Proof of Concept (PoC): Steganographic File Integrity Checker

NAME : CHAITANYA MAYEKAR

INTERN ID: 253

# 1. Introduction

This Proof of Concept (PoC) demonstrates a simple File Integrity Checker that uses steganography to embed the SHA256 hash of a file inside a PNG image. The purpose is to provide a method of verifying file integrity by comparing the extracted hidden hash against the actual computed hash of the target file.

# 2. Goals

The objectives of this PoC are:  
1. Compute the SHA256 hash of any file.  
2. Embed the hash into a cover PNG image using Least Significant Bit (LSB) steganography.  
3. Extract the hash back from the stego image.  
4. Verify the integrity of the target file by comparing hashes.  
5. Provide a Command-Line Interface (CLI) for demonstration.

# 3. Architecture

The PoC is split into two main modules:  
1. stego\_core.py – Core library containing hashing, embedding, extraction, and verification logic.  
2. app.py – CLI interface for embedding and verifying files.

# 4. Core Implementation

## 4.1 stego\_core.py

This module handles the core functionality of the PoC, including hashing, hiding, extracting, and verifying file hashes.

import hashlib  
from PIL import Image  
import os  
  
def make\_hash(file\_path: str) -> str:  
 h = hashlib.sha256()  
 with open(file\_path, "rb") as f:  
 for chunk in iter(lambda: f.read(4096), b""):  
 h.update(chunk)  
 return h.hexdigest()  
  
def hide\_hash(cover\_image: str, out\_image: str, hash\_text: str) -> None:  
 img = Image.open(cover\_image).convert("RGB")  
 pixels = list(img.getdata())  
 bits = ''.join(format(ord(c), '08b') for c in hash\_text)  
 if len(bits) > len(pixels):  
 raise ValueError("Cover image too small to hide hash.")  
 new\_pixels, i = [], 0  
 for r, g, b in pixels:  
 if i < len(bits):  
 r = (r & ~1) | int(bits[i])  
 i += 1  
 new\_pixels.append((r, g, b))  
 img.putdata(new\_pixels)  
 img.save(out\_image)  
  
def read\_hash(stego\_image: str, length: int = 64) -> str:  
 img = Image.open(stego\_image).convert("RGB")  
 pixels = list(img.getdata())  
 bits = "".join(str(r & 1) for r, \_, \_ in pixels[:length\*8])  
 chars = [chr(int(bits[i:i+8], 2)) for i in range(0, len(bits), 8)]  
 return ''.join(chars)  
  
def verify\_file(target\_file: str, stego\_image: str) -> bool:  
 h1 = make\_hash(target\_file)  
 h2 = read\_hash(stego\_image)  
 return h1 == h2

## 4.2 app.py

This module provides the CLI for embedding and verifying file hashes using the core library.

import argparse  
from stego\_core import make\_hash, hide\_hash, verify\_file  
  
def main():  
 parser = argparse.ArgumentParser(description="Steganographic File Integrity Checker")  
 subparsers = parser.add\_subparsers(dest="command")  
  
 embed\_parser = subparsers.add\_parser("embed", help="Embed hash into cover image")  
 embed\_parser.add\_argument("file", help="Target file to hash")  
 embed\_parser.add\_argument("cover", help="Cover image (PNG)")  
 embed\_parser.add\_argument("out", help="Output stego image")  
  
 verify\_parser = subparsers.add\_parser("verify", help="Verify file against stego image")  
 verify\_parser.add\_argument("file", help="Target file to check")  
 verify\_parser.add\_argument("stego", help="Stego image containing hash")  
  
 args = parser.parse\_args()  
  
 if args.command == "embed":  
 h = make\_hash(args.file)  
 print(f"Hash: {h}")  
 hide\_hash(args.cover, args.out, h)  
  
 elif args.command == "verify":  
 if verify\_file(args.file, args.stego):  
 print("File is OK (hash matches)")  
 else:  
 print("File changed (hash mismatch!)")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

# 5. Usage

Run the following commands to test the PoC:  
  
1. Embed hash:  
 python app.py embed report.pdf cover.png stego.png  
  
2. Verify hash:  
 python app.py verify report.pdf stego.png