CipherShield - A Secure File Encryption Utility

1. Introduction

CipherShield is a desktop application developed in Python that provides a user-friendly graphical interface for securely encrypting and decrypting files. The primary objective of this project is to offer a robust and accessible tool for protecting sensitive data. By leveraging industry-standard cryptographic libraries, CipherShield ensures that files are protected against unauthorized access. The application uses password-based key derivation, meaning the security of the files is tied to a user-provided password, eliminating the need to manage separate key files.

2. Core Features

- Password-Based Encryption: Files are encrypted using a key derived from a user-provided password.
- Strong Encryption Standard: Utilizes the Fernet symmetric encryption scheme, which is built on AES-128 in CBC mode with PKCS7 padding.
- Tamper-Proofing: A SHA-256 hash of the original file is stored with the encrypted data. During decryption, this hash is verified to ensure the file has not been altered or corrupted.
- Salted Key Derivation: Employs the PBKDF2 algorithm with a random salt for each encryption. This mitigates dictionary and rainbow table attacks, ensuring that identical passwords result in different encryption keys.
- Intuitive Graphical User Interface (GUI): Built with PyQt5, the application features an easy-to-navigate interface, enabling users to encrypt or decrypt files with just a few clicks.
- Robust User Experience: Features include a password confirmation dialog to prevent typos and a safe-overwrite mechanism to prevent accidental data loss during decryption.

3. Technologies and Libraries Used

- Language: Python 3
- GUI Framework: PyQt5
- Cryptography: cryptography library (specifically Fernet and PBKDF2HMAC).
- Hashing: hashlib module for SHA-256 integrity checks.

By Anushree Govekar

4. Key Code Snippets Explained

4.1 Password-Based Key Derivation (derive_key)

This function is the security core of the application. It takes a user's password and a random salt and runs them through the PBKDF2 algorithm 100,000 times. This computationally intensive process creates a secure 32-byte key that is extremely difficult to brute-force.

```
def derive_key(password: bytes, salt: bytes) -> bytes:
    """
    Derives a secure encryption key from a password and salt using PBKDF2.

    Args:
        password: The user's password in bytes.
        salt: A random salt in bytes.

    Returns:
        A URL-safe base64-encoded 32-byte key.
    """
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=salt,
        iterations=100000,
    )
    return base64.urlsafe_b64encode(kdf.derive(password))
```

4.2 File Encryption Process (encrypt_file)

The encryption process involves reading the file, calculating its hash, generating a new random salt, deriving the key, and then encrypting the data. The final output file is carefully structured by prepending the 16-byte salt to the encrypted ciphertext. This salt is not a secret and is required for decryption.

```
def encrypt_file(filename, password):
    """
    Encrypts a file using a password. The output file (.enc) contains:
    [16-byte salt][encrypted data]

    Args:
        filename: The path to the file to encrypt.
        password: The password to use for encryption.

    Returns:
        A string indicating the status of the operation.
    """
```

4.3 Secure Password Entry (PasswordDialog)

To enhance usability and prevent errors, a custom dialog was created. This dialog requires the user to enter their password twice. The "OK" button is only enabled when both fields are non-empty and the passwords match, providing immediate feedback to the user.

```
class PasswordDialog(QDialog):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.setWindowTitle("Enter Password")
        self.layout = QVBoxLayout(self)

        self.pass_label = QLabel("Password:")
        self.pass_input = QLineEdit()
        self.pass_input.setEchoMode(QLineEdit.Password)

        self.confirm_label = QLabel("Confirm Password:")
        self.confirm_input = QLineEdit()
        self.confirm_input.setEchoMode(QLineEdit.Password)

        self.message_label = QLabel("") # To show error messages like "Passwords
        self.message_label.setStyleSheet("color: red")

        self.buttons = QPushButton("OK")
        self.buttons.setEnabled(False) # Start with OK disabled
```

5. Application Workflow and Output

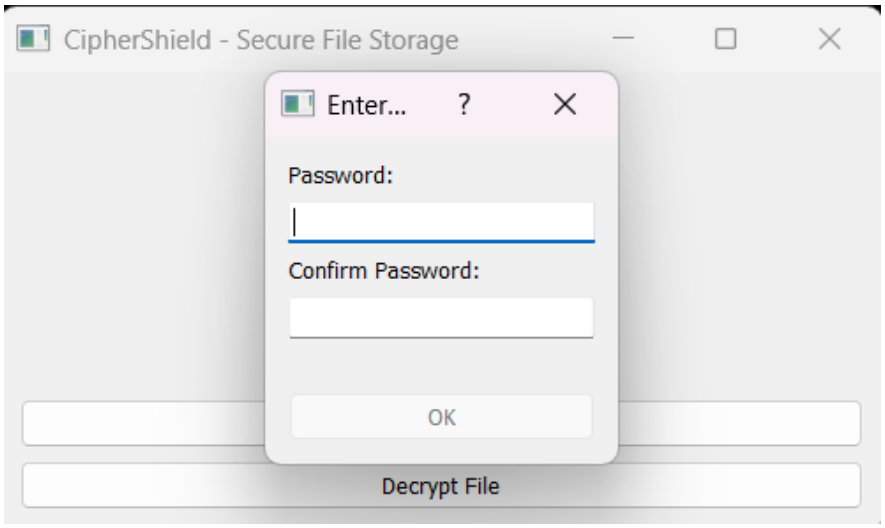
This section demonstrates the user's journey through the CipherShield application.

5.1 Main Window

The application opens to a clean and simple main window displaying the project name and the core actions.

5.2 Encryption Process

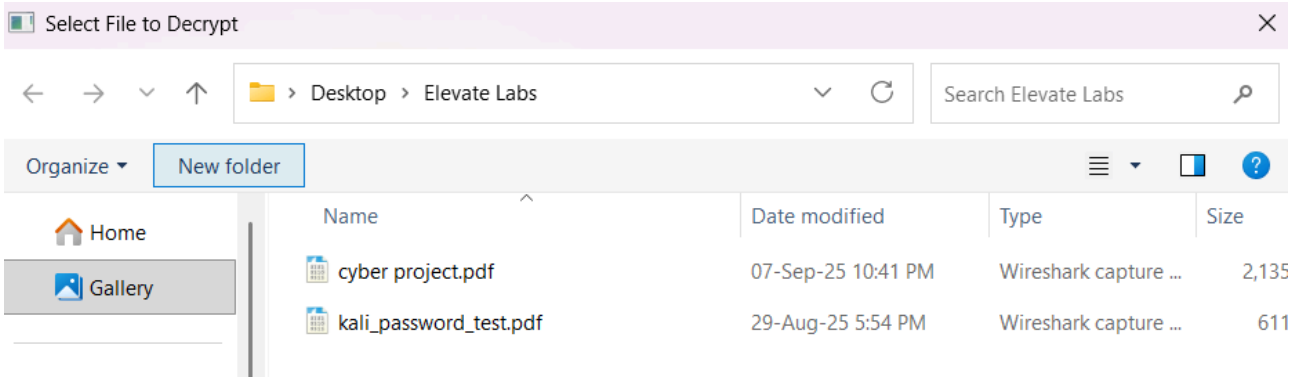
The user clicks "Encrypt File," selects a file, and is then prompted to enter and confirm a secure password.



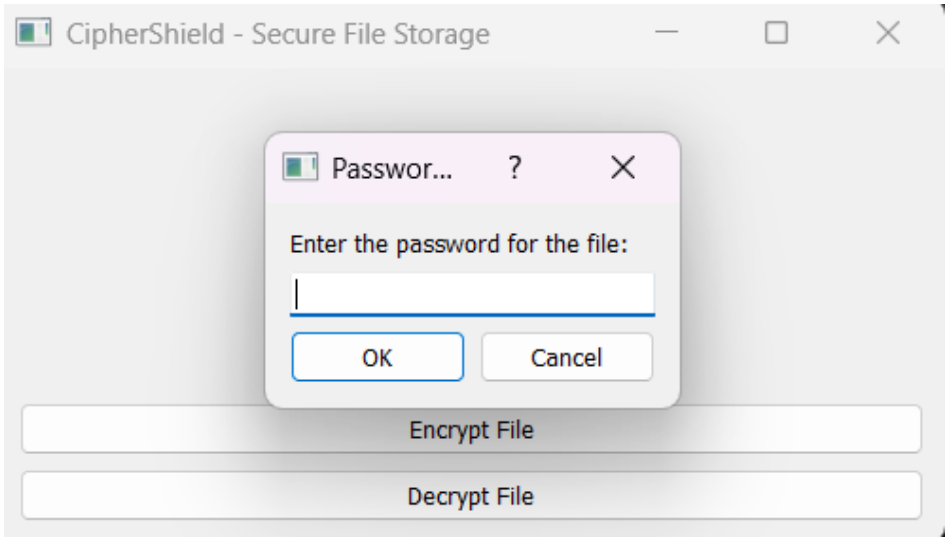
Upon successful encryption, a status message confirms the operation, and the new .enc file appears in the directory.

5.3 Decryption Process

The user selects an encrypted (.enc) file and is prompted for the password.

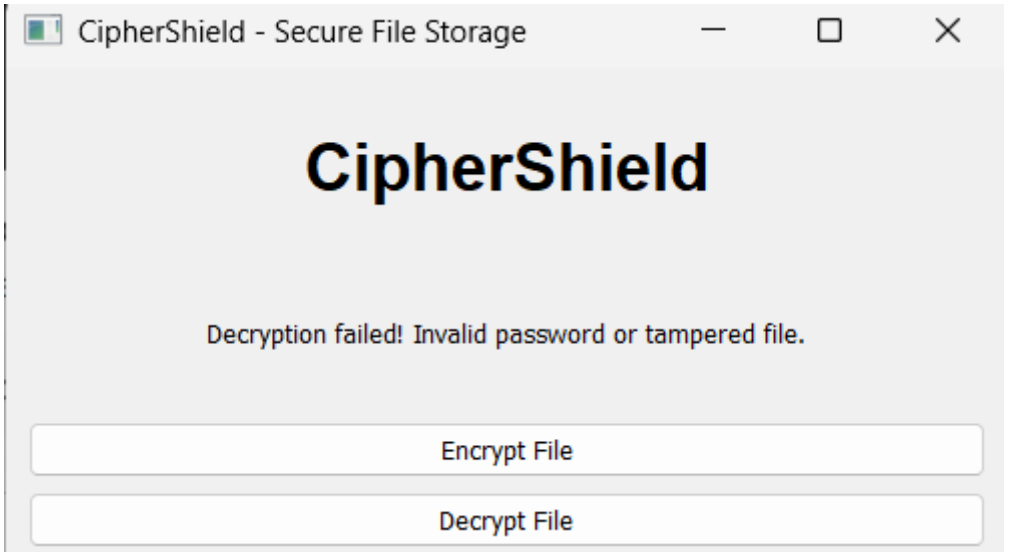


By Anushree Govekar



5.4 Handling Tampering and Incorrect Passwords

If the wrong password is used or if the encrypted file has been modified in any way, the application shows a clear error message, protecting the user from using corrupted data.



6. Conclusion

CipherShield successfully meets its goal of providing a secure and user-friendly file encryption tool. By implementing modern cryptographic principles and focusing on a seamless user experience, the project demonstrates a practical application of secure software development. Future enhancements could include support for encrypting entire folders or integrating alternative encryption algorithms.

By Anushree Govekar