Project Report: Steganography Tool for Image/File Hiding

- **Objective**

To develop a graphical user interface (GUI) tool that hides and extracts text or files inside images using steganography techniques, particularly Least Significant Bit (LSB) manipulation.

- **Tools and Technologies Used**
  - Programming Language: Python
  - Libraries Used:
    - **tkinter** – for GUI
    - **PIL (Pillow)** – for image processing
    - **OS , time** – for file handling and timestamps

- **Features and Functionalities**
  - **Text/File Embedding:** Supports embedding text or files (image, video, document) into cover images.
  - **Password Protection:** Encrypts hidden data using XOR encryption and locks out after 3 wrong password attempts.
  - **Drag-and-Drop GUI:** User-friendly buttons with icons and tooltips.
  - **Image Format Support:** Compatible with PNG and BMP formats.
  - **Image Generation:** Create blank carrier images with custom dimensions.

- **Guide**

Convert Message to Binary: Text or file content is converted to binary. Files include header info like extension and size.
Upload Image and Message/File: Users can upload a cover image and either type a message or choose a file.
Embed Using LSB: The binary data is embedded into the least significant 3 bits of RGB pixels.
Extraction and Decryption: Data is extracted from modified images and decrypted using the correct password.
Drag-and-Drop GUI: Clean interface with icons for 'Add', 'Encode', 'Decode', 'Generate', and 'Help'.
File Output: Extracted files are saved with the original extension.

- **Code :**

```
import tkinter as tk
from tkinter import messagebox, ttk, filedialog, simpledialog
from PIL import Image, ImageTk
import os
import time

class ToolTip:
    def __init__(self, widget, text):
        self.widget = widget
        self.text = text
        self.tip_window = None
        widget.bind("<Enter>", self.show)
        widget.bind("<Leave>", self.hide)
```

```
    def show(self, event=None):
        if self.tip_window or not self.text:
            return
        x, y, _, _ = self.widget.bbox("insert")
        x += self.widget.winfo_rootx() + 30
        y += self.widget.winfo_rooty() + 20
        self.tip_window = tw = tk.Toplevel(self.widget)
        tw.wm_overrideredirect(True)
        tw.geometry(f"+{x}+{y}")
        label = tk.Label(tw, text=self.text, justify='left',
                         background="#ffffe0", relief='solid',
borderwidth=1,
                         font=("Segoe UI", 9))
        label.pack(ipadx=4)

    def hide(self, event=None):
        if self.tip_window:
            self.tip_window.destroy()
            self.tip_window = None

class StegoTool:
    def __init__(self, window):
        self.window = window
        self.window.title("StegoTool")
        self.window.geometry("700x600")
        self.window.resizable(False, False)

        self.selected_mode = None
        self.selected_file_path = None
        self.failed_attempts = 0
```

```
        self.lockout_level = 0
        self.lockout_until = 0
        self.MAX_WIDTH = 10000
        self.MAX_HEIGHT = 10000

        self.setup_ui()

    def setup_ui(self):
        toolbar = tk.Frame(self.window, bd=1, relief=tk.RAISED)
        toolbar.pack(side=tk.TOP, fill=tk.X)

        buttons = {
            "Add": ("➕", "Add", "Add text or file to hide"),
            "Encode": ("🔒", "Encode", "Encode hidden data"),
            "Decode": ("🔓", "Decode", "Decode hidden data"),
            "Generate": ("⚙", "Generate", "Generate new image"),
            "Help": ("❓", "Help", "Show help or info")
        }

        for name, (emoji, label, tooltip) in buttons.items():
            frame = tk.Frame(toolbar)
            btn = tk.Button(frame, text=emoji, font=("Segoe UI
Emoji", 20), width=3,
                            command=lambda n=name:
self.button_action(n))
            btn.pack()
            lbl = tk.Label(frame, text=label, font=("Segoe UI", 9))
            lbl.pack()
            frame.pack(side=tk.LEFT, padx=5, pady=2)
```

```
            ToolTip(btn, tooltip)

        self.text_entry = tk.Text(self.window, height=5,
width=80)

        self.tree = ttk.Treeview(self.window, columns=("name",
"size", "date"), show="headings")
        self.tree.heading("name", text="File Name")
        self.tree.heading("size", text="Size")
        self.tree.heading("date", text="Date Modified")
        self.tree.column("name", width=300)
        self.tree.column("size", width=100, anchor="center")
        self.tree.column("date", width=150, anchor="center")
        self.tree.pack(fill=tk.BOTH, expand=True, padx=10,
pady=10)

    def button_action(self, name):
        if name == "Add":
            self.show_add_menu()
        elif name == "Encode":
            self.encode()
        elif name == "Decode":
            self.decode()
        elif name == "Generate":
            self.generate_large_image()
        elif name == "Help":
            self.show_help()

    def show_add_menu(self):
        menu = tk.Menu(self.window, tearoff=0)
```

```
        menu.add_command(label="Add Text",
command=lambda: self.set_mode("Text"))
        menu.add_command(label="Add Image",
command=lambda: self.set_mode("Image"))
        menu.add_command(label="Add Video",
command=lambda: self.set_mode("Video"))
        menu.add_command(label="Add Document",
command=lambda: self.set_mode("Document"))

        add_button =
self.window.nametowidget(".!frame.!frame.!button")
        x = add_button.winfo_rootx()
        y = add_button.winfo_rooty() +
add_button.winfo_height()
        menu.tk_popup(x, y)

    def set_mode(self, mode):
        self.selected_mode = mode
        if mode == "Text":
            self.text_entry.pack(pady=10, padx=10)
            self.selected_file_path = None
            for item in self.tree.get_children():
                self.tree.delete(item)
        else:
            self.text_entry.pack_forget()
            self.choose_file()

    def choose_file(self):
        self.selected_file_path =
filedialog.askopenfilename(filetypes=[("All Files", "*.*")])
```

```
        if self.selected_file_path:
            for item in self.tree.get_children():
                self.tree.delete(item)
            filename = os.path.basename(self.selected_file_path)
            size = os.path.getsize(self.selected_file_path)
            date = time.strftime('%Y-%m-%d',
time.localtime(os.path.getmtime(self.selected_file_path)))
            self.tree.insert("", tk.END, values=(filename,
f"{size/1024:.1f} KB", date))

    def handle_failed_attempts(self):
        """Handle escalating lockout periods after failed
attempts"""
        self.lockout_level += 1
        lockout_durations = [30, 120, 300, 600, 1800, 3600]  #
30s, 2m, 5m, 10m, 30m, 1h
        duration = lockout_durations[min(self.lockout_level-1,
len(lockout_durations)-1)]
        self.lockout_until = time.time() + duration

        time_unit = "seconds" if duration < 60 else "minutes"
        display_duration = duration if duration < 60 else
duration//60

        messagebox.showerror("Locked Out",
                             f"Too many failed attempts. Please try again in
{display_duration} {time_unit}.")
        self.failed_attempts = 0

    def decode_binary_from_image(self, image_path):
```

```
        progress = tk.Toplevel(self.window)
        progress.title("Decoding... Please wait")
        tk.Label(progress, text="Decoding in
progress...").pack(padx=20, pady=10)
        progress_bar = ttk.Progressbar(progress,
mode='determinate', length=300)
        progress_bar.pack(padx=20, pady=10)
        progress.update()

        try:
            with Image.open(image_path) as img:
                if img.mode != 'RGB':
                    img = img.convert('RGB')
                pixels = list(img.getdata())
                total_pixels = len(pixels)

            bits = []
            for i in range(8):
                r, g, b = pixels[i]
                bits.extend([(r & 7), (g & 7), (b & 7)])

            bits_str = ''.join(f'{x:03b}' for x in bits)
            ext_bin = bits_str[:40]
            size_bin = bits_str[40:72]
            size = int(size_bin, 2)
            total_bits_needed = 72 + (size * 8)

            batch_size = 10000
            processed_pixels = 8
```

```
            while processed_pixels < total_pixels and len(bits)*3 <
total_bits_needed:
                end_idx = min(processed_pixels + batch_size,
total_pixels)
                for i in range(processed_pixels, end_idx):
                    r, g, b = pixels[i]
                    bits.extend([(r & 7), (g & 7), (b & 7)])
                    if len(bits)*3 >= total_bits_needed:
                        break

                processed_pixels = end_idx

                if processed_pixels % 50000 == 0:
                    progress_bar['value'] = (processed_pixels /
total_pixels) * 100
                    progress.update()
                    self.window.update()

                if len(bits)*3 >= total_bits_needed:
                    break

            return ''.join(f'{x:03b}' for x in
bits)[:total_bits_needed]

        except Exception as e:
            messagebox.showerror("Decoding Error", f"Error
during decoding: {str(e)}")
            return ""
        finally:
            progress.destroy()
```

```
    def xor_encrypt(self, data_bytes, password):
        key = password.encode()
        return bytes(b ^ key[i % len(key)] for i, b in
enumerate(data_bytes))

    def xor_decrypt(self, data_bytes, password):
        return self.xor_encrypt(data_bytes, password)

    def file_to_binary(self, path, password=None):
        with open(path, 'rb') as file:
            content = file.read()
        if password:
            content = self.xor_encrypt(content, password)
        ext = os.path.splitext(path)[1]
        ext_bin = ''.join(f'{ord(c):08b}' for c in ext.ljust(5))
        size_bin = f'{len(content):032b}'
        data_bin = ''.join(f'{byte:08b}' for byte in content)
        return ext_bin + size_bin + data_bin

    def binary_to_file(self, bin_data, output_dir,
password=None):
        ext_bin = bin_data[:40]
        size_bin = bin_data[40:72]
        ext = ''.join(chr(int(ext_bin[i:i+8], 2)) for i in range(0,
40, 8)).strip()
        size = int(size_bin, 2)
        file_bin = bin_data[72:72 + (size * 8)]
        file_bytes = bytes(int(file_bin[i:i+8], 2) for i in range(0,
len(file_bin), 8))
```

```
        if password:
            file_bytes = self.xor_decrypt(file_bytes, password)

        default_name = f"extracted_file{ext}"
        out_path = filedialog.asksaveasfilename(
            initialfile=default_name,
            defaultextension=ext,
            filetypes=[("All Files", "*.*")]
        )
        if out_path:
            with open(out_path, 'wb') as f:
                f.write(file_bytes)
            messagebox.showinfo("Done", f"File saved as
{out_path}")

    def encode_binary_to_image(self, image_path, binary_data,
save_path):
        img = Image.open(image_path)
        if img.mode != 'RGB':
            img = img.convert('RGB')
        pixels = list(img.getdata())

        data_len = len(binary_data)
        max_capacity = len(pixels) * 9
        if data_len > max_capacity:
            raise Exception(f"Data too large to embed.\nMax:
{max_capacity} bits\nData: {data_len} bits")

        new_pixels = []
        idx = 0
```

```
        for r, g, b in pixels:
            if idx + 9 <= data_len:
                r = (r & ~7) | int(binary_data[idx:idx+3], 2)
                g = (g & ~7) | int(binary_data[idx+3:idx+6], 2)
                b = (b & ~7) | int(binary_data[idx+6:idx+9], 2)
                idx += 9
            new_pixels.append((r, g, b))

        img.putdata(new_pixels)
        img.save(save_path)

    def encode(self):
        if not self.selected_mode:
            messagebox.showwarning("Warning", "Please select
what to encode first (click Add)")
            return

        image_path =
filedialog.askopenfilename(filetypes=[("Image Files",
"*.png;*.bmp")])
        if not image_path:
            return

        use_pwd = messagebox.askyesno("Password", "Do you
want to set a password?")
        password = None
        if use_pwd:
            pwd = simpledialog.askstring("Password", "Enter
password:", show="*")
```

```
        confirm = simpledialog.askstring("Confirm
Password", "Confirm password:", show="*")
        if pwd != confirm:
            messagebox.showerror("Error", "Passwords do not
match.")
            return
        password = pwd

    try:
        if self.selected_mode == "Text":
            message = self.text_entry.get("1.0", tk.END).strip()
            if not message:
                messagebox.showerror("Error", "Enter a
message.")
                return
            msg_bytes = message.encode()
            if password:
                msg_bytes = self.xor_encrypt(msg_bytes,
password)
            data_bin = ''.join(f'{byte:08b}' for byte in msg_bytes)
+ '1111111111111110'
        else:
            if not self.selected_file_path:
                messagebox.showerror("Error", "Select a file to
hide.")
                return
            data_bin = self.file_to_binary(self.selected_file_path,
password)
```

```
            save_path =
filedialog.asksaveasfilename(defaultextension=".png",
filetypes=[("PNG files", "*.png")])
            if not save_path:
                return

            self.encode_binary_to_image(image_path, data_bin,
save_path)
            messagebox.showinfo("Success", f"Data embedded
and saved to:\n{save_path}")
        except Exception as e:
            messagebox.showerror("Error", str(e))

    def decode(self):
        current_time = time.time()
        if hasattr(self, 'lockout_until') and current_time <
self.lockout_until:
            remaining = int(self.lockout_until - current_time)
            time_unit = "seconds" if remaining < 60 else "minutes"
            display_time = remaining if remaining < 60 else
remaining//60
            messagebox.showwarning("Locked Out",
                                   f"Too many failed attempts. Please try again
in {display_time} {time_unit}.")
            return

        image_path =
filedialog.askopenfilename(filetypes=[("Image Files",
"*.png;*.bmp")])
        if not image_path:
```

```
            return

        try:
            data_bits =
self.decode_binary_from_image(image_path)
            if not data_bits:
                return

            if self.selected_mode == "Text":
                chars = []
                for i in range(0, len(data_bits), 8):
                    byte = data_bits[i:i+8]
                    if byte == '11111110':
                        break
                    chars.append(int(byte, 2))
                encrypted_bytes = bytes(chars)

                for attempt in range(1, 4):
                    password = simpledialog.askstring("Password",
                                                      f"Enter password (Attempt
{attempt}/3):",
                                                      show="*")
                    if not password:
                        return

                    try:
                        decrypted = self.xor_decrypt(encrypted_bytes,
password).decode(errors='strict')
                        self.failed_attempts = 0
                        self.lockout_level = 0
```

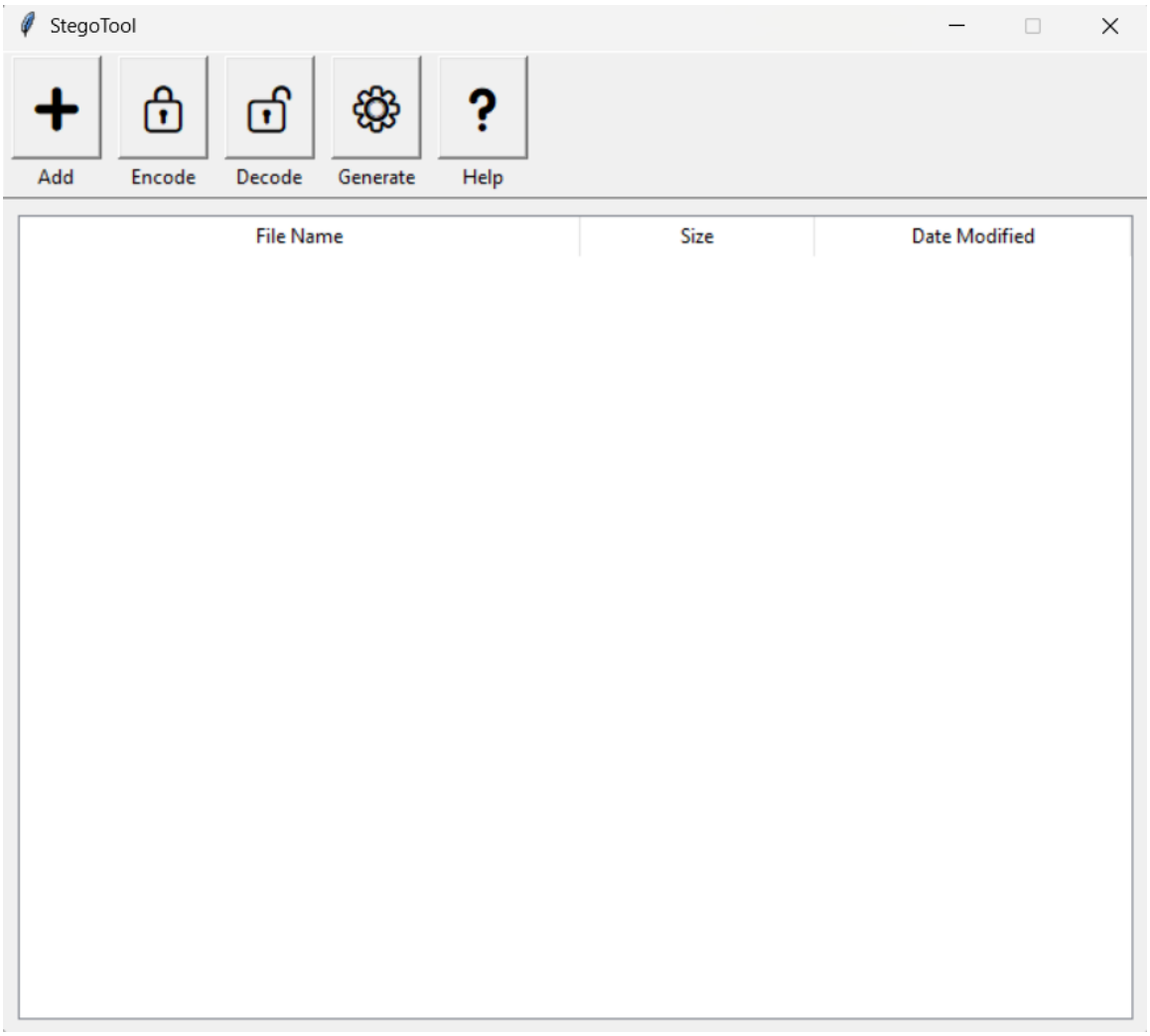
```
                self.text_entry.delete("1.0", tk.END)
                self.text_entry.insert(tk.END, decrypted)
                messagebox.showinfo("Success", "Message
extracted successfully.")
                return
            except:
                self.failed_attempts += 1
                if self.failed_attempts >= 3:
                    self.handle_failed_attempts()
                    return
                messagebox.showerror("Incorrect",
                                     f"Incorrect password. {3-
self.failed_attempts} attempts remaining.")

        else:
            for attempt in range(1, 4):
                password = simpledialog.askstring("Password",
                                                  f"Enter password (Attempt
{attempt}/3):",
                                                  show="*")
                if not password:
                    return

                try:
                    test_bits = data_bits[:100]
                    test_bytes = bytes(int(data_bits[i:i+8], 2) for i in
range(72, 172, 8))
                    if password:
                        test_bytes = self.xor_decrypt(test_bytes,
password)
```

```
                    self.failed_attempts = 0
                    self.lockout_level = 0
                    self.binary_to_file(data_bits,
os.path.dirname(image_path), password)
                    return
                except:
                    self.failed_attempts += 1
                    if self.failed_attempts >= 3:
                        self.handle_failed_attempts()
                        return
                    messagebox.showerror("Incorrect",
                                         f"Incorrect password. {3-
self.failed_attempts} attempts remaining.")

        except Exception as e:
            messagebox.showerror("Error", str(e))

    def generate_large_image(self):
        try:
            width = simpledialog.askinteger("Image Width",
f"Enter image width (max {self.MAX_WIDTH}):",
                                            minvalue=1000,
maxvalue=self.MAX_WIDTH)
            height = simpledialog.askinteger("Image Height",
f"Enter image height (max {self.MAX_HEIGHT}):",
                                             minvalue=1000,
maxvalue=self.MAX_HEIGHT)
            if not width or not height:
                return
```

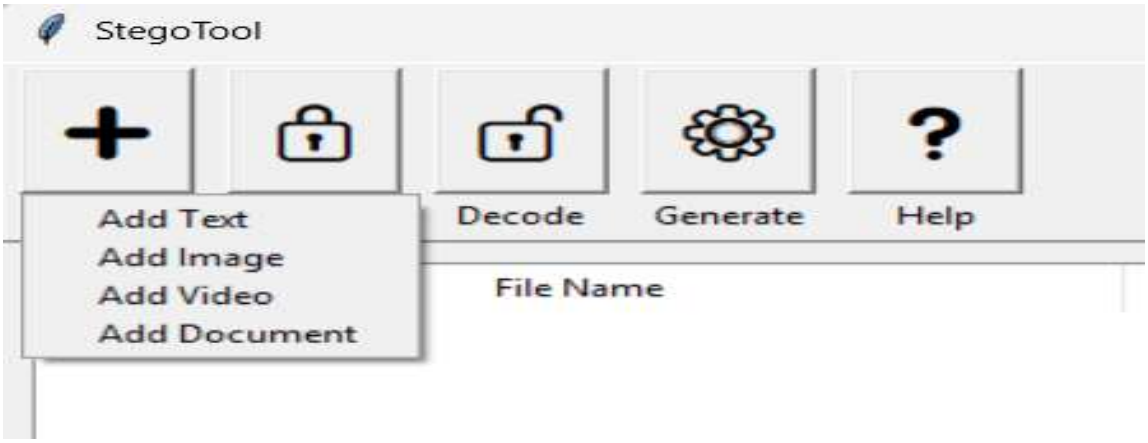
```
            img = Image.new('RGB', (width, height), color='white')
            save_path =
filedialog.asksaveasfilename(defaultextension=".png",
filetypes=[("PNG Files", "*.png")])
            if save_path:
                img.save(save_path)
                messagebox.showinfo("Done", f"Blank image
created:\n{save_path}")
        except Exception as e:
            messagebox.showerror("Error", str(e))

    def show_help(self):
        help_text = """StegoTool Help:

1. Click 'Add' to select what to hide:
   - Text: Type your secret message
   - File: Select an image, video, or document

2. Click 'Encode' to hide the data in an image
3. Click 'Decode' to extract hidden data
4. 'Generate' creates blank images for hiding data

Security Features:
- Password protection for encoded data
- Escalating lockout periods after failed attempts
- 3 attempts before temporary lockout
"""
        messagebox.showinfo("Help", help_text)

if __name__ == "__main__":
```

```
window = tk.Tk()
app = StegoTool(window)
window.mainloop()
```

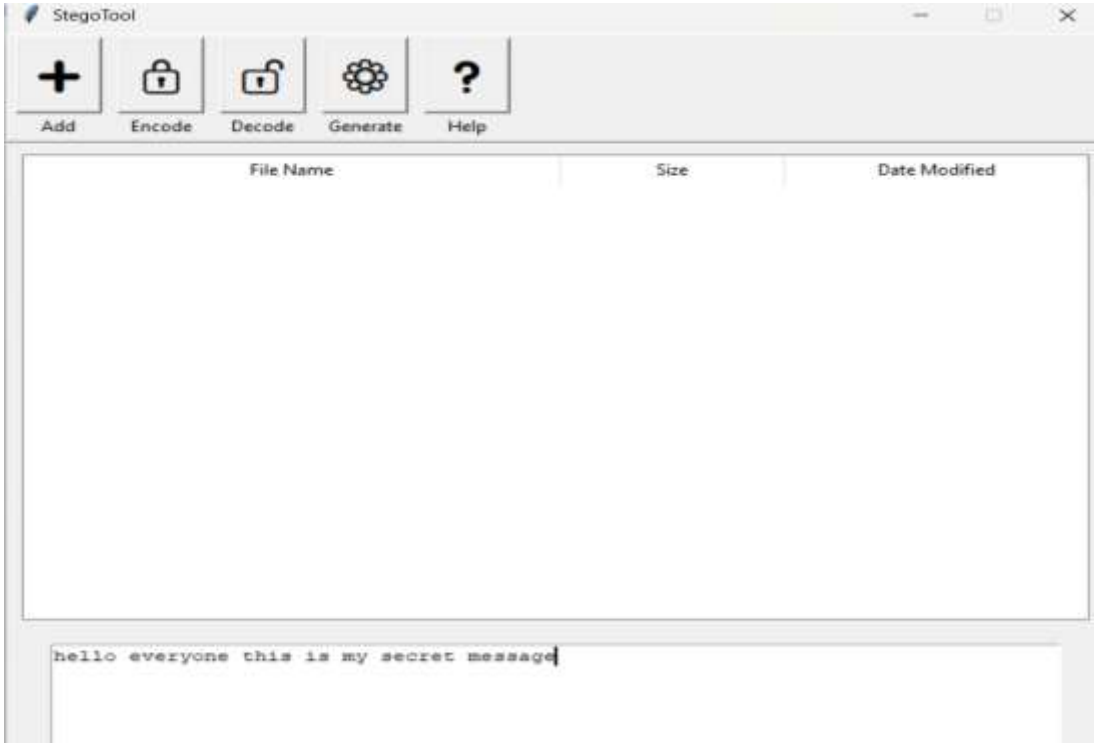
- **Screenshots :**



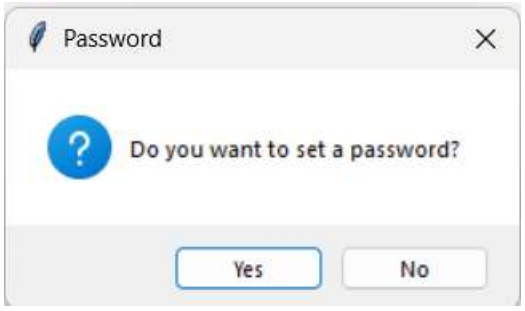
- **Add button to add type of file:**



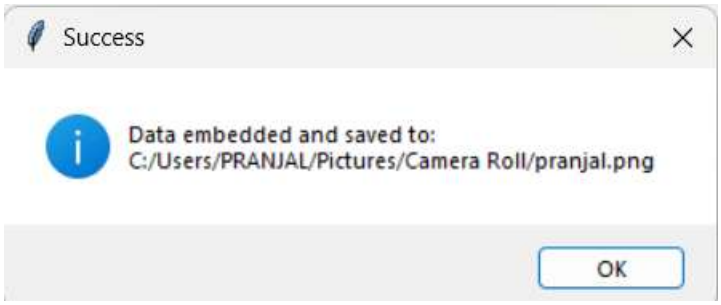
- **To add text:**



  - **Encoding the message:**
    - **Click on the encode icon and select your image file**



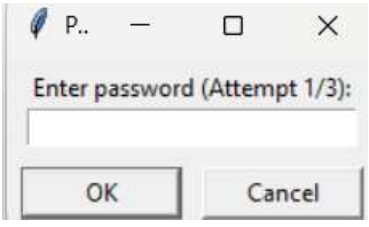
- **Add password if you want and save the image**

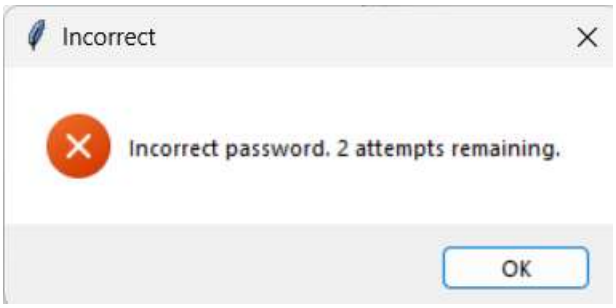


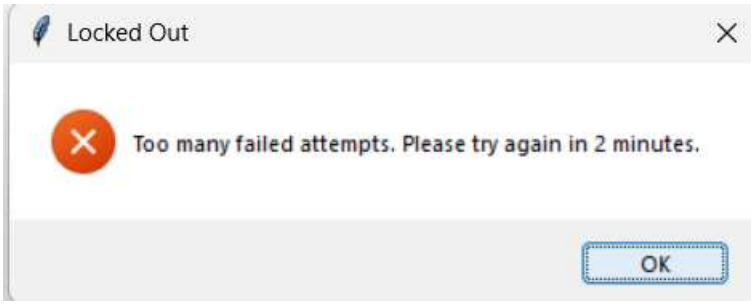
- **Message is been encoded.**

- **To decode click on decode icon and enter the password message will be decoded.**

- **Security Mechanisms**
  - XOR Encryption: Optional password-based encryption before embedding.
  - Lockout Logic:
    - After 3 incorrect password attempts, user is temporarily locked out.
    - Lockout time increases with each failure (30s → 1h).







- **Conclusion**

This tool provides a simple yet effective method of hiding sensitive data in images. It serves as an educational example of how steganography can be implemented using LSB and Python, with added usability and security features.