SECURE VAULT

BY

2211CS040009	ARELLA MANI KISHOR
2211CS040024	BRAHMARAOUTHU VISHWA TEJA
2211CS040055	GORANTLA JAGADEEP
2211CS040059	IMMADI SAI KIRAN

Under the Guidance of

Dr. B.Nageshwar Rao

Assistant Professor

MALLA REDDY UNIVERSITY

Department of Cyber Security

School of Engineering

Malla Reddy University

Hyderabad, Telangana, INDIA

Abstract

- Secure vault systems represent sophisticated infrastructures designed to protect valuable assets from unauthorized access, theft, or damage. A secure vault is a physical or digital space. A secure vault can also be an encrypted place where all important credentials are safely stored. All information added to a secure vault is encrypted at the device level. All information that you add in the vault is encrypted at your device level. To reduce risk, only trusted entities should have access to the secrets. Exposed secrets can result in undesired consequences such as unauthorized data access. The vaults verifies the user with a private passkey.

- Each file will be secured with a encryption algorithm so that one cannot access it unless he is an authorized user. Each file is specified in a separate location and can be accessed when it is decrypted or un-hided from the application.

1. Introduction

1.1 Problem Definition & Description

- **Problem Specification**
- The data which is made to store in the vault may have tampered or the master key may have been compromised.

- **Problem Description**
- The data that is stored in the vault can be tampered with or the Data Integrity may be lost or compromised. If the master password is compromised, all stored passwords can be accessed. Ransomware, phishing scams, and data breaches are just a few tactics employed by malicious actors to exploit vulnerabilities and steal valuable information. Compared to larger financial institutions, small businesses often lack the budget, expertise, and resources to invest in robust cybersecurity measures.

1.2 Objectives of the Project

- **Aim of the Project**
- Secure Vault's objective is to protect the confidential data from the unauthorized users and provide access control to the verified user with a private passkey.

- **Tasks and Deliverables**
- To protect the data from others, we implement the encryption algorithm to make the files secure and the data is protected by a personal passkey created by the user and by which the user can access the files or the data present in the vault. Each file is given an index value after storing it in the vault. The user can retrieve the data or view the required files by selecting the index value and a specific location will be provided where the file is present, and they can view it there.

1. Introduction

1.3 Scope of the project

- **Determining Goals**
- Providing Data Security
- Providing Access Control
- Protection of data with encryption algorithm
- Providing master key(used by the user every time whenever the user wants to view the files.)

- **Data & Constrains**
- In this application we use Cryptography techniques for data encryption and the application will be running on “Python Programming Language”.
- Data is restricted from the unauthorized users.

- **Workflow management strategies**
- Providing Protection for the files by encrypting the data after uploading it in the vault.
- Making sure User can access the files anytime with the index value in unhiding the files
- Using secure algorithms for encryption.
- Providing easy implementations to users.
- Making the application user friendly.

2. System Analysis

2.1 Existing System

- **Background & Literature Survey**
- Gokila Dorai, Sudhir Aggarwal, Neet Patel, and Charisa Powell. 2020. VIDE - Vault App Identification and Extraction System for iOS Devices. Forensic Science International: Digital Investigation 33 (Jul 2020), 301007. https://doi.org/10.1016/j. fsidi.2020.301007
- Debra Aho Williamson, "Worldwide Social Network Ad Spending: 2011 Outlook", February 2011. [online] http://www.emarketer.com/Reports/A11/ Emarketer-2000-757.aspx

- **Limitations of Existing System**
- Users often face the user interface problem, social engineering attacks
- The cost of maintenance
- File size error

2.2 Proposed System

- Less maintenance
- File size will be stored in backend(system storage)
- Secure access

- **Advantages of Proposed System**
- **Encryption:** Encrypted vaults ensure that only authorized people can access your data
- **Data confidentiality and integrity:** Maintains Data privacy. Stores and manages the users sensitive data
- **Shield sensitive data:** vault is a data storage on the computer that can be locked or unlocked with a password
- **Trust Building:** Maintaining a secure vault with reliability, enhancing trust with customers, partners

2. System Analysis

2.3 Software & Hardware Requirements

- **Software Requirements**
- Python supported software
- Windows 11 (32-bit or 64-bit), Windows 10 (32-bit or 64-bit), Windows 8 (32-bit or 64-bit), Windows 7 (32-bit or 64-bit), Windows Server 2008 – 2016
- Implementation of RSA algorithm
- Requires master password

- **Hardware Requirements**
- User experience on laptops or systems
- User friendly access(authorized access)
- Files are located in different locations and are accessed after un-hiding the files from the vault
- System is required to have a certain storage to save files

2. System Analysis

2.4 Feasibility Study

- **Technical Feasibility**
- Secure Vault exhibits strong technical feasibility with features like the secure protection and secure master key
- To enhance access control
- Secure protection while providing the encryption and protection of data

- **Robustness & Reliability**
- Offer a variety of protection for valuable data from unauthorized users
- It is reliable as it is provided with an encryption algorithm and one can control the storage only by a Master key provided by the user.

- **Economic Feasibility**
- In market the demand for secure storage solutions influences the economic feasibility of a vault

3. Architectural Design

3.1 Modules Design

3.1.1 Data Management:

- It is responsible for collecting and storing the data in the vault.
- All data stored in the vault is being encrypted using cryptographic algorithm.

3.1.2 File Selection Module:

- Extracts the data, analyses and modifies the data in the vault with security.
- Files can be selected by the users based on the index values.
- Data can be of any type and is protected with secure algorithms.

3.1.3 Design Module:

- Each file is protected and is stored in first-in-first-out order, so they can be called according to that.
- This application is classified into many parts to store and access the data.

3. Architectural Design

3.1.4 Availability Module:

- It checks whether the file is in proper format and is available to the authenticated user.
- Secure Connection and trustworthiness connections protecting the data from unauthorized access.

3.1.6 User Interface Module:

- Provides a user-friendly interface for users to interact with the app.
- Provide control over the data with the authorized users.
- It enables seamless interaction with the system and the user for securing the files.
- It implements mechanisms to authenticate and authorize user accessing into the vault.

3. Architectural Design

3.2 Method & Algorithm Design:

3.2.1 Encryption:

Encrypting the data stored in the vault ensures that even if unauthorized access is gained, the data remains unreadable. Strong encryption algorithms like AES (Advanced Encryption Standard) are commonly used.

3.2.2 Access Control:

Implementing strict access control mechanisms ensures that only authorized users can access the vault and only the user with the correct private key can enter and access the vault.

3.2.3 Algorithm (AES (Advanced Encryption Standard)):

AES is widely used for encrypting data due to its security, efficiency, and standardization. It stores and provides the security to the files that are added into the vault.

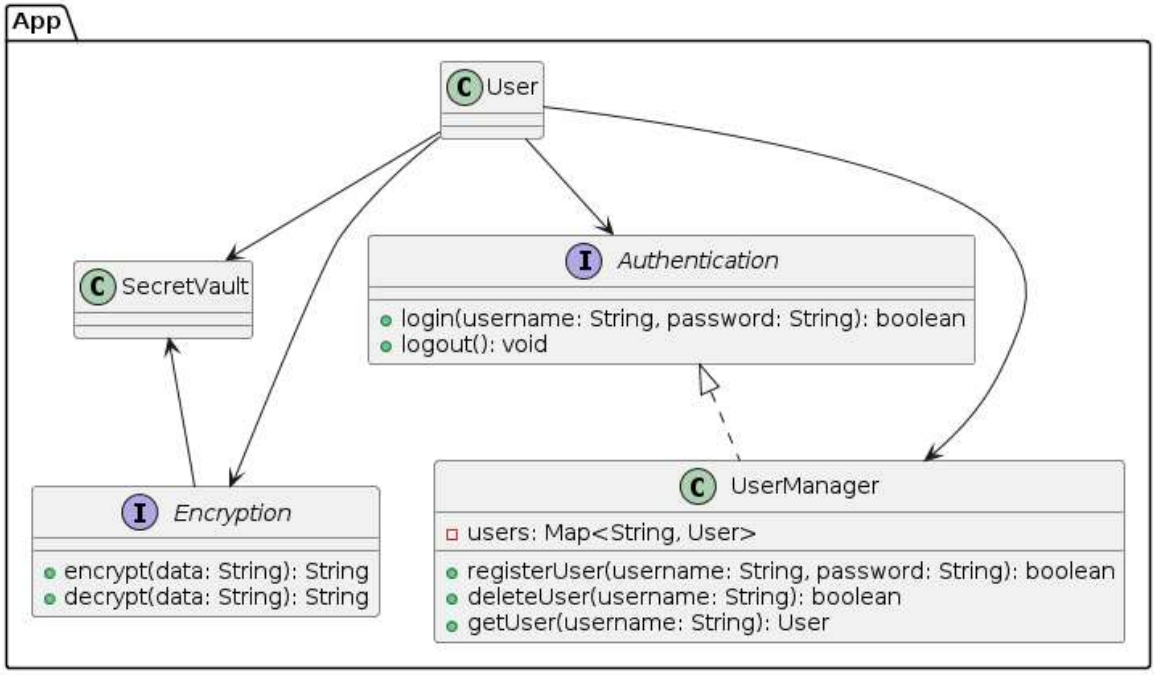
3.2.4 Access Control Mechanisms:

Role-Based Access Control (RBAC):

RBAC restricts access to the resources that are not accessed to the specific authenticated user and their roles are also restricted. It allows the files to the users based on the roles and permissions given to the user.

3.3 Project Architecture

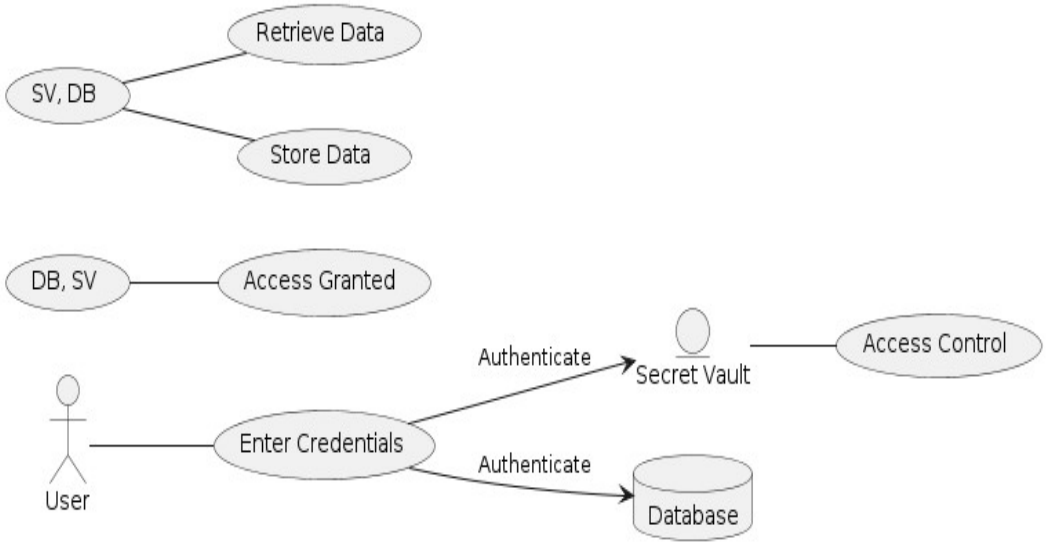
3.3.1 Architecture (Block Diagram)



- User has a private key to enter the vault. Only the user can enter the vault with his private key.
- The private key is then authenticated and verifies and validates whether the user is authenticated or not.
- The user manager will provide access if the private key matches with the database.
- The user can encrypt the files before adding them into the vault.
- The vault is available to the user and is ready to store the files.

3.3 Project Architecture

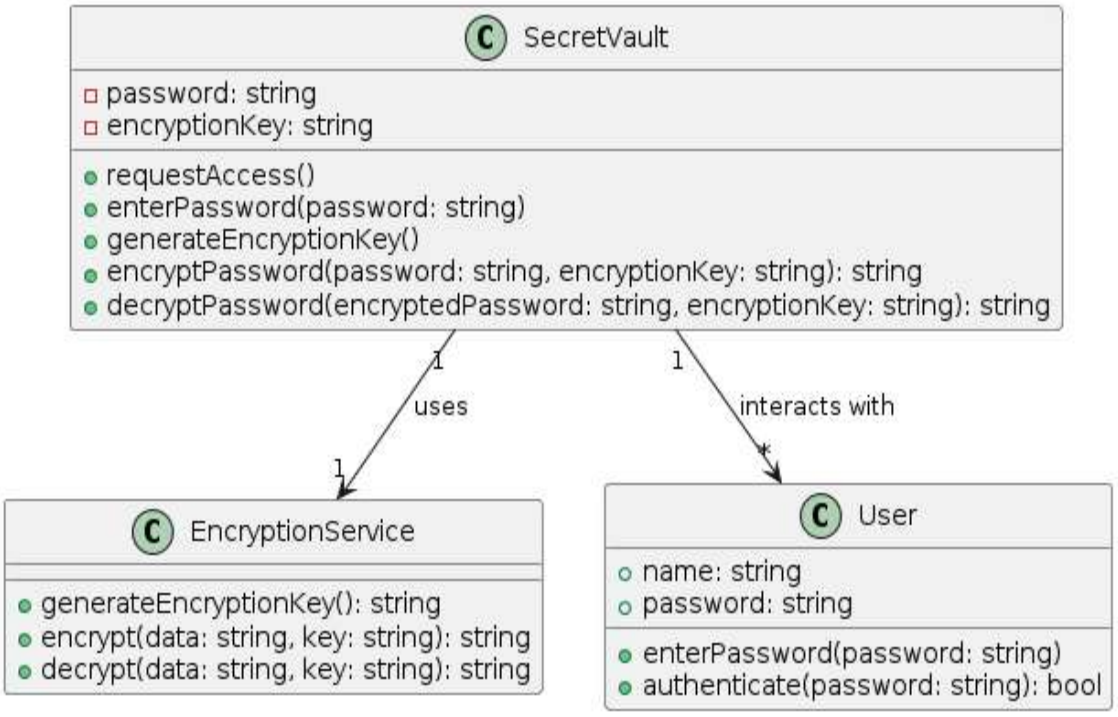
3.3.2 Data Flow Diagram



- The files/data received from the user is allotted in the system space internally.
- The Data is accessed to authorized users only. Storing and retrieving is done safely.
- Retrieval of data is easier and a specific file location is provided.
- Access control is restricted to the other users except for the authenticated user.

3.3 Project Architecture

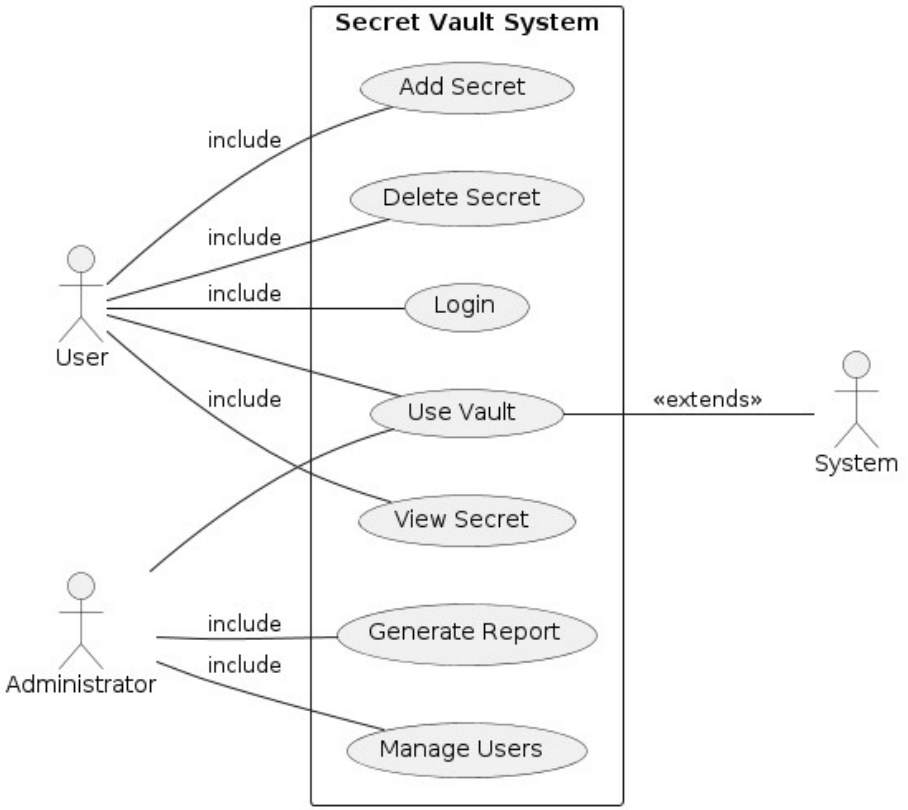
3.3.3 Class Diagram



- Secret Vault represents the vault where secrets are stored. It has attributes password and encryption key for managing access and encryption.
- Encryption Service provides encryption and decryption functionality. It has methods generate Encryption Key(), encrypt(), and decrypt().
- User represents the user of the secret vault application. It has attributes name and password, and methods enter Password() and authenticate().
- Secret Vault uses Encryption Service for encryption and decryption.

3.3 Project Architecture

3.3.4 Use Case Diagram



- User: Represents a regular user of the secret vault application.
- Administrator: Represents an administrator who manages the secret vault application.

- Use Cases:
- Login: Both users and administrators need to log in to access the application.

- Add Secret: Users can add secrets to the vault.

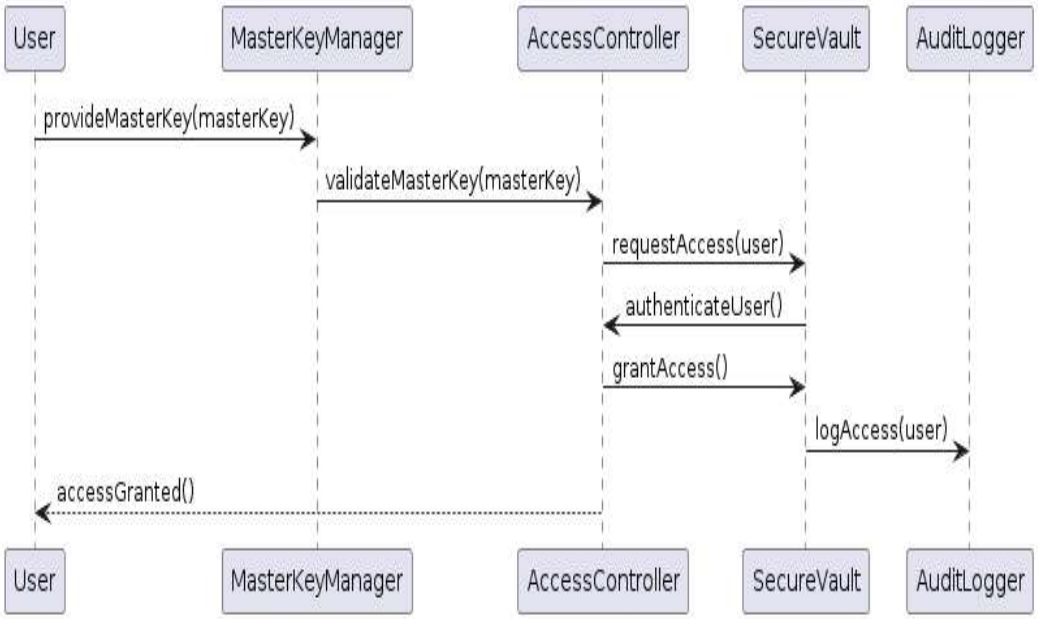
- Delete Secret: Users can delete secrets from the vault.

- View Vault: Users can view the contents of the vault.

- Generate Vault: Administrators can generate the vault

3.3 Project Architecture

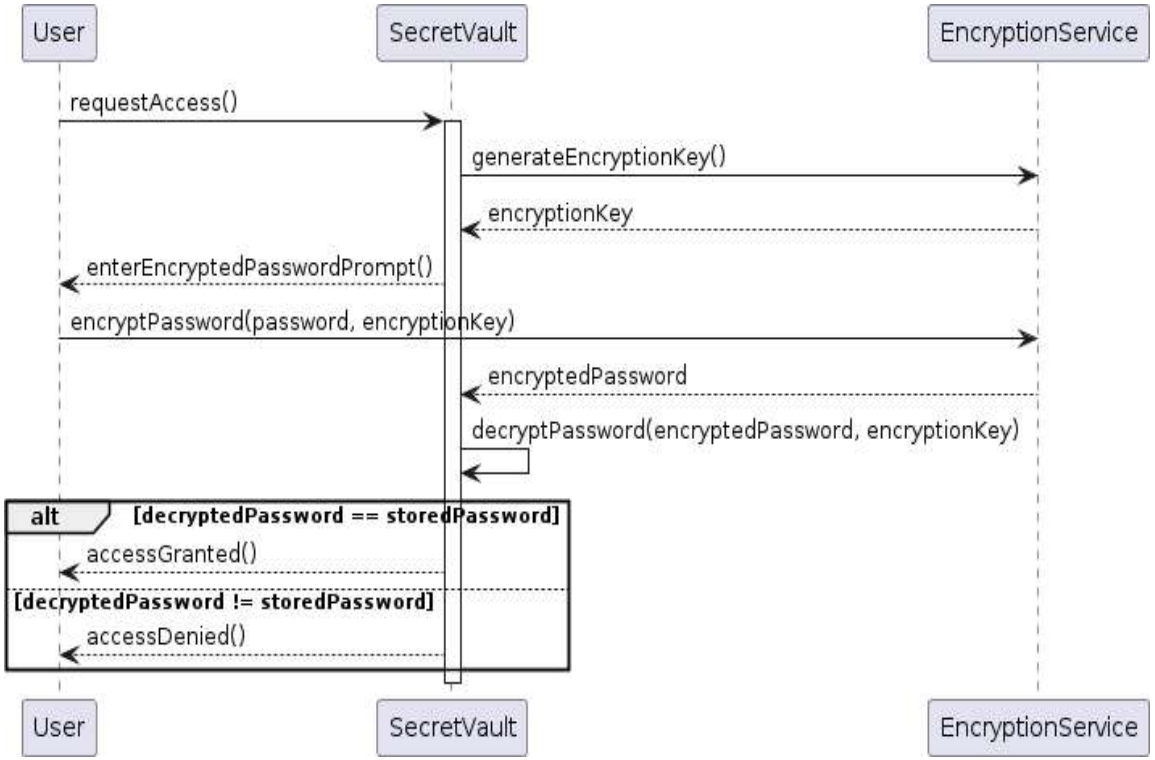
3.3.5 Sequence Diagram



- User sends a request Access() message to the Secret Vault.
- Secret Vault forwards the user credentials to the Authentication Server with authenticate(user Credentials).
- The Authenticator replies after providing the proper key.

3.3 Project Architecture

3.3.6 Activity Diagram



- The process starts at the Start node.
- The application checks if the user exists.
- If the user exists, the application authenticates the user.
- If the authentication is successful, the application displays the main menu.

4. Implementation

4.1 Coding Blocks

- Creating class file for secret_vault

```
class secret_vault:

        buffer_size = 64 * 1024

        def __init__(self, masterpwd):
                self.masterpwd = masterpwd

        def add_file(self, path, encrypt):
                if encrypt:
                        filenameWithExt = os.path.basename(path) + '.aes'
                        vaultpath = self.hid_dir + filenameWithExt
                        pyAesCrypt.encryptFile(path, vaultpath, self.key.decode(), self.buffer_size)
                else:
                        shutil.copy(path, self.hid_dir)

        def del_file(self, index):
                filenameWithExt = self.files[index]
                vaultpath = self.hid_dir + filenameWithExt
                if filenameWithExt.endswith('.aes'):
                        filename = filenameWithExt[:-4]
                        pyAesCrypt.decryptFile(vaultpath, filename, self.key.decode(), self.buffer_size)
                        os.remove(vaultpath)
                else:
                        shutil.copy(vaultpath, filenameWithExt)
                        os.remove(vaultpath)
```

```
def list_files(self):
        self.get_files()
        if not self.files:
                print("\nVault is empty!!!")
                return
        maxlen = max([len(x) for x in self.files])
        print('')
        print('-'*(maxlen+10))
        print("index\t|files")
        print('-'*(maxlen+10))
        for i, file in enumerate(self.files):
                print("{}\t|{}".format(i, file))
                print('-'*(maxlen+10))

def generate_key(self, salt=b"\xb9\x1f|}'S\xa1\x96\xeb\x154\x04\x88\xf3\xdf\x05", length=32):
    password = self.masterpwd.encode()

    kdf = PBKDF2HMAC(algorithm = hashes.SHA256(),
                     length = length,
                     salt = salt,
                     iterations = 100000,
                     backend = default_backend())

    self.key = base64.urlsafe_b64encode(kdf.derive(password))

def get_files(self):
        self.files = os.listdir(self.hid_dir)

def set_hid_dir(self):
        path = '~/.vault'
        hid_path = os.path.expanduser(path)
        self.hid_dir = hid_path + '/'
```

4. Implementation

4.2 Main Block

- Creating main block for the vault.

```
def main():
        print("Welcome to the secret vault!!!")
        path = os.path.expanduser('~/.vaultcfg')
        if os.path.exists(path):
                masterpwd = getpass("Enter your Master Password : ")
                vault = secret_vault(masterpwd)
                vault.generate_key()
                fernet = Fernet(vault.key)
                with open(path, 'rb') as f:
                        actual_mpwd = f.read()
                        try:
                                fernet.decrypt(actual_mpwd)
                                print('Welcome Back')
                        except:
                                print("Wrong Master Password!")
                                exit()
```

```
else:
        masterpwd = getpass("Create a Master Password : ")
        vault = secret_vault(masterpwd)
        vault.generate_key()
        fernet = Fernet(vault.key)
        enc_mpwd = fernet.encrypt(masterpwd.encode())
        with open(path, 'wb') as f:
                f.write(enc_mpwd)
                vault.set_hid_dir()
        try:
                os.makedirs(vault.hid_dir[:-1])
        except FileExistsError:
                pass

        if os.name == 'nt':
                call(["attrib", "+H", vault.hid_dir[:-1]])
                call(["attrib", "+H", path])

        print("Welcome")

vault.set_hid_dir()

choice = 0
```

```
while choice != 4:
        print("\nEnter 1 to hide a file\nEnter 2 to unhide a file\nEnter 3 to view hidden files\nEnter 4 to Exit\nEnter 5 to Reset the vault and delete all of its contents\n")
        try:
                choice = int(input("Enter your choice : "))
        except:
                print("\nUnknown value!")
                continue

        if choice == 1:
                print("\nTip : Drag and Drop the file")
                filepath = input("Enter the path of the file to hide : ")
                filepath = filepath.replace('\\', '')
                if filepath.endswith(' '):
                        filepath = filepath[:-1]
                if os.path.exists(filepath):
                        if os.path.isfile(filepath):
                                while True:
                                        enc_or_not = input("Do you want to encrypt the file? (Y or N) : ")
                                        if enc_or_not == 'y' or enc_or_not == 'Y':
                                                print('\nAdding file to the vault...')
                                                vault.add_file(filepath, 1)
                                                print("\nFile successfully added to the vault")
                                                print("You can now delete the original file if you want")
                                                break
                                        elif enc_or_not == 'n' or enc_or_not == 'N':
                                                print('\nAdding file to the vault...')
                                                vault.add_file(filepath, 0)
                                                print("\nFile successfully added to the vault")
                                                print("You can now delete the original file if you want")
                                                break
                                        else:
                                                print("Type Y or N")
                        else:
                                print("\nGiven path is a directory and not a file!")
                else:
                        print('\nFile does not exists!')
```

```
                elif choice == 2:
                        print('')
                        try:
                                file = int(input("Enter the index of the file from view hidden files : "))
                                vault.del_file(file)
                                print('\nFile unhided successfully')
                                print('The file will be present in {}'.format(os.getcwd()))
                        except:
                                print("\nInvalid index!")

                elif choice == 3:
                        vault.list_files()

                elif choice == 5:
                        while True:
                                confirm = input("\nDo you really want to delete and reset the vault?(Y or N) : ")
                                if confirm == 'y' or confirm == 'Y':
                                        pwdCheck = getpass("\nEnter the password to confirm : ")
                                        reset = secret_vault(pwdCheck)
                                        reset.generate_key()
                                        resetFernet = Fernet(reset.key)
                                        path = os.path.expanduser('~/.vaultcfg')
                                        with open(path, 'rb') as f:
                                                actual_mpwd = f.read()
                                                try:
                                                        resetFernet.decrypt(actual_mpwd)
                                                        print('Removing and resetting all data...')
                                                except Exception as e:
                                                        print(e)
                                                        print("\nWrong Master Password!")
                                                        print("Closing program now...")
                                                        exit()
                                        os.remove(path)
                                        shutil.rmtree(vault.hid_dir[:-1])
                                        print('\nReset done. Thank You')
                                        exit()
                                elif confirm == 'n' or confirm == 'N':
                                        print("\nHappy for that")
                                        break
                                else:
                                        print("Type Y or N")


if __name__ == '__main__':
        main()
```

4. Implementation

4.3 Execution Flow

4.3.1 Authentication:

The user launches the application and is prompted to authenticate themselves, usually by entering a password, PIN, or biometric authentication like fingerprint or facial recognition.

The application verifies the user's credentials against stored data to grant access.

4.3.2 Main Menu:

After successful authentication, the user is presented with the main menu of the application.

The main menu typically provides options for accessing, storing, or managing secret data.

4.3.3 Accessing Stored Data:

If the user chooses to access stored data, they may be presented with a list of categories or individual items stored in the vault.

Upon selecting an item, the user may need to provide additional authentication (such as a password) to decrypt the stored data.

4. Implementation

4.3.4 Storing Data:

If the user wants to store new data in the vault, they select an option to add a new item.

They may be prompted to enter the data they want to store (such as passwords, documents, or other sensitive information).

The application encrypts the data using strong encryption algorithms before storing it in the vault.

4.3.5 Managing Data:

The user may have options to edit, delete, or organize the stored data within the application.

These actions typically require additional authentication to ensure only authorized users can modify the vault contents.

4.3.6 Total Erase:

This new option allows the user to completely erase all data stored within the vault.

Upon selecting this option, the user may be prompted to confirm their decision to erase all data.

Once confirmed, the application securely deletes all stored data, ensuring it cannot be recovered.

4. Implementation

4.4 Results:

4.4.1 Calling the vault

```
(.venv) PS C:\Users\User\PycharmProjects\app vault> secret_vault
Welcome to the secret vault!!!
Enter your Master Password :
```

4.4.2 Menu

```
Enter 1 to hide a file
Enter 2 to unhide a file
Enter 3 to view hidden files
Enter 4 to Exit
Enter 5 to Reset the vault and delete all of its contents
```

4. Implementation

4.4 Results:

4.4.3 Adding files into the vault

```
Enter your choice : 1

Tip : Drag and Drop the file
Enter the path of the file to hide : C:/Users/User/Desktop/GJ/test.txt
Do you want to encrypt the file? (Y or N) : y
```

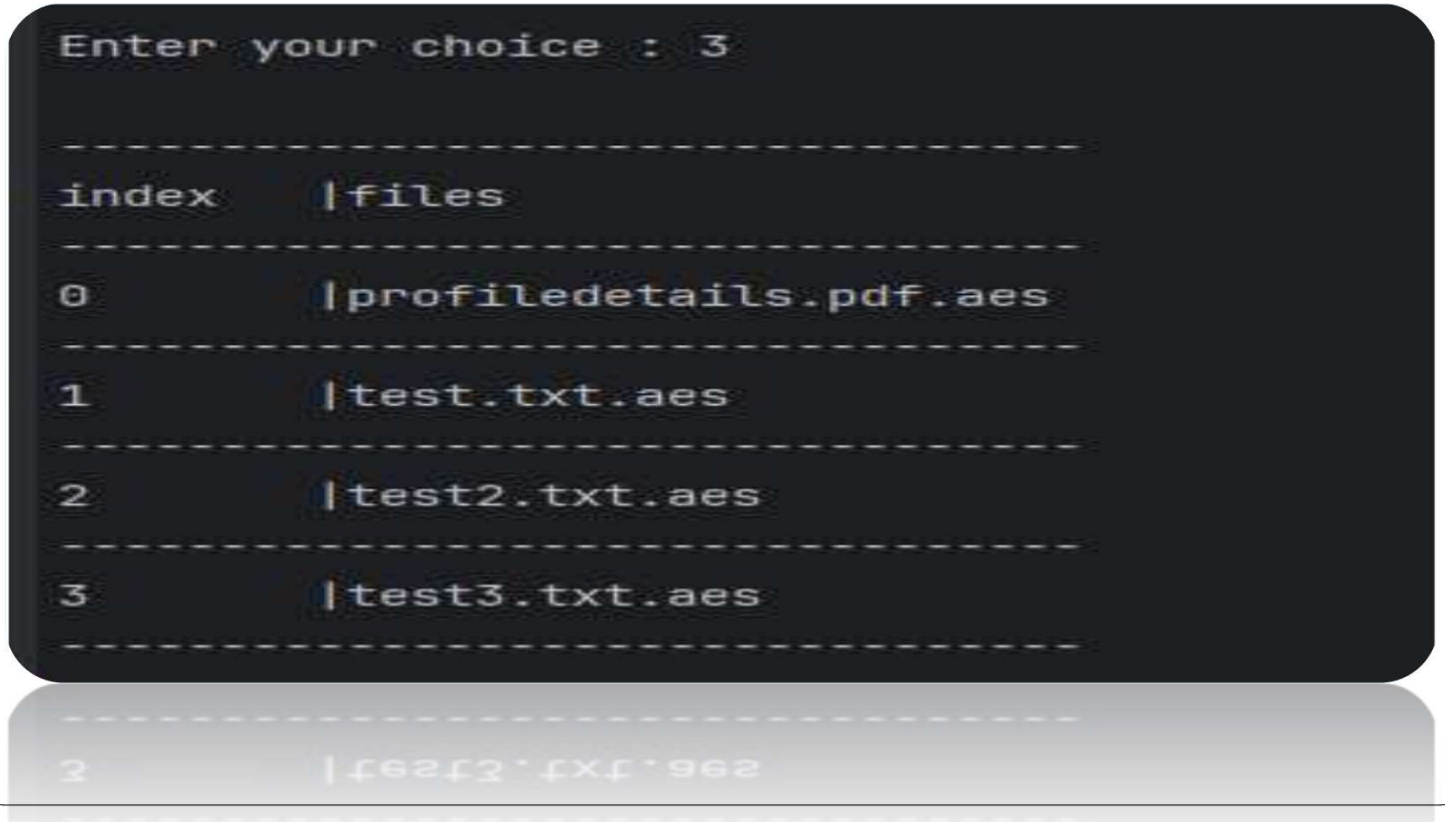
```
Adding file to the vault...

File successfully added to the vault
You can now delete the original file if you want
```

4. Implementation

4.4 Results:

4.4.4 Viewing of the Hidden Files



4. Implementation

4.4 Results:

4.4.5 Unhiding the Hidden Files

```
Enter your choice : 2

Enter the index of the file from view hidden files : 1

File unhided successfully
The file will be present in C:\Users\User\PycharmProjects\secret_vault
```

4. Implementation

4.4 Results:

4.4.6 Resetting of the vault (Total Erase)

```
Enter your choice : 5

Do you really want to delete and reset the vault?(Y or N) : y

Enter the password to confirm :
Removing and resetting all data...

Reset done. Thank You
```

4. Implementation

4.5 Conclusion:

- System vaults are used in many different fields, such as the safe keeping of cryptographic keys used in applications to encrypt and decrypt private information. Like a bank vault, a secure is a special secure. It is a virtual or maybe than a genuine secure where you can store records that require to be kept mystery from prying eyes. It was actualized to ensure touchy records and reports from prying eyes. Each record has a area and is continuously accessible. Each client is given a special key to get to the store where they can store all sorts of records counting PPTs, HTML web pages, records, pictures (JPG, PNG, GIF), APK records and more.

- The term "system vault" usually refers to happen within an operating system or software program where private data—such as passwords, cryptographic keys, or other credentials is encrypted and safeguarded. Protecting this sensitive data from theft, alteration, or unauthorized access is the goal of a system vault.

4. Implementation

4.6 Future scope:

In the future, as technology advances and security requirements change, the capabilities of a "System Vault" or other comparable secure storage techniques will probably change as well. The encryption techniques employed in System Vaults may grow increasingly complex and reliable as processing power rises. To combat new threats, this can entail implementing innovative cryptographic approaches or quantum-resistant algorithms. In addition to conventional passwords or cryptographic keys, biometric authentication techniques like fingerprint or facial recognition may become more and more important in future System Vaults. Access controls could be strengthened even more with multi-factor authentication.

Thank you!

- Queries?