**1. Project Setup**

* **Objective**: Encrypt and decrypt images using a password and a numerical key in Python, Tkinter, and SQLite.
* **Tools & Libraries**:
  + Tkinter for the GUI.
  + SQLite for storing encryption details.
  + PIL (Python Imaging Library) for handling image operations.

**2. Required Libraries**

Ensure the following libraries are installed:

bash

Copy code

pip install tkinter pillow sqlite3

**3. Project Structure**

* **Main Python file**: image\_encryption\_app.py
* **SQLite Database**: Automatically created (data.db) for storing user details (password and key).
* **Encrypted/Decrypted Files**: Image files chosen by the user.

**4. Key Functions Overview**

* **file\_browser()**: Lets the user select an image from the file system.
* **Encrypt()**: Opens a form where the user enters a password and key to encrypt the image.
* **encryptionimage(password, key)**: Encrypts the selected image using the XOR operation with the provided key.
* **Decrypt()**: Opens a form where the user enters a password and key to decrypt the image.
* **decryptionimage(password\_input, key)**: Decrypts the image by reversing the XOR operation using the same key.

**5. Code Walkthrough**

**Step 1: Import Libraries**

python

Copy code

import tkinter as tk

from tkinter import \*

from tkinter import messagebox as mg

from tkinter import filedialog as file

from PIL import Image, ImageTk

import os

import sqlite3

Tkinter and SQLite are used for the GUI and database, respectively, and PIL handles image loading and manipulation.

**Step 2: Create the Main Application Window**

python

Copy code

app = tk.Tk()

app.title("Encryption App")

app.geometry("900x600")

app.config(bg='#F5F5F5')

This creates the main application window, setting its title, dimensions, and background color.

**Step 3: File Browser Function**

python

Copy code

def file\_browser():

global fileses, img

fileses = file.askopenfilename(initialdir=os.getcwd(), title='Select image',

filetypes=(("PNG file", "\*.png"), ('JPEG files', '\*.jpg'), ('All files', '\*.\*')))

if fileses:

img = Image.open(fileses)

img = ImageTk.PhotoImage(img)

lbl.configure(image=img, width=200, height=200)

lbl.image = img

Allows the user to select an image file (PNG/JPEG). Displays the image on the app.

**Step 4: Encrypt Image**

python

Copy code

def Encrypt():

new = tk.Toplevel(app)

new.geometry('500x300')

new.config(bg='#F5F5F5')

label = tk.Label(new, text='Enter the details', bg='#F5F5F5', fg='black', font=("Calibri", 15))

label.grid(row=0, column=1, pady=10)

# Key entry

message = tk.Label(new, text="Key (Number):", fg='black', bg='#F5F5F5')

message.grid(row=1, column=0, padx=65, pady=10, sticky='w')

filentry = tk.Entry(new)

filentry.grid(row=1, column=1, padx=10, pady=10, ipadx=30, ipady=6, sticky='w')

# Password entry

passwords = tk.Label(new, text="Password:", fg='black', bg='#F5F5F5')

passwords.grid(row=3, column=0, padx=65, pady=10, sticky='w')

passwordsentry = tk.Entry(new, show="@")

passwordsentry.grid(row=3, column=1, padx=10, pady=10, ipadx=30, ipady=6, sticky='w')

# Confirm button

def ok():

filepath = filentry.get()

password = passwordsentry.get()

con = sqlite3.connect('data.db')

cur = con.cursor()

exists = cur.execute('SELECT name FROM sqlite\_master WHERE type="table" AND name="users"')

user\_result = exists.fetchone()

if user\_result is None:

cur.execute('CREATE TABLE users (password TEXT, key INTEGER)')

cur.execute('INSERT INTO users (password, key) VALUES (?, ?)', (password, filepath))

con.commit()

con.close()

encryptionimage(password, filepath)

new.destroy()

ok\_button = tk.Button(new, text='OKEY', bg='red', fg='black', command=ok)

ok\_button.grid(row=7, column=0, columnspan=3, pady=27)

A separate window allows the user to input a password and encryption key. Once entered, the image is encrypted.

**Step 5: Encryption Logic**

python

Copy code

def encryptionimage(password, key):

global fileses

if os.path.exists(fileses):

with open(fileses, 'rb') as file:

image = bytearray(file.read())

for index, value in enumerate(image):

image[index] = value ^ int(key)

with open(fileses, 'wb') as encrypted\_file:

encrypted\_file.write(image)

mg.showinfo('Success', 'Image encrypted successfully!')

The selected image is encrypted using an XOR operation with the key.

**Step 6: Decrypt Image**

python

Copy code

def Decrypt():

new = tk.Toplevel(app)

new.geometry('500x300')

new.config(bg='#F5F5F5')

label1 = tk.Label(new, text='Enter the details', bg='#F5F5F5', fg='black', font=("Calibri", 15))

label1.grid(row=0, column=1, pady=10)

message1 = tk.Label(new, text="Key (Number):", fg='black', bg='#F5F5F5')

message1.grid(row=1, column=0, padx=65, pady=10, sticky='w')

filentry1 = tk.Entry(new)

filentry1.grid(row=1, column=1, padx=10, pady=10, ipadx=30, ipady=6, sticky='w')

passworddecrypt = tk.Label(new, text="Password:", fg='black', bg='#F5F5F5',)

passworddecrypt.grid(row=3, column=0, padx=65, pady=10, sticky='w')

passworddecryptentry = tk.Entry(new, show="@")

passworddecryptentry.grid(row=3, column=1, padx=10, pady=10, ipadx=30, ipady=6, sticky='w')

def ok():

file\_key = filentry1.get()

password\_input = passworddecryptentry.get()

decryptionimage(password\_input, file\_key)

new.destroy()

ok\_button = tk.Button(new, text='OKEY', bg='red', fg='black', command=ok)

ok\_button.grid(row=7, column=0, columnspan=3, pady=27)

A form where users input the password and key to decrypt the image.

**Step 7: Decryption Logic**

python

Copy code

def decryptionimage(password\_input, key):

global fileses

if os.path.exists(fileses):

with open(fileses, 'rb') as encrypted\_file:

image = bytearray(encrypted\_file.read())

for index, value in enumerate(image):

image[index] = value ^ int(key)

with open(fileses, 'wb') as decrypted\_file:

decrypted\_file.write(image)

mg.showinfo('Success', 'Image decrypted successfully!')

The image is decrypted by applying the same XOR operation with the key.

**6. User Interface Overview**

* **Main Window**: Displays the project title, an "Open Image" button to browse files, and two buttons: "Encrypt" and "Decrypt".
* **Sub-windows**: For both encryption and decryption, a pop-up appears where users input the required key and password.

**7. SQLite Database**

SQLite is used to store the password and key for image encryption. The database (data.db) automatically creates a table named users to store the credentials.

**8. Steps to Run the Project**

**Step 1: Clone or Create the Project**

Ensure you have the complete source code either by cloning a repository (if hosted) or copying the files to your local machine.

**Step 2: Install Dependencies**

Use the following command to install necessary dependencies:

bash

Copy code

pip install pillow sqlite3

**Step 3: Run the Application**

Run the main Python file to start the Tkinter GUI:

bash

Copy code

python image\_encryption\_app.py

**Step 4: Select an Image**

* Click on the **Open Image** button to browse and select an image file.
* Supported formats: PNG and JPEG.

**Step 5: Encrypt the Image**

* Click **Encrypt** to open a new window.
* Enter a password and a numerical key.
* The image will be encrypted and stored back at the same location.

**Step 6: Decrypt the Image**

* Click **Decrypt** to open the decryption window.
* Enter the same password and key used for encryption.
* The image will be decrypted and restored to its original form.

**9. Error Handling**

* **File Not Found**: If the selected image file cannot be found, an error message will be shown.
* **Incorrect Password or Key**: If the entered password or key does not match the stored credentials, a message will notify the user.
* **Key Validation**: The key must be an integer. If a non-integer value is entered, an error will be raised.

**10. Security Considerations**

* **Password Encryption**: The passwords used are stored in plain text in the SQLite database. For a more secure implementation, consider using a hashing algorithm (like SHA-256) to store hashed passwords instead of plain text.
* **Key Validation**: Ensure users enter a valid numerical key, as this is essential for both encryption and decryption.

**11. Enhancements for the Future**

* **File Type Extensions**: Currently, only PNG and JPEG formats are supported. Future enhancements could include support for other file formats.
* **Password Hashing**: Enhance security by storing hashed passwords in the database rather than plain text.
* **GUI Improvements**: Add a progress bar for large files and better user feedback during encryption/decryption.

**12. Code Maintenance**

* **Database Integrity**: Regularly ensure that the SQLite database (data.db) is not corrupted or inaccessible. Backup data if needed.
* **Error Logging**: Add logging functionality to keep track of any errors that occur during execution. This would assist in debugging and improving the project.

**Conclusion**

This project provides a basic yet effective way to encrypt and decrypt image files using Tkinter and Python. The process is secure enough for personal use cases, and with a few enhancements, it can be adapted for more advanced applications.