Protecting User Password Keys at Rest (on the Disk)

Category: System Software, Security

Team Scope: Developing an application for file/folder encryption, which is in turn protected by user passphrase

Pre-requisite: Linux File System Operations, Crypto Algorithms. Programming in any Language suited for System Software like C, C++, Python, etc.

Unique Idea Brief (Solution)

HPJ_CRYP

HPJ_CRYP is a desktop application that is developed to help **encrypt** (lock) and **decrypt** (unlock) files or folders on your computer(**Disk**). Encryption transforms your data into a scrambled format that **can't be read by anyone**, while decryption changes it back to **its original form**.

Main Concepts :

- **Encryption:** The process of converting information into a secret code to hide its contents.
- **Decryption:** The process of converting the encrypted code back to the original information.
- **Password:** A secret word or phrase that is used to access encrypted files.
- **Key:** A piece of information used in the encryption and decryption processes.
- **Salt:** Random data added to your password to make it harder for attackers to guess.
- **AES (Advanced Encryption Standard) :** A widely used encryption method that secures data.

Key Derivation Function (PBKDF2):

- Directly using a password for encryption is insecure. PBKDF2 turns a password into a stronger, more complex key.

AES Cipher in CFB Mode :

- AES (Advanced Encryption Standard) is a widely used encryption algorithm. CFB (Cipher Feedback) mode is a way to apply AES to encrypt data.
- CFB mode allows AES to encrypt data of any size, making it flexible and secure for different file types.
- **Initialization Vector (IV):** A random block of data used to start the encryption.
- **Feedback:** Each piece of data is encrypted using the previous encrypted piece, providing strong security.

HMAC (Hash-Based Message Authentication Code):

- A way to check the integrity of data and ensure it hasn’t been tampered.
- HMAC helps verify that the data is unchanged and authentic, especially when using passwords for encryption.

File Encryption Flow :

- Encrypting files involves turning readable data into an unreadable format using a key. This key is securely managed using the user’s password.

File Decryption Flow :

- Decrypting files involves reversing the encryption process to restore the original data using the password.

Features of HPJ_CRYP

- **File Encryption:**

a. Converts your files into an unreadable format to protect their content.
b. Uses AES (Advanced Encryption Standard) in CFB (Cipher Feedback) mode to securely encrypt files.
c. Keeps sensitive information safe from unauthorized access.

- **File Decryption:**

a. Converts your encrypted files back to their original readable format.
b. Uses AES in CFB mode to decrypt files, requiring the correct password.
c. Allows you to access your files when needed securely.

- **Directory Encryption and Decryption:**

a. Encrypts or Decrypts all files within a selected folder.
b. Encrypts or decrypts each file in the folder and its subfolders recursively.
c. Provides a convenient way to protect or access all files in a directory.

- **Password-Based Encryption:**

a. Uses your password to create a strong encryption key for securing files.
b. Derives a key from your password using PBKDF2 (Password-Based Key Derivation Function 2) with a salt (random data).
c. Ensures that only those with the correct password can encrypt or decrypt files.

➢ Random File Key Generation:

a. Generates a unique encryption key for each file.
b. Uses a secure method to create random 32-byte keys.
c. Enhances security by using a different key for each file.

➢ Key Encryption:

a. Encrypts the file's encryption key with a derived key from your password.
b. Uses AES in CFB mode to encrypt the file's key using a key derived from your password.
c. Protects the file's encryption key, making it accessible only with the correct password.

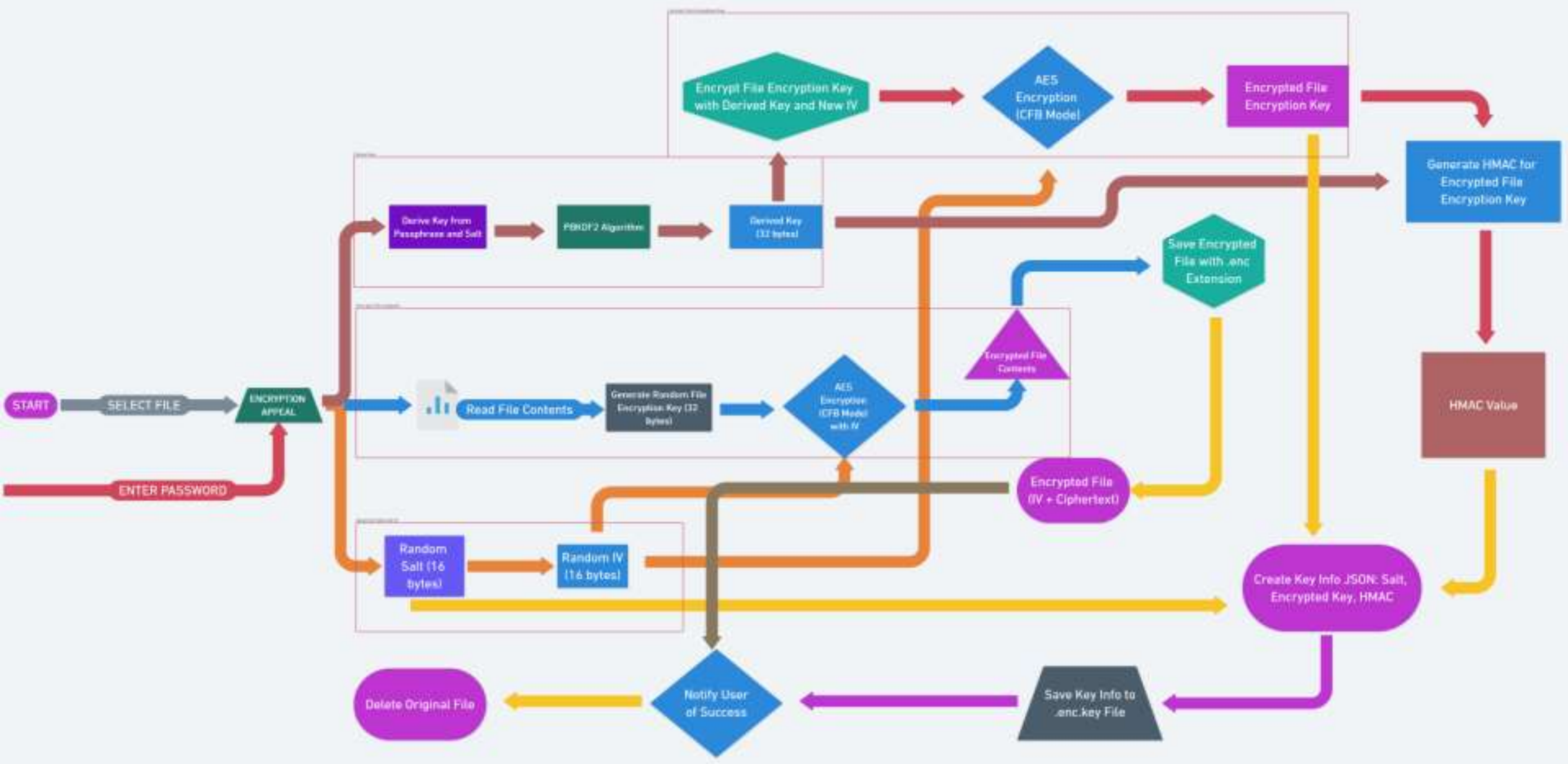
➢ Key Storage:

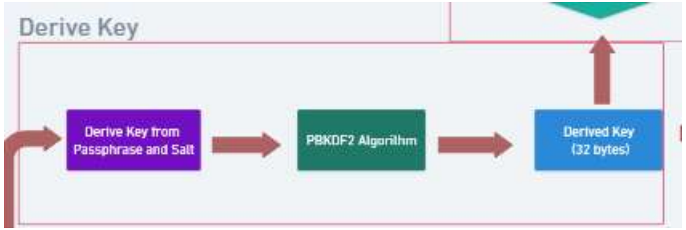
a. Store encrypted keys and additional data needed for decryption in a “.enc.key” file.
b. Saves the encrypted file key, salt, and HMAC in a JSON format.
c. Keeps all necessary information to decrypt a file securely in one place.

➢ HMAC Verification:

a. Verifies the integrity and authenticity of the encrypted file.
b. Uses HMAC (Hash-Based Message Authentication Code) to check that the data hasn’t been tampered with.
c. Ensures that the encrypted file is valid and that the correct password has been used.

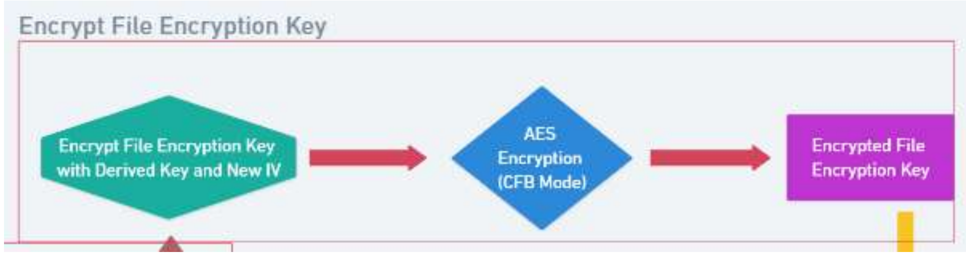
PROCESS FLOW ENCRYPTION


START
SELECT FILE
ENCRYPTION APPEAL
ENTER PASSWORD
Derive Key from Passphrase and Salt
PBKDF2 Algorithm
Derived Key (32 bytes)
Encrypt File Encryption Key with Derived Key and New IV
AES Encryption (CFB Mode)
Encrypted File Encryption Key
Generate HMAC for Encrypted File Encryption Key
HMAC Value
Read File Contents
Generate Random File Encryption Key (32 bytes)
AES Encryption (CFB Mode) with IV
Encrypted File Contents
Save Encrypted File with .enc Extension
Encrypted File (IV + Ciphertext)
Random Salt (16 bytes)
Random IV (16 bytes)
Create Key Info JSON: Salt, Encrypted Key, HMAC
Save Key Info to .enc.key File
Notify User of Success
Delete Original File



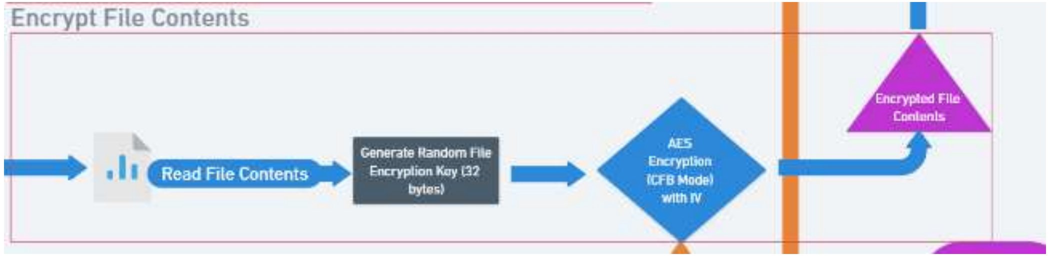
The process of deriving a cryptographic key from a passphrase.

1. A passphrase and salt are combined and fed into the PBKDF2 (Password-Based Key Derivation Function.
2. Algorithm, which performs numerous hashing operations.
3. Derived key of 32 bytes, suitable for use in encryption algorithms like AES-256.


Encrypt File Encryption Key
Encrypt File Encryption Key with Derived Key and New IV
AES Encryption (CFB Mode)
Encrypted File Encryption Key

Encrypt a file encryption key using AES in CFB (Cipher Feedback) mode.

1. File encryption key is combined with a derived key and a new Initialization Vector (IV).
2. This combined data is then encrypted with AES-CFB.
3. The encrypted file encryption key.

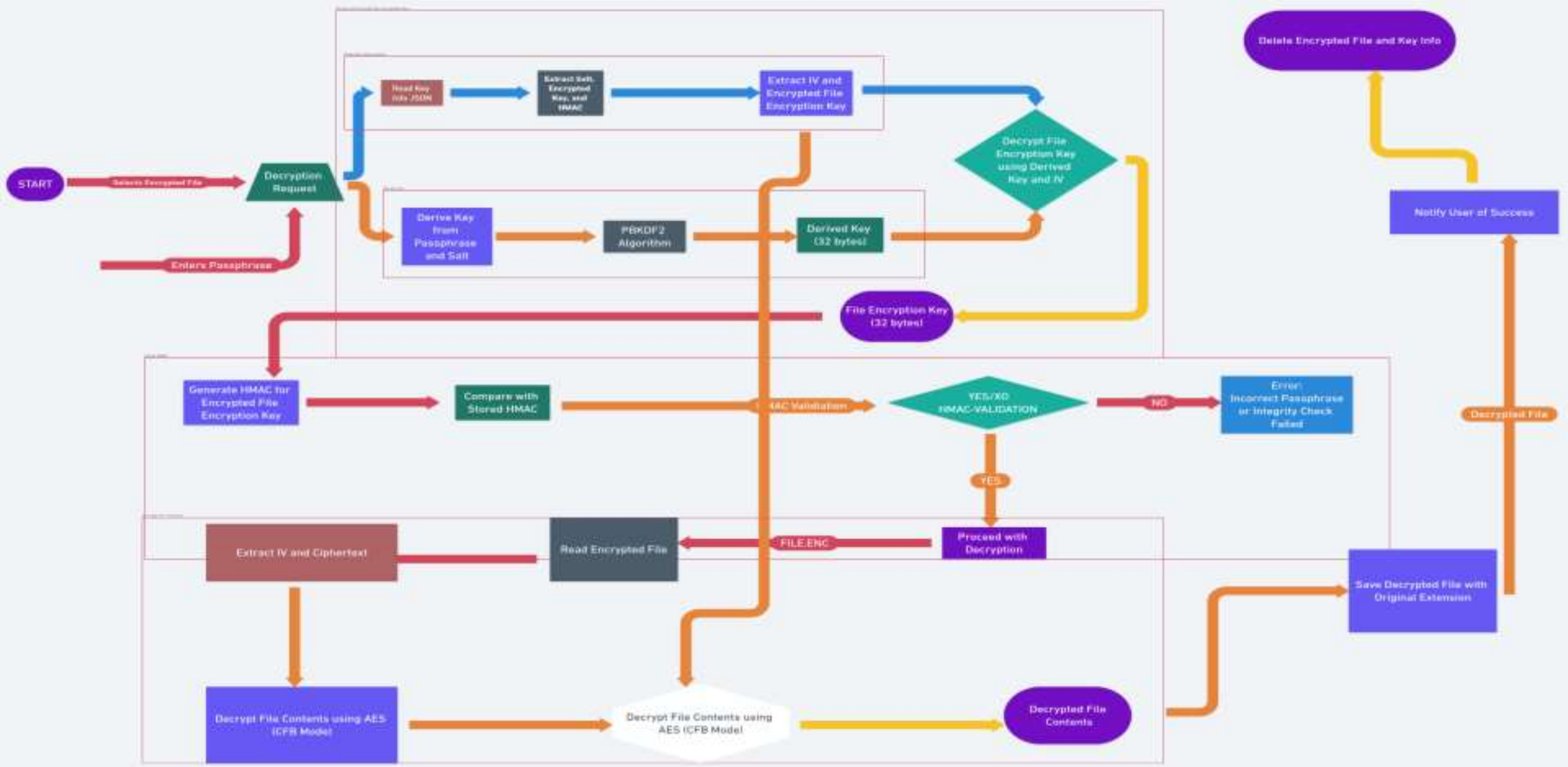

Encrypt File Contents
Read File Contents
Generate Random File Encryption Key (32 bytes)
AES Encryption (CFB Mode) with IV
Encrypted File Contents

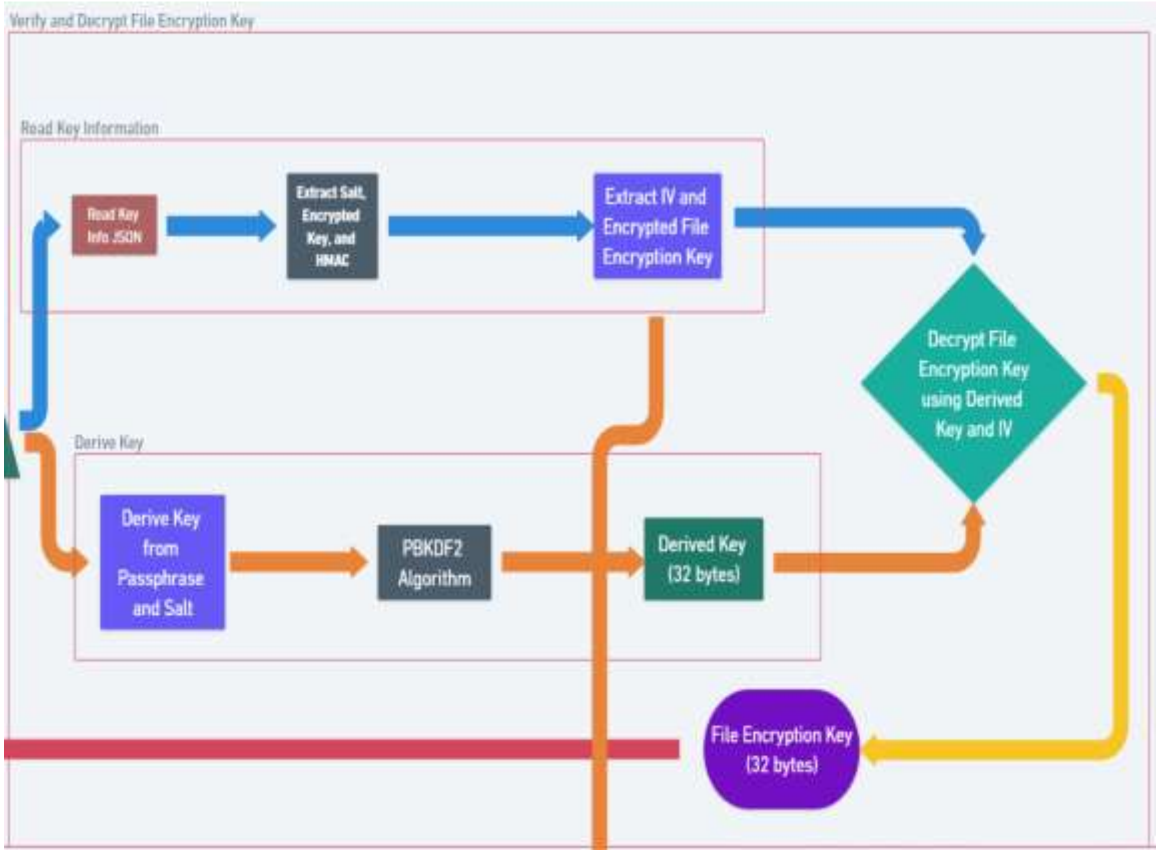
Encrypt file contents.

1. Read the file contents to be encrypted.
2. Generate a 32-byte random file encryption key.
3. Encrypt the file contents using AES-CFB mode with the random key and IV, resulting in the encrypted file contents.

REFERENCE LINK: CLEARLY REFER IMAGE CLICK HERE: | https://whimsical.com/83nA4x8jbAL2Kj5Dyyenqn

PROCESS FLOW DECRYPTION


START
Selects Encrypted File
Decryption Request
Enters Passphrase
Read Key Info JSON
Extract Salt, Encrypted Key, and HMAC
Extract IV and Encrypted File Encryption Key
Derive Key from Passphrase and Salt
PBKDF2 Algorithm
Derived Key (32 bytes)
Decrypt File Encryption Key using Derived Key and IV
File Encryption Key (32 bytes)
Generate HMAC for Encrypted File Encryption Key
Compare with Stored HMAC
HMAC Validation
YES/NO HMAC-VALIDATION
NO
Error: Incorrect Passphrase or Integrity Check Failed
YES
Proceed with Decryption
FILE.ENC
Read Encrypted File
Extract IV and Ciphertext
Decrypt File Contents using AES (CFB Mode)
Decrypt File Contents using AES (CFB Mode)
Decrypted File Contents
Save Decrypted File with Original Extension
Decrypted File
Notify User of Success
Delete Encrypted File and Key Info



Verify and Decrypt File Encryption Key
Read Key Information
Read Key Info JSON
Extract Salt, Encrypted Key, and HMAC
Extract IV and Encrypted File Encryption Key
Decrypt File Encryption Key using Derived Key and IV
Derive Key
Derive Key from Passphrase and Salt
PBKDF2 Algorithm
Derived Key (32 bytes)
File Encryption Key (32 bytes)

Read Key Information:

•Read key info JSON.

•Extract salt, encrypted key, and HMAC.

•Extract IV and encrypted file encryption key.

Derive Key:

•Derive key from passphrase and salt.

•Use PBKDF2 algorithm.

•Obtain derived key (32 bytes).

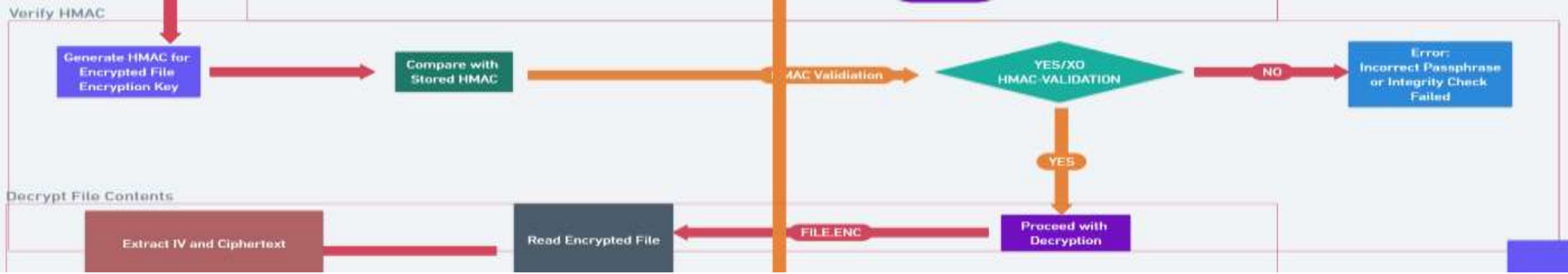
Decrypt File Encryption Key:

•Use derived key and IV.

•Decrypt file encryption key.

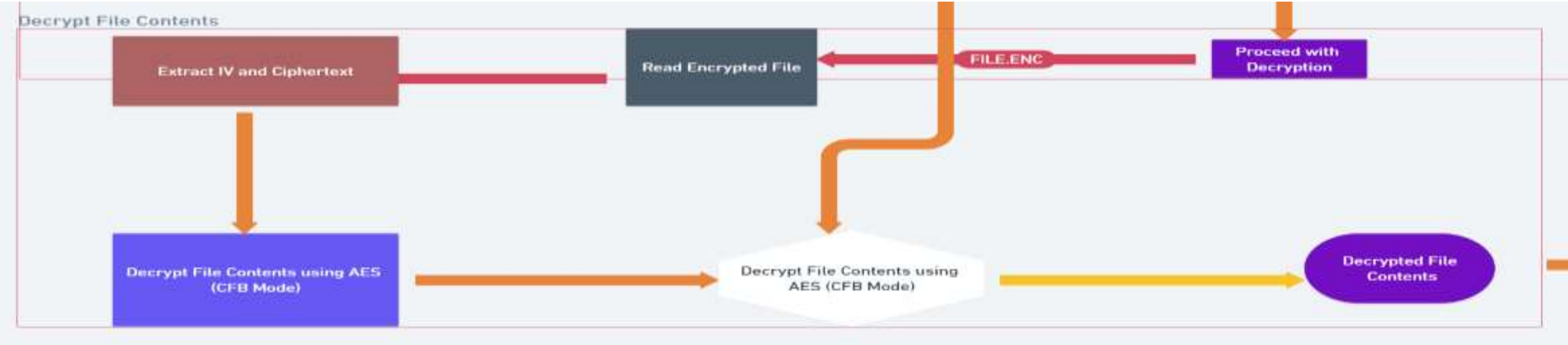
•Obtain file encryption key (32 bytes).

REFERENCE LINK: CLEARLY REFER IMAGE CLICK HERE | https://whimsical.com/83nA4x8jbAL2Kj5Dyyenqn



The process of verifying the HMAC for the encrypted file encryption key and decrypting the file contents.

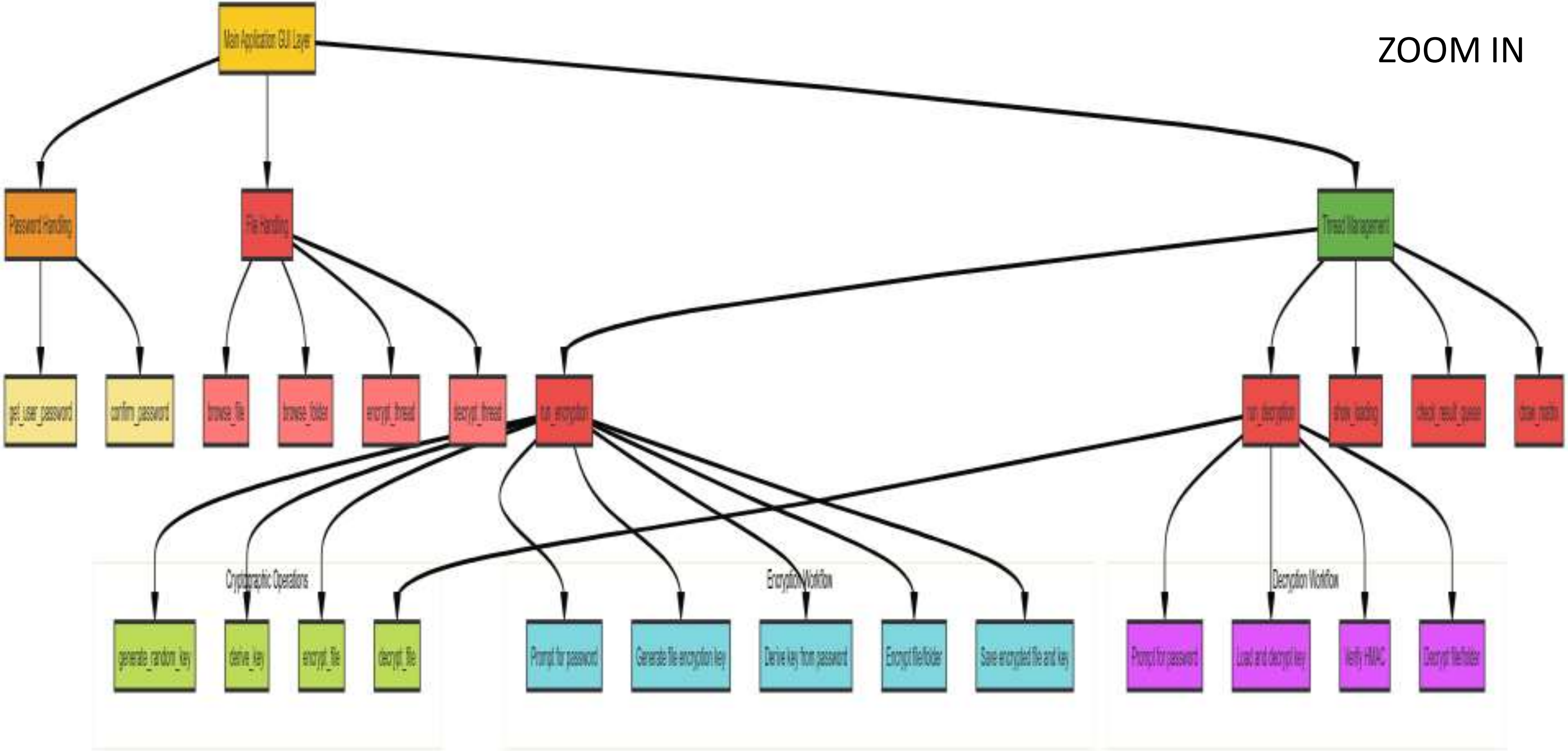
- The HMAC for the encrypted key is generated and compared with the stored HMAC.
- If they match, the decryption proceeds by extracting the IV and ciphertext from the encrypted file and decrypting the file using the file encryption key.


Decrypt File Contents
Extract IV and Ciphertext
Read Encrypted File
FILE.ENC
Proceed with Decryption
Decrypt File Contents using AES (CFB Mode)
Decrypt File Contents using AES (CFB Mode)
Decrypted File Contents

- The decryption process of file contents involves the initialization vector (IV) and ciphertext being extracted from the encrypted file,
- which is then read for decryption. Using AES in CFB mode, the file contents are decrypted, resulting in the original decrypted file contents.

Architecture Diagram

ZOOM IN



Technologies Used

Python Standard Libraries:

- **os**: Provides functions for interacting with the operating system, including file and directory manipulation.
- **json**: Handles reading from and writing to JSON files, which is used for storing encrypted key information.
- **shutil**: Provides high-level file operations like copying and removing directories.
- **time**: Used for creating delays.
- **threading**: Enables concurrent execution of functions using threads.
- **queue**: Implements a thread-safe queue to handle inter-thread communication.
- **random**: Generates random numbers and selections for drawing random characters in the Matrix effect.

Third-Party Libraries:

tkinter: The standard GUI library for Python is used to create the graphical user interface.

- **filedialog**: Opens file and folder selection dialogs.
- **simpledialog**: Prompts for user input.
- **ttk**: Provides themed widget sets for Tkinter.
- **messagebox**: Displays message boxes.

pycryptodome: A self-contained Python package of cryptographic primitives.

- **AES**: Used for symmetric encryption and decryption (AES-256 in CFB mode).
- **get_random_bytes**: Generates random bytes for initialization vectors (IVs) and keys.

- **PBKDF2**: A key derivation function to derive a secure encryption key from a password.
- **HMAC**: Provides hashing for data integrity checks.
- **SHA256**: A hashing algorithm used with HMAC for generating digests.

Key Functional Components:

1. **Password Handling**
2. **Key Management**
3. **File Encryption/Decryption**
4. **File and Directory Operations**
5. **Multithreading**
6. **Visual Effects**

Team Members and Contributions

PIYUSH KUMAR

Cryptography and Security Implementation

- ➢ Design and implement encryption and decryption algorithms using AES-256 in CFB mode.
- ➢ Develop key management functions, including key generation and derivation using PBKDF2 and HMAC for integrity checks.
- ➢ Ensure secure password handling and user authentication.
- ➢ **SKILLS:** Strong understanding of cryptography, Python programming, and security best practices.

ATHUKURI VENKATA SIVA SAI JAYANTH

GUI Development

- ➢ Design and implement the graphical user interface.
- ➢ Create and manage GUI components such as entry widgets, buttons, progress bars, and message boxes.
- ➢ Implement visual effects like the "Matrix" style display using the Canvas widget.
- ➢ **SKILLS:** Experience with GUI development, familiarity with ‘tkinter’ and user-friendly design.

RACHERLA HIMABINDU

File and Directory Operations, Multithreading

- ➢ Implement file and directory handling functions, including reading, writing, and deleting files securely.
- ➢ Develop recursive directory processing for encryption and decryption.
- ➢ Manage inter-thread communication and synchronization using queues.
- ➢ **SKILLS:** Proficiency in file system operations, experience with multithreading in Python, and problem-solving skills.

❑ Conclusion:

1. Cryptography - implementing robust encryption/decryption algorithms (with the Advanced Encryption Standard [AES] in CFB mode). Knowing the value of good key management, I implemented PBKDF2 to derive strong keys from passwords and used HMACs for data integrity checks.
2. The thing that resulted in this biggest increase in overall performance was implementing multithreading, as multi-threading allowed me to run the encryption and decryption tasks simultaneously, meaning my GUI would remain responsive. It showed me how to handle thread-safe operations and inter-thread communication (using queues).
3. File I/O: Worked with File operations to manually manipulate files and folders. This consisted of reading, writing, and securely deleting files, as well as managing directories in a recursive way.
4. Python Programming: This helped to strengthen my Python programming skills in standard libraries for system interactions and third parties because of more advanced functionalities.
5. User Authentication: Secure implementation of user authentication mechanisms, protecting sensitive operations and making sure to handle passwords securely (e.g., plaintext is never exposed in input).