**INTERNSHIP REPORT**

**Intel-Unnati**

**On**

**By**

**BARENKALA BHUVAN SUBRAMANYAM - BU21CSEN0400086**

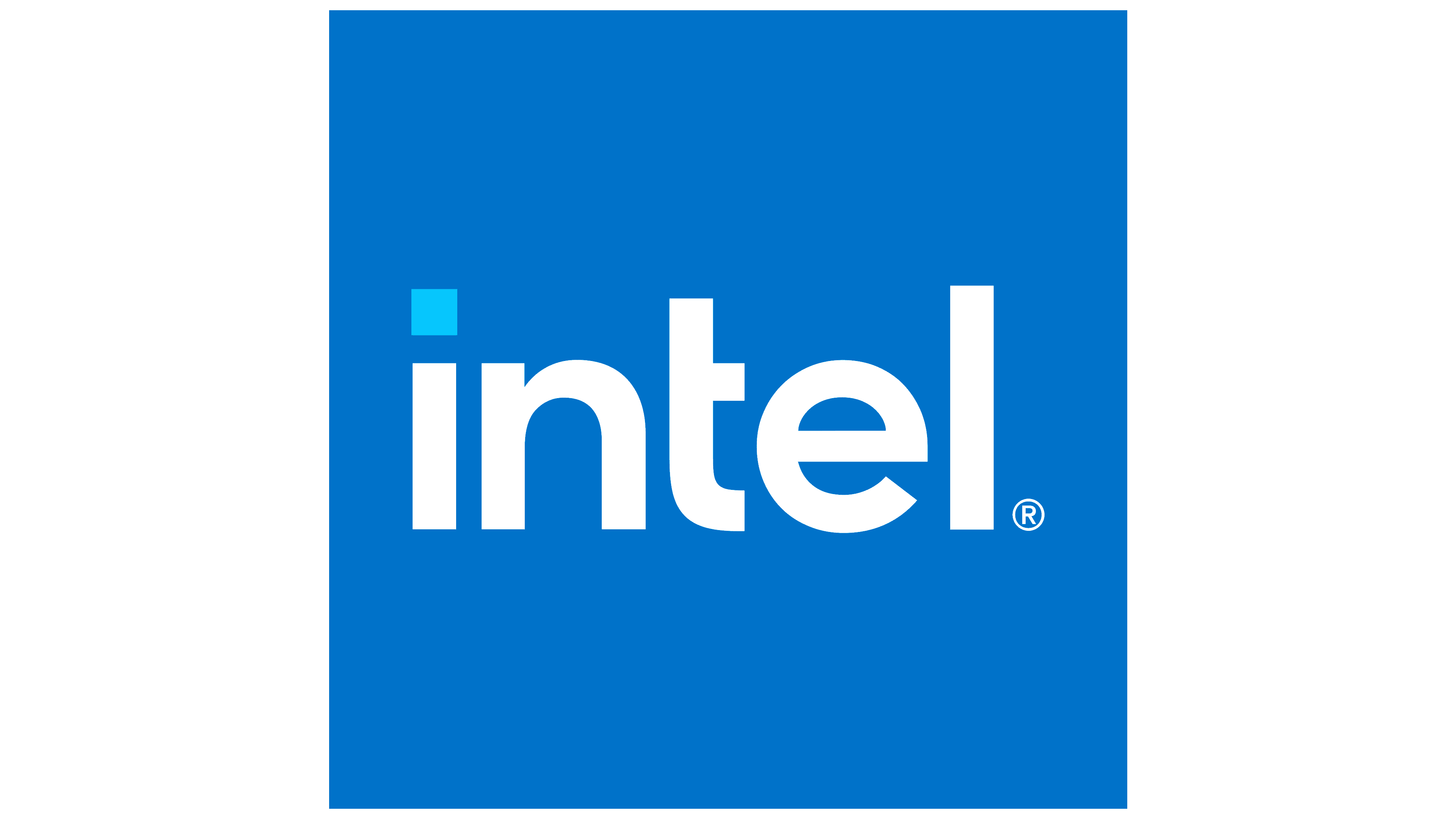
**ESWARAPRAMOD EDARA - BU21CSEN0400102**

**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**

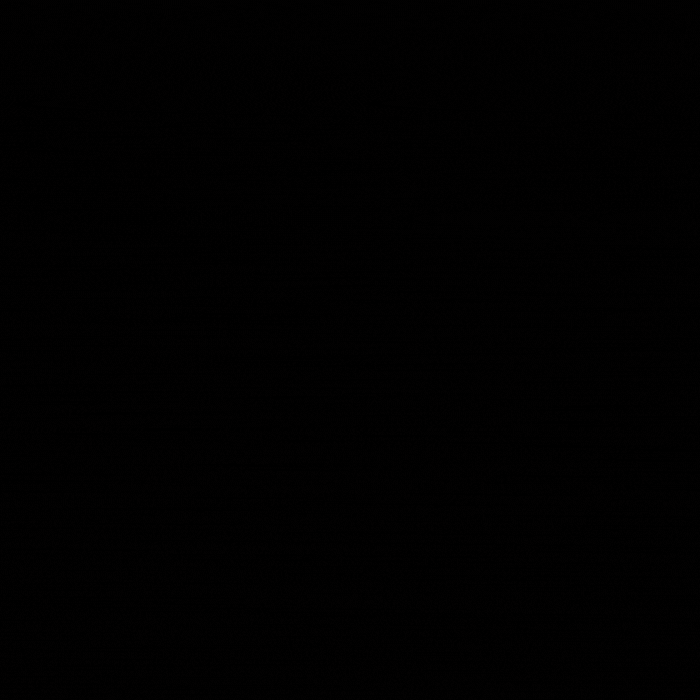
**(DEEMED TO BE A UNIVERSITY)**

**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.

**User Launches the Application:**

* With Kurukshetra.bat tap on Windows or Kurukshetra.sh on Linux.

**Main Screen:**

* Offers an ability to encrypt them or to decrypt, if needed.

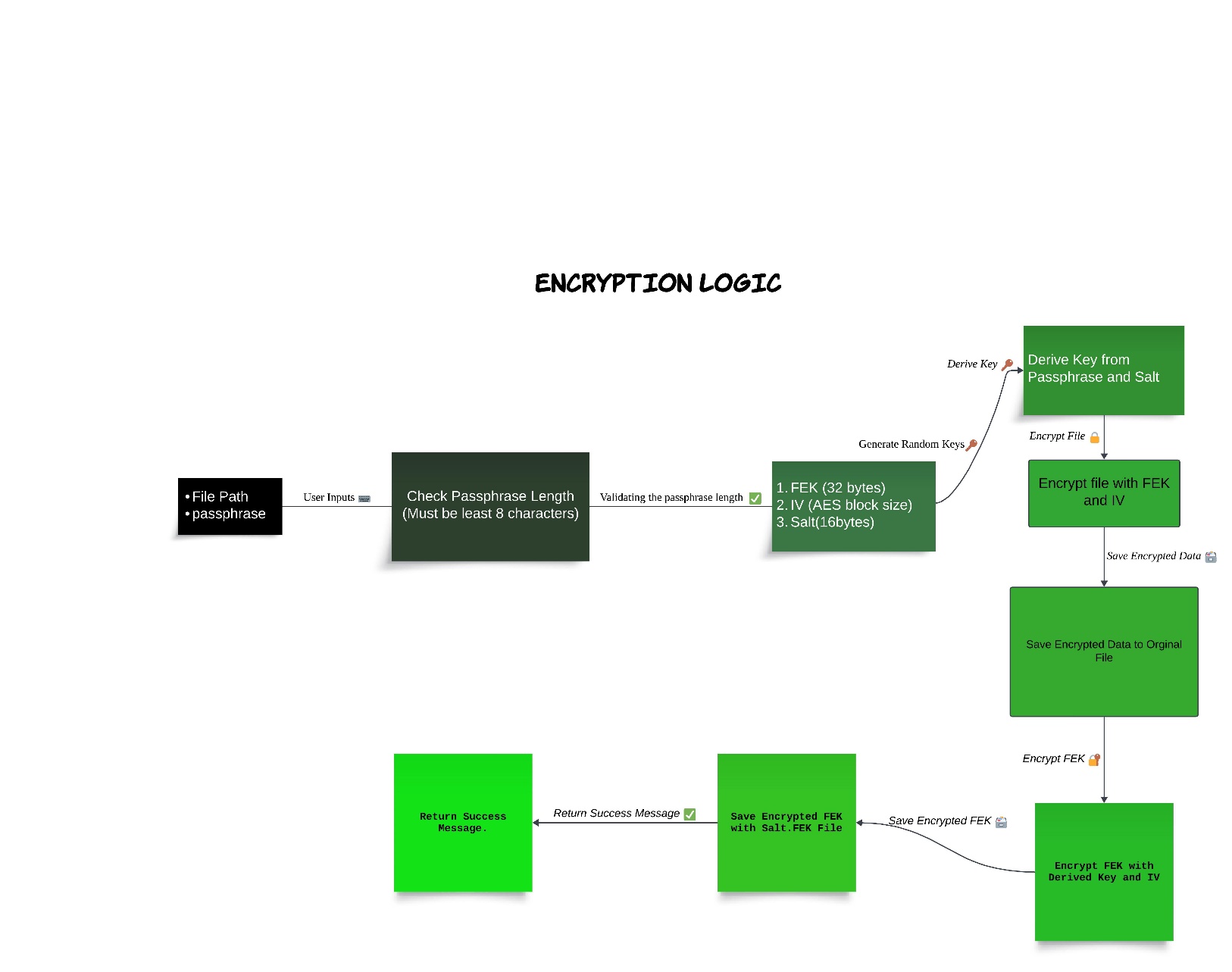
**Encryption Process:**

* User chooses a file, types a password and the particular file is locked.

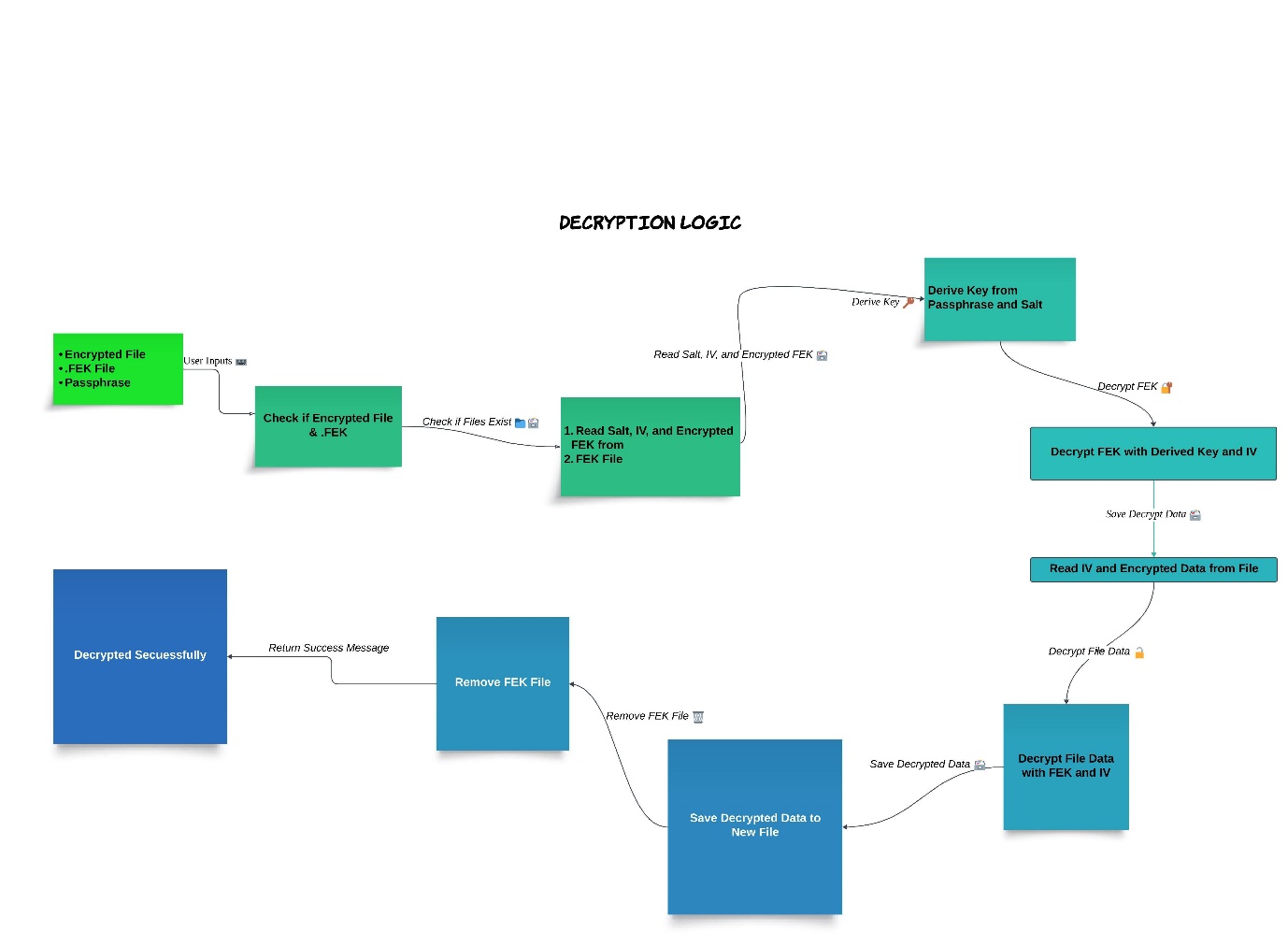
**Decryption Process:**

* User chooses an encrypted file chooses the correct password and the file is decrypted.

**Encryption logic**



**Decryption logic**

****

**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.

**Why AES-256?**

1. **Security:**

* Strength: AES-256 entails keying of 256 bits which makes the encryption much more secure than AES-128 or AES-192. Due to the 2^256 possible key combinations it can be stated that brute force attack is practically impossible.
* Cryptanalysis Resistance: AES-256 is immune to all known types of cryptographic attacks that can be practically applied. The algorithm has been studied and criticised thoroughly due to its designation as the standard for encryption by NIST (National Institute of Standards and Technology).

1. **Performance:**

* Efficiency: AES is extensively used secure and it is found efficient in both software as well in the hardware implementation. AES-256 though it requires a longer key size it is designed for better performance and has the capability of encrypting large amounts of data in a short span of time.
* Speed: The encryption and decryption processes in AES are fast so that they can be processed with little computational expenses to provide flexibility for several applications, such as the applications of real-time encryption.
* Standardization: Widespread Adoption: AES-256 is an international standard according to ISO/IEC 18033-3 encryption standard and is popular with many businesses, or industries such as finance, health care, and government.
* Compliance: AES-256 is the perfect fit for the mentioned legislation and doesn’t contradict other important international acts like GDPR, HIPAA, or FIPS 140-2 concerning the use of strong encryption for data protection.

1. **Symmetric Encryption:**

* Simplicity: Since AES-256 is a symmetric encryption algorithm the same key is used for encrypting or decrypting the information. This makes key management easier at the same time propelling the need for secure methods in exchanging the keys.
* Speed and Efficiency: AES is a symmetric encryption technique and it takes lesser time than asymmetric encryption techniques like RSA for encrypting big data.

1. **Flexibility:**

* Versatility: AES-256 can be applied to modes of operations (e. g. , CBC, GCM) that can also offer other features like; authentication (integrity check) and support for nonce based encryption in order to prevent replay attacks.
* Scalability: The proposed algorithms can be easily scaled up and can be run on various platforms varying from low power devices to high end servers.

**Key Derivation Function (KDF)**

**Security Enhancement:**

* Password Strengthening: KDFs enhance security by; Maps passwords into cryptographic keys securely in a way that if one tries to guess the original password or brute force the key, he or she is most likely not to guess it right.

**Salted Hashing:**

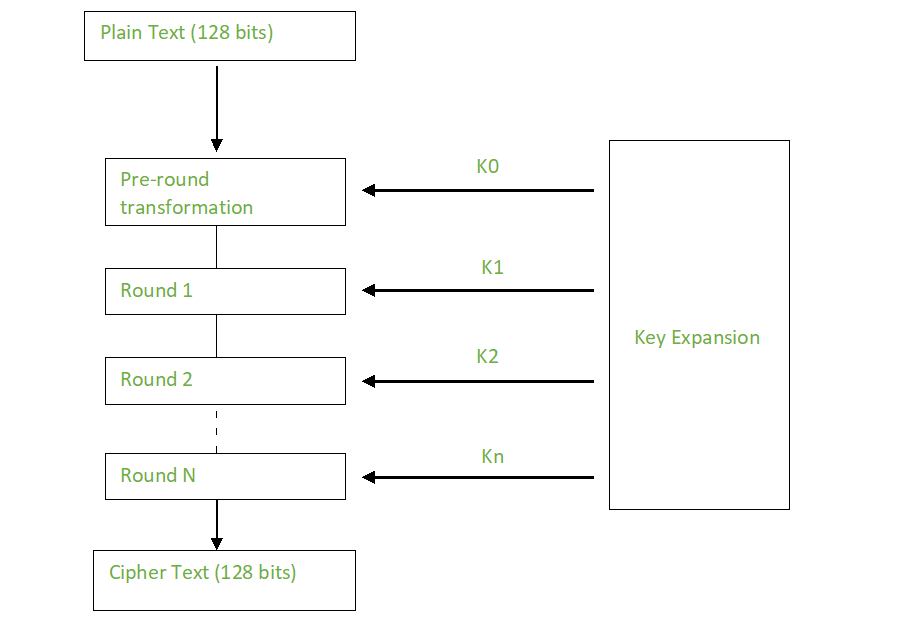
* Unique Keys: KDFs ensure that if one would use the same password for the encryption operation then the salt would ensure that the keys to be different for the purpose of avoiding showers to the rainbow table attacks.

**Justification for AES-256**

* Data Sensitivity: This project is expected to ensure that the users’ private files are secure, and AES-256 is adequately sufficient to protect the files from read and write access by unauthorized persons.
* Compliance: Through AES256 the project adheres to the set industry practices and national and international legislation concerning the use of data encryption in order to satisfy the necessity for the protection of the encrypted data.
* Performance: Therefore, owing to the extremely high rate of AES-256, the formation of the application indicates that it could indicate the possibility of the equalization of the roles of encryption and decryption with minimal impact on the performance of the system and the relief of the end user.
* Future-Proofing: AES-256 is particularly dependable for this purpose since it is future proofed and the actual encryption from the use of massive supercomputers and cryptanalysis.

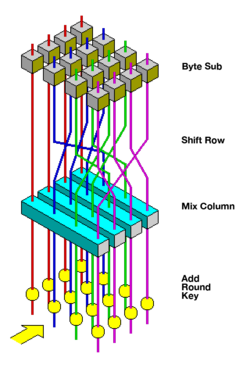
Thus, by integrating AES-256 encryption into the application to encrypt and decrypt files, the project employs a fast and secure encryption algorithm that can protect users’ information from modern threats and attacks.

### **Creation of Round Keys**



Cipher Text (256 bits)

Plain 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

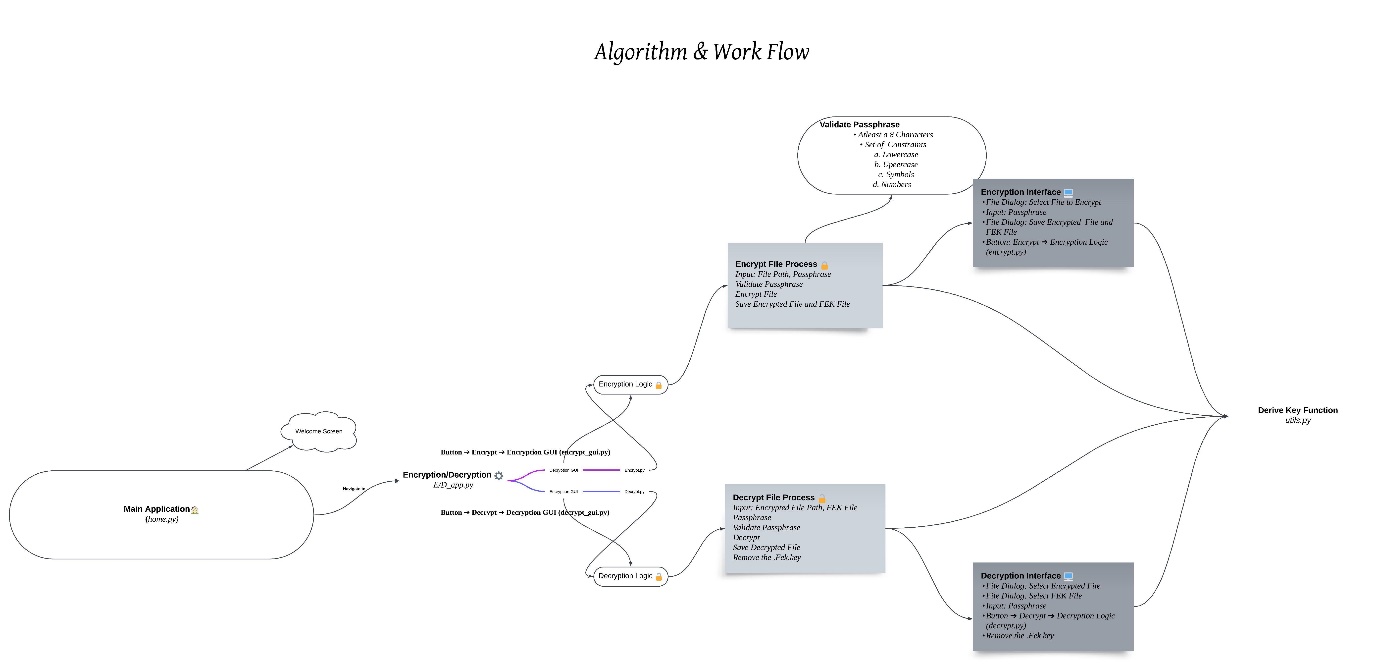
* **Python Cryptography Library (pycryptodome):** For AES-256 encryption and decryption.
* **Linux File System Operations:** For handling file input/output operations.
* **Key Derivation Function (KDF):** For generating secure keys from the user passphrase.

## 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.

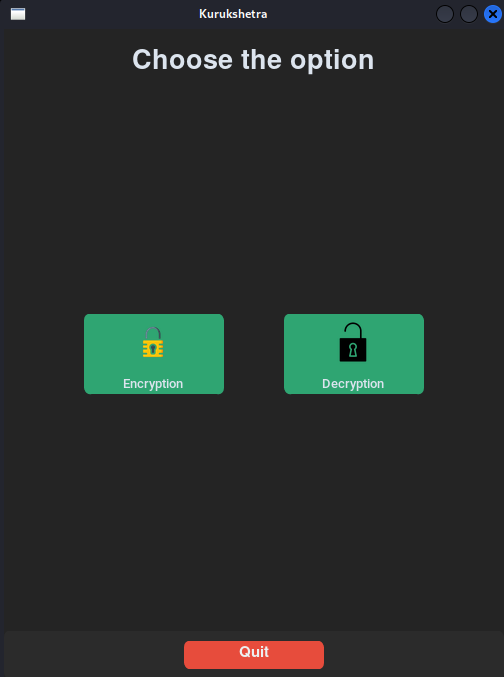
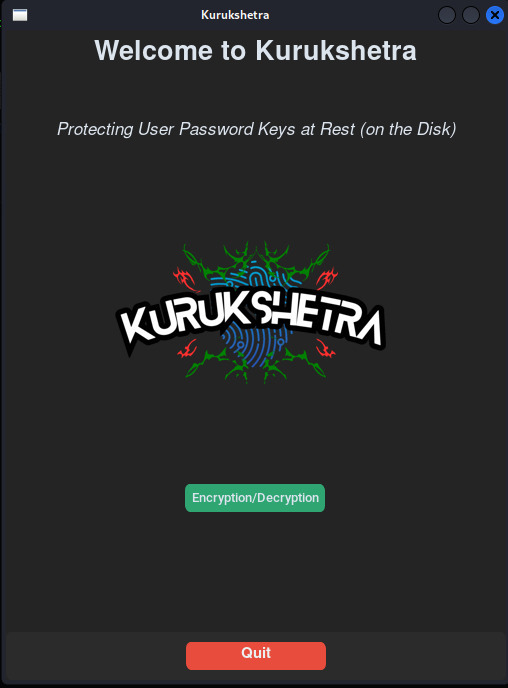
## Algorithm

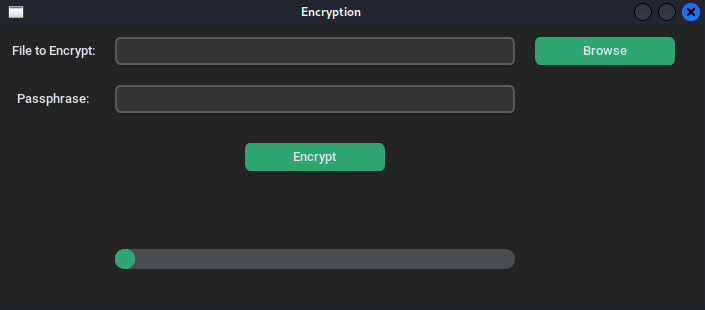


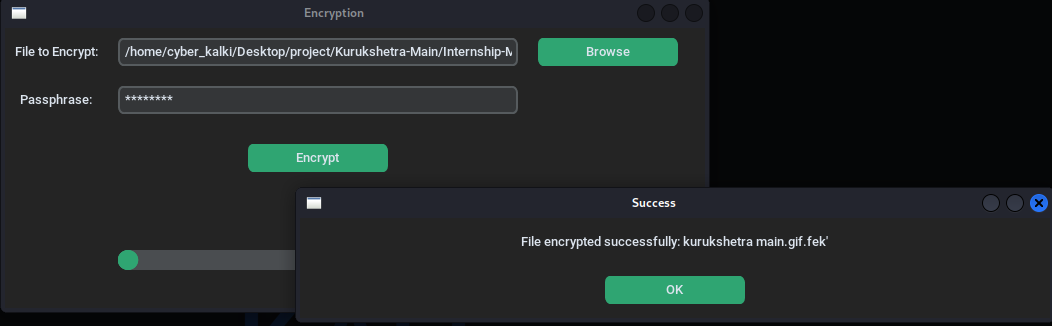
**Learning Outcomes**

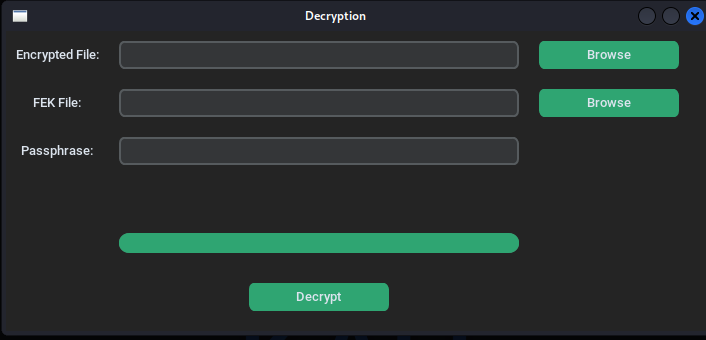
1. **Partitioning the High-Level Problem Statement:** Understanding how to break down a complex problem into smaller, manageable tasks and workflows.
2. **Understanding Crypto Algorithms:** Gaining practical knowledge of different cryptographic algorithms and their usage models in securing data.

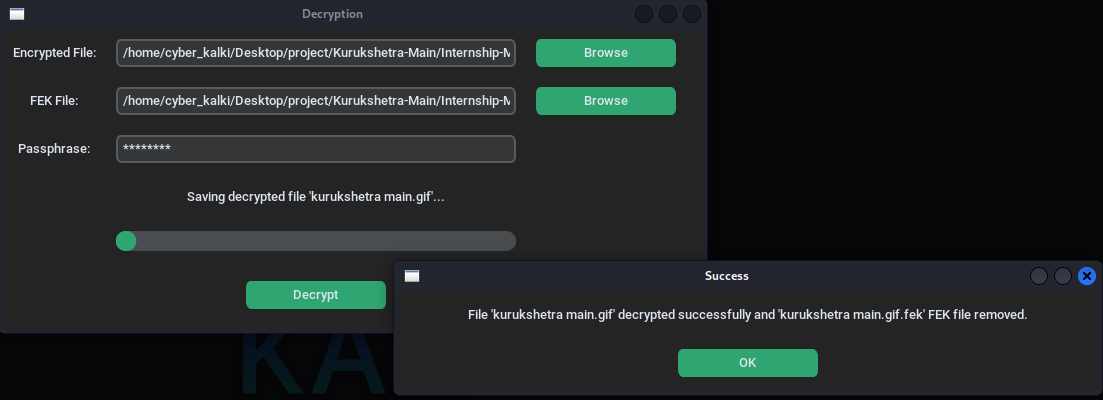
**Result**





****

****

****

**Demo**

****

**Conclusion**

This project well serves to show the implementation of a secure and GUI based application to encrypt and decrypt files using AES-256 encryption and passphrase controlled key management. This way, the application solves one of the most urgent tasks of a growing number of companies – data security when they are stored at rest, and become unavailable for unauthorized access. The focus of the project on such aspects as compliance, best practices, and scalability makes it possible to classify the result of the given work as the useful contribution to the further development of information security, protection of data stored on disks and popularization of absolute encryption methodologies.

**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>