

File Encryptor GUI

Author: Daniel Chisasura

Version: 1.0

Purpose:

This Python application provides a simple graphical interface for encrypting and decrypting files using password-based encryption. It's designed for users who want to secure sensitive files without needing command-line tools.

Overview

The script uses the cryptography library to perform secure encryption and decryption using the **Fernet** symmetric encryption scheme. A password entered by the user is transformed into a secure key using a **Key Derivation Function (KDF)**, ensuring strong protection against brute-force attacks.

The GUI is built with **Tkinter**, making it easy to use for non-technical users.

Features

- Password-based encryption using PBKDF2 and SHA-256.
- Secure file encryption and decryption with Fernet.
- Simple and intuitive GUI built with Tkinter.
- Error handling for missing passwords, invalid files, and incorrect decryption attempts.


Skills Demonstrated

- GUI development with Tkinter
- Cryptographic key derivation using PBKDF2
- File encryption and decryption with Fernet

- Secure password handling
- Exception handling and user feedback

How to Use

1. **Install dependencies:**
2. **Run the application:**
3. **In the GUI:**
 - a. Enter a password.
 - b. Click **Encrypt File** to select and encrypt a file.
 - c. Click **Decrypt File** to select and decrypt a previously encrypted file.

 Encrypted files will be saved with a `.enc` extension. Decrypted files will restore the original name.

Notes

- A **fixed salt** is used for simplicity. In production, use a unique salt per file and store it securely.
- The encryption key is derived using **100,000 iterations** of PBKDF2 for strong security.
- The GUI prevents empty password submissions and handles common errors gracefully.

Sample Workflow

- ✓ User enters password
- 📁 Selects file to encrypt
- 🔒 File is encrypted and saved as `filename.ext.enc`
- 🔒 Later, selects encrypted file and enters same password
- ✓ File is decrypted and restored

Lessons Learned

- **Cryptography requires careful design:** Using a KDF like PBKDF2 ensures passwords are transformed into strong keys.
- **User experience matters:** A GUI makes encryption accessible to non-technical users.

- **Error handling improves reliability:** Catching exceptions and guiding users prevents frustration.
- **Security trade-offs:** Using a fixed salt simplifies the demo but isn't ideal for real-world use.
- **Modular design helps clarity:** Separating encryption logic from GUI code improves maintainability.