A white background with blue text

Description automatically generated

A close-up of a text

Description automatically generated

A white paper with black text

Description automatically generated

Contents

[Abstract: 2](#_Toc142608102)

[Objectives: 2](#_Toc142608103)

[Introduction 5](#_Toc142608104)

[Algorithm 6](#_Toc142608105)

[The Distribution: 8](#_Toc142608106)

[Vault\_cli.py: 8](#_Toc142608107)

[Vault\_gui.py 11](#_Toc142608108)

[Main.py 14](#_Toc142608109)

[Execution 16](#_Toc142608110)

[Conclusion 25](#_Toc142608111)

# Table of Figures:

[Figure 1 16](#_Toc142608403)

[Figure 2 17](#_Toc142608404)

[Figure 3 18](#_Toc142608405)

[Figure 4 18](#_Toc142608406)

[Figure 5 18](#_Toc142608407)

[Figure 6 18](#_Toc142608408)

[Figure 7 18](#_Toc142608409)

[Figure 8 19](#_Toc142608410)

[Figure 9 19](#_Toc142608411)

[Figure 10 19](#_Toc142608412)

[Figure 11 19](#_Toc142608413)

[Figure 12 19](#_Toc142608414)

[Figure 13 20](#_Toc142608415)

[Figure 14 21](#_Toc142608416)

[Figure 15 21](#_Toc142608417)

[Figure 16 21](#_Toc142608418)

[Figure 17 22](#_Toc142608419)

[Figure 18 22](#_Toc142608420)

[Figure 19 23](#_Toc142608421)

[Figure 20 23](#_Toc142608422)

[Figure 21 23](#_Toc142608423)

[Figure 22 23](#_Toc142608424)

[Figure 23 24](#_Toc142608425)

# Introduction

The combination of Python scripts, namely "main.py," "vault\_cli.py," and "vault\_gui.py," introduces a robust file vault application. Users are given the choice between two interaction methods: the Command-Line Interface (CLI) or the Graphical User Interface (GUI). This introduction sheds light on the essence of these scripts and how they work together.

"main.py" serves as the starting point. It welcomes users and lets them decide whether to use CLI or GUI. This decision leads to distinct paths, tailored to different user preferences and technical familiarity.

Opting for CLI triggers "vault\_cli.py," handling text-based interactions in the command-line. Users can perform tasks like registering, logging in, uploading files, viewing content, and deleting items. The CLI's simplicity makes navigation easy, encapsulating features in straightforward commands.

For a more visual experience, "vault\_gui.py" comes into play. With the "tkinter" library, it creates an interactive graphical environment. Users can register, login, manage files, and view content through buttons and visual displays.

Security is a priority, regardless of the chosen mode. User credentials are securely stored in an SQLite database, while encryption keys generated with Fernet cryptography enhance data security.

In essence, the collaboration of these scripts showcases Python's versatility. By offering different interface options, this application bridges technology and user comfort. The harmony between CLI's efficiency and GUI's visual appeal demonstrates Python's power in making technology accessible.

In summary, the combination of "main.py," "vault\_cli.py," and "vault\_gui.py" marks a user-centered approach to secure file management. By providing choices for interaction, this trio exemplifies Python's adaptability in simplifying tasks and personalizing technology.

# Algorithm

1. Execution Start:
   1. User runs `main.py` script.
2. Welcome and Interface Choice:
   1. Display a welcome message.
   2. Prompt user to choose between CLI (1) or GUI (2) interaction.
3. CLI Interaction:
   1. If user chooses CLI:
      1. Execute `vault\_cli.py`.
4. CLI Mode - User Actions:
   1. Initialize session variables.
   2. Connect to the SQLite database.
   3. Loop:
      1. Display main menu with options: Register, Login, Quit.
      2. Get user's choice.
      3. If Register:
         * Prompt user for username and password.
         * Hash the password and insert data into the database.
      4. If Login:
         * Prompt user for username and password.
         * Authenticate user from the database.
         * If authenticated, display secondary menu: Upload, View, Delete, Logout.
      5. If Upload:
         * Ask user for file path.
         * Generate user-specific key.
         * Encrypt and move the file to the user's directory.
      6. If View:
         * Display a list of files in the user's vault.
         * Ask user to choose a file to view.
         * Decrypt and display the selected file's content.
      7. If Delete:
         * Confirm user's decision.
         * Delete all files from the user's directory except the key.
      8. If Logout:
         * Clear session variables and return to the main menu.
      9. If Quit:
         * Exit the loop and the program.
5. GUI Interaction:
   1. If user chooses GUI:
      1. Execute `vault\_gui.py`.
6. GUI Mode - User Actions:
   1. Initialize session variables.
   2. Create GUI window with welcome message and buttons: Register, Login, Quit.
   3. If Register Button Clicked:
      1. Open registration dialog.
      2. Prompt user for username and password.
      3. Hash the password and insert data into the database.
   4. If Login Button Clicked:
      1. Open login dialog.
      2. Prompt user for username and password.
      3. Authenticate user from the database.
      4. If authenticated, display user actions dialog: Upload, View, Delete.
   5. If Upload Button Clicked:
      1. Open file dialog to choose a file.
      2. Generate user-specific key.
      3. Encrypt and move the file to the user's directory.
   6. If View Button Clicked:
      1. Open a new window with a list of files in the user's vault.
      2. Allow user to select a file to view.
      3. Decrypt and display the selected file's content.
   7. If Delete Button Clicked:
      1. Open confirmation dialog.
      2. If user confirms, delete all files from the user's directory except the key.
   8. If Quit Button Clicked:
      1. Close the GUI window and the program.
7. Conclusion:
   1. The application concludes, offering secure and flexible file management through both CLI and GUI interactions.

# The Distribution:

## Vault\_cli.py:

"vault\_cli.py" is a versatile script that empowers secure file management through a text-based interface. This comprehensive tool combines database management, encryption, and user interaction to ensure the confidentiality of data, all without relying on graphical elements. It provides a set of functions and methods that enable users to perform various operations such as user registration, file uploading, viewing, deletion, and more, all within a command-line environment. By integrating encryption techniques, password hashing, and personalized key management, "vault\_cli.py" ensures robust security while maintaining ease of use. Whether users are safeguarding personal files or managing sensitive data, this script offers a reliable and efficient solution for secure file handling and interaction through the command line.

1. **initialize\_database` Function:**  
   This function establishes an SQLite database, creating a 'users' table to securely store user data.
2. **`get\_user\_key` Function:**  
   The purpose of this function is to fetch a user-specific encryption key for personalized data protection.
3. **`generate\_key` Function:**  
   This function generates a new encryption key, significantly enhancing the cryptographic security for various operations.
4. **`encrypt\_file` Function:**  
   Using the Fernet encryption method, this function encodes the content of files, ensuring confidentiality during storage and transfer.
5. **`decrypt\_file` Function:**  
   The function's role is to decrypt previously encrypted files using the appropriate key, thus restoring the original content.
6. **`authenticate\_user` Function:**  
   This function validates user access by comparing hashed passwords with the entered credentials, enhancing security.
7. **`hash\_password` Function:**  
   In this function, passwords are transformed into hashed forms to enhance security during storage and comparison.
8. **`register` Function:**  
   The function oversees user registration, including tasks such as password hashing, individual key generation, and secure storage.
9. **`upload\_file` Function:**  
   With the objective of maintaining data confidentiality, this function encrypts files before uploading, then relocates them to the user's vault.
10. **`view\_files` Function:**  
    This function leverages Tkinter to present a graphical interface that lists user files, enhancing user interaction.
11. **`open\_file` Function:**  
    By decrypting and presenting selected file content, this function simplifies user access and viewing.
12. **`delete\_vault` Function:**  
    The function ensures secure removal of vault contents, with user confirmation to mitigate accidental data loss.
13. **`login` Function:**   
    With the task of validating user identity through password checks, this function grants access to encrypted files upon successful login.
14. **`read\_file\_content` Function:**  
    This function reads and returns specific file content, simplifying further processing and display.
15. **`show\_file\_content` Function:**  
    By displaying decrypted file content, this function enhances the user experience during content review.
16. **`show\_register\_dialog` & `show\_login\_dialog` Methods:**  
    These methods create pop-up dialogs for user registration and login, featuring fields for credentials.
17. **`register\_user` & `login\_user` Methods:**  
    These methods manage user registration and login, ensuring complete inputs and displaying error messages when needed.
18. **`show\_user\_actions` Method:**  
    The method presents a dialog featuring buttons for actions like file upload, viewing, and deletion, enhancing user interaction.
19. **`main` Function:**  
    The main function orchestrates the overall program flow by initializing the database.

## Vault\_gui.py

"vault\_gui.py" is a comprehensive script designed to facilitate secure file management through an intuitive graphical interface. This script merges the functionalities of database management, encryption, and user interaction to ensure the confidentiality and accessibility of data. It offers a range of functions and methods that allow users to perform tasks like user registration, file uploading, viewing, and deletion, all within a graphical environment. Through the integration of encryption techniques, password hashing, and personalized key management, "vault\_gui.py" prioritizes data security while offering a user-friendly experience. Whether users are organizing personal files or overseeing sensitive information, this script presents a robust and convenient solution for secure file management and interaction, all within an engaging graphical interface.

1. **`initialize\_database` Function:**  
   This function establishes an SQLite database to securely store user data within a 'users' table.
2. **`get\_user\_key` Function:**The function retrieves a user-specific encryption key for tailored data protection.
3. **`generate\_key` Function:**  
   This function generates a new encryption key, enhancing cryptographic security for file operations.
4. **`encrypt\_file` Function:**  
   The function uses Fernet encryption to encode file content, ensuring confidentiality during storage and transfer.
5. **`decrypt\_file` Function:**This function decrypts encrypted files using the appropriate key, restoring the original content.
6. `**authenticate\_user` Function:**  
   The function validates user access by comparing hashed passwords with the entered credentials.
7. **`hash\_password` Function:**  
   This function converts passwords into hashed forms, boosting security during storage and comparison.
8. **`register` Function:**  
   This function manages user registration, including tasks like password hashing, individual key generation, and secure storage.
9. **`upload\_file` Function:**  
   The function encrypts files before uploading, ensuring data confidentiality, and moves them to the user's vault.
10. **`view\_files` Function:**  
    This function lists user files using Tkinter, creating a graphical interface for improved interaction.
11. **`open\_file` Function:**  
    The function opens and decrypts selected files for viewing, enhancing user accessibility to content.
12. **`delete\_vault` Function:**  
    This function securely removes all vault contents with user confirmation, preventing accidental data loss.
13. **`login` Function:**  
    The function validates user identity through password checks, granting access to encrypted files upon successful login.
14. **`read\_file\_content` Function:**  
    This function reads and returns specific file content, facilitating further processing and display.
15. **`show\_file\_content` Function:**  
    This function displays decrypted file content, offering an improved user experience for content review.
16. **`FileVaultGUI` Class:**  
    This class defines a GUI application, enabling user-friendly interactions with secure file management.
17. **`show\_register\_dialog` & `show\_login\_dialog` Methods:**  
    These methods create pop-up dialogs for user registration and login, featuring input fields for credentials.
18. **`register\_user` & `login\_user` Methods:**  
    These methods manage user registration and login procedures, ensuring complete inputs and error message display.
19. **`show\_user\_actions` Method:**  
    This method displays a dialog with buttons for various actions like file upload, viewing, and deletion.
20. **`main` Function:**  
    This function orchestrates program flow by initializing the database, creating the GUI instance, and entering Tkinter's event loop.

## Main.py

"main.py" serves as the entry point for initiating the secure vault application. This script offers users a choice between two distinct interfaces, each catering to different preferences. Upon execution, the script greets users and prompts them to select either the Command-Line Interface (CLI) or the Graphical User Interface (GUI).

If the user chooses the CLI option, the script attempts to run "vault\_cli.py," which provides a text-based interaction mode. Similarly, selecting the GUI option triggers an attempt to run "vault\_gui.py," offering a graphical approach to interacting with the secure vault application.

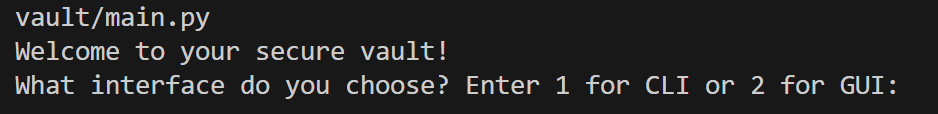
This script ensures a seamless transition between the chosen interface, providing a tailored experience based on user preference. By facilitating a flexible selection between CLI and GUI modes, "main.py" empowers users to access the secure vault application using their preferred interaction style, thereby enhancing usability and accommodating diverse user needs.

1. **Greeting and Interface Choice:**
   1. This section displays a welcoming message to introduce users to the secure vault application.
   2. It prompts users to input their preferred interface by entering '1' for the CLI or '2' for the GUI.
2. **Interface Selection:**
   1. This part employs a conditional structure to determine the chosen interface based on the user's input.
   2. If the user enters '1', the script proceeds to execute the "vault\_cli.py" script using a subprocess.
   3. If '2' is chosen, the script initiates the execution of "vault\_gui.py."
3. **CLI or GUI Execution:**
   1. Depending on the selected interface, this segment utilizes the `subprocess.run()` function to execute the corresponding interface script.
   2. If the user chooses the CLI option, the "vault\_cli.py" script is run in the command-line environment.
   3. Similarly, if the user prefers the GUI, the "vault\_gui.py" script is executed, providing graphical interaction.
4. **Error Handling:**
   1. This part handles potential errors that may arise if the specified interface script ("vault\_cli.py" or "vault\_gui.py") is not found in the directory.
   2. In such cases, the script displays an error message, indicating the absence of the respective script and suggesting potential solutions.
5. **User Interaction Flexibility:**
   1. By allowing users to choose between CLI and GUI interfaces, this segment offers flexibility in interaction with the secure vault application.
   2. Users can select the mode that aligns with their comfort and familiarity, ensuring a personalized and user-friendly experience.

# Execution

Certainly, let's delve into a more detailed and comprehensive breakdown of the technical execution of the "main.py" script, covering both option 1 (CLI) and option 2 (GUI):

* **Initialization:**
  + The "main.py" script begins by importing the necessary modules, including `subprocess`, which enables running other Python scripts from within the current script.



Figure

* **User Choice Prompt:**
  + A welcoming message is displayed, inviting the user to experience the secure vault application.
  + The user is prompted to choose between two interaction modes: '1' for the Command-Line Interface (CLI) and '2' for the Graphical User Interface (GUI).
* **User Input Processing:**
  + The script captures the user's input using the `input()` function.
  + This input will determine whether the CLI or GUI mode is initiated.

**Choice 1: Command-Line Interface (CLI):**

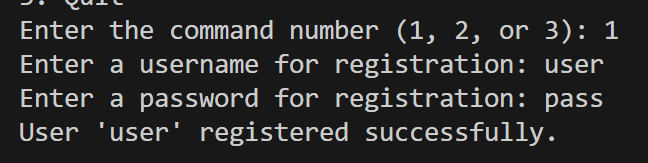
* **Execute "vault\_cli.py":**
  + If the user enters '1', the script proceeds to execute the "vault\_cli.py" script using the `subprocess.run()` function.
  + This action launches the execution of the CLI interface.

A screenshot of a computer

Description automatically generated

Figure

* **Database Initialization:**
  + "vault\_cli.py" initializes a connection to an SQLite database, which is essential for storing user information.
  + The `initialize\_database` function is called to create a table named "users" if it doesn't already exist.
  + This table will store user credentials for authentication.
* **Key Generation:**
  + The `generate\_key` function generates a unique encryption key using the Fernet cryptography library.
  + This key will be used for encrypting and decrypting files in the user's vault.
* **CLI-Based Interaction:**
  + The script presents users with a command menu.
  + Users can choose to register, login, upload files, view the contents of their vault, and delete items from their vault.
  + Each of these interactions corresponds to a specific function, such as `register`, `login`, `upload\_file`, `view\_files`, and `delete\_vault`.



Figure

A screenshot of a computer

Description automatically generated

Figure

A black background with white text

Description automatically generated

Figure

A screenshot of a computer screen

Description automatically generated

Figure

A screen shot of a computer

Description automatically generated

Figure

A screenshot of a computer

Description automatically generated

Figure

A screenshot of a computer program

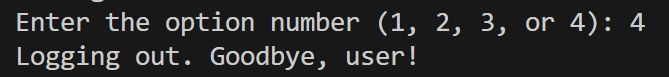
Description automatically generated

Figure

A black screen with white text

Description automatically generated

Figure



Figure

A black background with white text

Description automatically generated

Figure

**Choice 2: Graphical User Interface (GUI):**

* **Execute "vault\_gui.py":**
  + If the user enters '2', the script proceeds to execute the "vault\_gui.py" script using the `subprocess.run()` function.
  + This action launches the execution of the GUI interface.

A screenshot of a computer

Description automatically generated

Figure

* **GUI Initialization:**
  + "vault\_gui.py" initializes the GUI environment using the `tkinter` library.
  + It creates a main window with labels and buttons, providing users with a visually interactive experience.
* **User Registration and Login:**
  + Users can interact with the GUI by clicking buttons such as "Register" or "Login."
  + These actions trigger the appearance of dialog boxes, prompting users to input their desired username and password.

A screenshot of a computer screen

Description automatically generated

Figure

* **User Actions and Interaction:**
  + Upon successful registration or login, users are presented with buttons that allow them to perform various actions.
  + These include uploading files, viewing the contents of their vault, and deleting items from their vault.
  + Clicking these buttons triggers corresponding functions, enabling the user to seamlessly manage their data.

A screenshot of a login box

Description automatically generated

Figure

A screenshot of a computer screen

Description automatically generated

Figure

A screenshot of a computer

Description automatically generated

Figure

A screen shot of a computer

Description automatically generated

Figure

* **File Viewing and Deletion:**
  + When users click the "View Files" button, a new window opens, listing the files present in their vault.
  + Clicking on a specific file triggers the `show\_file\_content` function, which displays the content of the selected file in a text widget.

A screenshot of a computer

Description automatically generated

Figure

A screenshot of a computer

Description automatically generated

Figure

A screenshot of a computer

Description automatically generated

Figure

A screenshot of a computer

Description automatically generated

Figure

* **Exit the Application:**
  + Users can gracefully exit the GUI application by clicking the "Quit" button, which closes the application's main window.

A screenshot of a computer

Description automatically generated

Figure

* **Program Termination:**
  + Regardless of whether the user chooses the CLI or GUI mode, the script execution concludes after the chosen interface completes its execution.
  + This ensures that the user experience is seamless and consistent, regardless of their chosen mode of interaction.

# Conclusion

In conclusion, the trio of Python scripts - "main.py," "vault\_cli.py," and "vault\_gui.py" - collaboratively create a robust and versatile secure file vault application, offering users the flexibility to interact either through the Command-Line Interface (CLI) or the Graphical User Interface (GUI). This application employs a well-structured architecture to provide users with secure data storage and management options.

The journey begins with the execution of "main.py," which acts as the orchestrator of user interactions. The script initiates by presenting a welcoming message and prompting users to select their preferred mode of interaction. This dynamic decision-making process, influenced by the user's choice, leads to the execution of either the CLI or GUI script.

For users opting for the CLI experience, "vault\_cli.py" takes the reins. The script elegantly manages interactions with the terminal. It commences by establishing connections with an SQLite database, enabling user authentication and data storage. The generation of encryption keys using the Fernet cryptography library ensures secure file encryption and decryption. The CLI prompt guides users through registration, login, file uploads, vault content views, and item deletion, all of which are distinctly encapsulated within their respective functions.

Meanwhile, users favoring a more visual and interactive interface are directed to "vault\_gui.py." Here, the `tkinter` library creates an intuitive GUI environment. The journey unfolds with user-friendly registration and login dialogs, setting the stage for seamless data management. Aesthetic buttons for uploading files, viewing vault contents, and deleting items provide a rich user experience. Clicking these buttons triggers dedicated functions, encapsulating the desired actions within the realm of graphical elegance.

The application ensures an elegant exit from both interfaces, be it the GUI or the CLI. The seamless termination process preserves the user experience, regardless of their interaction mode. Notably, throughout this journey, stringent security measures are maintained. Encrypted keys and hashed passwords contribute to safeguarding user data and enhancing overall security.

Ultimately, this integrated system empowers users to manage their data effectively, embracing modern technology while maintaining user-friendliness. The design philosophy encapsulates the principles of convenience, security, and adaptability, catering to diverse user preferences. Whether through a series of terminal prompts or a visually appealing graphical interface, the vault application exemplifies the versatility and power of Python programming in building applications that seamlessly bridge the gap between user needs and technical functionality.

In this holistic ecosystem, the collaboration of these three Python scripts transforms mere code into a dynamic, secure, and user-centered file vault application. Through meticulous design, thoughtful interaction flows, and robust security mechanisms, this trio showcases the profound impact that programming can have on simplifying complex tasks, exemplifying the synergy between technology and user-centered design.