File Encryption and Decryption Tool:

**Operation.**

\*\*\*\*Important\*\*\*\*

This tool on loading will create a new encryption key file (encryption\_key.key} in the same directory which will need to be stored safely to enable you to decrypt the files in the future otherwise you will not be able to recover the encrypted files. If this file is already located in the directory the application will use that file and not create a new one however is best practice to import the original key used to encrypt the files for decryption to be ensured.

You can also save and import keys as required.

The tool will not destroy the original files after encryption or decryption for your own safety.

If you do require the files to be destroyed there is a shredder option included.

\*\*\*Please be careful when using this tool\*\*\*

**This was created for educational purposes only. Feel free to alter the code freely.**

**Features**:

1. File Encryption: Allow users to encrypt files using strong encryption algorithms such as AES (Advanced Encryption Standard).

2. File Decryption: Enable users to decrypt encrypted files using the appropriate decryption key or password.

3. Strong Encryption Algorithms: Implement support for industry-standard encryption algorithms with configurable key sizes.

4. Password Protection: Require users to set a strong password for encrypting files and prompt for it during decryption.

5. File Shredding: Include an option to securely delete original unencrypted files after they've been successfully encrypted to prevent unauthorized access.

6. Batch Processing: Allow users to encrypt or decrypt multiple files simultaneously to save time and streamline workflow.

7. Progress Indication: Provide visual feedback on the progress of encryption and decryption processes, especially for large files or batches of files.

8. File Explorer Integration: Enable users to encrypt or decrypt files directly from the file explorer context menu for ease of use.

9. Customizable Encryption Settings: Allow users to customize encryption settings such as encryption algorithm, key size, and encryption mode (e.g., CBC, ECB).

10. Cross-Platform Compatibility: Ensure compatibility with major desktop operating systems like Windows, macOS, and Linux to maximize accessibility.

11. Error Handling: Implement robust error handling mechanisms to gracefully handle errors such as invalid file formats, incorrect passwords, or file corruption during encryption or decryption.

12. Logging and Audit Trail: Maintain logs of encryption and decryption activities, including timestamps and file paths, for audit and troubleshooting purposes.

13. User-Friendly Interface: Design an intuitive and user-friendly interface with clear instructions and prompts for encrypting and decrypting files.

14. File Format Support: Support a wide range of file formats for encryption and decryption to accommodate diverse user needs.

15. Data Integrity Verification: Implement mechanisms to verify the integrity of encrypted files during decryption to detect tampering or corruption.

16. Automatic Updates: Provide automatic update functionality to ensure users have the latest security patches and feature enhancements.

**Requirements**:

1. Programming Language: Choose a suitable programming language for desktop application development, such as Java, C#, or Python.

2. Encryption Library: Select a reliable encryption library or framework that supports strong encryption algorithms and provides necessary functionalities for encryption and decryption.

3. User Interface Toolkit: Use a user interface toolkit or framework for building the graphical user interface (GUI) of the application, ensuring it is visually appealing and easy to navigate.

4. Security Best Practices: Adhere to security best practices throughout the development process, including secure handling of encryption keys, password hashing, and protection against common security vulnerabilities like buffer overflows or injection attacks.

5. Testing and Quality Assurance: Conduct thorough testing of the application to identify and address any bugs, compatibility issues, or security vulnerabilities before deployment.

6. Documentation: Prepare comprehensive documentation covering installation instructions, usage guidelines, troubleshooting tips, and security recommendations for users.

7. Compliance: Ensure compliance with relevant data protection regulations and standards, such as GDPR (General Data Protection Regulation) or HIPAA (Health Insurance Portability and Accountability Act), if applicable.

8. Feedback Mechanism: Implement a mechanism for users to provide feedback, report bugs, or request new features, facilitating continuous improvement of the application.

By incorporating these features and meeting the specified requirements, you can develop a robust and secure File Encryption and Decryption Tool that meets the needs of users who require secure handling of sensitive files.

Example python code outline:

import os

import shutil

from cryptography.fernet import Fernet

class FileEncryptorDecryptor:

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

self.key = key

self.cipher = Fernet(key)

def encrypt\_file(self, input\_file, output\_file):

# Encrypts a single file using the provided key

with open(input\_file, 'rb') as f:

data = f.read()

encrypted\_data = self.cipher.encrypt(data)

with open(output\_file, 'wb') as f:

f.write(encrypted\_data)

def decrypt\_file(self, input\_file, output\_file):

# Decrypts a single file using the provided key

with open(input\_file, 'rb') as f:

data = f.read()

decrypted\_data = self.cipher.decrypt(data)

with open(output\_file, 'wb') as f:

f.write(decrypted\_data)

def batch\_encrypt\_files(self, input\_dir, output\_dir):

# Encrypts multiple files in a directory

for filename in os.listdir(input\_dir):

if os.path.isfile(os.path.join(input\_dir, filename)):

input\_file = os.path.join(input\_dir, filename)

output\_file = os.path.join(output\_dir, filename + '.encrypted')

self.encrypt\_file(input\_file, output\_file)

def batch\_decrypt\_files(self, input\_dir, output\_dir):

# Decrypts multiple files in a directory

for filename in os.listdir(input\_dir):

if os.path.isfile(os.path.join(input\_dir, filename)):

input\_file = os.path.join(input\_dir, filename)

output\_file = os.path.join(output\_dir, filename[:-10]) # remove '.encrypted' from filename

self.decrypt\_file(input\_file, output\_file)

def shred\_file(self, file\_path):

# Securely deletes a file by overwriting it with random data before deletion

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

length = f.tell()

f.seek(0)

f.write(os.urandom(length))

os.remove(file\_path)

# Additional methods for UI integration, error handling, logging, etc. can be added here

# Main function for CLI usage

def main():

key = Fernet.generate\_key() # Generate a random encryption key

encryptor\_decryptor = FileEncryptorDecryptor(key)

# Example usage:

input\_file = 'example.txt'

encrypted\_file = 'example.txt.encrypted'

decrypted\_file = 'example\_decrypted.txt'

encryptor\_decryptor.encrypt\_file(input\_file, encrypted\_file)

encryptor\_decryptor.decrypt\_file(encrypted\_file, decrypted\_file)

input\_dir = 'input\_directory'

output\_dir = 'output\_directory'

encryptor\_decryptor.batch\_encrypt\_files(input\_dir, output\_dir)

encryptor\_decryptor.batch\_decrypt\_files(output\_dir, input\_dir)

file\_to\_shred = 'file\_to\_shred.txt'

encryptor\_decryptor.shred\_file(file\_to\_shred)

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

main()

This code outline demonstrates the basic structure of a Python application for file encryption and decryption. You'll need to install the cryptography library (pip install cryptography) for the encryption functionality. Additionally, you'll want to extend this outline with features like user interface integration, error handling, logging, and more, based on your specific requirements and design choices.

**Final solution:**

import customtkinter as tk

from customtkinter import filedialog

from tkinter import messagebox

from cryptography.fernet import Fernet

import os

class CustomFileEncryptorDecryptor:

    def \_\_init\_\_(self):

        self.key\_file = "encryption\_key.key"

        self.key = None

        self.cipher = None

        self.input\_file = None

        self.output\_file = None

        self.input\_dir = None

        self.output\_dir = None

        self.load\_or\_generate\_key()

    def load\_or\_generate\_key(self):

        if os.path.exists(self.key\_file):

            with open(self.key\_file, 'rb') as f:

                self.key = f.read()

        else:

            self.key = Fernet.generate\_key()

            with open(self.key\_file, 'wb') as f:

                f.write(self.key)

        self.cipher = Fernet(self.key)

    def save\_key(self):

        if self.key:

            with open(self.key\_file, 'wb') as f:

                f.write(self.key)

            messagebox.showinfo("Success", "Encryption key saved successfully!")

        else:

            messagebox.showerror("Error", "No encryption key found!")

    def import\_key(self):

        file\_path = filedialog.askopenfilename()

        if file\_path:

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

                self.key = f.read()

                self.cipher = Fernet(self.key)

                messagebox.showinfo("Success", "Encryption key imported successfully!")

        else:

            messagebox.showerror("Error", "No file selected!")

    def select\_file(self):

        self.input\_file = filedialog.askopenfilename()

        return self.input\_file

    def select\_output\_file(self):

        self.output\_file = filedialog.asksaveasfilename(defaultextension=".encrypted")

        return self.output\_file

    def select\_directory(self):

        self.input\_dir = filedialog.askdirectory()

        return self.input\_dir

    def select\_output\_directory(self):

        self.output\_dir = filedialog.askdirectory()

        return self.output\_dir

    def encrypt\_file(self):

        self.select\_file()

        if self.input\_file:

            self.select\_output\_file()

            if self.output\_file:

                try:

                    with open(self.input\_file, 'rb') as f:

                        data = f.read()

                        encrypted\_data = self.cipher.encrypt(data)

                    with open(self.output\_file, 'wb') as f:

                        f.write(encrypted\_data)

                    messagebox.showinfo("Success", "File encrypted successfully!")

                except Exception as e:

                    messagebox.showerror("Error", f"Encryption failed: {str(e)}")

    def decrypt\_file(self):

        self.select\_file()

        if self.input\_file:

            self.select\_output\_file()

            if self.output\_file:

                try:

                    with open(self.input\_file, 'rb') as f:

                        data = f.read()

                        decrypted\_data = self.cipher.decrypt(data)

                    with open(self.output\_file, 'wb') as f:

                        f.write(decrypted\_data)

                    messagebox.showinfo("Success", "File decrypted successfully!")

                except Exception as e:

                    messagebox.showerror("Error", f"Decryption failed: {str(e)}")

    def batch\_encrypt\_files(self):

        self.select\_directory()

        if self.input\_dir:

            self.select\_output\_directory()

            if self.output\_dir:

                try:

                    for filename in os.listdir(self.input\_dir):

                        if os.path.isfile(os.path.join(self.input\_dir, filename)):

                            input\_file = os.path.join(self.input\_dir, filename)

                            output\_file = os.path.join(self.output\_dir, filename + '.encrypted')

                            with open(input\_file, 'rb') as f:

                                data = f.read()

                                encrypted\_data = self.cipher.encrypt(data)

                            with open(output\_file, 'wb') as f:

                                f.write(encrypted\_data)

                    messagebox.showinfo("Success", "Batch encryption completed!")

                except Exception as e:

                    messagebox.showerror("Error", f"Batch encryption failed: {str(e)}")

    def batch\_decrypt\_files(self):

        self.select\_directory()

        if self.input\_dir:

            self.select\_output\_directory()

            if self.output\_dir:

                try:

                    for filename in os.listdir(self.input\_dir):

                        if os.path.isfile(os.path.join(self.input\_dir, filename)):

                            input\_file = os.path.join(self.input\_dir, filename)

                            output\_file = os.path.join(self.output\_dir, filename[:-10])  # remove '.encrypted' from filename

                            with open(input\_file, 'rb') as f:

                                data = f.read()

                                decrypted\_data = self.cipher.decrypt(data)

                            with open(output\_file, 'wb') as f:

                                f.write(decrypted\_data)

                    messagebox.showinfo("Success", "Batch decryption completed!")

                except Exception as e:

                    messagebox.showerror("Error", f"Batch decryption failed: {str(e)}")

    def shred\_file(self):

        self.select\_file()

        if self.input\_file:

            try:

                with open(self.input\_file, 'rb+') as f:

                    length = f.tell()

                    f.seek(0)

                    f.write(os.urandom(length))

                os.remove(self.input\_file)

                messagebox.showinfo("Success", "File shredded successfully!")

            except Exception as e:

                messagebox.showerror("Error", f"File shredding failed: {str(e)}")

    def show\_about\_dialog(self):

        messagebox.showinfo("About", "Custom File Encryptor/Decryptor\nVersion 1.0\n\nDeveloped by DPH Security")

def create\_custom\_gui():

    root = tk.CTk()

    root.title("File Encryptor/Decryptor")

    label = tk.CTkLabel(root, text="Select an operation:")

    label.grid(row=0, column=0, columnspan=2, pady=10)

    encrypt\_button = tk.CTkButton(root, text="Encrypt File", command=app.encrypt\_file)

    encrypt\_button.grid(row=1, column=0, pady=5, padx=5)

    decrypt\_button = tk.CTkButton(root, text="Decrypt File", command=app.decrypt\_file)

    decrypt\_button.grid(row=1, column=1, pady=5, padx=5)

    batch\_encrypt\_button = tk.CTkButton(root, text="Batch Encrypt Files", command=app.batch\_encrypt\_files)

    batch\_encrypt\_button.grid(row=2, column=0, pady=5, padx=5)

    batch\_decrypt\_button = tk.CTkButton(root, text="Batch Decrypt Files", command=app.batch\_decrypt\_files)

    batch\_decrypt\_button.grid(row=2, column=1, pady=5, padx=5)

    shred\_button = tk.CTkButton(root, text="Shred File", command=app.shred\_file)

    shred\_button.grid(row=3, column=0, pady=5, padx=5)

    about\_button = tk.CTkButton(root, text="About", command=app.show\_about\_dialog)

    about\_button.grid(row=3, column=1, pady=5, padx=5)

    save\_key\_button = tk.CTkButton(root, text="Save Encryption Key", command=app.save\_key)

    save\_key\_button.grid(row=4, column=0, pady=5, padx=5)

    import\_key\_button = tk.CTkButton(root, text="Import Encryption Key", command=app.import\_key)

    import\_key\_button.grid(row=4, column=1, pady=5, padx=5)

    root.mainloop()

def main():

    global app

    app = CustomFileEncryptorDecryptor()

    create\_custom\_gui()

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

    main()

There are some features which are not included yet but his is a functioning tool.