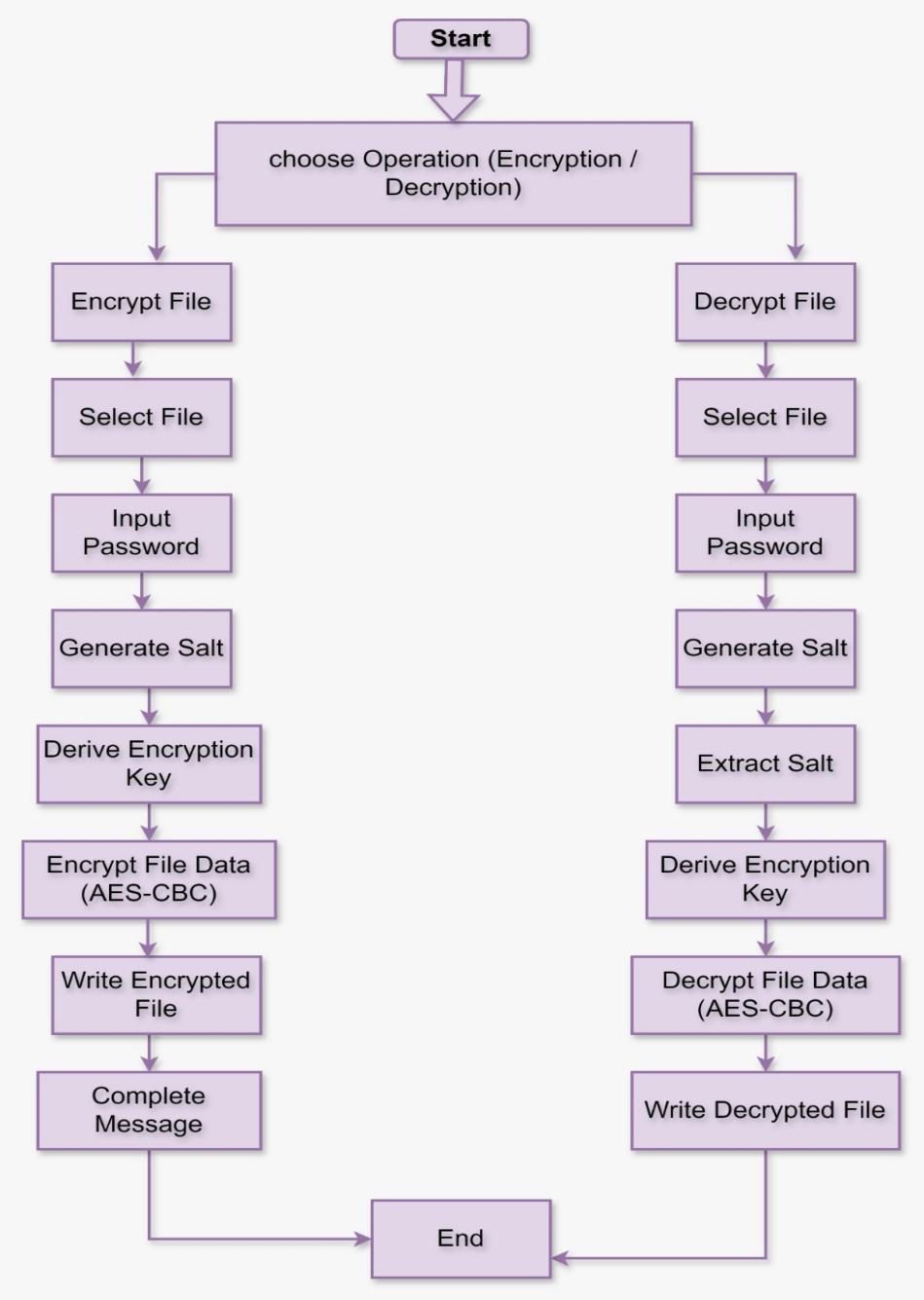
**Title: Protecting User Password Keys at Rest (on the Disk)**

**Technical Approach**

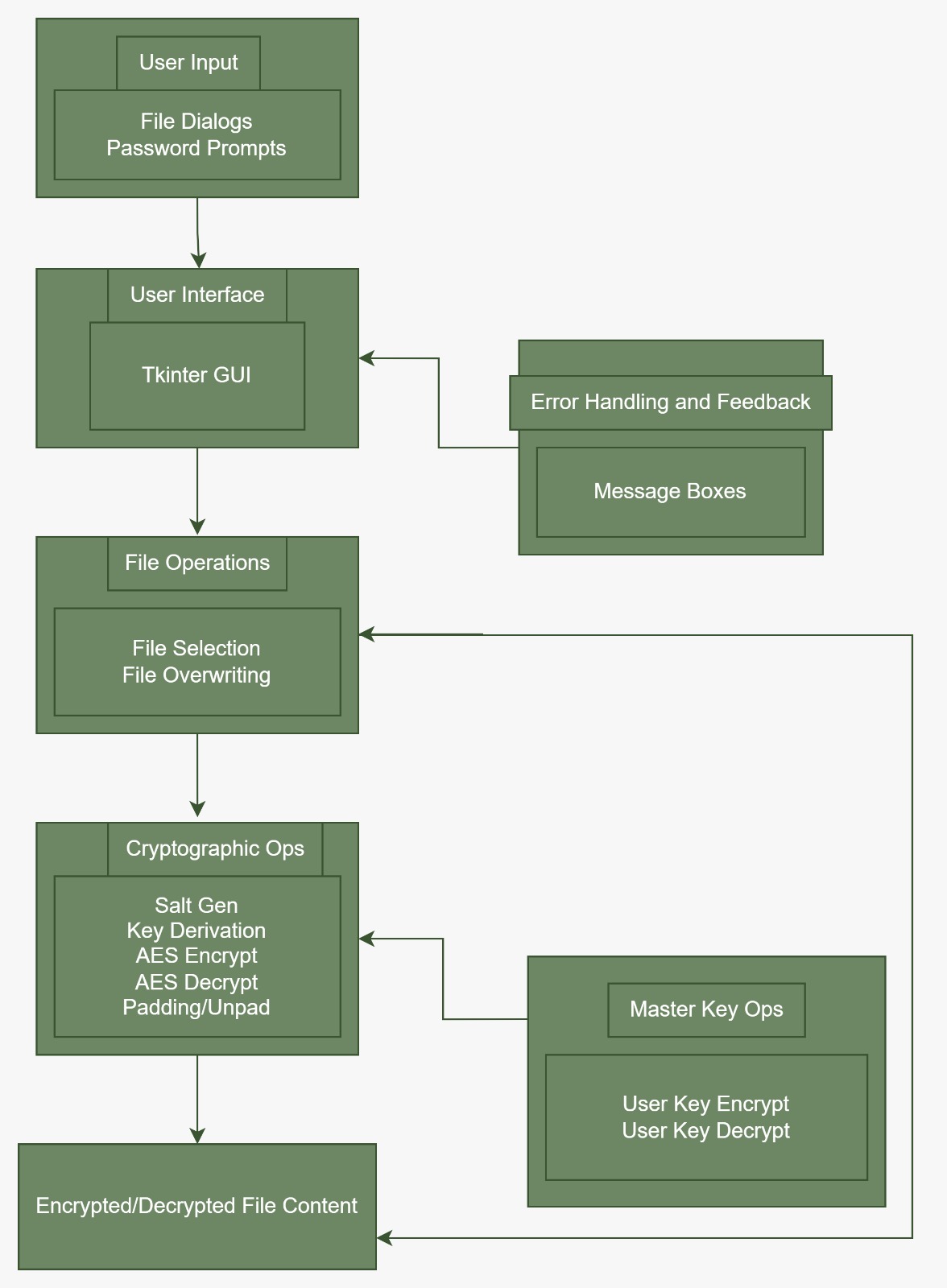
The project aims to secure user password keys when stored on the disk by utilizing a combination of cryptographic techniques. Here's a detailed breakdown of the approach:

1. **Key Derivation from Password**
   * The user provides a password which is then combined with a randomly generated salt.
   * To derive the encryption key from the password, we use the PBKDF2HMAC function. This function implements the Password-Based Key Derivation Function 2 (PBKDF2) with HMAC (Hash-based Message Authentication Code) and SHA-256 as the underlying hash algorithm.
   * The process involves 100000 iterations to enhance security by making brute-force attacks computationally expensive. This approach produces a 32-byte key which serves as the encryption key for further processes.
2. **Master Key Utilization**
   * The master key, which is hardcoded for demonstration purposes, is used to encrypt the user-derived key. In a real-world scenario, this master key should be stored and managed securely, potentially using hardware security modules (HSMs) or key management services (KMS).
   * For the encryption of the user-derived key, we use the AES (Advanced Encryption Standard) algorithm in Cipher Block Chaining (CBC) mode. AES-CBC requires a unique initialization vector (IV) for each encryption operation to ensure security.
   * Padding is applied using the PKCS7 scheme to ensure the plaintext length is a multiple of the block size (16 bytes for AES). The encrypted user key is then concatenated with the IV and stored securely.
3. **Data Encryption and Decryption**
   * The derived key from the user’s password is used for the actual encryption and decryption of the file contents. The AES algorithm in CBC mode is utilized here as well, with new IVs generated for each encryption operation.
   * The encryption process involves padding the plaintext to match the block size and then performing the AES encryption with the derived key and IV. The encrypted data along with the IV is stored or transmitted as needed.
   * During decryption, the IV is extracted from the encrypted data and the AES decryption is performed using the derived key. The decrypted data is then unpadded to retrieve the original plaintext.

**Flow Diagram**



**Architectural Diagram**

****

**Codelink:** https://github.com/Varun-13-sys/Protecting-User-Key/blob/main/passwordprotector.py

**Issues Faced**

1. **Key Management**
   * One of the primary issues was securely storing and retrieving the master key. In a production environment, a more secure approach such as a hardware security module (HSM) or a key management service (KMS) would be necessary. Ensuring the master key's confidentiality and integrity is crucial as its compromise would undermine the security of all encrypted keys.
2. **Password Handling**
   * Ensuring the user-provided password is strong enough to resist brute force attacks was another challenge. This could be mitigated by enforcing password complexity rules and employing password policies. Passwords should be long and complex enough to make brute force attacks impractical, and users should be educated on the importance of using strong, unique passwords.
3. **Error Handling**
   * Handling various exceptions during file operations and cryptographic processes was crucial to ensure the application did not crash and provided meaningful error messages to the user. Errors could arise from issues such as incorrect file paths, insufficient permissions, or corrupted files. Proper error handling ensures the application remains robust and user-friendly, providing clear instructions for resolving issues.
4. **User Experience**
   * Providing a user-friendly interface while maintaining security was a delicate balance. Ensuring the GUI was intuitive and straightforward for non-technical users was an ongoing challenge. The interface needed to clearly guide users through the encryption and decryption processes while ensuring they understood the importance of securely managing their passwords and keys.

**Results**

The implemented solution successfully encrypts and decrypts files based on user-provided passwords. Key results include:

1. **Secure Encryption and Decryption**
   * The files are securely encrypted with AES-256 in CBC mode, and the encryption keys are securely managed using a master key. The use of AES-256 ensures a high level of security, and CBC mode, while requiring careful management of the IV, provides a strong encryption scheme for large amounts of data.
2. **User-friendly Interface**
   * The application provides an intuitive GUI for users to select files and perform encryption or decryption with ease. The GUI was designed with simplicity in mind, allowing users to quickly understand how to use the application. It includes clear prompts for password entry, file selection dialogs, and informative messages about the encryption and decryption status.
3. **Robust Error Handling**
   * The application gracefully handles errors, providing informative messages to the user without compromising security. Error messages are designed to be clear and helpful, guiding users to correct any issues encountered during the encryption or decryption processes. This ensures that even less technically proficient users can use the application effectively.