Name:- Virali Shailesh Gada

Intern id:-303

Topic:- Stenographic File Integrity Checker project.

1. Objective

The tool aims to ensure file integrity by:

- Generating a cryptographic hash (SHA256) of a target file.

- Hiding this hash inside a benign cover image using steganography.

- Later extracting the hidden hash to verify if the target file has been modified.

2. Core Concepts

a. Cryptographic Hashing

- A hash function (SHA256) takes an input file and produces a fixed-length string (hash) uniquely representing the file content.

- Even a small change in the file results in a drastically different hash.

- This property helps detect any modification in the file.

b. Steganography (Least Significant Bit - LSB)

- Steganography hides secret data within a cover medium (here, an image) without noticeable changes.

- LSB technique modifies the least significant bit of pixel color values to embed data.

- Since the LSB contributes minimally to the color value, changes are visually imperceptible.

3. How the Code Works

a. Generating the Hash

- The function `generate\_hash(file\_path)` reads the target file in binary mode.

- It updates the SHA256 hash object with chunks of the file.

- Returns the hexadecimal string representation of the hash.

b. Embedding the Hash into the Image

- The hash string is converted into a binary string (each character to 8 bits).

- A delimiter (`'1111111111111110'`) is appended to mark the end of the hash.

- The cover image is loaded and converted to RGB mode.

- The code iterates over each pixel’s red channel and replaces its LSB with one bit of the hash binary string.

- Once all bits are embedded, the modified image is saved as the stego image.

c. Extracting the Hash from the Image

- The stego image is loaded and pixels are read.

- The LSB of the red channel of each pixel is collected to reconstruct the binary string.

- Every 8 bits are converted back to a character.

- Extraction stops when the delimiter is detected.

- The resulting string is the embedded hash.

d. Verifying File Integrity

- The current hash of the target file is generated.

- The embedded hash is extracted from the stego image.

- Both hashes are compared:

- If they match, the file is intact.

- If they differ, the file has been modified.

4. GUI Workflow

- Select Target File: User chooses the file to protect.

- Select Cover Image: User chooses a PNG image to hide the hash.

- Embed Hash :The hash is embedded into the cover image and saved.

- Select Stego Image: User selects the image containing the hidden hash.

- Verify Integrity: The tool compares the current file hash with the extracted hash and displays the result.

5. Limitations and Considerations

- The cover image must be large enough to hold the entire hash (256 bits + delimiter).

- Only the red channel’s LSB is used; this limits embedding capacity but keeps distortion minimal.

- The delimiter is essential to know where the embedded hash ends.

- This method is vulnerable to image compression or format changes that alter pixel data.

- The tool currently supports only PNG images (lossless format).

6. Summary

This project combines cryptographic hashing and steganography to create a simple yet effective file integrity checker. The GUI facilitates user interaction, making it easy to embed and verify hashes without dealing with command-line complexity.

import tkinter as tk

from tkinter import filedialog, messagebox

from PIL import Image

import hashlib

def generate\_hash(file\_path):

hasher = hashlib.sha256()

with open(file\_path, 'rb') as f:

while chunk := f.read(8192):

hasher.update(chunk)

return hasher.hexdigest()

def embed\_hash\_in\_image(cover\_image\_path, hash\_str, output\_image\_path):

img = Image.open(cover\_image\_path).convert('RGB')

pixels = img.load()

bin\_hash = ''.join(format(ord(c), '08b') for c in hash\_str) + '1111111111111110' # delimiter

width, height = img.size

idx = 0

for y in range(height):

for x in range(width):

if idx < len(bin\_hash):

r, g, b = pixels[x, y]

r = (r & ~1) | int(bin\_hash[idx])

pixels[x, y] = (r, g, b)

idx += 1

else:

img.save(output\_image\_path)

return True

raise ValueError("Cover image too small to hold the hash")

def extract\_hash\_from\_image(stego\_image\_path):

img = Image.open(stego\_image\_path).convert('RGB')

pixels = img.load()

width, height = img.size

bits = []

for y in range(height):

for x in range(width):

r, g, b = pixels[x, y]

bits.append(str(r & 1))

chars = []

for i in range(0, len(bits), 8):

byte = bits[i:i+8]

if len(byte) < 8:

break

char = chr(int(''.join(byte), 2))

chars.append(char)

if ''.join(chars[-2:]) == '\xff\xfe': # delimiter

break

return ''.join(chars).rstrip('\xff\xfe')

class StegoApp:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("File Integrity Checker PoC")

self.target\_file = ""

self.cover\_image = ""

self.stego\_image = ""

tk.Button(root, text="Select Target File", command=self.select\_target\_file).grid(row=0, column=0, padx=10, pady=10)

self.target\_label = tk.Label(root, text="No file selected")

self.target\_label.grid(row=0, column=1)

tk.Button(root, text="Select Cover Image", command=self.select\_cover\_image).grid(row=1, column=0, padx=10, pady=10)

self.cover\_label = tk.Label(root, text="No image selected")

self.cover\_label.grid(row=1, column=1)

tk.Button(root, text="Embed Hash", command=self.embed\_hash).grid(row=2, column=0, padx=10, pady=10)

self.embed\_status = tk.Label(root, text="")

self.embed\_status.grid(row=2, column=1)

tk.Button(root, text="Select Stego Image", command=self.select\_stego\_image).grid(row=3, column=0, padx=10, pady=10)

self.stego\_label = tk.Label(root, text="No image selected")

self.stego\_label.grid(row=3, column=1)

tk.Button(root, text="Verify Integrity", command=self.verify\_integrity).grid(row=4, column=0, padx=10, pady=10)

self.verify\_status = tk.Label(root, text="")

self.verify\_status.grid(row=4, column=1)

def select\_target\_file(self):

self.target\_file = filedialog.askopenfilename(title="Select Target File")

self.target\_label.config(text=self.target\_file if self.target\_file else "No file selected")

def select\_cover\_image(self):

self.cover\_image = filedialog.askopenfilename(title="Select Cover Image", filetypes=[("PNG Images", "\*.png")])

self.cover\_label.config(text=self.cover\_image if self.cover\_image else "No image selected")

def embed\_hash(self):

if not self.target\_file or not self.cover\_image:

messagebox.showerror("Error", "Please select both target file and cover image.")

return

try:

file\_hash = generate\_hash(self.target\_file)

output\_path = filedialog.asksaveasfilename(defaultextension=".png", filetypes=[("PNG Images", "\*.png")], title="Save Stego Image As")

if not output\_path:

return

embed\_hash\_in\_image(self.cover\_image, file\_hash, output\_path)

self.embed\_status.config(text=f"Hash embedded and saved to:\n{output\_path}")

self.stego\_image = output\_path

self.stego\_label.config(text=self.stego\_image)

except Exception as e:

messagebox.showerror("Error", str(e))

def select\_stego\_image(self):

self.stego\_image = filedialog.askopenfilename(title="Select Stego Image", filetypes=[("PNG Images", "\*.png")])

self.stego\_label.config(text=self.stego\_image if self.stego\_image else "No image selected")

def verify\_integrity(self):

if not self.target\_file or not self.stego\_image:

messagebox.showerror("Error", "Please select both target file and stego image.")

return

try:

current\_hash = generate\_hash(self.target\_file)

extracted\_hash = extract\_hash\_from\_image(self.stego\_image)

if current\_hash == extracted\_hash:

self.verify\_status.config(text="File is intact. Hash matches.", fg="green")

else:

self.verify\_status.config(text="File has been modified! Hash mismatch.", fg="red")

except Exception as e:

messagebox.showerror("Error", str(e))

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

app = StegoApp(root)

root.mainloop()