File Encryption: Security that Matters

Submitted in partial fulfilment of the

Requirements for the award of the degree

Of

Bachelor of Technology

in

Computer Science & Engineering

By:

Divjot Kaur (032/CSE1/2023)

Har Ranjan Singh (040/CSE1/2023)

Harleen Kaur (042/CSE1/2023)

Harman Kaur (044/CSE1/2023)

Under the guidance of:

Dr. Jasleen Kaur Sethi

Department of Computer Science & Engineering

Guru Tegh Bahadur Institute of Technology

Guru Gobind Singh Indraprastha University

Dwarka, New Delhi

Year 2023-2024

DECLARATION

We hereby declare that all the work presented in this Minor Project Report for the partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in **Computer Science & Engineering**, Guru Tegh Bahadur Institute of Technology, affiliated to Guru Gobind Singh Indraprastha University, Delhi is an authentic record of our own work carried out under the guidance of **Dr. Jasleen Kaur**.

Date: 7/12/2023

Divjot Kaur (032/CSE1/2020)

Har Ranjan Pal Singh (040/CSE1/2020)

Harleen Kaur (042/CSE1/2020)

Harman Kaur (044/CSE1/2020)

CERTIFICATE

This is to certify that the project report entitled "**File Encryption: Security that Matters**", submitted by **Ms. Divjot Kaur, Mr. Har Ranjan Pal Singh, Ms. Harleen Kaur,** and **Ms. Harman Kaur** in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in **Computer Science & Engineering**, Guru Tegh Bahadur Institute of Technology, New Delhi is an authentic record of the candidate's own work carried out by them under our guidance. The matter embodied in this report is original and has not been submitted for the award of any other degree.

Jasleen Kaur
(Project Guide)

Aashish Bhardwaj
(Head of Department)
Computer Science & Engineering

Date: 7/12/2023

ACKNOWLEDGMENT

We would like to express our gratitude towards our supervisor **Dr. Jasleen Kaur** for her valuable guidance and frequent suggestions incorporated together with long hours of precious time to help us during this project and for making everything worthwhile and fruitful throughout the project. Without her guidance, we could not have presented this work up to the present standard. She has helped us surmount many hurdles encountered during the course of this project and has been a great source of inspiration.

We would also like to take this opportunity to thank the HOD of Computer Science Department **Dr. Aashish Bhardwaj** and other faculties who gave us support during this project or in other aspects of our study at Guru Tegh Bahadur Institute of Technology. Last but not the least we would like to thank our friends and family for supporting us and giving us the best of guidance.

Date: 7/12/2023

Divjot Kaur (032/CSE1/2020)
divjot18kaur@gmail.com

Har Ranjan Pal Singh (040/CSE1/2020)
ranjansingh271@gmail.com

Harleen Kaur (042/CSE1/2020)
harleenchandi@gmail.com

Harman Kaur (044/CSE1/2020)
harmankaurmalik@gmail.com

ABSTRACT

In response to the escalating reliance on digital data storage and the widespread proliferation of sensitive information, the imperative for implementing robust file security measures has reached unprecedented heights. This groundbreaking project introduces File Encryption, an intricately designed file folder encryption and decryption system that stands as a bulwark against unauthorized access to confidential data.

Employing cutting-edge encryption algorithms, File Encryption ensures the fortification of files within designated folders, thereby upholding both data integrity and confidentiality with unparalleled sophistication. Seamlessly integrating with prevailing file management workflows, the system boasts a remarkably user-friendly interface, simplifying encryption and decryption processes for users across various proficiency levels.

At its core, File Encryption leverages AES encryption algorithms to imbue files with an exceptional level of cryptographic strength, rendering sensitive data impervious to unauthorized decryption attempts. The interface's intuitiveness further streamlines the encryption and decryption procedures, minimizing disruption to existing file management practices while maximizing user convenience.

Offering users a spectrum of flexibility, the system empowers them to selectively encrypt or decrypt specific files or entire folders, enabling granular control over data protection. This feature proves particularly invaluable for prioritizing the security of critical files, aligning the system with diverse user needs.

A testament to its comprehensive approach, the system incorporates robust key management mechanisms, presenting users with options for password-based encryption or integration with external key management services. This meticulous approach ensures that only authorized individuals can gain access to encrypted files, reinforcing the system's formidable security posture.

Beyond mere encryption, the system meticulously maintains an audit trail of encryption and decryption activities, furnishing administrators with a comprehensive panorama of file security events. This feature not only enhances accountability but also facilitates compliance with stringent data protection regulations, positioning File Encryption as a stalwart guardian of data integrity.

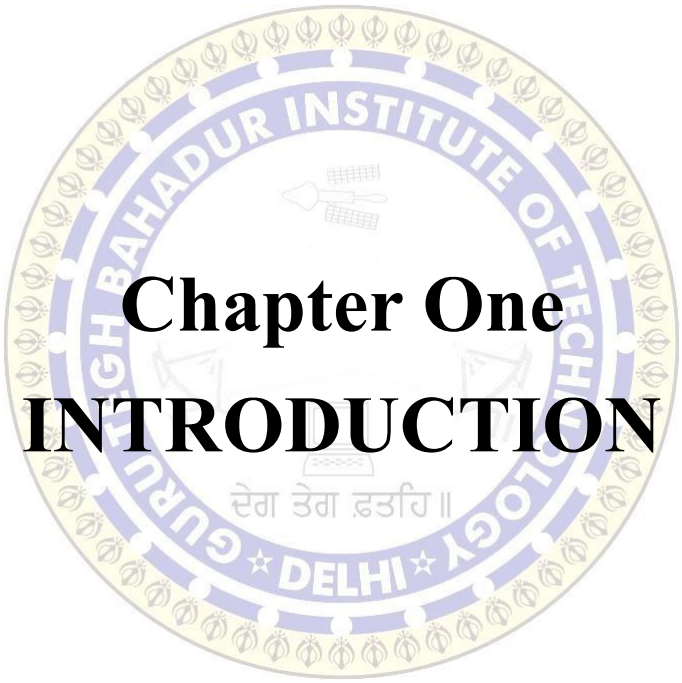
Designed with cross-platform compatibility in mind, File Encryption supports various operating systems, adapting seamlessly to diverse user environments and organizational infrastructures. This flexibility underscores the system's adaptability, making it a versatile choice for entities with heterogeneous computing landscapes.

In conclusion, File Encryption emerges as a comprehensive solution tailored for both organizations and individuals seeking a reliable, efficient, and user-friendly file folder encryption and decryption system. By harmoniously blending advanced encryption technologies with an intuitively crafted design, this project adeptly addresses the burgeoning need for robust data protection in the dynamic landscape of today's digital realm.

LIST OF TABLES

LIST OF FIGURES

CONTENTS



Chapter One
INTRODUCTION

INTRODUCTION

With the increasing reliance on digital data storage and the proliferation of sensitive information, the need for robust file security measures has become paramount. This project introduces File Encryption, an advanced file folder encryption and decryption system designed to safeguard confidential data from unauthorized access.

File Encryption employs state-of-the-art encryption algorithms to protect files within designated folders, ensuring data integrity and confidentiality. The system integrates seamlessly with existing file management workflows, providing a user-friendly interface for encryption and decryption processes.

It leverages AES encryption algorithms to secure files with a high level of cryptographic strength, ensuring that sensitive data remains confidential and resistant to unauthorized decryption attempts.

The system offers an intuitive and user-friendly interface, allowing users to easily encrypt and decrypt files within specified folders. The integration is designed to minimize disruption to existing file management practices.

It provides users with the flexibility to selectively encrypt or decrypt specific files or entire folders, allowing for granular control over data protection. This feature is particularly useful for prioritizing the security of critical files.

The system incorporates robust key management mechanisms, including options for password-based encryption or integration with external key management services. This ensures that only authorized users can access encrypted files.

It maintains an audit trail of encryption and decryption activities, providing administrators with a comprehensive view of file security events. This feature enhances accountability and facilitates compliance with data protection regulations.

The system is designed to be cross-platform, supporting various operating systems to accommodate diverse user environments. This flexibility enables seamless integration into different organizational infrastructures.

File Encryption is a comprehensive solution for organizations and individuals seeking a reliable, efficient, and user-friendly file folder encryption and decryption system. By combining advanced encryption technologies with intuitive design, this project addresses the growing need for robust data protection in today's digital landscape.

1.1 Objective

The primary goal of the File Encryption project is to develop a robust and versatile file encryption system that ensures the secure and efficient encryption of both individual files and entire folders. This project aims to provide users with a reliable tool for safeguarding sensitive data, employing advanced encryption algorithms to uphold confidentiality, integrity, and user-friendly functionality. The overarching objective is to offer a comprehensive solution that seamlessly integrates into existing workflows, minimizes disruption, and addresses the escalating need for heightened data protection in today's digital landscape.

The process of encryption and decryption in a file encryption system typically involves several steps and relies on cryptographic algorithms to secure and protect data. Here is an overview of the general steps involved in both encryption and decryption:

Encryption:

1. **Key Generation:**
    - A cryptographic key is generated using a secure key generation algorithm. This key is a crucial component in the encryption process and is used to transform the plaintext data into ciphertext.
2. **Plaintext Input:**
    - The user provides the plaintext data, which is the original and readable form of the information that needs to be protected.
3. **Encryption Algorithm:**
    - The encryption algorithm takes the plaintext data and the cryptographic key as input and performs a series of mathematical operations to transform the data into ciphertext. The ciphertext is the encrypted and unreadable form of the information.

4. **Ciphertext Output:**

    - The encrypted data (ciphertext) is produced and can now be stored or transmitted securely. Without the corresponding key, it should be challenging for unauthorized users to decipher and understand the information.

Decryption:

1. **Key Input:**

    - The user, or a system, provides the cryptographic key used during the encryption process. This key is essential for decrypting the data.

2. **Ciphertext Input:**

    - The encrypted data (ciphertext) is input into the decryption algorithm along with the cryptographic key.

3. **Decryption Algorithm:**

    - The decryption algorithm processes the ciphertext and the key to reverse the encryption process, transforming the ciphertext back into the original plaintext.

4. **Plaintext Output:**

    - The decrypted data (plaintext) is produced, and it is now in its original, readable form. Authorized users can access and utilize the information.

Key Management:

- **Secure Storage and Distribution:**

    - The cryptographic key must be securely stored and managed to prevent unauthorized access. In some systems, key management includes secure distribution of keys to authorized users.

1.2 Methodology

An easy-to-use python-based tool has been built to help the users to have a successful communication. Users need to integrate this module in their conversation technologies access all the features of the tool. For this project we followed a methodology that can

be applied to almost all the projects to achieve the project's objectives and goals. Anybody can build a tool, but it takes a specialized routine that needs to be followed in order to make it successful. The routine consisted of various phases which are briefly described as follows:

A.) Analysis:

The initiation of a project begins when a problem or opportunity is identified. The problem is then thoroughly researched, and needs are analysed including technology options, feasibility and appropriateness. All needs are approved, and a concept is devised which is then documented in the planning phase. This phase presents findings that include recommended features, order flow and guidance on overall tool structure. We have analysed all the needs of this project based on the problem and keeping in mind the security goals, File Encryption has been designed.

B.) Requirement Analysis Phase

User requirements are formally defined in terms of data, system performance, security, and maintainability requirements for the system. All requirements are defined to a level of detail sufficient for systems design to proceed. All requirements need to be measurable, testable, and relatable to the opportunity identified in the Analysis Phase. This phase identifies the type of files and folders that would use the tool. All the system requirements are mentioned in the SRS document. Based on the requirement analysis, the architecture is planned. The SRS document is presented in the next chapter.

C.) Planning Phase

The concept is further developed to describe how the business will operate once the approved system is implemented, and to assess how the system will impact the users. To ensure the products and /or services provide the required capability on-time and within budget, project resources, activities, schedules, tools, and reviews are defined. Additionally, security certification and accreditation activities begin with the identification of system security requirements and the completion of a high level vulnerability assessment. This stage gives the details needed to make informed recommendations for the design and development of the tool to secure the data. Whole of the problem-solving process was planned and then the design phase was brought into execution. The various tools & technologies used throughout, is a Linux based

machine (for better solutions), python libraries and encryption algorithms. The system execution process was planned. All the small details as in when will the mails & messages be sent were decided. The chat methodology, feedback concept was all planned. The journey of the user on the app was finalized.

D.) Design Phase

The physical characteristics of the system are designed during this phase. The operating environment is established, major subsystems and their inputs and outputs are defined, and processes are allocated to resources. The physical characteristics of the system are specified, and a detailed design is prepared. This phase is very important as the complete working structure is delivered that'll dictate how the site works. Also, if there are any changes to be made, this is the best time to make them, and not during the development stage.

E.) Development Phase

The detailed specifications produced during the design phase are translated into hardware, communications, and executable software. Software shall be unit tested, integrated, and retested in a systematic manner. Hardware is assembled and tested. This is where the product vision is brought to life.

The different Linux OS Types were tried and tested for the results. Ended on Kali to the best of their ability. The types of files/folders at first stage were designed. All the modules were designed and coded individually. Various Encryption/Decryption algorithms were linked and the concept was implemented. The dashboard was designed, the mathematical concepts involved in various dashboard functionalities were all finalized and integrated.

The tool process was all implemented at this stage. It consisted of two steps.

Step 1: Individual Module Construction

1. **User Interface (UI) Module:**
    - **Objective:** Create an intuitive and user-friendly interface for interacting with the encryption/decryption tool.
    - **Components:**

- File selection interface
- Encryption/decryption settings and options
- Progress indicators and status updates
- Notifications and alerts

2. **Encryption Module:**
    - **Objective:** Implement the core functionality of encrypting files and folders securely.
    - **Components:**
        - Encryption algorithm implementation (e.g., AES)
        - Key generation and management
        - File and folder parsing
        - Encryption process control flow
3. **Decryption Module:**
    - **Objective:** Develop the functionality to decrypt files and folders securely.
    - **Components:**
        - Decryption algorithm implementation
        - Key input and authentication
        - File and folder parsing
        - Decryption process control flow
4. **Key Management Module:**
    - **Objective:** Handle the generation, storage, distribution, rotation, and recovery of cryptographic keys.
    - **Components:**
        - Key generation algorithms

- Secure key storage mechanisms
- Key distribution protocols
- Key rotation and recovery processes

5. **User Authentication Module:**
    - **Objective:** Ensure secure user authentication to control access to encryption/decryption capabilities.
    - **Components:**
        - User authentication methods (passwords, multi-factor authentication)
        - Access control mechanisms
        - User roles and permissions
6. **File and Folder Selection Module:**
    - **Objective:** Facilitate the selection of files and folders for encryption or decryption.
    - **Components:**
        - File and folder navigation interface
        - Selection controls (checkboxes, drag-and-drop functionality)
        - Validation for supported file types and sizes

Step 2: Integration and Beautification

1. **File and Folder Selection Integration:**
    - Integrate the file and folder selection module with the encryption and decryption processes.
    - Ensure a smooth and intuitive user experience when selecting files and folders.
2. **User Interface (UI) Enhancements:**
    - Optimize the UI for clarity and ease of use.

- Ensure a consistent design language and visual hierarchy throughout the tool.

3. **Visual Feedback:**
    - Provide visual cues and feedback during encryption/decryption processes to keep users informed.
    - Use progress indicators, success messages, and error notifications.

The various functionalities developed in this phase are listed below:

1. File Encryption

I. User Authentication:
   a. Objective: Authenticate users to ensure that only authorized individuals can perform encryption.
   b. Implementation: Implement user authentication mechanisms (e.g., password, biometrics). Enforce access control to restrict unauthorized access.

II. File Selection:
   a. Objective: Allow users to choose the files or folders they want to encrypt.
   b. Implementation: Provide a user interface for file/folder selection, supporting multiple selections. Validate selected files to ensure they are accessible and compatible.

III. Encryption Algorithm:
   a. Objective: Implement a robust encryption algorithm to transform plaintext into ciphertext securely.
   b. Implementation: Choose a strong encryption algorithm (e.g., AES) and integrate it into the tool. Set parameters such as key size and mode of operation based on security requirements.

IV. Key Management:
   a. Objective: Generate, handle, and securely manage cryptographic keys used in the encryption process.
   b. Implementation: Implement key generation algorithms. Integrate key storage, distribution, rotation, and recovery mechanisms.

V. Encryption Process:
    a. Objective: Execute the encryption process, transforming selected files into encrypted form.
    b. Implementation: Integrate the chosen encryption algorithm with the selected files. Manage the encryption process flow, considering user feedback and progress indicators.

VI. Ciphertext Output:
    a. Objective: Generate encrypted files (ciphertext) for storage or transmission.
    b. Implementation: Save the encrypted files in the specified location or provide options for users to choose a destination.

VII. Notification and Feedback:
    a. Objective: Keep users informed about the status of the encryption process.
    b. Implementation: Display clear notifications, success messages, or error alerts. Provide feedback on completion and prompt users to take necessary actions.

2. File Decryption

I. User Authentication:
    a. Objective: Authenticate users to verify their identity before allowing access to the decryption process.
    b. Implementation: Implement user authentication mechanisms, similar to those used in the encryption process. Enforce access controls to prevent unauthorized access.

II. File Selection:
    a. Objective: Enable users to choose the encrypted files or folders they want to decrypt.
    b. Implementation: Provide a user interface for file/folder selection, allowing users to choose multiple files if needed. Validate selected files to ensure they are encrypted and accessible.

III. Decryption Algorithm:
    a. Objective: Implement a robust decryption algorithm to transform ciphertext back into plaintext.

b. Implementation: Choose the same encryption algorithm used for encryption (e.g., AES) and integrate it into the decryption process. Set parameters consistent with the encryption process.

IV. Key Management:

a. Objective: Manage cryptographic keys for decryption securely, ensuring that only authorized users can access the decrypted data.

b. Implementation: Authenticate users and obtain the necessary decryption keys. Integrate key management mechanisms for secure key handling during decryption.

V. Decryption Process:

a. Objective: Execute the decryption process, transforming encrypted files back into their original plaintext form.

b. Implementation: Integrate the decryption algorithm with the selected encrypted files. Manage the decryption process flow, considering user feedback and progress indicators.

VI. Plaintext Output:

a. Objective: Generate decrypted files (plaintext) for user access.

b. Implementation: Save the decrypted files in the specified location or provide options for users to choose a destination.

3. Folder Encryption

I. User Authentication:

a. Objective: Authenticate users to verify their identity before allowing access to the decryption process.

b. Implementation: Implement user authentication mechanisms, similar to those used in the encryption process. Enforce access controls to prevent unauthorized access.

II. Folder Selection:

a. Objective: Allow users to choose the folder/directory they want to encrypt.

b. Implementation: Provide a user interface for folder selection, ensuring users can navigate and select entire directories. Validate selected folders to ensure they contain files eligible for encryption.

III. File Encryption Integration:
    a. Objective: Integrate the existing file encryption functionality to encrypt all files within the selected folder.
    b. Implementation: Iterate through each file in the selected folder and apply the file encryption process to each.

IV. Notification and Feedback:
    a. Objective: Keep users informed about the completion of the folder encryption process.
    b. Implementation: Display a notification or success message upon successful completion. Alert users in case of any errors or issues during the encryption.

4. Folder Decryption

I. User Authentication:
    a. Objective: Authenticate users to verify their identity before allowing access to the decryption process.
    b. Implementation: Implement user authentication mechanisms, similar to those used in the encryption process. Enforce access controls to prevent unauthorized access.

II. Folder Selection:
    a. Objective: Allow users to choose the folder/directory they want to decrypt.
    b. Implementation: Provide a user interface for folder selection, enabling users to navigate and select entire directories. Validate selected folders to ensure they contain encrypted files.

III. File Decryption Integration:
    a. Objective: Integrate the existing file decryption functionality to decrypt all files within the selected folder.
    b. Implementation: Iterate through each encrypted file in the selected folder and apply the file decryption process to each.

IV. Notification and Feedback:
    a. Objective: Keep users informed about the completion of the folder decryption process.

b. Implementation: Display a notification or success message upon successful completion. Alert users in case of any errors or issues during the decryption.

F.) Integration and Testing Phase

The various components of the system are integrated and systematically tested. The user tests the system to ensure that the functional requirements, as defined in the functional requirements document, are satisfied. Any faults or errors are rectified in this stage and after this the product goes into its final phase.

All the modules which were implemented in the development phase are all tested in this stage. The various features which were integrated are properly tested and then the system moves to the implementation phase.

The Encryption/Decryption module and other modules were developed independently initially and were integrated in this phase.

G.) Implementation Phase

This is the last step of the project. The tool is made operational and launched in a production environment. The phase is initiated only after the system has been properly tested and accepted by the user. This phase continues until the tool is operating in production in accordance with the defined requirements. If there are any updates or modifications, then they are accordingly maintained and updated.

This project is in its last stage. The tool is operational and will soon be hosted. We plan to develop a complete security application for this concept, very soon.

Chapter Two
SOFTWARE REQUIREMENTS AND SPECIFICATIONS

SOFTWARE REQUIREMENTS AND SPECIFICATIONS

2.1 Introduction

The following subsections of the Software Requirements Specifications (SRS) document will provide an overview of the entire tool – File Encryption. This specification document includes description of the functions and the specifications of the project. In this section a review of the entire project is provided. The reader will get familiarized with the specifications before further details are provided.

2.1.1 Purpose

This specification document describes the capabilities that will be provided by the tool 'File Encryption'. It will explain the purpose and features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate and how the system will react to external stimuli. This document is intended for both the stakeholders and the developers of the system and will be proposed to the Computer Science Head of Department for his approval.

File Encryption aims to provide the following functionalities:

1. File Encryption

VIII. User Authentication:

a. Objective: Authenticate users to ensure that only authorized individuals can perform encryption.

b. Implementation: Implement user authentication mechanisms (e.g., password, biometrics). Enforce access control to restrict unauthorized access.

IX. File Selection:

a. Objective: Allow users to choose the files or folders they want to encrypt.

b. Implementation: Provide a user interface for file/folder selection, supporting multiple selections. Validate selected files to ensure they are accessible and compatible.

X. Encryption Algorithm:

a. Objective: Implement a robust encryption algorithm to transform plaintext into ciphertext securely.

b. Implementation: Choose a strong encryption algorithm (e.g., AES) and integrate it into the tool. Set parameters such as key size and mode of operation based on security requirements.

XI. Key Management:

a. Objective: Generate, handle, and securely manage cryptographic keys used in the encryption process.

b. Implementation: Implement key generation algorithms. Integrate key storage, distribution, rotation, and recovery mechanisms.

XII. Encryption Process:

a. Objective: Execute the encryption process, transforming selected files into encrypted form.

b. Implementation: Integrate the chosen encryption algorithm with the selected files. Manage the encryption process flow, considering user feedback and progress indicators.

XIII. Ciphertext Output:

a. Objective: Generate encrypted files (ciphertext) for storage or transmission.

b. Implementation: Save the encrypted files in the specified location or provide options for users to choose a destination.

XIV. Notification and Feedback:

a. Objective: Keep users informed about the status of the encryption process.

b. Implementation: Display clear notifications, success messages, or error alerts. Provide feedback on completion and prompt users to take necessary actions.

2. File Decryption

VII. User Authentication:

a. Objective: Authenticate users to verify their identity before allowing access to the decryption process.

b. Implementation: Implement user authentication mechanisms, similar to those used in the encryption process. Enforce access controls to prevent unauthorized access.

VIII. File Selection:

a. Objective: Enable users to choose the encrypted files or folders they want to decrypt.

b. Implementation: Provide a user interface for file/folder selection, allowing users to choose multiple files if needed. Validate selected files to ensure they are encrypted and accessible.

IX. Decryption Algorithm:

a. Objective: Implement a robust decryption algorithm to transform ciphertext back into plaintext.

b. Implementation: Choose the same encryption algorithm used for encryption (e.g., AES) and integrate it into the decryption process. Set parameters consistent with the encryption process.

X. Key Management:

a. Objective: Manage cryptographic keys for decryption securely, ensuring that only authorized users can access the decrypted data.

b. Implementation: Authenticate users and obtain the necessary decryption keys. Integrate key management mechanisms for secure key handling during decryption.

XI. Decryption Process:

a. Objective: Execute the decryption process, transforming encrypted files back into their original plaintext form.

b. Implementation: Integrate the decryption algorithm with the selected encrypted files. Manage the decryption process flow, considering user feedback and progress indicators.

XII. Plaintext Output:

a. Objective: Generate decrypted files (plaintext) for user access.

b. Implementation: Save the decrypted files in the specified location or provide options for users to choose a destination.

3. Folder Encryption

V. User Authentication:

a. Objective: Authenticate users to verify their identity before allowing access to the decryption process.

b. Implementation: Implement user authentication mechanisms, similar to those used in the encryption process. Enforce access controls to prevent unauthorized access.

VI. Folder Selection:

a. Objective: Allow users to choose the folder/directory they want to encrypt.

b. Implementation: Provide a user interface for folder selection, ensuring users can navigate and select entire directories. Validate selected folders to ensure they contain files eligible for encryption.

VII. File Encryption Integration:

a. Objective: Integrate the existing file encryption functionality to encrypt all files within the selected folder.

b. Implementation: Iterate through each file in the selected folder and apply the file encryption process to each.

VIII. Notification and Feedback:

a. Objective: Keep users informed about the completion of the folder encryption process.

b. Implementation: Display a notification or success message upon successful completion. Alert users in case of any errors or issues during the encryption.

4. Folder Decryption

V. User Authentication:

a. Objective: Authenticate users to verify their identity before allowing access to the decryption process.

b. Implementation: Implement user authentication mechanisms, similar to those used in the encryption process. Enforce access controls to prevent unauthorized access.

VI. Folder Selection:

a. Objective: Allow users to choose the folder/directory they want to decrypt.

b. Implementation: Provide a user interface for folder selection, enabling users to navigate and select entire directories. Validate selected folders to ensure they contain encrypted files.

VII. File Decryption Integration:

a. Objective: Integrate the existing file decryption functionality to decrypt all files within the selected folder.

b. Implementation: Iterate through each encrypted file in the selected folder and apply the file decryption process to each.

VIII. Notification and Feedback:

a. Objective: Keep users informed about the completion of the folder decryption process.

b. Implementation: Display a notification or success message upon successful completion. Alert users in case of any errors or issues during the decryption.

2.1.2 Intended Audience & Intended Use

The target audience for this file encryption system encompasses individuals who prioritize and recognize the significance of maintaining a secure digital environment. It is designed to cater to a diverse group of users who share a common understanding that security is a paramount aspect that demands serious attention in today's digital landscape. The primary focus is on individuals who adhere to the ideology of a balanced internet use, emphasizing the importance of digital protection and a thoughtful approach to utilizing digital data.

The system is tailored for those who actively seek to enhance their digital security. These individuals are likely to have a heightened awareness of potential threats and understand the importance of implementing robust security measures to safeguard their data.

The target audience includes individuals who adopt a balanced approach to their internet usage. They appreciate the convenience and benefits of digital interactions but are equally mindful of the potential risks and vulnerabilities associated with online activities.

Individuals who advocate for digital protection and recognize the need for proactive measures to secure sensitive information are within the target audience. This may include professionals, activists, or enthusiasts who actively promote and practice digital security.

The system caters to those who critically assess and evaluate their digital data usage. Users who understand the importance of securing digital information and engage in thoughtful practices to assess and manage their data fall within the intended audience.

The primary purpose of the system is to support individuals in maintaining a secure conversation environment. It targets users who value secure communication channels and seek tools that enable them to transform their conversations with confidence, knowing that their data is protected.

The overarching goal is to empower users with the means to keep the "security fire burning." By offering a reliable file encryption system, the project aims to assist and guide individuals in adopting secure practices in their digital interactions, thereby contributing to an overall safer online environment.

The target audience comprises individuals who share a collective commitment to digital security, balanced internet use, and the assessment of digital data practices. The file encryption system is positioned as a valuable tool to support and enhance the security posture of users who prioritize safeguarding their digital information.

2.1.3 Constraints

1. **Performance Impact:**
    - Encryption and decryption processes can introduce a computational overhead, potentially slowing down the performance of systems, particularly on older or resource-constrained devices. The level of impact depends on the encryption algorithm's complexity and the system's processing capabilities.
2. **Key Management Complexity:**
    - Managing cryptographic keys effectively is crucial for the security of the system. However, key management can become complex, especially in large-scale deployments or when dealing with multiple users. Key distribution, rotation, and storage require careful planning and implementation.
3. **User Accessibility and Usability:**
    - Striking a balance between security and user accessibility can be challenging. Complex encryption processes or frequent authentication requirements may impact user experience and hinder widespread

adoption. Achieving a user-friendly interface while maintaining robust security is a delicate balance.

4. **Recovery and Backup Concerns:**
    - In scenarios where users forget passwords or encounter other issues, recovery mechanisms become essential. However, implementing secure recovery without compromising the overall system security poses a challenge. Additionally, ensuring backup procedures for encrypted data is crucial to prevent data loss.
5. **Limited Granularity in Control:**
    - Some file encryption systems may have limitations in providing granular control over individual files or specific folders. Balancing security with user flexibility to selectively encrypt or decrypt files requires careful design considerations.
6. **Legal and Compliance Considerations:**
    - Compliance with legal regulations and data protection laws can be a constraint. Certain industries or regions may have specific requirements for data encryption and may impose limitations on the types of encryption algorithms that can be used.
7. **Resource Consumption:**
    - Resource-intensive encryption algorithms may consume significant system resources, impacting battery life on mobile devices or affecting the performance of systems with limited processing power.
8. **Communication Overhead:**
    - When encrypting files for transmission over a network, there can be an added communication overhead due to the increased size of encrypted data. This can impact bandwidth utilization and transmission speed.
9. **Algorithm Vulnerabilities:**
    - The security of file encryption systems depends on the strength of the encryption algorithms used. If vulnerabilities are discovered in these

algorithms over time, it may necessitate updates or changes to maintain a secure system.

2.1.4 Scope of Development

1. **Security Features:**
    - **Encryption Algorithms:** Implementing strong and widely accepted encryption algorithms, such as AES (Advanced Encryption Standard), to ensure the confidentiality of the data.
    - **Key Management:** Developing robust key management mechanisms, including secure generation, storage, distribution, rotation, and recovery processes.
    - **Authentication:** Incorporating secure authentication methods to ensure that only authorized users have access to encryption/decryption capabilities.
2. **User Interface and Usability:**
    - **Intuitive Design:** Creating a user-friendly and intuitive interface for both technical and non-technical users to easily encrypt and decrypt files/folders.
    - **Granular Control:** Providing users with options for selective encryption or decryption, enabling them to choose specific files or folders to secure.
    - **Notifications and Alerts:** Implementing clear notifications and alerts to keep users informed about the status of encryption/decryption processes.
3. **Cross-Platform Compatibility:**
    - **Operating System Support:** Designing the tool to be compatible with various operating systems, ensuring seamless integration into diverse computing environments.

- **Device Compatibility:** Ensuring compatibility with different devices, including computers, mobile devices, and network-attached storage (NAS) systems.

4. **Performance Optimization:**
    - **Optimized Algorithms:** Balancing security with performance by selecting encryption algorithms that offer a reasonable level of security without excessive computational overhead.
    - **Parallel Processing:** Implementing techniques such as parallel processing to enhance the speed of encryption/decryption operations.
5. **Key Recovery and Backup:**
    - **Recovery Mechanisms:** Including secure and user-friendly mechanisms for key recovery in case users forget their passwords or encounter other issues.
    - **Backup Solutions:** Providing users with tools or recommendations for securely backing up their cryptographic keys to prevent data loss.

2.2 Specific Requirements

2.2.1 Functional Requirements

- The user must have Linux Platform, other OS might not support all features.
- The tool requires a stable system with better resources are mentioned below.

2.2.2 Hardware requirements

- Ram Required: 4 GB
- Processor Required: Intel Pentium 4 and later
- Application Linux Size: Minimum 20 GB

2.2.3 Software Interfaces

The files or folders stored in the system, goes through the encryption algorithm and using the AES algorithm it can convert it to encrypted file and files in folder. The

communication between the tool and file consists of operations concerning reading, writing, and modifying the data and its HEX Code.

2.2.4 Technical Requirements

- Languages: Python
- Applications: Visual Studio, GitHub

2.2.5 Non-Functional Requirements

In non-functional requirement limits are provided, which means restriction is attached with the requirement and one has to fulfil or satisfy that limit.

- Time: The project has to be completed within two months.
- Input: The software registers new deployment with a key/code into the database and accepts any type files and passes them to the model for the magic.
- General Requirements:
    - Physical Environment: Physical environment requirements such as where is the equipment to function located, are there any environmental restrictions like temperature, humidity, magnetic interference, etc are to be gathered prior to the development phase of the system. No special physical requirements are needed in our project.
    - Interface: Interface requirements are such as is the input coming from one or more systems, is output going to one or more systems, is there any prescribed medium that the data must use should be gathered. In our project the input comes from user as any file.
    - User and Human Factors: User and Human factors consist of requirements as who will use the system, will there be different types of users, what is the skill levels of each type of user, what kind of training is required for each user, and how easy will it be for a user to understand and use the system are required. In this project no special training is required to use the project.
    - Data: Data requirements like what should be the format of both the input and output, how accurate must they be, should any data be retained for any period of time should be known.
    - Error handling: Application shall handle expected and non-expected errors in ways that prevent loss in information.

2.2.6 Non-Functional Characteristics

- Usability: The system is easy to use. The user reaches the summarized text with basic numeric options, very quickly.
- Reliability: The system is reliable in terms of usage and journey. Also, for the admin it is very easy to update the database.
- Performance: Calculation time and response time is very little because various features are timesaving. Whole cycle of summarizing a page/file does not take more than 30 seconds, in order to load the document. The capacity of servers is very high. Calculation and response times are very low, and this comes with so many users at the same time.1 minute degradation or response time is not acceptable. The certain session limit is also acceptable at early stages of development.
- Supportability: The system requires knowledge of python for maintenance. If any problem is faced in tool, it requires coding knowledge.

2.3 Risk Analysis

These steps are performed in risk analysis for designing the system because the future of the system is our concern. We identified what risks might create problem in the life of the system. We also identified that what change in the user requirements, technologies, hardware and all other entries connected to the system will affect the system.

2.3.1 Risk Identification

We were able to identify the risks under the following categories:

- Project risk
- Technical risk
- Business risk

Following list was identified under the categories mentioned above:

- Enough number of people were available, as estimated, to complete the system.
- All staff involved in the system was not fully trained on the platform to be used for development. We also had to study various things about the platform and the system.

2.3.2 Probability of Risk

The probability for the project risks such as schedule, resources, customer, requirement problems and their impact on the system was negligible. There was a risk on the technical grounds because the system was developed with a new technology hence the experience on the tools was taking which faced the management to think whether the choice made was right or wrong. But a survey done on the use of new platform gave us the confidence of continuing on this decision. As we know system design is a solution a “how to” approach to the creation of a new system. This important phase is composed of several steps. It provides the understanding and procedural details necessary for implementing the system recommended in the feasibility study.

Chapter Three
TECHNOLOGY USED

TECHNOLOGY USED

3.1 Python

Python is an interpreter, high-level and general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant white space. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically typed, and garbage collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Python was created in the late 1980s, and first released in 1991, by Guido van Rossum as a successor to the ABC programming language. Python 2.0, released in 2000, introduced new features, such as list comprehensions, and a garbage collection system with reference counting, and was discontinued with version 2.7 in 2020. Python 3.0, released in 2008, was a major revision of the language that is not completely backward compatible, and much Python 2 code does not run unmodified on Python 3.Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, a free and open sourcereference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development. It currently ties with Java as the second most popular programming language in the world.

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by meta programming and meta objects (magic methods)).

Many other paradigms are supported via extensions, including design by contract and logic programming. Python uses dynamic typing and a combination of reference counting and a cycle detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Python's design offers some support for functional programming in the Lisp tradition. It has filter, map, and reduce functions; list comprehensions, dictionaries, sets, and generator expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.

Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.

Python strives for a simpler, less-cluttered syntax and grammar while giving developers a choice in their coding methodology. In contrast to Perl's "there is more than one way to do it" motto, Python embraces a "there should be one—and preferably only one—obvious way to do it" design philosophy.Alex Martelli, a Fellow at the Python Software Foundation and Python book author, writes that "To describe something as 'clever' is not considered a compliment in the Python culture."

The language's core philosophy is summarized in the document The Zen of Python (PEP 20), which includes aphorisms such as:

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts

Python's developers strive to avoid premature optimization, and reject patches to non-critical parts of the CPython reference implementation that would offer marginal increases in speed at the cost of clarity.

When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or use PyPy, a just-in-time compiler.

CPython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

It is used for:

- Web development (server-side),
- Software development,
- Mathematics,
- System scripting.

An important goal of Python's developers is keeping it fun to use. This is reflected in the language's name—a tribute to the British comedy group Monty Python —and in occasionally playful approaches to tutorials and reference materials, such as examples that refer to spam and eggs (from a famous Monty Python sketch) instead of the standard foo and bar.

A common neologism in the Python community is pythonic, which can have a wide range of meanings related to program style. To say that code is pythonic is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called unpythonic.

Users and admirers of Python, especially those considered knowledgeable or experienced, are often referred to as Pythonistas.

Syntax & Semantics of Python

Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases than C or Pascal.

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents the program's semantic structure. This feature is sometimes termed the off-side rule, which some other languages share, but in most languages indentation doesn't have any semantic meaning.

3.2 Crypto Library - from Crypto import Random

The term "crypto library" typically refers to cryptographic libraries that offer a set of functions and algorithms for secure communication, data protection, and other cryptographic operations. These libraries are crucial in ensuring the confidentiality, integrity, and authenticity of data in various applications, such as secure communication, digital signatures, and encryption.

In this Python example, the cryptography library is used to perform a SHA-256 hash on the string "Hello, World!". The library provides a set of modules for various cryptographic operations, and in this case, it includes functions for hashing using the SHA-256 algorithm.

Random Module:

The Random module is a standard module found in many programming languages, including Python. It is used for generating pseudo-random numbers. Pseudo-random numbers are not truly random but are generated by algorithms that produce sequences that exhibit statistical randomness. These numbers are commonly used in applications such as simulations, games, and cryptographic key generation.

In this Python example, the random module is used to generate a random integer between 1 and 10 (inclusive). The random module provides various functions for generating random numbers, including integers, floating-point numbers, and sequences.

Note:

The specific implementations and available features of crypto libraries and random modules may vary across programming languages.

It's important to consult the documentation of the specific library or module you are using to understand the details of their implementation, available functions, and best practices for usage.

If you have a specific programming language or library in mind, please provide more details, and I can offer more context-specific information.

3.3 GetPass Library - from getpass import getpass

The getpass library in Python provides a way to interact with the user for password input without echoing the characters on the screen. This is particularly useful when you want to securely handle password input, such as when prompting a user to enter a password in a console or terminal environment.

Key Functions:

1. **getpass.getpass(prompt='Password: ', stream=None)**:
    - This function prompts the user for a password, hiding the input from the screen. The optional **prompt** argument allows you to customize the input prompt, and the optional **stream** argument specifies the stream to which the prompt is written.
2. **getpass.getuser()**:
    - This function returns the username of the user executing the Python script or interactively using the interpreter.

Use Cases:

- **Secure Password Input:** The primary use case of the **getpass** library is to securely handle password input without revealing characters on the screen. This is crucial for applications that require sensitive information, such as login credentials.
- **Customized Prompts:** The **getpass** library allows you to customize the prompt displayed to the user, providing a clear indication of the expected input.

Note:

- The **getpass** library is particularly useful in console or terminal environments where secure input is needed.
- It's important to note that while this library provides a level of security for password input, it may not protect against more sophisticated attacks. For a higher level of security, consider using dedicated libraries for authentication and encryption.

- The behavior of **getpass** may vary across different operating systems and environments.

When using the **getpass** library, always be mindful of security best practices and consider additional measures for securing sensitive information in your applications.

3.4 Crypto.Cipher Library - Crypto.Cipher import AES

The **Crypto.Cipher** module is part of the **pycryptodome** library in Python, which is a self-contained Python package of low-level cryptographic primitives. The **Crypto.Cipher** module specifically provides a collection of block ciphers that can be used for various encryption and decryption operations.

Overview of Crypto.Cipher:

1. **Block Ciphers:**
    - Block ciphers are cryptographic algorithms that encrypt and decrypt data in fixed-size blocks. Each block typically consists of a fixed number of bits (e.g., 128 bits for AES). Block ciphers use a key to perform encryption and decryption operations on blocks of plaintext or ciphertext.
2. **Cipher Modes:**
    - The **Crypto.Cipher** module supports various cipher modes, which define how blocks are processed and combined during encryption and decryption. Common cipher modes include Electronic Codebook (ECB), Cipher Block Chaining (CBC), and Galois/Counter Mode (GCM).
3. **Symmetric Encryption:**
    - The module supports symmetric-key cryptography, where the same key is used for both encryption and decryption. Common symmetric encryption algorithms include Advanced Encryption Standard (AES), Triple DES (3DES), and Blowfish.

The **Crypto.Cipher.AES.new** method is used to create an AES cipher object with a specified key, mode (CBC), and an initialization vector (IV). The **encrypt** and **decrypt** functions demonstrate how to use the AES cipher for encryption and decryption.

Note:

- **Initialization Vector (IV):**
    - The IV is a random value that is used in certain cipher modes (like CBC) to ensure that the same plaintext does not encrypt to the same ciphertext when encrypted multiple times with the same key.
- **Key Size:**
    - The key size used in the example is 16 bytes (128 bits) for AES. The key size can vary depending on the chosen cipher algorithm (e.g., 16, 24, or 32 bytes for AES).
- **Security Considerations:**
    - When using cryptographic algorithms, it's essential to follow security best practices, including using strong, random keys and understanding the specific security properties of the chosen cipher mode.
- **Cryptodome Library:**
    - Ensure that you have the **pycryptodome** library installed (**pip install pycryptodome**) to use the **Crypto.Cipher** module.

The example demonstrates a basic usage of the **Crypto.Cipher** module, but it's important to adapt the code based on specific security requirements and best practices for your application.

3.5 Time Library - import time

The **time** module in Python provides functionality related to time, including measuring time intervals, formatting time values, and working with the system time. Here's a detailed explanation of the key features and functions in the **time** module:

1. Getting the Current Time:

- The **time()** function returns the current system time in seconds since the epoch (the epoch is a predefined point in time, usually the start of 1970, on Unix systems).

2. Converting Time to a Readable Format:

- The **ctime()** function converts a time in seconds since the epoch to a string representing a local time.

3. Sleeping:

- The **sleep()** function suspends the execution of the current thread for a specified number of seconds.

4. Measuring Time Intervals:

- You can use **time()** to record the start and end times and calculate the elapsed time.

5. Formatting Time Values:

- The **strftime()** function is used to format a **struct_time** object or a time in seconds since the epoch.

6. Getting Process Time:

- The **process_time()** function returns the current process time in seconds, which is useful for measuring CPU time consumed by a program.

7. Time Tuple:

- The **gmtime()** and **localtime()** functions return a **struct_time** object representing the current UTC time or local time, respectively.

8. Performance Measurement:

- The **perf_counter()** function returns a high-resolution performance counter that can be used for measuring short durations.

Note:

- The **time** module provides various other functions and constants for dealing with time-related operations.

- The **struct_time** object returned by functions like **gmtime()** and **localtime()** has attributes like year, month, day, hour, minute, second, etc.
- When working with time-related tasks, consider using the **datetime** module for more advanced functionalities and better timezone support.

The **time** module is fundamental for dealing with time-related operations in Python, and its functions can be employed for various scenarios, from measuring performance to formatting time for display.

3.6 ASCII Art

ASCII art is a captivating form of visual expression that utilizes characters from the ASCII (American Standard Code for Information Interchange) character set to create images and designs. In ASCII art, artists use characters such as letters, numbers, and symbols to form patterns, shapes, and intricate designs, producing an artistic representation when viewed in a monospaced font.

Historical Roots:

ASCII art finds its roots in the early days of computing when graphical displays were limited, and text-based interfaces dominated computer screens. In this era, artists and enthusiasts creatively leveraged the simplicity of ASCII characters to craft visual designs. The humble beginnings of ASCII art can be traced back to teleprinters and early computer terminals, where limited graphical capabilities inspired users to invent a new form of digital art using characters available on their keyboards.

Key Concepts and Techniques:

1. Character Choice:

Artists carefully choose ASCII characters based on their shapes, sizes, and shading qualities. Characters like "@" or "#" might be used for bold strokes, while "." or " " (space) can represent lighter areas.

2. Shading and Contrast:

ASCII art relies on the strategic arrangement of characters to create shading effects and contrasts. By varying the density of characters in different regions, artists convey depth and texture.

3. Line Art and Symmetry:

ASCII art often incorporates line art techniques to create outlines, contours, and intricate patterns. Achieving symmetry is crucial, and artists employ precise character placement to maintain balance in their compositions.

4. Size and Aspect Ratio:

The size and aspect ratio of the canvas impact the final appearance of ASCII art. Artists may choose specific dimensions to optimize the viewing experience, especially when sharing their creations online.

5. Tools and Editors:

While ASCII art can be created using simple text editors, specialized tools and online generators have emerged to streamline the process. These tools offer features like character selection, colorization, and real-time preview.

Modern Resurgence:

In recent times, ASCII art has experienced a resurgence in popularity as a nostalgic and creative medium. Social media platforms, coding communities, and online forums embrace ASCII art as a unique way for users to express creativity, humor, and individuality. It has become a cultural phenomenon, with artists using ASCII characters to depict memes, portraits, and even recreate famous artworks.

Challenges and Innovations:

Despite its simplicity, ASCII art poses challenges, especially when creating detailed and intricate designs. Artists often overcome these challenges through experimentation, innovation, and a deep understanding of character nuances. Some modern variations of ASCII art include ANSI art, which adds color to the traditional black-and-white palette, and animated ASCII art, where sequences of characters create motion and dynamism.

Conclusion:

ASCII art stands as a testament to the creative potential of simplicity. In a digital age dominated by high-resolution graphics, this humble art form captivates audiences with its nostalgic charm and the ingenuity of artists who transform characters into canvases.

From its humble beginnings as a necessity in early computing to its modern-day revival as a unique and cherished form of expression, ASCII art continues to bridge the gap between technology and creativity, reminding us that art can flourish even within the constraints of characters and pixels.

3.7 Kali Linux

Kali Linux is a Debian-derived Linux distribution designed for digital forensics, penetration testing, ethical hacking, and other cybersecurity tasks. It is an open-source operating system that provides a comprehensive toolkit for security professionals and enthusiasts to assess and secure computer systems. Kali Linux is maintained and funded by Offensive Security, a leading provider of information security training and certification.

Key Features and Components of Kali Linux:

Penetration Testing Tools: Kali Linux comes pre-installed with a vast array of penetration testing tools, making it a go-to choice for cybersecurity professionals. These tools include network scanners, vulnerability analysis tools, password cracking utilities, wireless network assessment tools, and more.

Custom Kernel and Package Repository: Kali Linux features a custom kernel that includes various patches and drivers to support a wide range of wireless devices and network adapters. It has its own package repository, ensuring that the tools and software included in Kali are up-to-date and tailored for security testing.

Forensic Tools: Kali Linux is equipped with tools for digital forensics and incident response. These tools assist in the identification, preservation, and analysis of digital evidence, making Kali an invaluable resource for forensic professionals.

Ease of Use: Kali Linux is designed to be user-friendly, even for those new to cybersecurity. It provides a well-organized menu structure, categorized tools, and a user-friendly interface, making it accessible to both beginners and experienced security practitioners.

Live Boot Capability: Kali Linux can be booted directly from a live USB or DVD without the need for installation. This allows security professionals to use Kali on the go, without modifying the host system.

Community and Support: Kali Linux has a vibrant and active community of security professionals, ethical hackers, and enthusiasts. The community contributes to ongoing development, provides support through forums and documentation, and shares knowledge on various security-related topics.

Security Training and Certification: Offensive Security, the organization behind Kali Linux, offers training and certification programs such as "Offensive Security Certified Professional" (OSCP) and "Kali Linux Certified Professional" (KLCP). These certifications are highly regarded in the cybersecurity industry.

Use Cases:

Penetration Testing: Kali Linux is widely used for penetration testing and ethical hacking. Security professionals use its tools to identify vulnerabilities in systems, networks, and applications to secure them against malicious attacks.

Forensics and Incident Response: Digital forensics experts use Kali Linux to investigate security incidents, gather evidence, and analyse the aftermath of cybersecurity breaches.

Security Auditing: Organizations and security consultants use Kali Linux for security auditing and risk assessments to ensure the robustness of their information systems.

Security Training: Kali Linux is commonly used as a training platform in cybersecurity courses and workshops. Its hands-on approach and extensive toolset make it an ideal environment for learning security concepts and techniques.

Conclusion:

Kali Linux has become a cornerstone in the toolkit of cybersecurity professionals, providing a powerful and versatile platform for conducting security assessments. Its commitment to keeping tools up to date, its user-friendly interface, and the support of a dedicated community make Kali Linux a valuable asset for anyone involved in cybersecurity and ethical hacking activities.

3.8 VMWare

VMware is a leading provider of virtualization and cloud computing software and services. The company offers a range of products that enable organizations to optimize and manage their IT infrastructure through virtualization, allowing for more efficient

resource utilization, enhanced flexibility, and improved scalability. VMware's solutions are widely used in data centers, enterprises, and cloud environments to streamline operations and increase the agility of IT systems.

Key Components and Offerings:

1. **VMware vSphere:**
    - VMware vSphere is a comprehensive virtualization platform that provides the foundation for building and managing virtualized infrastructure. It includes features such as hypervisor (ESXi), centralized management (vCenter Server), and various tools for virtual machine (VM) management.
2. **ESXi Hypervisor:**
    - ESXi is VMware's bare-metal hypervisor, which allows for the creation and execution of virtual machines on physical servers. It provides a lightweight, efficient, and secure virtualization layer, enabling multiple virtual machines to run on a single physical server.
3. **vCenter Server:**
    - vCenter Server is a centralized management platform that allows administrators to manage and monitor multiple ESXi hosts and their associated virtual machines. It provides features like resource management, performance monitoring, and advanced capabilities such as vMotion for live VM migration.
4. **VMware Workstation and Fusion:**
    - VMware Workstation (for Windows and Linux) and Fusion (for macOS) are desktop virtualization solutions that enable users to run multiple operating systems on a single physical machine. These products are popular for development, testing, and running different operating systems on a desktop or laptop.
5. **VMware Horizon:**

- VMware Horizon is a virtual desktop infrastructure (VDI) solution that allows organizations to deliver virtualized desktops and applications to end-users. It enhances mobility and provides secure access to desktop environments from various devices.

6. **VMware Cloud Foundation:**

   - VMware Cloud Foundation is an integrated platform that combines compute virtualization (vSphere), storage virtualization (vSAN), and network virtualization (NSX) to deliver a complete and integrated cloud infrastructure.

7. **NSX:**

   - VMware NSX is a network virtualization and security platform that provides software-defined networking (SDN) capabilities. It allows organizations to create and manage virtualized network infrastructure, improving agility and security.

Use Cases:

1. **Server Virtualization:**

   - VMware's server virtualization solutions enable organizations to consolidate multiple physical servers into a single server, optimizing hardware utilization and reducing operational costs.

2. **Desktop Virtualization:**

   - VMware Workstation, Fusion, and Horizon facilitate desktop virtualization, allowing users to run multiple operating systems or access virtual desktops securely from different devices.

3. **Cloud Infrastructure:**

   - VMware's cloud solutions, including VMware Cloud Foundation, help organizations build and manage cloud infrastructure, whether on-premises or in public clouds.

4. **Disaster Recovery:**

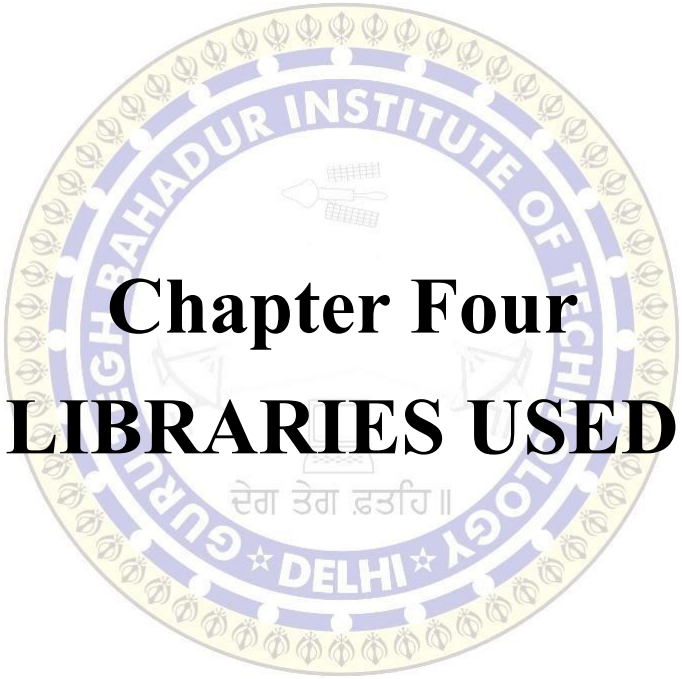
- VMware's virtualization platform supports efficient disaster recovery solutions, allowing organizations to replicate and recover virtual machines in case of hardware failures or other disasters.

5. **Software Development and Testing:**

   - Developers use VMware Workstation and Fusion to create isolated development and testing environments, enabling them to test software on different operating systems and configurations.

Conclusion:

VMware has played a pivotal role in transforming IT infrastructure by popularizing virtualization technologies. Its products and solutions offer organizations the flexibility, scalability, and efficiency needed to adapt to evolving business requirements. Whether in data centers, on desktops, or in the cloud, VMware continues to be a key player in the virtualization and cloud computing landscape.



Chapter Four
LIBRARIES USED

LIBRARY USED

4.1 Crypto Library - from Crypto import Random

The term "crypto library" typically refers to cryptographic libraries that offer a set of functions and algorithms for secure communication, data protection, and other cryptographic operations. These libraries are crucial in ensuring the confidentiality, integrity, and authenticity of data in various applications, such as secure communication, digital signatures, and encryption.

In this Python example, the cryptography library is used to perform a SHA-256 hash on the string "Hello, World!". The library provides a set of modules for various cryptographic operations, and in this case, it includes functions for hashing using the SHA-256 algorithm.

Random Module:

The Random module is a standard module found in many programming languages, including Python. It is used for generating pseudo-random numbers. Pseudo-random numbers are not truly random but are generated by algorithms that produce sequences that exhibit statistical randomness. These numbers are commonly used in applications such as simulations, games, and cryptographic key generation.

In this Python example, the random module is used to generate a random integer between 1 and 10 (inclusive). The random module provides various functions for generating random numbers, including integers, floating-point numbers, and sequences.

Note:

The specific implementations and available features of crypto libraries and random modules may vary across programming languages.

It's important to consult the documentation of the specific library or module you are using to understand the details of their implementation, available functions, and best practices for usage.

If you have a specific programming language or library in mind, please provide more details, and I can offer more context-specific information.

4.2 GetPass Library - from getpass import getpass

The getpass library in Python provides a way to interact with the user for password input without echoing the characters on the screen. This is particularly useful when you want to securely handle password input, such as when prompting a user to enter a password in a console or terminal environment.

Key Functions:

3. **getpass.getpass(prompt='Password: ', stream=None)**:
    - This function prompts the user for a password, hiding the input from the screen. The optional **prompt** argument allows you to customize the input prompt, and the optional **stream** argument specifies the stream to which the prompt is written.
4. **getpass.getuser()**:
    - This function returns the username of the user executing the Python script or interactively using the interpreter.

Use Cases:

- **Secure Password Input:** The primary use case of the **getpass** library is to securely handle password input without revealing characters on the screen. This is crucial for applications that require sensitive information, such as login credentials.
- **Customized Prompts:** The **getpass** library allows you to customize the prompt displayed to the user, providing a clear indication of the expected input.

Note:

- The **getpass** library is particularly useful in console or terminal environments where secure input is needed.

- It's important to note that while this library provides a level of security for password input, it may not protect against more sophisticated attacks. For a higher level of security, consider using dedicated libraries for authentication and encryption.
- The behavior of **getpass** may vary across different operating systems and environments.

When using the **getpass** library, always be mindful of security best practices and consider additional measures for securing sensitive information in your applications.

4.3 Crypto.Cipher Library - Crypto.Cipher import AES

The **Crypto.Cipher** module is part of the **pycryptodome** library in Python, which is a self-contained Python package of low-level cryptographic primitives. The **Crypto.Cipher** module specifically provides a collection of block ciphers that can be used for various encryption and decryption operations.

Overview of Crypto.Cipher:

4. **Block Ciphers:**
    - Block ciphers are cryptographic algorithms that encrypt and decrypt data in fixed-size blocks. Each block typically consists of a fixed number of bits (e.g., 128 bits for AES). Block ciphers use a key to perform encryption and decryption operations on blocks of plaintext or ciphertext.
5. **Cipher Modes:**
    - The **Crypto.Cipher** module supports various cipher modes, which define how blocks are processed and combined during encryption and decryption. Common cipher modes include Electronic Codebook (ECB), Cipher Block Chaining (CBC), and Galois/Counter Mode (GCM).

6. **Symmetric Encryption:**

    - The module supports symmetric-key cryptography, where the same key is used for both encryption and decryption. Common symmetric encryption algorithms include Advanced Encryption Standard (AES), Triple DES (3DES), and Blowfish.

The **Crypto.Cipher.AES.new** method is used to create an AES cipher object with a specified key, mode (CBC), and an initialization vector (IV). The **encrypt** and **decrypt** functions demonstrate how to use the AES cipher for encryption and decryption.

Note:

- **Initialization Vector (IV):**
    - The IV is a random value that is used in certain cipher modes (like CBC) to ensure that the same plaintext does not encrypt to the same ciphertext when encrypted multiple times with the same key.
- **Key Size:**
    - The key size used in the example is 16 bytes (128 bits) for AES. The key size can vary depending on the chosen cipher algorithm (e.g., 16, 24, or 32 bytes for AES).
- **Security Considerations:**
    - When using cryptographic algorithms, it's essential to follow security best practices, including using strong, random keys and understanding the specific security properties of the chosen cipher mode.
- **Cryptodome Library:**
    - Ensure that you have the **pycryptodome** library installed (**pip install pycryptodome**) to use the **Crypto.Cipher** module.

The example demonstrates a basic usage of the **Crypto.Cipher** module, but it's important to adapt the code based on specific security requirements and best practices for your application.

4.4 Time Library - import time

The **time** module in Python provides functionality related to time, including measuring time intervals, formatting time values, and working with the system time. Here's a detailed explanation of the key features and functions in the **time** module:

1. Getting the Current Time:

- The **time()** function returns the current system time in seconds since the epoch (the epoch is a predefined point in time, usually the start of 1970, on Unix systems).

2. Converting Time to a Readable Format:

- The **ctime()** function converts a time in seconds since the epoch to a string representing a local time.

3. Sleeping:

- The **sleep()** function suspends the execution of the current thread for a specified number of seconds.

4. Measuring Time Intervals:

- You can use **time()** to record the start and end times and calculate the elapsed time.

5. Formatting Time Values:

- The **strftime()** function is used to format a **struct_time** object or a time in seconds since the epoch.

6. Getting Process Time:

- The **process_time()** function returns the current process time in seconds, which is useful for measuring CPU time consumed by a program.

7. Time Tuple:

- The **gmtime()** and **localtime()** functions return a **struct_time** object representing the current UTC time or local time, respectively.

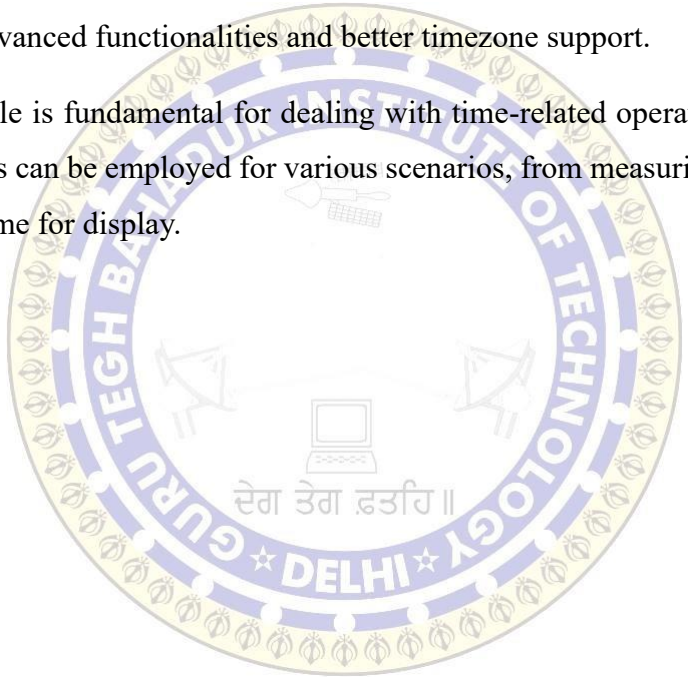
8. Performance Measurement:

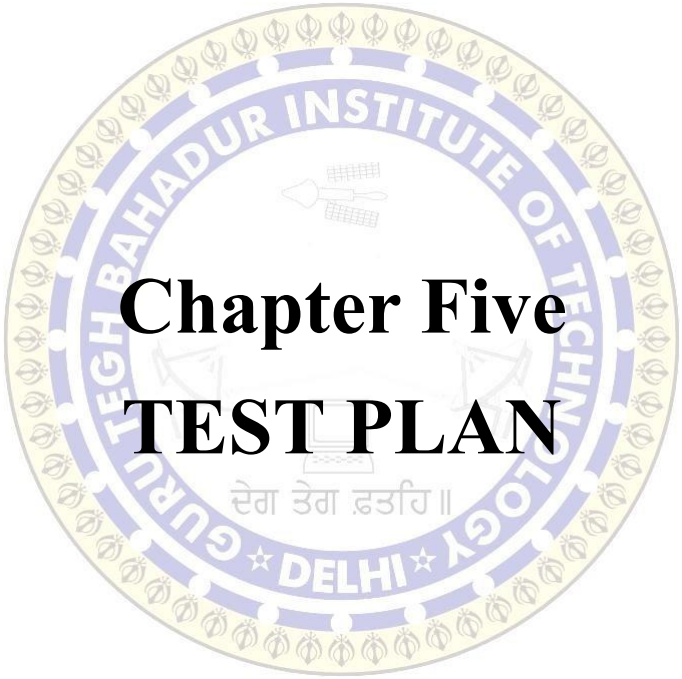
- The **perf_counter()** function returns a high-resolution performance counter that can be used for measuring short durations.

Note:

- The **time** module provides various other functions and constants for dealing with time-related operations.
- The **struct_time** object returned by functions like **gmtime()** and **localtime()** has attributes like year, month, day, hour, minute, second, etc.
- When working with time-related tasks, consider using the **datetime** module for more advanced functionalities and better timezone support.

The **time** module is fundamental for dealing with time-related operations in Python, and its functions can be employed for various scenarios, from measuring performance to formatting time for display.





Chapter Five
TEST PLAN

TEST PLAN

5.1 Introduction

The Test Plan is designed to describe the scope, approach, resources, and schedule of all testing activities of the project File encryption. The plan identifies the items to be tested, the features to be tested, the types of testing to be performed, the personnel responsible for testing, the schedule and resources required to complete testing, and the risks associated with the plan. The objective is to guarantee the operation of File encryption is in alignment with the Requirement. The purpose of this test plan is to ensure the proper functionality, security, and performance of the file encryption software. The test plan covers functional, security, and performance testing of the file encryption software.

5.1.1 Objective

- Verify that the solution meets the requirements
    - This is to prove that file encryption: and security that matters address the problem and satisfy the requirement; making sure to serve all the basic and important needs.
- Ensure the solution is optimized for the environment
    - This is to ensure that it works well in a real-world environment. This means providing tests that simulate real-life conditions and situations.
- Ensure overall quality
    - The third goal of our testing is to help us ensure a high-level quality of product. Since software changes become exponentially more expensive as a project advances through its life cycle, the goal is to identify defects as early as possible.
- Verify that the encryption and decryption processes work correctly:
    - To Create test cases to encrypt files of various sizes and formats. Then, ensure the decryption process accurately retrieves the original data.
    - Test extreme scenarios, like very small or very large files, to ensure the tool behaves as expected without encountering errors or data loss.
- Ensure the software handles different file types and sizes:

- Test the tool with various file formats (text, images, videos, etc.) to ensure it can handle diverse data types without issues.
- Assess how the tool manages different file sizes, from small documents to large multimedia files. Ensure encryption and decryption remain efficient and reliable across these variations.

- Validate the security features to protect against unauthorized access:
  - Evaluate the encryption algorithms used. Ensure they meet industry standards and resist known attacks, such as brute force or cryptographic attacks.
- Evaluate the performance under various conditions:
  - Measure encryption and decryption times for files of different sizes. Analyse how the tool performs on systems with varying hardware capabilities.

5.1.2 Features to be tested

Password setup

- User can set a valid password
- Error message is displayed when a user registers with an invalid password

Login

- User can log in with valid credentials and is taken to the main page of the program
- Error message is displayed when a user registers with invalid credentials

Exit program

- The program is designed to allow the user to exit by typing 'exit' explicitly.
- If the user closes the tab directly, the program is set up to exit accordingly.

Encrypt file

- User Input Handling: The user inputs the file path they want to encrypt within the program. Supports encryption for various file types: images, audio, documents, etc.

- File Encryption Process: The program encrypts the specified file once the valid file path is provided. Displays a confirmation message upon successful encryption: "File encrypted successfully."
- Error Handling for Invalid Paths:
    - If the user enters an incorrect or inaccessible file path, an error message is displayed.
    - The error message notifies the user about the incorrect path and prompts for a valid one.

Decrypt file

- The program allows the user to input a file path for decryption.
- Upon decryption, the program displays a message confirming that the file has been successfully decrypted.
- It supports decryption of various file types, such as images or audio files.
- In case of an incorrect file path entry, the program triggers an error message indicating the mistake to the user.

Encrypt folder

- The program facilitates the encryption of an entire folder located in the same directory where the program operates.
- Upon initiating the encryption process, the program traverses through the specified folder, encrypting each file within it.
- After successfully encrypting all files within the folder, the program prints a message informing the user that the entire folder has been encrypted.
- The encryption procedure ensures that all file types present within the folder, regardless of their formats (text, images, videos, etc.), are uniformly encrypted to maintain security.

Decrypt folder

- The program supports the decryption of an entire folder located in the program's directory.
- Upon triggering the decryption process, the program traverses through the specified folder, decrypting each encrypted file contained within it.

- Once all files within the folder have been successfully decrypted, the program displays a message indicating the completion of the folder decryption process.
- The decryption procedure ensures that all encrypted files within the folder, irrespective of their file types, are uniformly decrypted, restoring them to their original formats.

Types of files that can be encrypted

- Prepare a set of diverse file types, including text documents, images, audio files, video files, and other formats commonly used by your intended user base.
- Use the program to encrypt each of these files, ensuring that the encryption process works consistently across all file types.
- After encryption, confirm that the encrypted files retain their integrity and are not corrupted.
- Following successful encryption, decrypt these files and confirm that they are restored to their original formats without any loss of data or quality.

Sending encrypted files and decrypting files at the receiver end

- Encryption on Sender's System:
    - Encrypt the file using the encryption tool on the sender's system.
    - Ensure the encryption method used is compatible with the decryption capabilities of the receiving system.
- Secure File Transfer:
    - Use a secure and reliable method to transfer the encrypted file from the sender's system to the receiver's system. This could involve methods like secure file transfer protocols, encrypted email attachments, or secure cloud storage.
- Receiving and Decrypting on the Receiver's System:
    - Once the encrypted file is received on the other system, use the same encryption tool or a compatible one to decrypt the file.
    - Ensure that the decryption process successfully restores the original file without any loss or corruption of data.
- Verification:
    - Check the decrypted file on the receiver's end to confirm that it matches the original file from the sender's system.

    - Verify that the decryption process on the receiver's system works consistently for files shared from various sources.
- Cross-Platform Compatibility:
    - Ensure that the encryption tool used on both systems is compatible across different operating systems to facilitate smooth decryption.
- Handling Errors:
    - Test scenarios where the transfer might face interruptions or errors, and verify the file's integrity post-transfer and decryption.

5.1.3 Testing Approaches

The testing focus will Center on two key aspects of the file encryption tool. Firstly, the encryption functionality enables users to initiate encryption and manage keys. Secondly, the decryption interface allows users to decrypt various file types. Testing aims to ensure seamless transitions between these areas and validate functionalities like encryption, decryption, key management, and file handling across different tool sections or tabs.

5.1.4 Testing Methodology

Our testing philosophy aligns with industry-accepted principles, emphasizing meticulous planning, setting clear objectives, employing diverse testing methods like functional, performance, and security testing, conducting thorough validation processes, maintaining detailed documentation, and continuously refining through feedback and analysis. This approach ensures that every aspect of testing is thorough, effective, and meets industry standards.

5.1.5 Test Execution

During test execution, the crucial step involves physically running the tests against the system. It's imperative to verify that the observed outcomes align precisely with the expected results. Any deviations between the actual and expected results are flagged and channeled through a structured defect management process. This process involves documenting, prioritizing, and rectifying these disparities to ensure that the final product meets the desired specifications and quality standards.

5.2 Testing Scope and Environment

- Smoke Testing
    - Test the major functionalities for bugs. If 'must have' functionalities have defects, the testing phase will not start. The product would be sent back to the developer.
    - Entry Criteria: Major functionalities are added and working.
    - Exit Criteria: No Bugs found in testing
    - Environment: Test
- Integration Testing
    - The scope of integration testing is to ensure that specific functionalities work in sync with each other.
    - Prepare test cases for all the functionality where there is a linkage between modules.
    - Test the flow of data among different modules of the Application under test.
    - Entry Criteria: No bugs in smoke testing
    - Exit Criteria:
        - All the integration test cases have been executed.
        - No critical and Priority P1 & P2 defects are opened.
    - Environment: Test
- System Testing
    - The objectives of system tests are to test major business functionality, and ensure functional and quality requirements are met.
    - Ensure the system supports the use cases that have been defined
    - Entry Criteria: No critical or P1, P2 Priority bug is in open state.
    - Exit Criteria: No critical or Priority bugs should be in an open state
    - Environment: Test
- User Interface Testing
    - Ensure the presence and alignment of elements as defined in requirements.
    - Identify the elements for which User Interface testing has to be performed.
    - Prepare test cases for the presence and look of elements.

- Execute the test cases designed.
- Entry Criteria:
  - The system should have passed the exit criteria of system testing
  - No critical or Priority P1, P2 bug in an open state.
    - Exit Criteria: No critical or Priority bugs should be in an open state
    - Environment: Test

- Progression Testing
  - The objective should be to validate the functionality according to client requirements.
  - Identify progression scenarios.
  - Prepare test cases for progression.
  - Execute newly created test cases for the added requirement
  - Entry Criteria:
    - After the exit criteria for the smoke test are met.
    - No open P1, P2 defects open.
  - Exit Criteria:
    - All test cases are executed.
    - No critical or Priority P1, P2 bug in an open
  - Environment: Test
- Regression Testing
  - Test that no new defects are introduced after the addition of requirements.
  - Identify regression scenarios.
  - Prepare test cases.
  - Automate regression test cases for the previously present functionalities.
  - Entry Criteria: After progression testing no P1, or P2 open bugs found
  - Exit Criteria:
    - All previous bugs are tested.
    - No critical or Priority P1, P2 bug in an open state
- Environment: Test

- User Acceptance Testing
    - Receive sign-off from project manager, test lead, and development lead.
    - Set the pre-production environment.
    - Train the user on how to use the application.
    - Record the results and responses while a user performs testing
    - Entry Criteria: Sign off from Team Member.
    - Exit Criteria:
        - No critical defects open
        - The business process works satisfactorily
- Environment: Pre-Production

5.3 Defect management

The testing team recognizes that the bug-reporting process is a critical communication tool within the testing process. Without effective communication of bug information and other issues, the development and release process will be negatively impacted.

When logging a Bug, the tester should provide a priority to the defect to assist the developer in prioritizing the order in which they review and fix defects. Priority 1 and priority 2 defects must be addressed for acceptance (see definitions below).

1. Bug Priority: Critical

Priority = 1-Critical

Priority Definition: System down or a critical defect inhibiting the majority of users from performing key tasks with no workarounds. These incidents must be resolved before implementation.

Example: Form Tool defect preventing submission.

2. Bug Priority: Very Important

Priority = 2-High

Priority Definition: Important defect that significantly impacts the performance of a large number of users, but a workaround exists; a critical defect inhibiting a small number of users from performing key tasks with no workarounds. The incident requires resolution.

Example: Round-robin assignment logic defect that causes manual assignment workaround.

3. Bug Priority: Important

Priority = 3-Medium

Priority Definition: Enhancement that improves usability for the majority of users or a non[1]critical defect, with workarounds.

Example: Form does not format properly across less common devices/browsers.

4. Bug Priority: Nice to Have

Priority = 4-Low

Priority Definition: Enhancement that improves usability for a small set of users or is cosmetic; features that are not critical to the functionality of the solution.

Example: Incorrect capitalization, punctuation or font; tabbing order; size of text box does not correspond to max text length.

- Bug Triage
    - The Test candidate and Development candidate should all be involved in these triage meetings.
    - The Test Lead will provide the required documentation and reports on bugs for all attendees. The purpose of the triage is to determine the type of resolution for each bug and to prioritize and determine a schedule for all “To Be Fixed Bugs’. The development will then assign the bugs to the appropriate person for fixing and report the resolution of each bug back to Enrich.
    - The Test Lead will be responsible for tracking and reporting on the status of all bug resolutions.

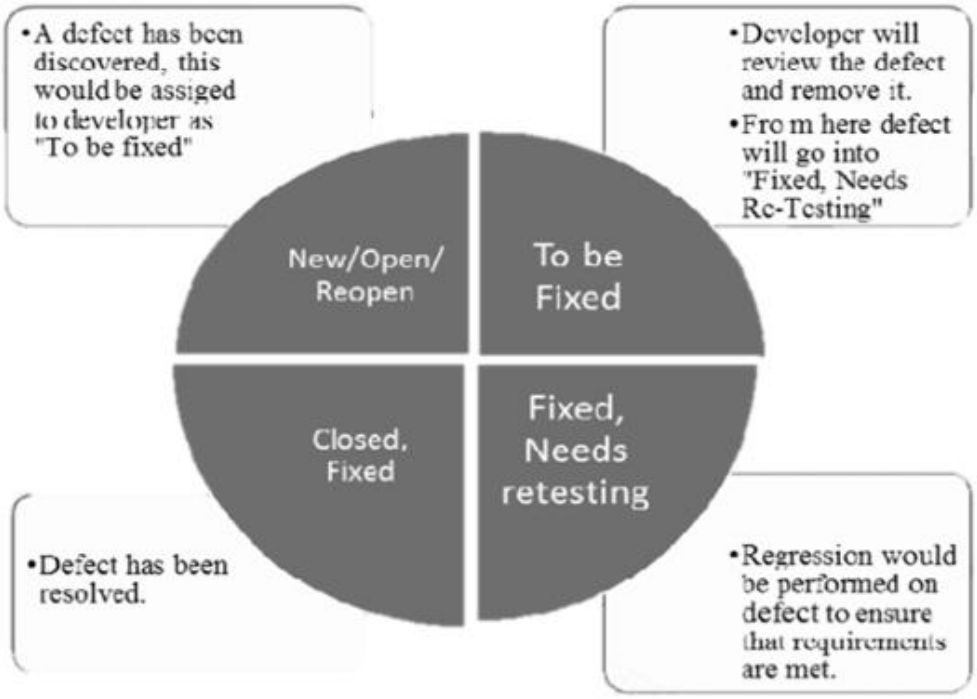


Fig.: 1

Bug Regression will be a central tenant throughout all testing phases. All bugs that are resolved as "Fixed, Needs Re-Testing" will be regressed when the testing team is notified of the new drop containing the fixes. When a bug passes regression it will be considered "Closed, Fixed". If a bug fails regression, the adopter's testing team will notify the development team by entering notes into notes. When a Priority 1 bug fails regression, the testing team should also put out an immediate email to development. The Test Lead will be responsible for tracking and reporting.

5.4 Criteria

- Entry Criteria
  - Testing would begin when Unit testing of developed modules/components is complete and no bugs were found in the smoke test.
- Suspension Criteria
  - Testing will be suspended when Smoke Test or Critical test case bugs are discovered
  - Testing will be suspended if there is a critical scope change that impacts the requirement

- Suspend testing until the Development team fixes all the failed cases.

- Exit Criteria
  - When the app is stable, the Team agrees that the application meets functional requirements.
  - Script execution of all the test cases in each area has passed.
  - All priority 1 and 2 bugs have been resolved and closed.

Each test area has been signed off as completed by the Test Lead.

5.5 Test Cases and Expected Results

Test	Ensure the user can set a password
Type	Integration
Pre-condition	Download essential Python packets and get the system ready.
Steps	▪ Enter a password ▪ Re-enter the password ▪ Show a message to restart the program to complete the setup or display an error message if any error occurs.
Expected result	The user should be able to set up a password and the program will prompt to restart the program.
Remarks	Test case Pass

Test	Ensure users can log in with valid credentials.
Type	Integration
Pre-condition	Restart the program after the initial setup
Steps	▪ Enter the password and hit enter ▪ Show an error message in case the password does not match.
Expected result	The user should be able to log in to the program and the program menu will appear in front of the user.
Remarks	Test case Pass

Test	Verify File Encryption Process.
Type	Integration
Pre-condition	Restart the program after the initial setup
Steps	▪ Enter the password and hit enter. ▪ Select the encrypt file option. ▪ Enter the path of the file to be encrypted. ▪ Show an error message if an incorrect file path is typed. ▪ Display a message that the file is encrypted after a successful encryption process has taken place.
Expected result	▪ Upon encryption completion, a confirmation message appears on the screen. ▪ The encrypted file is appended with a ".enc" suffix, and when accessed, its contents are not visible, ensuring secure encryption.
Remarks	Test case Pass

Test	Verify the File Decryption process.
Type	Integration
Pre-condition	Restart the program after the initial setup
Steps	▪ Enter the password and hit enter. ▪ Select the decrypt file option. ▪ Enter the path of the file to be decrypted. ▪ Show an error message if an incorrect file path is typed. ▪ Display a message that the file is decrypted after a successful decryption process has taken place.
Expected result	▪ The decryption process concludes successfully, accompanied by a displayed confirmation message. ▪ Post-decryption, the file content remains unchanged, ensuring integrity throughout the process.
Remarks	Test case Pass

Test	Verify Folder Encryption Process.
Type	Integration
Pre-condition	Restart the program after the initial setup
Steps	▪ Enter the password and hit enter. ▪ Select the encrypt folder option. ▪ Display a message that the folder is encrypted after a successful encryption process has taken place.
Expected result	▪ Upon encryption completion, a confirmation message appears on the screen. ▪ Every file within the folder undergoes encryption and is distinguished by the addition of a ".enc" extension at the end.
Remarks	Test case Pass

Test	Verify Folder Decryption Process.
Type	Integration
Pre-condition	Restart the program after the initial setup
Steps	▪ Enter the password and hit enter. ▪ Select the decrypt folder option. ▪ Display a message that the folder is decrypted after a successful decryption process has taken place.
Expected result	▪ Upon decryption completion, a confirmation message appears on the screen. ▪ Following decryption, the content of all files within the folder remains unaltered, ensuring the integrity of the data throughout the process.
Remarks	Test case Pass

Test	Verify types of files that can be encrypted and decrypted.
Type	Integration
Pre-condition	Restart the program after the initial setup
Steps	▪ Enter the password and hit enter. ▪ Select the encrypt option. ▪ Test on different types of files if the encryption process is working correctly or not (Eg: text file, image file, video file, mp3 files, etc.). ▪ Select the decrypt option ▪ Test on different types of files if the decryption process is working correctly or not (Eg: text file, image file, video file, mp3 files, etc.).
Expected result	▪ Users can encrypt a diverse range of file types including text documents, images, videos, and audio files using this tool. ▪ Users can decrypt a diverse range of file types including text documents, images, videos, and audio files using this tool.
Remarks	Test case Pass

Test	Verify Sending encrypted files and decrypting files at the receiver end.
Type	Integration
Pre-condition	Restart the program after the initial setup
Steps	▪ At the sender's end, Enter the password and press enter. ▪ Select the encrypt file option. ▪ Enter the path of the file to be encrypted. ▪ Show an error message if an incorrect file path is typed. ▪ Display a message that the folder is encrypted after a successful encryption process has taken place. ▪ Send the encrypted file to another computer. ▪ On the receiver side, Enter the password and hit enter. ▪ Select the decrypt file option. ▪ Enter the path of the file to be decrypted. ▪ Show an error message if an incorrect file path is typed. ▪ Display a message that the file is decrypted after a successful decryption process has taken place.

Expected result	Upon completion of the encryption process, the recipient should seamlessly decrypt the file, accessing the intended message without compromising its integrity.
Remarks	Test case Pass

Test	Verify exit mode
Type	Integration
Pre-condition	Restart the program after the initial setup
Steps	▪ Enter the password and hit enter. ▪ Select the exit option.
Expected result	The program allows users to exit smoothly without encountering any errors or disruptions.
Remarks	Test case Pass

RESULTS

RESULTS

As part of our project, we set out to develop a Python tool named 'File Encryption: Security that Matters,' focusing on security principles and ensuring secure communication. I'm thrilled to announce that our initial goals have been accomplished with resounding success.

The tool embodies a responsive design, offering seamless user interaction. Each functionality has been meticulously crafted and rigorously tested, guaranteeing its reliability and effectiveness in securing data and facilitating secure communication. Our commitment to security has been at the forefront throughout the development process, resulting in a robust and user-friendly tool.

The Python tool 'File Encryption: Security that Matters' encapsulates a comprehensive set of functionalities focused on data security. It begins with a robust password setup for access control, followed by secure login authentication. Users can encrypt and decrypt individual files and folders, ensuring data confidentiality through encryption algorithms. Additionally, the tool facilitates the secure sharing of encrypted files, enabling receivers to decrypt files securely. A vital aspect is a safe program exit, ensuring data integrity. Finally, the tool's versatility allows encryption and decryption of diverse file types, offering a holistic approach to data protection while maintaining user-friendliness and stringent security measures.

1) Setup Password: This involves establishing a secure password or passphrase that grants access to the tool or application. It's crucial to ensure the password creation process is robust, suggesting the use of strong, unique passwords that combine letters, numbers, and special characters.

2) Login with Valid Credentials: Once the password is set up, users can authenticate themselves by providing the correct credentials, typically a username and password combination. Validating these credentials against stored data ensures that only authorized users can access the functionalities of the tool.

3) Encrypt File: This function enables users to encrypt individual files using encryption algorithms. It converts the content of the file into an unreadable format, protecting its

data from unauthorized access. Options to choose encryption algorithms and methods can enhance security.

4) Decrypt File: This function allows authorized users to reverse the encryption process, converting the encrypted file back to its original, readable format. Proper authentication is essential before decryption to maintain security.

5) Encrypt Folder: Similar to encrypting a file, this functionality extends encryption to entire folders. It secures all files within a designated folder using encryption techniques, adding an extra layer of protection to multiple files simultaneously.

6) Decrypt Folder: This reverses the encryption process for an entire folder, restoring the encrypted files back to their original, accessible state. Proper authentication and access rights are crucial before decryption occurs.

7) Send an Encrypted File and Decrypt at the Receiver Side: This feature facilitates secure file sharing by allowing users to encrypt a file and then securely transmit it to another party. The receiver, using the appropriate credentials and decryption keys, can decrypt and access the file.

8) Encrypt and Decrypt Any Type of Files: This broadens the scope of file compatibility for encryption and decryption, allowing users to secure various file formats and types, ensuring comprehensive data protection.

9) Exit Program Safely: This ensures a proper shutdown of the application, securely closing all active sessions, releasing resources, and maintaining data integrity. It's crucial for preventing unauthorized access or data loss.

Each of these functionalities, when implemented securely and intuitively, contributes to the overall robustness and user-friendliness of the file encryption tool, ensuring data security and seamless user experience.

SUMMARY AND CONCLUSIONS

SUMMARY AND CONCLUSIONS

The File Encryption Cybersecurity Project represents a pivotal initiative aimed at fortifying digital data security through advanced encryption mechanisms. In a world where the exponential growth of digital information is accompanied by increasingly sophisticated cyber threats, the need for robust file protection measures has never been more pronounced.

The project's overarching goal is to develop an innovative file encryption system that goes beyond conventional methods. It seeks to address the shortcomings of existing encryption techniques by integrating cutting-edge cryptographic algorithms, a user-friendly interface, and a secure key management framework. Through a multi-faceted approach, the system aims to ensure the confidentiality and integrity of files, safeguarding sensitive information from unauthorized access and potential tampering.

The project begins with a comprehensive analysis of existing encryption methodologies, identifying strengths and weaknesses. Building upon this foundation, it introduces a multi-layered encryption strategy that combines both symmetric and asymmetric cryptographic algorithms. This approach not only enhances the security posture of the system but also positions it as adaptable to emerging cybersecurity threats.

User experience is a central consideration in the project, with the development of an intuitive graphical user interface (GUI) that simplifies the encryption and decryption processes. Recognizing that security measures are most effective when seamlessly integrated into user workflows, the interface aims to make robust file encryption accessible to a wide range of users.

Key management, often a vulnerable point in encryption systems, is a critical focus of the project. The implementation of a secure key management framework aligns with industry best practices, ensuring that cryptographic keys are generated, distributed, and stored in a manner that enhances overall system security.

Testing and evaluation are integral components of the project, validating the effectiveness and resilience of the enhanced file encryption system. Rigorous assessments encompass resistance to common cyber threats, performance benchmarks, and compatibility with diverse file types and sizes. The outcomes of these tests will contribute to the project's overall success and provide stakeholders with confidence in the system's capabilities.

Conclusions:

In conclusion, the File Encryption Cybersecurity Project represents a significant stride forward in the realm of data security. By combining technological innovation, user-centric design principles, and stringent security measures, the project aims to redefine the standards for file protection in the face of evolving cyber threats.

The proposed file encryption system not only addresses current vulnerabilities but also anticipates future challenges, positioning itself as a proactive defines against a dynamic threat landscape. The multi-layered encryption strategy, intuitive user interface, and robust key management framework collectively contribute to the project's overarching objective of establishing a new paradigm in file security.

As the project progresses, collaboration with cybersecurity experts, user feedback, and adherence to industry standards will be paramount. Ongoing refinement based on real-world scenarios and emerging threats will be critical to ensuring the system's continued efficacy in safeguarding sensitive information.

In essence, the File Encryption Cybersecurity Project aspires to empower individuals and organizations with a comprehensive and accessible solution for securing their digital assets. By combining state-of-the-art technology with user-friendly design and a commitment to rigorous security practices, the project is poised to make a lasting impact on the landscape of digital security. As the system advances from development to deployment, it is expected to contribute significantly to the collective efforts in creating a more secure and resilient digital environment.

REFERENCES

REFERENCES

[1] Bhargav, S., Majumdar, A., & Ramudit, S. (2008, Spring). 128-bit AES decryption. Retrieved November 21, 2020, from http://www.cs.columbia.edu/~sedwards/classes/2008/4840/reports /AES.pdf

[2] Clark, A. (2018, August 2). How much encryption is too much: 128, 256, or 512-bit? Retrieved November 21, 2020, from https://discover.realvnc.com/blog/ how-much-encryption-is-too-much-128-256-or-512-bit

[3] Hoang Trang and Nguyen Van Loi, "An Efficient FPGA Implementation of The Advanced Encryption Standard algorithm", IEEE International Conference on Computing and Communication Technology, pages 1-4, Ho Chi Minh City, 2012.

[4] Kamali S.H, Shakerian R, Hedayati M, and Rahmani M, "A new modified version of Advanced Encryption Standard based algorithm for image encryption", (ICEIE) International Conference On Electronics and Information Engineering, volume 1, page 1250-1255, Aug 2010

[5] AES Encryption: study and Evaluation (PDF) AES Encryption: Study & Evaluation (researchgate.net)

[6] What is file Encryption: (https://www.ssh.com/academy/encryption/what-is-file-encryption)

[7] Thakkar, J. (2020, June 2). DES vs. AES: Everything to Know About AES 256 and DES Encryption. Retrieved November 21, 2020, from https://sectigostore.com/blog/des-vs-aes-everything-to-know-about-aes-256-and-des-encryption/

[8] Townsend Security. (2020, June 1). AES vs. DES Encryption: Why AES has replaced DES, 3DES, and TDEA. Retrieved November 21, 2020, from https://www.precisely.com/blog/data-security/ aes-vs-des-encryption-standard-3des-tdea

[9] Mustafeez, A. Z. (n.d.). What is the AES algorithm? Retrieved November 20, 2020, from https://www.educative.io/edpresso/what-is-the-aes-algorithm

[10] M. Y. Rhee, \"Internet Security Cryptographic Principles, Algorithms and Protocols\", John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, England, 2003.

[11] O. A. Dawood, A. S. Rahma, and A. J. Abdul Hossein, "The New Block Cipher Design (Tigris Cipher)", I.J.Computer Network and Information Security (IJCNIS).

[12] William, "Cryptography and Network Security Principles and Practice", Fifth Edition, Pearson Education, Prentice Hall, 2011

[13] Schneier B., "Applied Cryptography", John Wiley& Sons Publication, New York, 1994.

[14] Agrawal Monika, Mishra Pradeep, "A Comparative Survey on Symmetric Key Encryption Techniques", International Journal on Computer Science and Engineering (IJCSE), Vol. 4 No. 05 May 2012, pp. 877-882.

[15] Seth ShashiMehrotra, Mishra Rajan, "Comparative analysis of Encryption algorithm for data communication", International Journal of Computer Science and Technology, vol. 2, Issue 2, June 2011, pp. 292-294.

[16] AlamMd Imran, Khan Mohammad Rafeek. "Performance and Efficiency Analysis of Different Block Cipher Algorithms of Symmetric Key Cryptography", International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 10, October 2013, pp.713-720.

[17] MandalPratap Chandra, "Superiority of Blowfish Algorithm" IJARCSSE, volume 2, Issue 9, September 2012, pp. 196-201

[18] MarwahaMohit, Bedi Rajeev, Singh Amritpal, Singh Tejinder, "Comparative Analysis of Cryptographic Algorithms", International Journal of Advanced Engineering Technology/IV/III/July-Sep, 2013/16-18.

[19] Abdul D.S, Kader H.M Abdul, Hadhoud, M.M., "Performance Evaluation of Symmetric Encryption Algorithms", Communications of the IBIMA, Volume 8, 2009, pp. 58-64.

[20] Apoorva, Kumar Yogesh, "Comparative Study of Different Symmetric Key Cryptography", IJAIEM, vol. 2, Issue 7, July 2013, pp. 204-206.

[21] W. Stallings; "Cryptography and Network Security" 2nd Edition, Prentice Hall,1999

[22] Bruce Schneir: Applied Cryptography, 2nd edition, John Wiley & Sons, 1996

[23] A. Kakkar and P. K. Bansal, "Reliable Encryption Algorithm used for Communication", M. E. Thesis, Thapar University, 2004.

[24] N. Koblitz, “Elliptic Curve Cryptosystems”, Journal of Mathematics of Computation. Published by American Mathematical Society, Vol. 48, No. 177, pp. 203-209, 1987.

[25] H. Krawczyk, “The Order of Encryption and Authentication for Protecting Communications, http://eprint.iacr.org/2001.

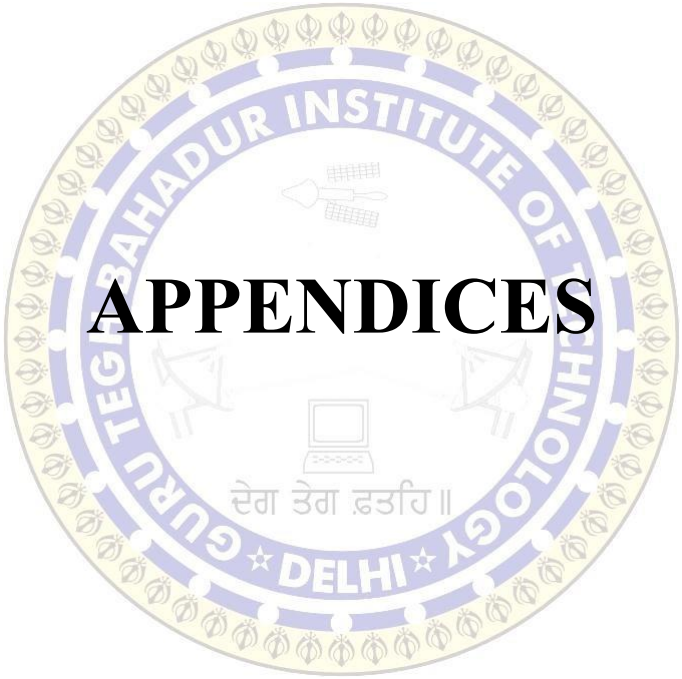
[26] L. Eschenauer, V. D. Gligor, “A Key Management Scheme for Distributed Sensor Networks”, ACM conference on Computer Security, Vol.2, pp. 41-47, 2002.

[27] Jung. W. Lo, M. S. Hwang, C. H. Liu, “An efficient key assignment scheme for access control in a large leaf class hierarchy”, Journal of Information Sciences Elsevier Science, Vol. 4, pp. 917-925, 2003.

[28] B. B. Madan, K. G. Popstojanova, K. Vaidyanathan and K.S. Trivedi, “A Method for Modeling and Quantifying the Security Attributes of Intrusion Tolerant Systems”, Journal of Performance Evaluation, Elsevier Science Publishers, Vol. 56, No. 1, pp. 167-186, 2004.

[29] T. Jamil, “The Rijndael Algorithm”, IEEE Potential, Vol.1, pp. 1-4, 2004.

[30] H.W. Kim, S. Lee, “Design and Implementation of a Private and Public Key Crypto Processor and Its Application to a Security System”, IEEE Transactions on Consumer Electronics, Vol. 50, No. 1, pp. 214-224, 2004.

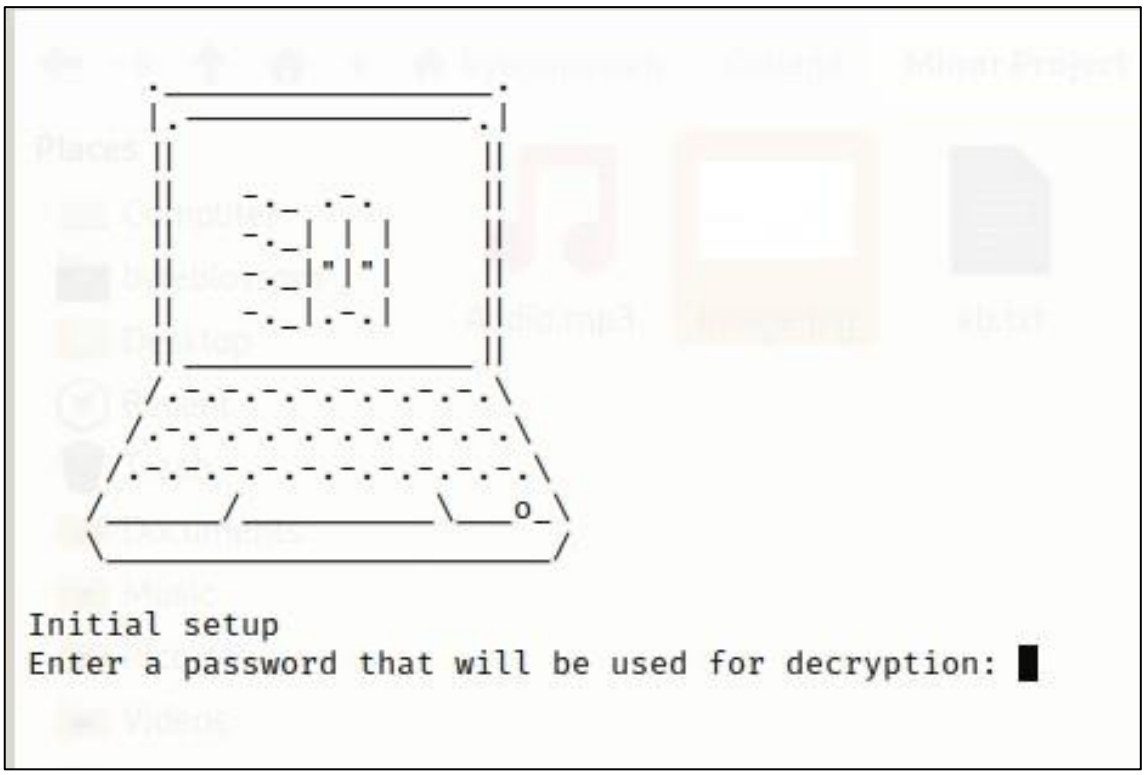


APPENDICES

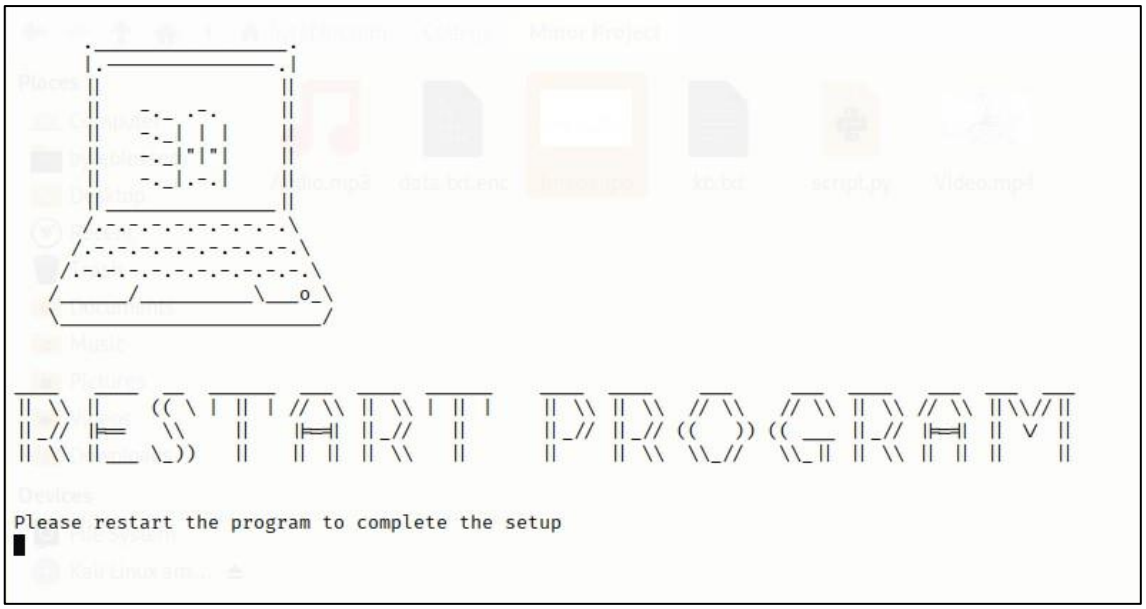
APPENDIX – A
SCREEN SHOTS

Basic Setup -

```
┌──(byteblossom㉿kali)-[~/College/Minor Project]
└─$ python3 script.py
```

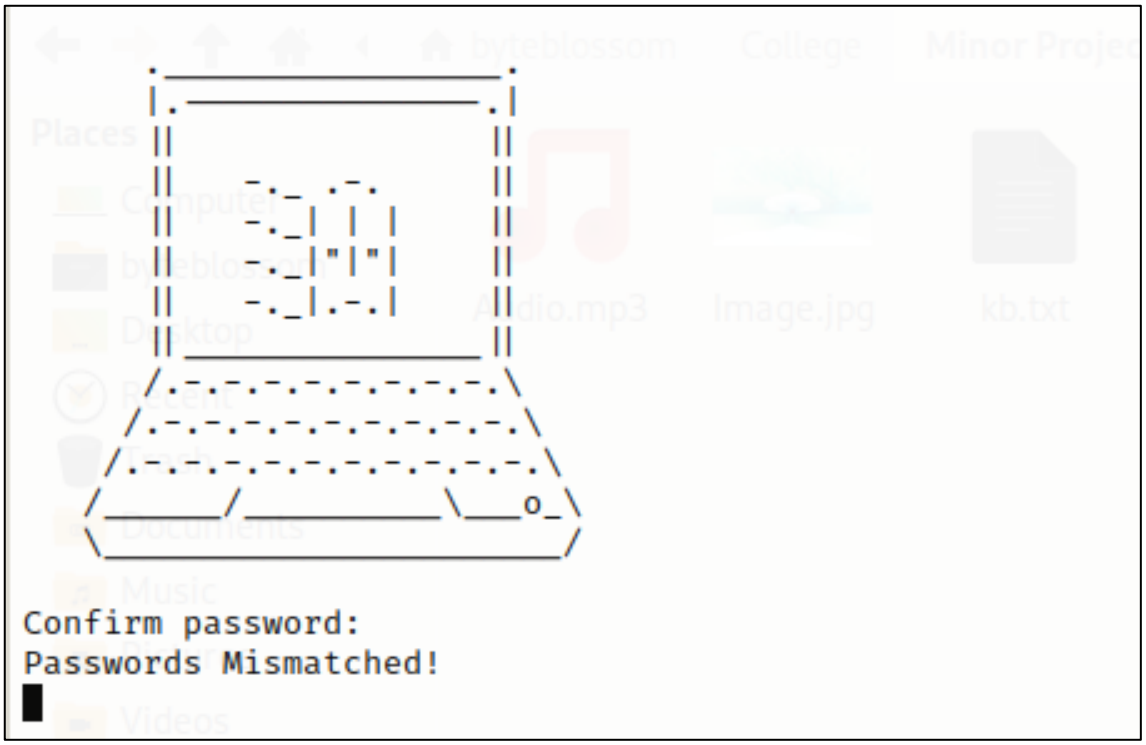


```
Confirm password:
```

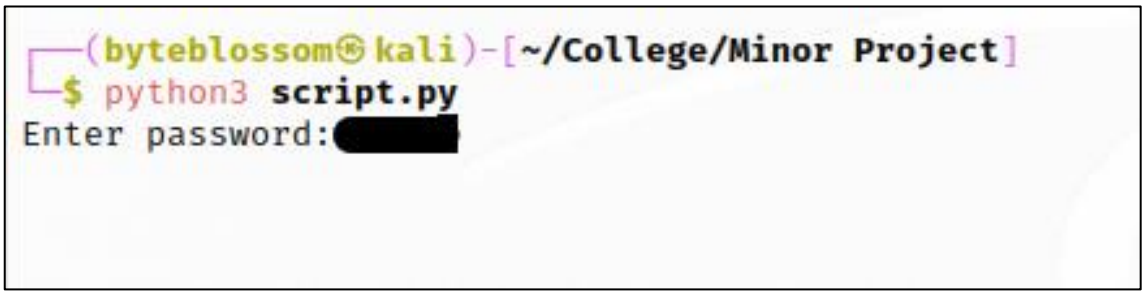


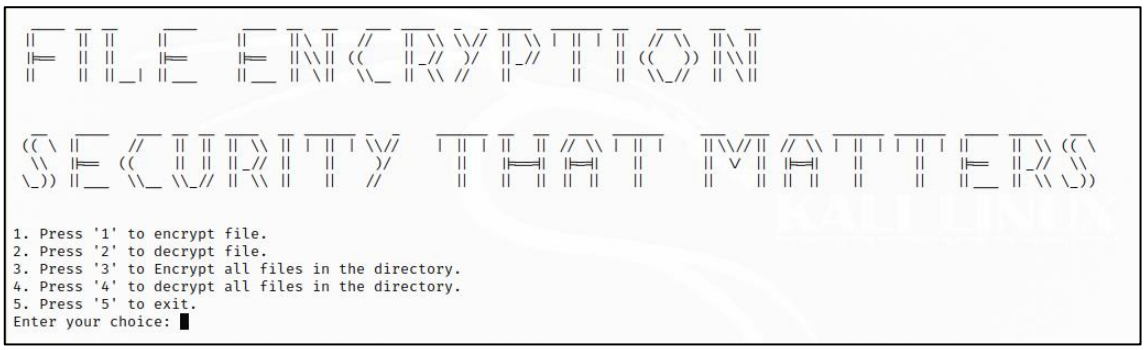
```
Please restart the program to complete the setup
```

Setup Password Mismatch –

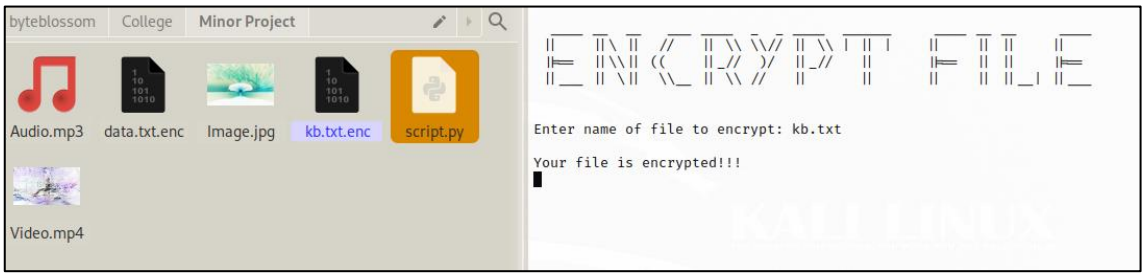


Initial Running -

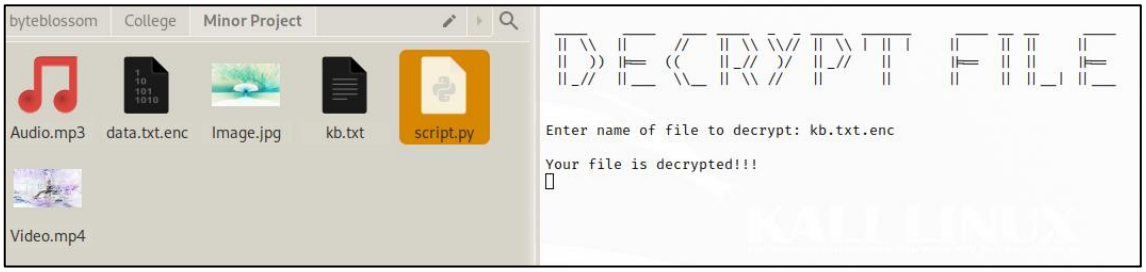




Encryption of Text File -

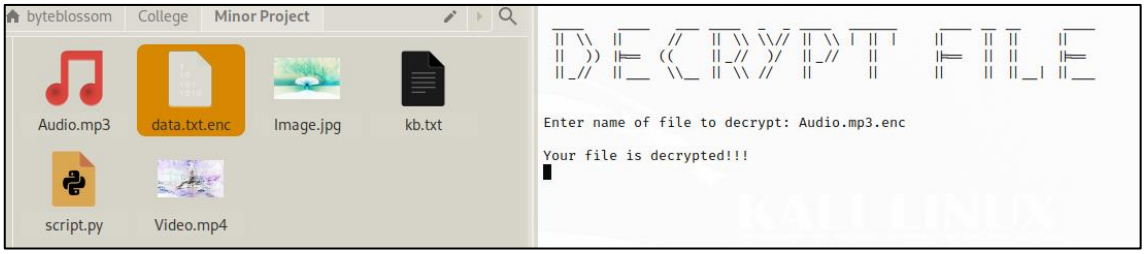


Decryption of Text File -

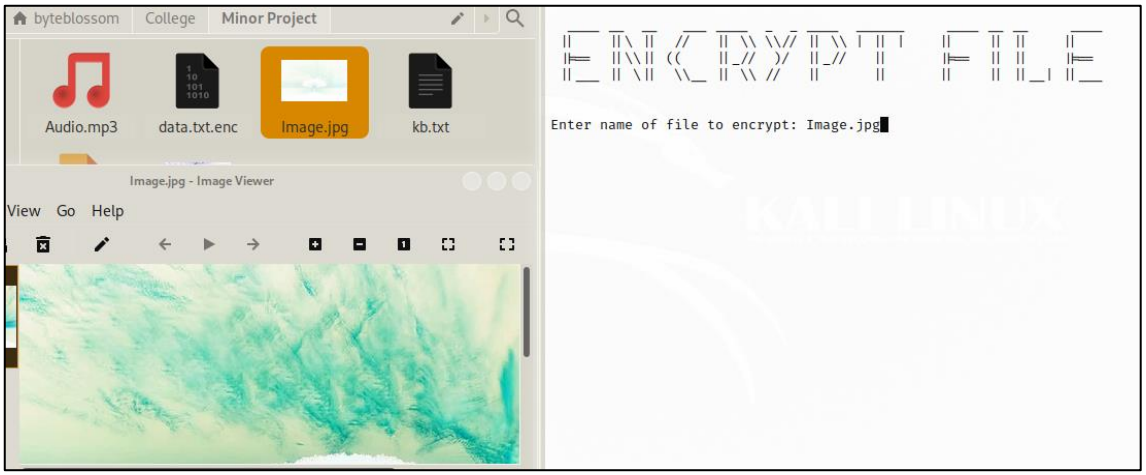


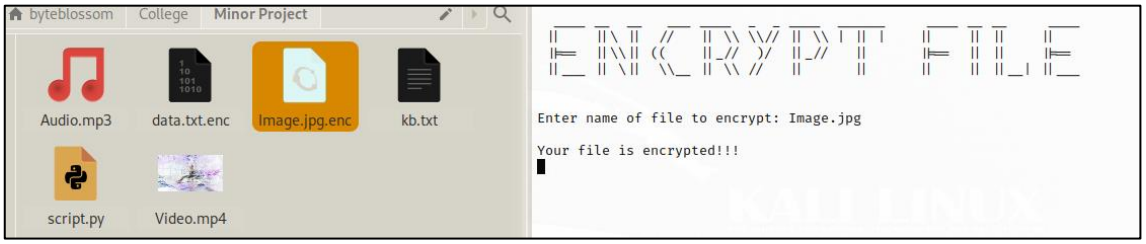
Encryption of Audio File -

Decryption of Audio File –

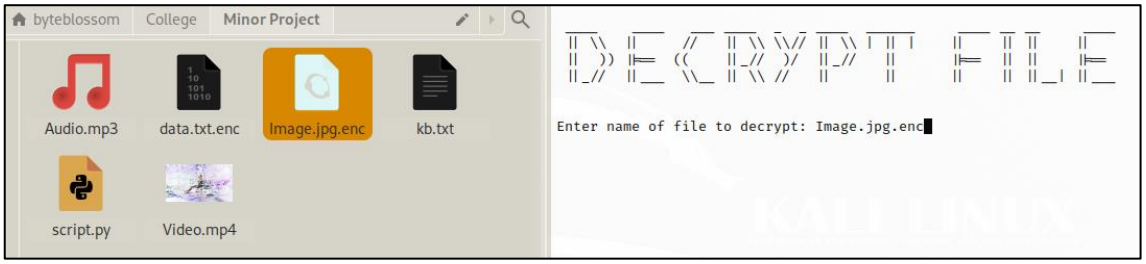


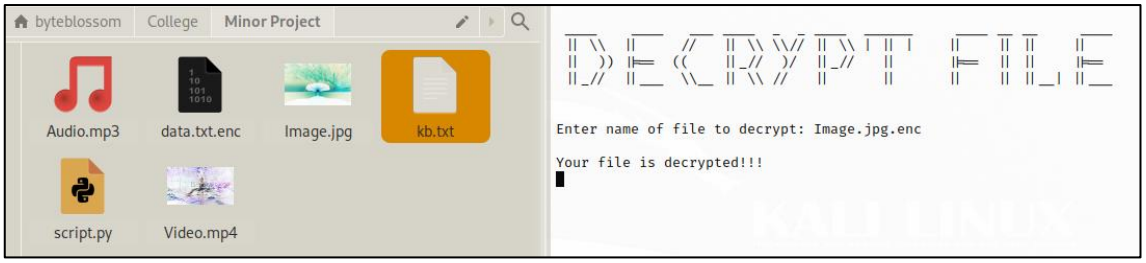
Encryption of Image File –



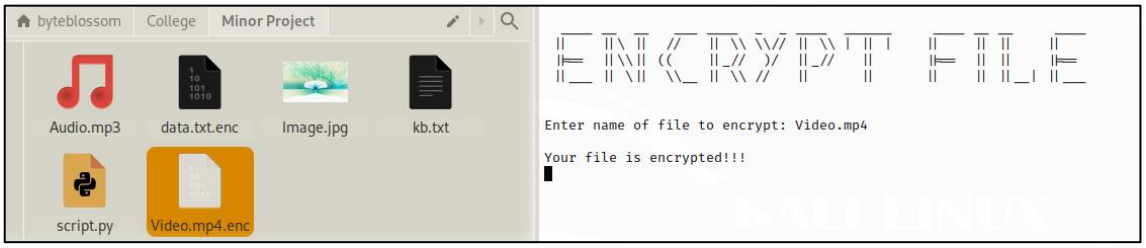


Decryption of Image File –

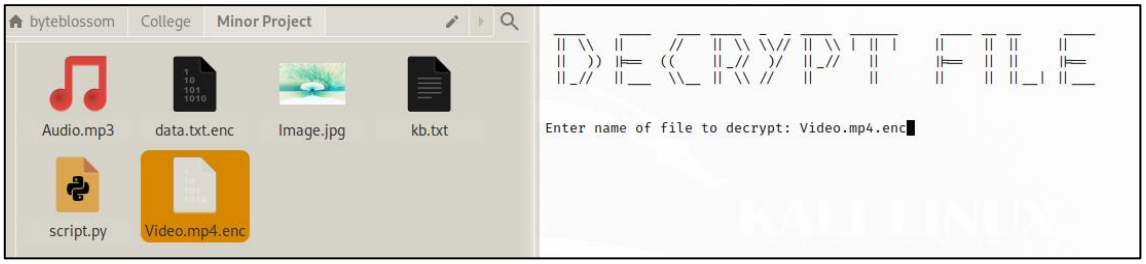


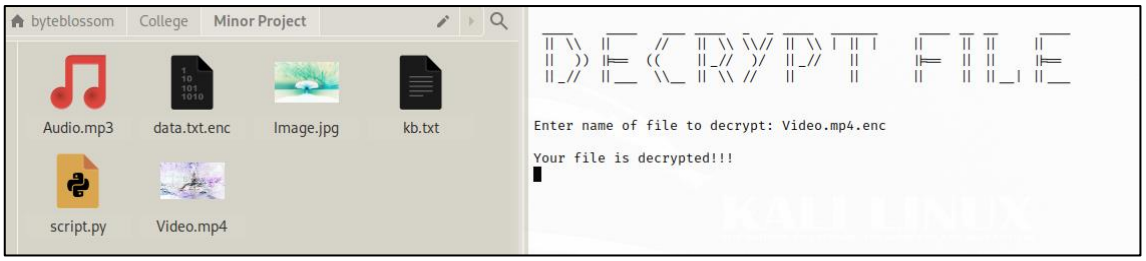


Encryption of Video File –

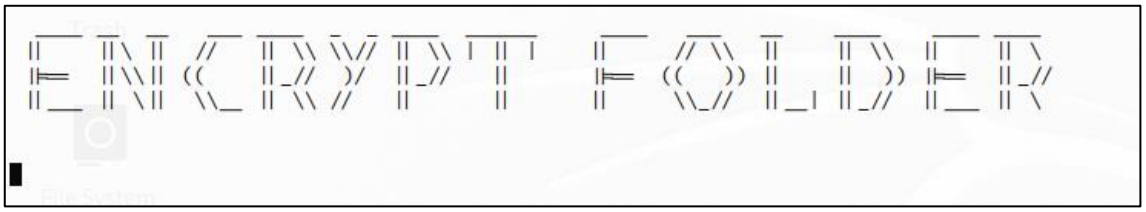


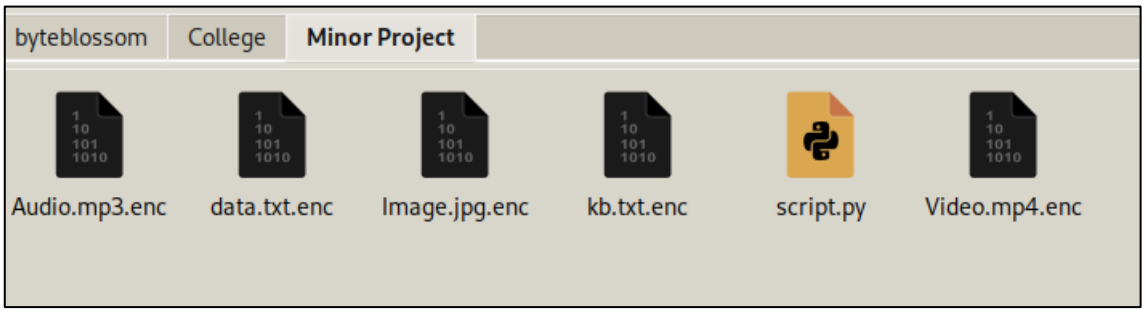
Decryption of Video File –



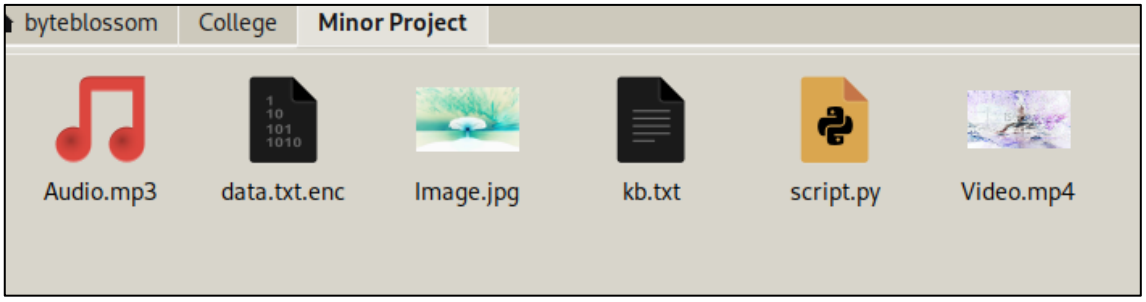


Encryption of Folder –

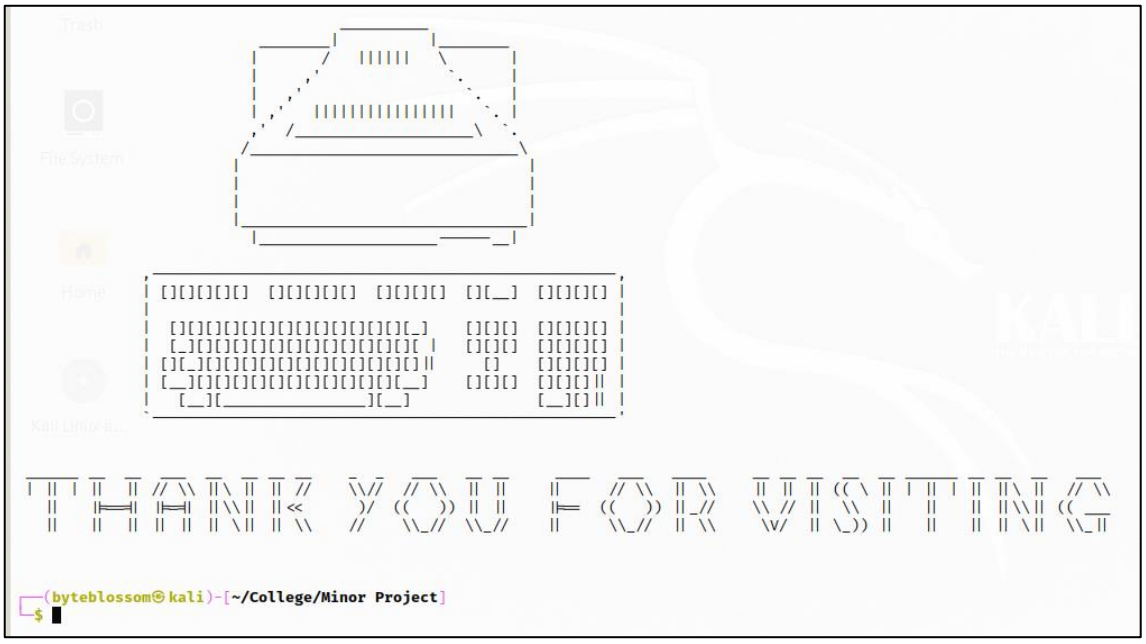


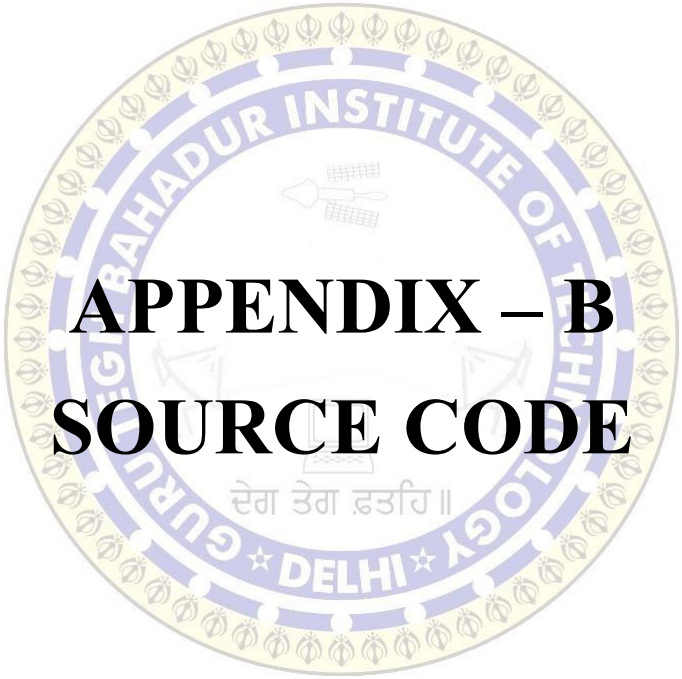


Decryption of Folder –



Ending of the Tool –





APPENDIX – B
SOURCE CODE

```
#!/usr/bin/python3

from Crypto import Random
from Crypto.Cipher import AES
import os
import os.path
from os import listdir
from os.path import isfile, join
import time
from getpass import getpass

logo1 = '''
 ____ __ __    _____    ________ ____ ____ _ _____ _______ __  ____  __ __
||   || ||   ||      ||   ||\ ||  //   || \\\ \\V/ || \\\ | || | ||  // \\\  ||\ ||
||== || ||   ||==    ||== ||\\|| ((   ||_//  )/  ||_//   ||  || ((   )) ||\\||
||   || ||__| ||___  ||___ || \|| \\\__ || \\\ //  ||     ||  ||  \\\_// || \||
'''
logo2= '''
 __  ____  ___ __ ___ ________ ______ _ _  ________ ____ __ ______  ___ ___
___ _______ ______ ____ _____ __
 (( \ ||    //  || || || \\\ || | || | \\V/  | || | || || // \\\ | || |   ||\\V/|| // \\\ | || | | || | ||    || \\\ (( \\
 \\\ ||== ((   || || ||_// ||  ||   )/     ||  ||==|| ||=||  ||     || V || ||=||  ||    ||  ||== ||_//  \\\
 \_)) ||___  \\\__ \\\_// || \\\ ||  ||  //      ||  || || || ||  ||     ||   || || ||  ||    ||  ||___ || \\\ \_))

'''
logo3 = '''

                    ___________
               _________|         |_________
              |      /   |||||   \      |
              |     ,'              `.    |
              |   ,'                  `.  |
              | ,'   |||||||||||||||   `. |
```

```
                ,' /_____________________\ `.
               /_____________________________\\
              |                               |
              |                               |
              |                               |
              |_______________________________|
               |_____________________------__|


          ,-----------------------------------------------,
          |[][][][][] [][][][][] [][][][] [][__] [][][][]|
          |                                               |
          | [][][][][][][][][][][][_] [][][] [][][][]|
          | [_][][][][][][][][][][][| [][][] [][][][]|
          |[][_][][][][][][][][][][]||   [] [][][][]|
          |[__][][][][][][][][][][__] [][][] [][][]||
          |  [__][_____________][__]        [__][]||
          `-----------------------------------------------'
'''

logo4 = '''
 ____ __ __  ___ ____ _ _ ____ ____ ____ ____ ____
||  ||\ ||  //  || \\\ \\\/ || \\\ | || |  ||  || ||  ||
||== ||\\\|| ((  ||_//  )/  ||_//  ||  ||== || ||  ||==
||___ || \||  \\\__ || \\\ //  ||  ||  ||  || ||__| ||___

'''

logo5 = '''
 ____  ____  ___ ____ _ _ ____ ____ ____ ___ __ __  ____
|| \\\ ||  //  || \\\ \\\/ || \\\ | || |  ||  || ||  ||
||  )) ||== ((  ||_//  )/  ||_//  ||  ||== || ||  ||==
||_//  ||___  \\\__ || \\\ //  ||  ||  ||  || ||__| ||___
```

'''

logo6 = '''

 _____ __ __   ___ _____ _ _ _____ _______   _____   ___   __   _____   _____ _____
|| ||\ || // || \\\ \\\/ || \\\ | || | || // \\\ || || \\\ || || \\
||== ||\\\|| ((||_//)/ ||_// || ||== ((   )) || ||)) ||== ||_//
||___ || \|| __ || \\\ // || || || _// ||__| ||_// ||___ || \\

'''

logo7 = '''

 _____   _____   ___ _____ _ _ _____ _______   _____   ___   __   _____   _____ _____
|| \\\ || // || \\\ \\\/ || \\\ | || | || // \\\ || || \\\ || || \\
||)) ||== ((||_//)/ ||_// || ||== ((   )) || ||)) ||== ||_//
||_// ||___ __ || \\\ // || || || _// ||__| ||_// ||___ || \\

'''

logo8 = '''

 _______ __ __   ___   __ __ _____ _____ _ _   _____   ___   _____   __ __ __
__ __ _______ __ __ __   ___
| || | || || // \\\ ||\ || || //   \\\// // \\\ || || || // \\\ || \\\ || || || ((\ || | || | || ||\ || // \\\
  || ||==|| ||=|| ||\\\|| ||<<   )/ ((   )) || || ||== ((   )) ||_//   \\\ // || \\\ || || || ||\\\|| ((___
  || || || || || || \\|| || \\\   //   _// _// || _// || \\\   \V/ || _)) || || || || \|| _||

'''

logo9 = '''

    .______________________.
    |.----------------.|
    || ||

```
    ||   -._ .-.     ||
    ||   -._| | |    ||
    ||   -._|"|"|    ||
    ||   -._|.-.|    ||
    ||_________________||
    /.-.-.-.-.-.-.-.-.\\
   /.-.-.-.-.-.-.-.-.-.\\
  /.-.-.-.-.-.-.-.-.-.-.\\
 /______/__________\\___o_\\
 \\_________________________/
'''


logo10 = '''
____   ____  __ _______ ____  ____  ______    ____  ____    ___     ___  ____   ___
___  ___
|| \\\ ||    (( \\ | || | // \\\ || \\\ | || |    || \\\ || \\\  // \\\   // \\\ || \\\ // \\\ ||\\V/||
||_// ||==   \\\    ||   ||=|| ||_//   ||     ||_// ||_// ((   )) (( ___ ||_// ||=|| || \\V ||
|| \\\ ||___ \\_))   ||   || || || \\\   ||     ||   || \\\  \\\_//   \\\_|| || \\\ || || ||    ||

'''
class Encryptor:
    def __init__(self, key):
        self.key = key

    def pad(self, s):
        return s + b"\0" * (AES.block_size - len(s) % AES.block_size)

    def encrypt(self, message, key, key_size=256):
        message = self.pad(message)
        iv = Random.new().read(AES.block_size)
        cipher = AES.new(key, AES.MODE_CBC, iv)
        return iv + cipher.encrypt(message)
```

```
def encrypt_file(self, file_name):
    with open(file_name, 'rb') as fo:
        plaintext = fo.read()
    enc = self.encrypt(plaintext, self.key)
    with open(file_name + ".enc", 'wb') as fo:
        fo.write(enc)
    os.remove(file_name)

def decrypt(self, ciphertext, key):
    iv = ciphertext[:AES.block_size]
    cipher = AES.new(key, AES.MODE_CBC, iv)
    plaintext = cipher.decrypt(ciphertext[AES.block_size:])
    return plaintext.rstrip(b"\0")

def decrypt_file(self, file_name):
    with open(file_name, 'rb') as fo:
        ciphertext = fo.read()
    dec = self.decrypt(ciphertext, self.key)
    with open(file_name[:-4], 'wb') as fo:
        fo.write(dec)
    os.remove(file_name)

def getAllFiles(self):
    dir_path = os.path.dirname(os.path.realpath(__file__))
    dirs = []
    for dirName, subdirList, fileList in os.walk(dir_path):
        for fname in fileList:
            if (fname != 'script.py' and fname != 'data.txt.enc'):
                dirs.append(dirName + "/" + fname)
    return dirs
```

```
    def encrypt_all_files(self):
        dirs = self.getAllFiles()
        for file_name in dirs:
            self.encrypt_file(file_name)

    def decrypt_all_files(self):
        dirs = self.getAllFiles()
        for file_name in dirs:
            self.decrypt_file(file_name)


key = b'[EX\xc8\xd5\xbfI{\xa2$\x05(\xd5\x18\xbf\xc0\x85)\x10nc\x94\x02)j\xdf\xcb\xc4\
x94\x9d(\x9e'
enc = Encryptor(key)
clear = lambda: os.system('clear')

if os.path.isfile('data.txt.enc'):
    while True:
        password = getpass(prompt = "Enter password: ")
        enc.decrypt_file("data.txt.enc")
        p = ''
        with open("data.txt", "r") as f:
            p = f.readlines()
        if p[0] == password:
            enc.encrypt_file("data.txt")
            break

    while True:
        clear()
        print(logo1)
        print(logo2)
        choice = int(input(
```

```
"1. Press '1' to encrypt file.\n2. Press '2' to decrypt file.\n3. Press '3' to Encrypt all files in the directory.\n4. Press '4' to decrypt all files in the directory.\n5. Press '5' to exit.\nEnter your choice: "))
    clear()
    if choice == 1:
        print(logo4)
        enc.encrypt_file(str(input("Enter name of file to encrypt: ")))
        print("\nYour file is encrypted!!!")
        time.sleep(5)
    elif choice == 2:
        print(logo5)
        enc.decrypt_file(str(input("Enter name of file to decrypt: ")))
        print("\nYour file is decrypted!!!" )
        time.sleep(3)
    elif choice == 3:
        print(logo6)
        time.sleep(5)
        enc.encrypt_all_files()
        print("\nAll files in the folder are now encrypted!!!")
        time.sleep(3)
    elif choice == 4:
        print(logo7)
        time.sleep(5)
        enc.decrypt_all_files()
        print("\nAll files in the folder are now decrypted!!!")
        time.sleep(3)
    elif choice == 5:
        print(logo3)
        print(logo8)
        exit()
    else:
        print("Please select a valid option!")
```

```
else:
    while True:
        clear()
        print(logo9)
        print("Initial setup")
        password = getpass(prompt = "Enter a password that will be used for decryption:
")
        clear()
        print(logo9)
        repassword = getpass(prompt ="Confirm password: ")
        if password == repassword:
            break
        else:
            print("Passwords Mismatched!")
            time.sleep(5)
    f = open("data.txt", "w+")
    f.write(password)
    f.close()
    enc.encrypt_file("data.txt")
    clear()
    print(logo9)
    print(logo10)
    print("Please restart the program to complete the setup")
    time.sleep(5)
```