**INTERNSHIP REPORT**

**Intel-Unnati**

**On**

**By**

**BARENKALA BHUVAN SUBRAMANYAM - BU21CSEN0400086**

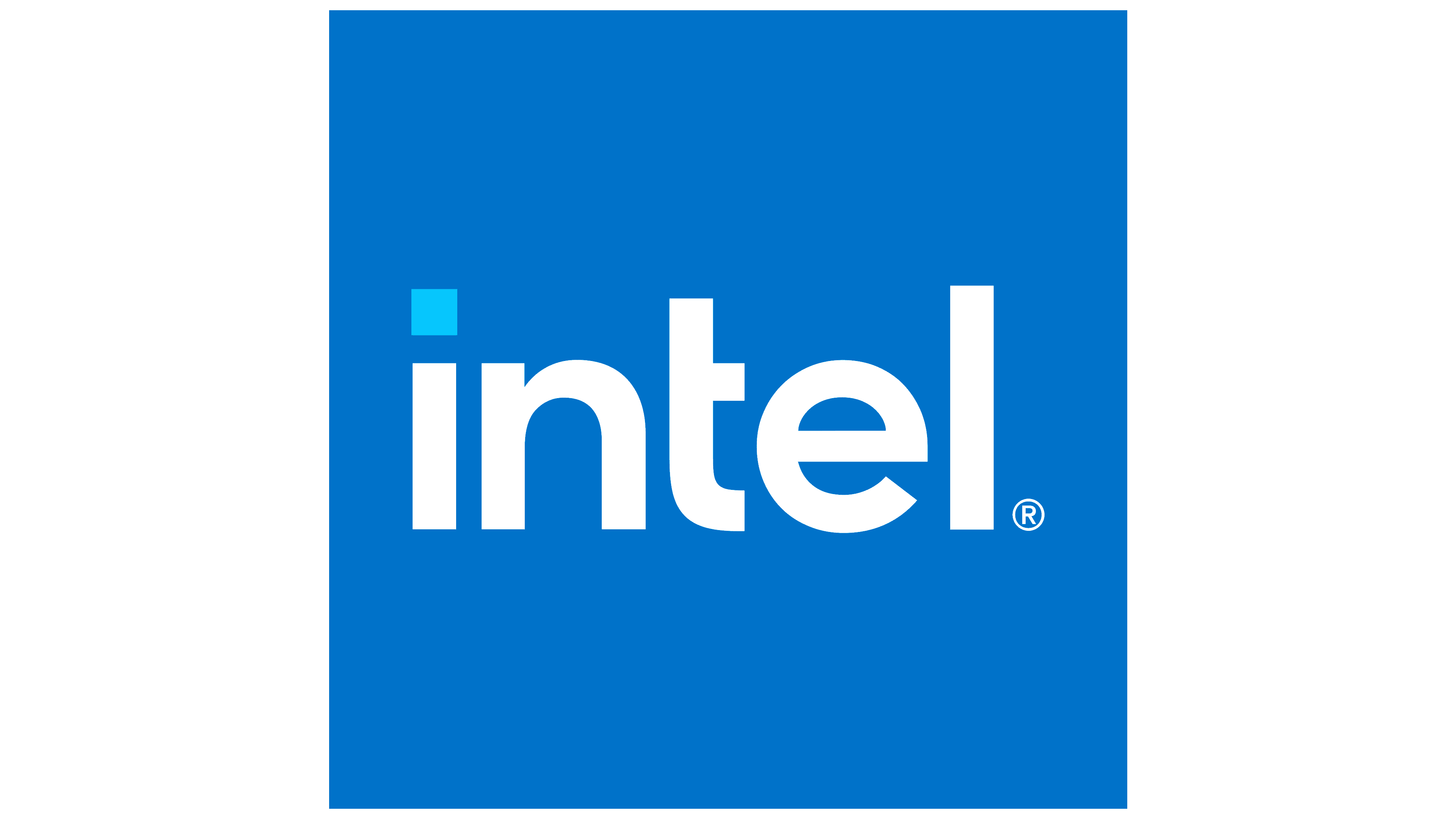
**ESWARAPRAMOD EDARA - BU21CSEN0400102**

**SATTARU VENKATA ABHILASH - BU21CSEN0400144**

**YUGANDHAR REDDY - BU21CSEN0400224**

**EEDIGA VISHNU - BU21CSEN0400099**

**(Duration: 14-05-2024 to 10-07-2024)**

**** 

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**Gandhi Institute of Technology and Management**

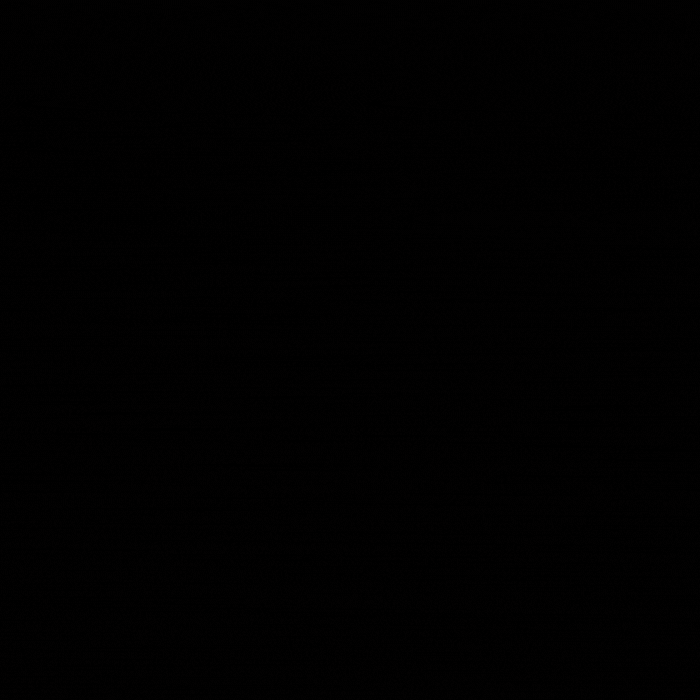
**(DEEMED TO BE A UNIVERSITY)**

**BENGALURU, KARNATAKA, INDIA**

**SESSION: 2021-2025**

**Abstract:**

This project presents an authorization application that securely encrypts and decrypts user-selected files and folders using AES-256 encryption. Each encryption operation generates a File Encryption Key (FEK), which is then encrypted with a password using a Key Derivation Function. This ensures that the data can only be accessed and decrypted by authorized individuals with the right passcode. The report discusses the application workflow, AES-256 encryption, KDF usage, the reasoning behind cryptographic decisions, an open-source library, system routine usage, and detailed testing scenarios. It gives access to the source code repository. This project illustrates a strong implementation of cryptographic standards that provide data security and secrecy.



**CONTENTS**

|  |  |
| --- | --- |
| S.No | Index |
| 1. | **Introduction** |
| 2. | **Problem Statement** |
| 3. | **Scope** |
| 4. | **Infrastructure Requirements** |
| 5. | **High-Level Features** |
| 6. | **Application workflow** |
| 7. | **High-Level Algorithm** |
| 8. | **Justification for Crypto Algorithms Used** |
| 9. | **Open Source and System Routines Used** |
| 10. | **Test Plan** |
| 11. | **Source Code** |
| 12. | **Learning Outcomes** |
| 12. | **Results** |
| 13. | **Conclusion** |
| 14. | **References** |

**Introduction:**

This project's goal is to provide an authorization application that secures user password keys at rest. The application will encrypt a user-specified file or directory with AES-256 encryption, protect the encryption key with a user passphrase, and guarantee that neither the passphrase nor the encryption key is saved in plain text.

**Problem Statement:**

Protecting User Password Keys at Rest (on the Disk)

**Scope:**

The scope of this project comprises the creation of a file encryption application with a user passphrase. The project needs knowledge of Linux file system operations, cryptography techniques, and Python system programming.

**Infrastructure Requirements**

**Hardware:**

* Any x86-based Desktop or Server with Linux

**Software:**

* Vs code
* Python
* Linux Operation System (Kali Linux)

**Requirements:**

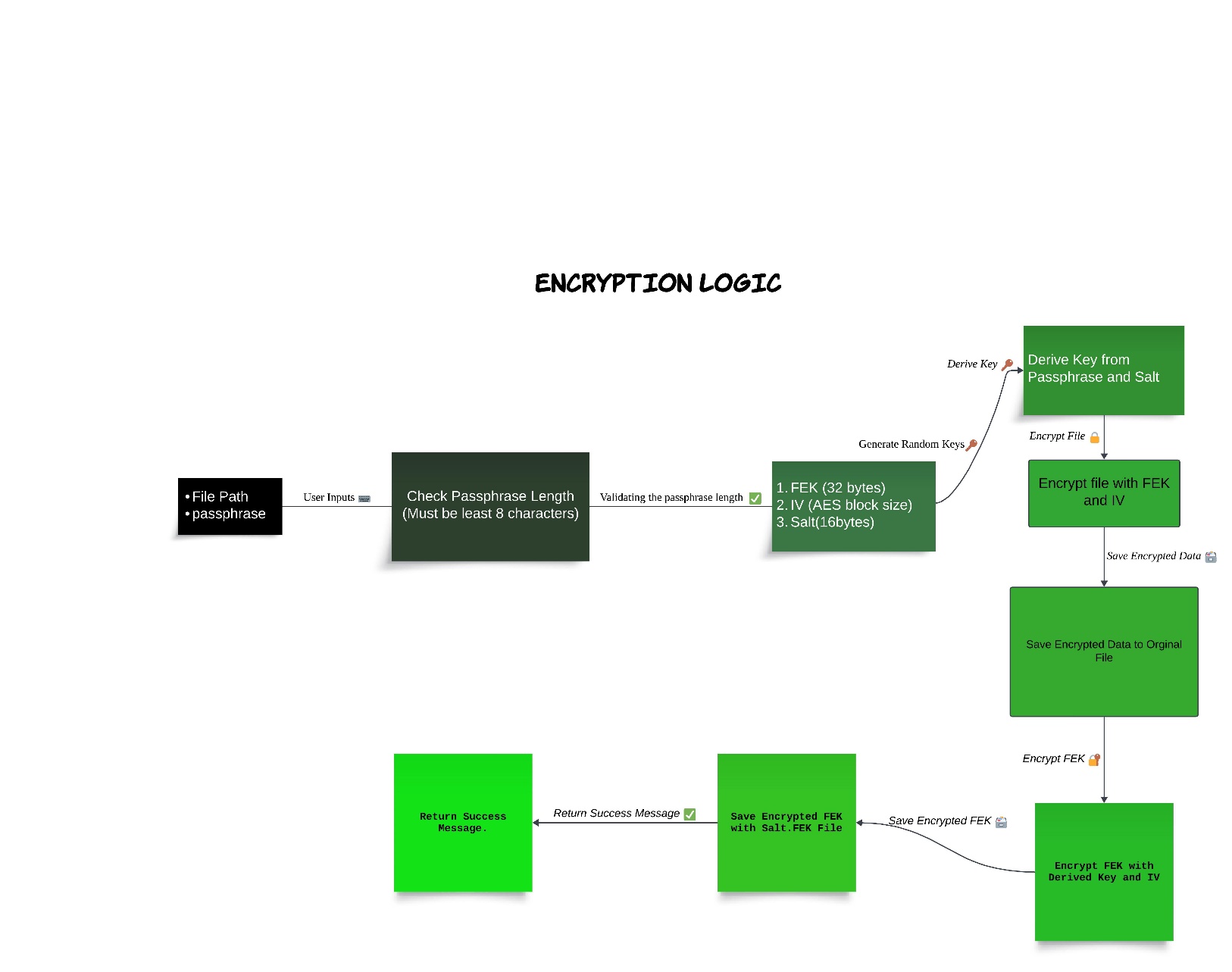
* cryptography
* pycryptodome
* utils
* customtkinter

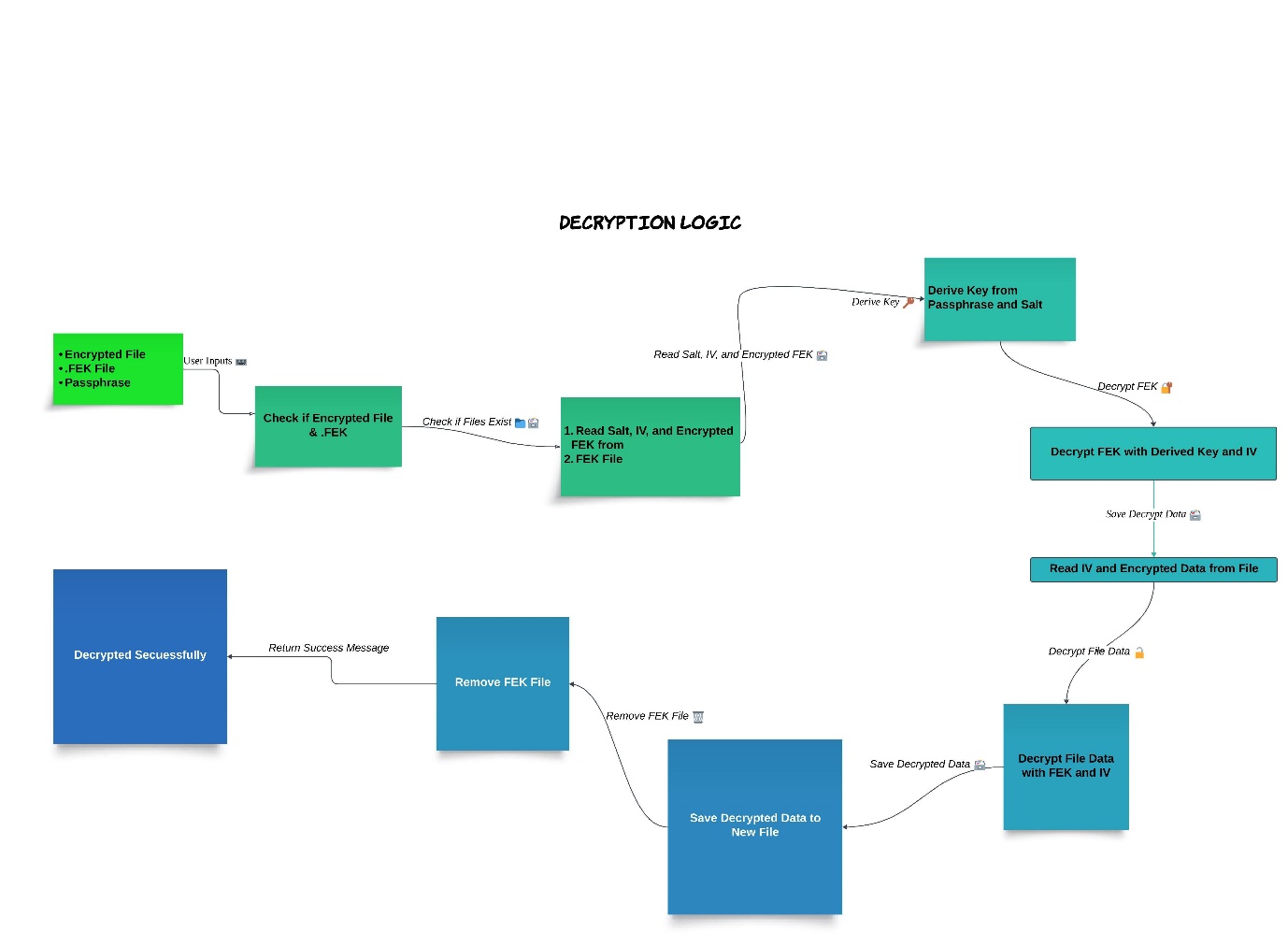
**High-Level Features**

1. Encrypt a user-chosen file or directory using AES-256 with a randomly generated File Encryption Key (FEK).
2. Store the FEK in a file protected by a user passphrase.
3. Ensure the user passphrase and FEK are not stored in plain text.
4. Authenticate the user passphrase to decrypt and retrieve the FEK, then use it to decrypt the file.

**Application Workflow**

1. **User Input:**
   * User selects a file or directory to encrypt.
   * User provides a passphrase.
2. **Encryption Process:**
   * Generate a random FEK.
   * Encrypt the file or directory using AES-256 and the FEK.
   * Use a Key Derivation Function (KDF) to generate a key from the user passphrase.
   * Encrypt the FEK using the derived key and store it.
3. **Decryption Process:**
   * User provides the passphrase.
   * Use the KDF to generate the key from the passphrase.
   * Decrypt the FEK using the derived key.
   * Decrypt the file or directory using the FEK.



****

**High-Level Algorithm**

**File Encryption:**

* Generate a random 256-bit FEK.
* Encrypt the file using AES-256 with the FEK.
* Derive a key from the user passphrase using a KDF (e.g., PBKDF2, bcrypt).
* Encrypt the FEK with the derived key.
* Store the encrypted FEK.

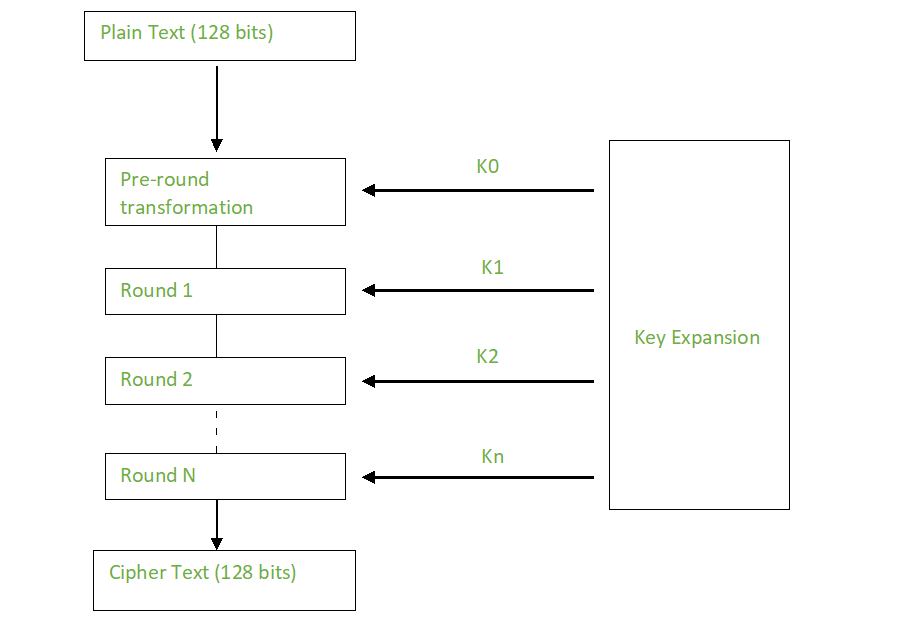
**File Decryption:**

* Derive a key from the user passphrase using the same KDF.
* Decrypt the stored FEK with the derived key.
* Decrypt the file using the decrypted FEK.

## Justification for Crypto Algorithms Used

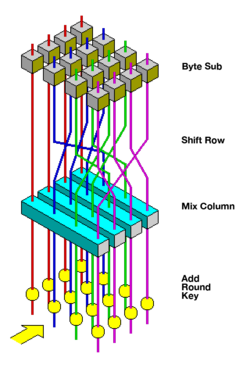
* **AES-256:** Advanced Encryption Standard (AES) is a highly trusted **encryption algorithm** used to secure data by converting it into an unreadable format without the proper key. Developed by the National Institute of Standards and Technology (NIST), **AES encryption** uses various **key lengths** (128, 192, or 256 bits) to provide strong protection against unauthorized access. This **data security** measure is efficient and widely implemented in securing **internet communication**, protecting **sensitive data**, and encrypting files. AES, a cornerstone of modern cryptography, is recognized globally for its ability to keep information safe from cyber threats.
* Chosen for its strong encryption standard, widely recognized and accepted for secure data encryption.
* 256-bit key – 14 rounds

### **Creation of Round Keys**



Plain Text (256 bits)

Cipher Text (256 bits)



* **KDF (e.g., PBKDF2, bcrypt):** Chosen to securely derive a key from the user passphrase, adding an extra layer of security by making it computationally expensive to guess the passphrase.

## Open Source and System Routines Used

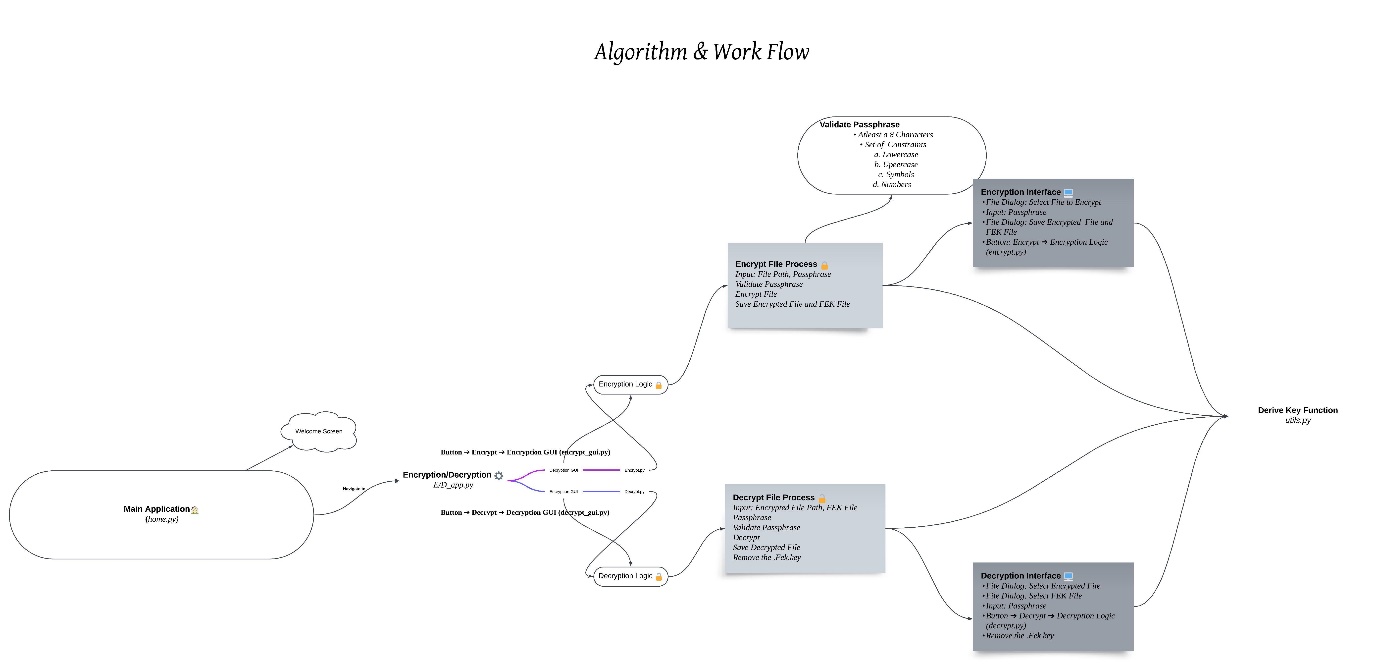
* **Python Cryptography Library (pycryptodome):** For AES-256 encryption and decryption.
* **Linux File System Operations:** For handling file input/output operations.
* **Key Derivation Function (KDF):** For generating secure keys from the user passphrase.

## Test Plan

### Test Cases

1. **Simple Case:**
   * Encrypt and decrypt a single small text file.
   * Verify the content matches the original file after decryption.
2. **Directory Case:**
   * Encrypt and decrypt a directory containing multiple files.
   * Verify the content and structure of the directory matches the original after decryption.
3. **Passphrase Authentication Failure:**
   * Attempt decryption with an incorrect passphrase.
   * Verify that decryption fails and no data is exposed.
4. **Edge Cases:**
   * Very large files.
   * Files with special characters in the names.
   * Empty files and directories.

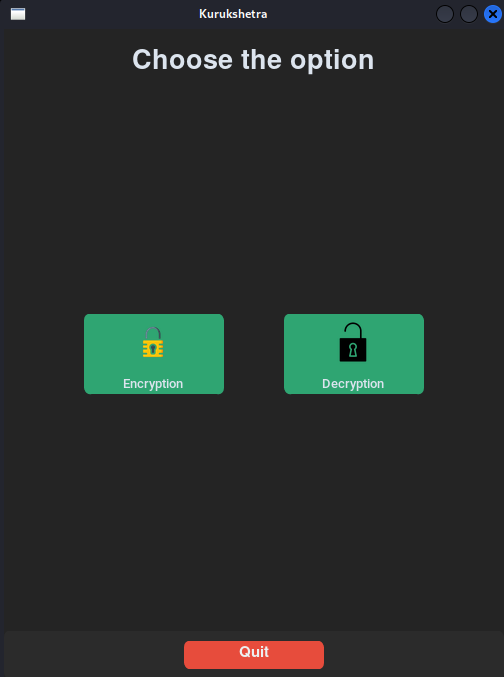
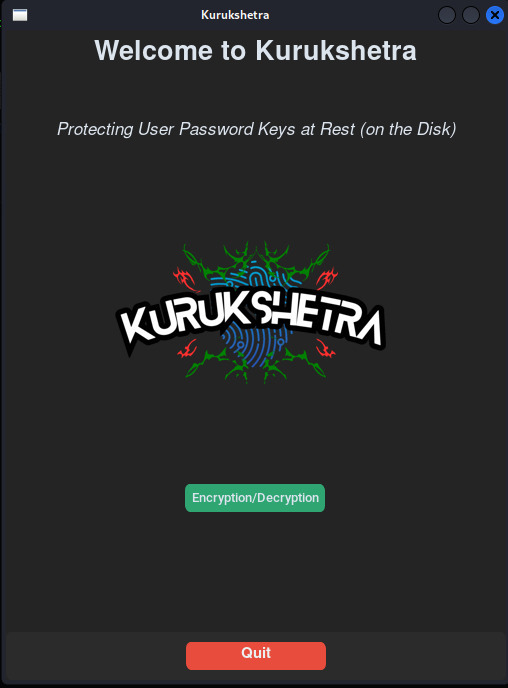
## Algorithm

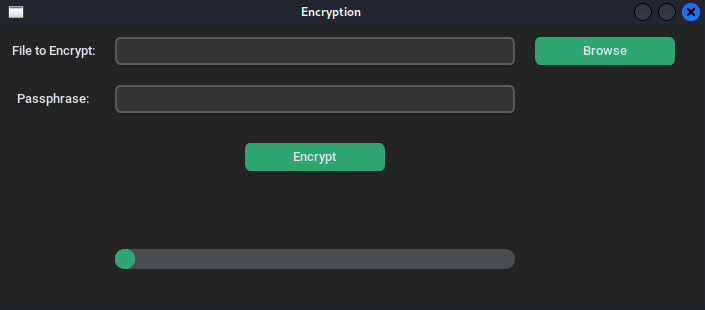


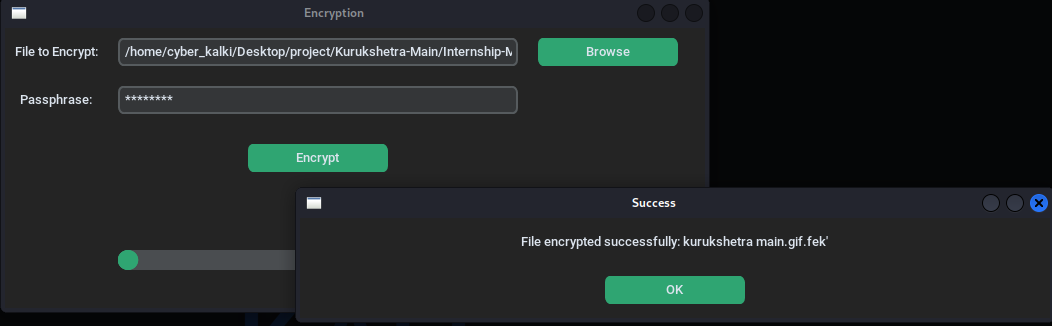
**Learning Outcomes**

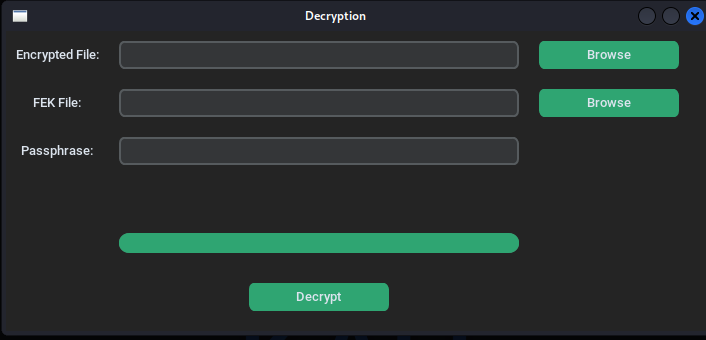
1. **Partitioning the High-Level Problem Statement:** Understanding how to break down a complex problem into smaller, manageable tasks and workflows.
2. **Understanding Crypto Algorithms:** Gaining practical knowledge of different cryptographic algorithms and their usage models in securing data.

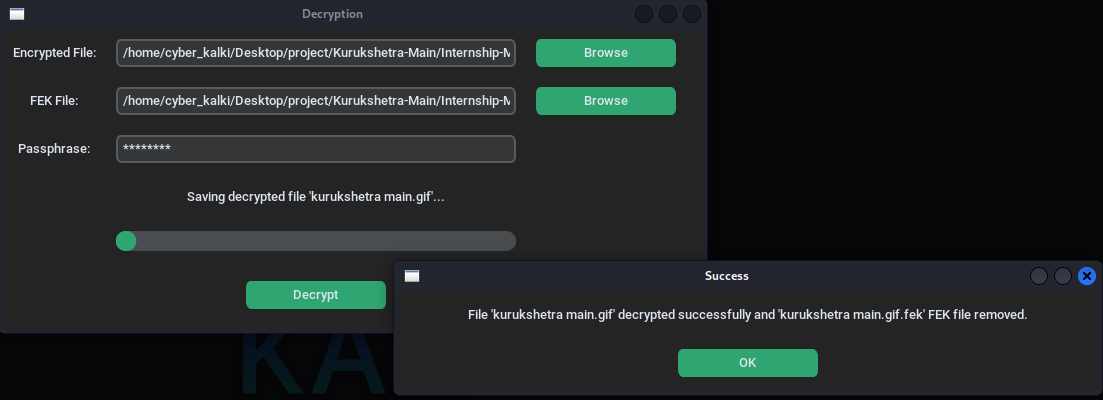
**Result**





****

****

****

**Demo**

****

**Conclusion**

This project demonstrates the development of an application to protect user password keys at rest using AES-256 encryption and secure passphrase-based key management. It highlights the importance of not storing sensitive information in plain text and provides a robust solution for securing data on disk.

**References**

<https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf>

<https://pycryptodome.readthedocs.io/en/latest/>

<https://refspecs.linuxfoundation.org/FHS_3.0/fhs-3.0.pdf>