
    img = Image.new('RGB', (width, height), color=(123, 222, 64))
    fp = tempfile.NamedTemporaryFile(delete=False, suffix='.png')
    img.save(fp.name, format='PNG')
    return fp.name
def test_hash_and_pack_unpack():
    f = make_temp_file(b'hello world')
    h = compute_hash(f, 'sha256')
    assert len(h) == 64
    d = {f: h}
    payload = pack_payload(d, 'sha256')
    algo, obj = unpack_payload(payload)
    assert algo == 'sha256'
    assert obj[f] == h
def test_embed_extract_roundtrip():
    tmp_png = make_temp_png(64, 64)
    f1 = make_temp_file(b'data1')
    d = compute_hashes([f1], 'sha256')
    payload = pack_payload(d, 'sha256')
    out = tmp_png + '.stego.png'
    embed_into_png_lsb(tmp_png, out, payload)
    extracted = extract_from_png_lsb(out)
    algo, obj = unpack_payload(extracted)
    assert algo == 'sha256'
    assert list(obj.keys())[0] == f1

```

- **sf_checker.py :**

sf_checker.py 1 | stego.png | report.pdf | notes.txt | test_sf_che

sf_checker.py > ...

```python
from __future__ import annotations
import argparse
import hashlib
import json
import os
import struct
import sys
from typing import Dict, List, Tuple

from PIL import Image


# ---- Intern Metadata ----
INTERN_NAME = "Aditya Borade"
INTERNSHIP_ORG = "Digisuraksha foundation"
INTERN_ID = "131"
# -------------------------


MAGIC = b"SFIC"  # 4-byte magic header
HEADER_FMT = ">4sB I"  # magic (4s), algo_id (B), payload_len (I)
# algo_id: 1=sha256, 2=sha3_256

ALGO_MAP = {"sha256": 1, "sha3_256": 2}
ALGO_INV = {1: "sha256", 2: "sha3_256"}


def compute_hash(path: str, algo: str = "sha256") -> str:
    """Compute hex digest for a single file."""
    h = None
    if algo == "sha256":
        h = hashlib.sha256()
    elif algo == "sha3_256":
        h = hashlib.sha3_256()
    else:
        raise ValueError("Unsupported algorithm: %s" % algo)
```

sf_checker.py 1 | stego.png | report.pdf | notes.txt | test_sf_checker.py 1 | make_cover.py 1

sf_checker.py > ...

```python
def compute_hash(path: str, algo: str = "sha256") -> str:
    else:
        raise ValueError("Unsupported algorithm: %s" % algo)

    with open(path, "rb") as f:
        for chunk in iter(lambda: f.read(8192), b""):
            h.update(chunk)
    return h.hexdigest()


def compute_hashes(paths: List[str], algo: str = "sha256") -> Dict[str, str]:
    results = {}
    for p in paths:
        results[p] = compute_hash(p, algo)
    return results


def pack_payload(hashes: Dict[str, str], algo: str) -> bytes:
    """Pack JSON payload and prepend header with magic and algo id and 4-byte length."""
    payload_json = json.dumps(hashes, sort_keys=True).encode("utf-8")
    algo_id = ALGO_MAP.get(algo)
    if not algo_id:
        raise ValueError("Unknown algorithm")
    header = struct.pack(HEADER_FMT, MAGIC, algo_id, len(payload_json))
    return header + payload_json


def unpack_payload(data: bytes) -> Tuple[str, Dict[str, str]]:
    """Unpack header + JSON payload and return algo name and dict of hashes."""
    if len(data) < struct.calcsize(HEADER_FMT):
        raise ValueError("Data too short for header")
    magic, algo_id, payload_len = struct.unpack(HEADER_FMT, data[:struct.calcsize(HEADER_FMT)])
    if magic != MAGIC:
        raise ValueError("Magic header mismatch")
    algo = ALGO_INV.get(algo_id)
    if not algo:
        raise ValueError("Unknown algorithm id: %s" % algo_id)
```

sf_checker.py 1 | stego.png | report.pdf | notes.txt | test_sf_checker.py 1 | make_cove

sf_checker.py > ...

```python
def unpack_payload(data: bytes) -> Tuple[str, Dict[str, str]]:
    if not algo:
        raise ValueError("Unknown algorithm id: %s" % algo_id)
    payload = data[struct.calcsize(HEADER_FMT):struct.calcsize(HEADER_FMT) + payload_len]
    obj = json.loads(payload.decode("utf-8"))
    return algo, obj


def _bytes_to_bits(b: bytes) -> List[int]:
    bits = []
    for byte in b:
        for i in range(7, -1, -1):
            bits.append((byte >> i) & 1)
    return bits


def _bits_to_bytes(bits: List[int]) -> bytes:
    out = bytearray()
    for i in range(0, len(bits), 8):
        byte = 0
        for j in range(8):
            if i + j < len(bits):
                byte = (byte << 1) | bits[i + j]
            else:
                byte <<= 1
        out.append(byte)
    return bytes(out)


def embed_into_png_lsb(cover_path: str, out_path: str, payload: bytes) -> None:
    """Embed payload into PNG LSB of RGB bytes. Simple single-LSB per color channel.

    Capacity calculation: number of pixels * 3 bits (R,G,B) >= payload_bits
    """
    img = Image.open(cover_path)
    if img.mode not in ("RGB", "RGBA"):
        img = img.convert("RGBA")
```

sf_checker.py 1 | stego.png | report.pdf | notes.txt | test_sf_checker.py 1 | make_cover.py 1

sf_checker.py > ...

```python
def embed_into_png_lsb(cover_path: str, out_path: str, payload: bytes) -> None:
        img = img.convert("RGBA")
    pixels = img.load()
    width, height = img.size
    total_channels = width * height * 3  # ignore alpha channel
    payload_bits = _bytes_to_bits(payload)
    if len(payload_bits) > total_channels:
        raise ValueError(f"Cover image too small. Need {len(payload_bits)} bits, have {total_channels} bits.")

    bit_idx = 0
    for y in range(height):
        for x in range(width):
            if bit_idx >= len(payload_bits):
                break
            px = list(pixels[x, y])  # tuple
            for c in range(3):  # R, G, B
                if bit_idx >= len(payload_bits):
                    break
                px[c] = (px[c] & ~1) | payload_bits[bit_idx]
                bit_idx += 1
            pixels[x, y] = tuple(px)
        if bit_idx >= len(payload_bits):
            break

    img.save(out_path, format="PNG")


def extract_from_png_lsb(stego_path: str) -> bytes:
    img = Image.open(stego_path)
    if img.mode not in ("RGB", "RGBA"):
        img = img.convert("RGBA")
    pixels = img.load()
    width, height = img.size
    bits = []
    # read all LSBs into a bitstream, later parse header/payload
    for y in range(height):
        for x in range(width):
```

sf_checker.py 1 X | stego.png | report.pdf | notes.txt | test_sf_checker.py 1 | make_cover.py 1

sf_checker.py > ...

```
131 def extract_from_png_lsb(stego_path: str) -> bytes:
139     for y in range(height):
140         for x in range(width):
141             px = pixels[x, y]
142             for c in range(3):
143                 bits.append(px[c] & 1)
144
145     data = _bits_to_bytes(bits)
146     # Now we need to extract header first to know payload length; since header is small, we can parse up to header size
147     header_size = struct.calcsize(HEADER_FMT)
148     needed_bytes = header_size + 4  # header + at least 4 bytes of payload len; but header already contains payload len
149     # ensure we have enough bytes
150     if len(data) < header_size:
151         raise ValueError("Not enough data extracted to contain header")
152     # read header to get payload length
153     magic, algo_id, payload_len = struct.unpack(HEADER_FMT, data[:header_size])
154     if magic != MAGIC:
155         raise ValueError("Magic header mismatch during extraction")
156     total_needed = header_size + payload_len
157     # ensure we have at least total_needed bytes (bits -> bytes may have extra trailing zeros)
158     if len(data) < total_needed:
159         # It's possible we have trailing zero bits not making a full byte stream; so we try to reconstruct exact number of bits
160         # Rebuild bits with exact length = total_needed * 8
161         total_bits_needed = total_needed * 8
162         if total_bits_needed > len(bits):
163             raise ValueError("Not enough stego capacity to read full payload")
164         truncated_bits = bits[:total_bits_needed]
165         data = _bits_to_bytes(truncated_bits)
166     # slice exact
167     extracted = data[:total_needed]
168     return extracted
169
170
171 def embed_command(args):
172     targets = args.targets
173     cover = args.cover
174     out = args.out
```

sf_checker.py > ...

```python
    def embed_command(args):
        cover = args.cover
        out = args.out
        algo = args.algo
        hashes = compute_hashes(targets, algo)
        payload = pack_payload(hashes, algo)
        embed_into_png_lsb(cover, out, payload)
        print(f"Embedded hashes for {len(hashes)} file(s) into {out}")
        print("Intern placeholders: ", INTERN_NAME, INTERNSHIP_ORG, INTERN_ID)


    def extract_command(args):
        stego = args.stego
        extracted = extract_from_png_lsb(stego)
        algo, obj = unpack_payload(extracted)
        print("Algorithm:", algo)
        print("Extracted hashes:")
        print(json.dumps(obj, indent=2))


    def verify_command(args):
        targets = args.targets
        stego = args.stego
        extracted = extract_from_png_lsb(stego)
        algo, obj = unpack_payload(extracted)
        print(f"Using algorithm: {algo}")
        all_ok = True
        for t in targets:
            if t not in obj:
                print(f"Warning: no stored hash for {t}")
                all_ok = False
                continue
            current = compute_hash(t, algo)
            stored = obj[t]
            ok = "OK" if current == stored else "MISMATCH"
            print(f"{t}: {ok}")
            if ok == "MISMATCH":
```

sf_checker.py > ...

```
192    def verify_command(args):
207            print(f"{t}: {ok}")
208            if ok == "MISMATCH":
209                all_ok = False
210        if all_ok:
211            print("All verified: files match stored hashes")
212        else:
213            print("One or more files differ from stored hashes")
214
215
216    def build_arg_parser():
217        p = argparse.ArgumentParser(prog="sf_checker.py", description="Stenographic File Integrity Checker - PoC")
218        sub = p.add_subparsers(dest="cmd")
219
220        pe = sub.add_parser("embed", help="Embed hashes into cover image")
221        pe.add_argument("--targets", nargs="+", required=True, help="Target files to hash and store")
222        pe.add_argument("--cover", required=True, help="Cover PNG image path")
223        pe.add_argument("--out", required=True, help="Output stego PNG path")
224        pe.add_argument("--algo", choices=["sha256", "sha3_256"], default="sha256")
225        pe.set_defaults(func=embed_command)
226
227        px = sub.add_parser("extract", help="Extract hidden hashes from stego image")
228        px.add_argument("--stego", required=True, help="Stego PNG path")
229        px.set_defaults(func=extract_command)
230
231        pv = sub.add_parser("verify", help="Verify target files against hashes embedded in stego image")
232        pv.add_argument("--targets", nargs="+", required=True, help="Target files to verify")
233        pv.add_argument("--stego", required=True, help="Stego PNG path")
234        pv.set_defaults(func=verify_command)
235
236        return p
237
238
239    def main(argv=None):
240        parser = build_arg_parser()
241        args = parser.parse_args(argv)
242        if not hasattr(args, "func"):
```

sf_checker.py 1 | stego.png | report.pdf | notes.txt | test_sf_checker.py 1 | make_cover.py 1

sf_checker.py > ...

```
216    def build_arg_parser():
223        pe.add_argument("--out", required=True, help="Output stego PNG path")
224        pe.add_argument("--algo", choices=["sha256", "sha3_256"], default="sha256")
225        pe.set_defaults(func=embed_command)
226
227        px = sub.add_parser("extract", help="Extract hidden hashes from stego image")
228        px.add_argument("--stego", required=True, help="Stego PNG path")
229        px.set_defaults(func=extract_command)
230
231        pv = sub.add_parser("verify", help="Verify target files against hashes embedded in stego image")
232        pv.add_argument("--targets", nargs="+", required=True, help="Target files to verify")
233        pv.add_argument("--stego", required=True, help="Stego PNG path")
234        pv.set_defaults(func=verify_command)
235
236        return p
237
238
239    def main(argv=None):
240        parser = build_arg_parser()
241        args = parser.parse_args(argv)
242        if not hasattr(args, "func"):
243            parser.print_help()
244            return
245        args.func(args)
246
247
248    if __name__ == "__main__":
249        main()
250
```