# PASSWORD MANAGER REPORT

dumpalapravalika.cse2021@citchennai.net

210421104045

## TABLE OF CONTENTS

# 1. PASSWORD MANAGER

## 1.1 Introduction:

It is a basic password management system employing the `cryptography` library's `Fernet` module for encryption. This system offers essential functionalities such as key generation, password file management, password addition, and retrieval. Upon execution, users are presented with a menu-driven interface enabling operations like creating or loading cryptographic keys, managing password files, adding new passwords, and retrieving stored passwords. Through symmetric encryption, passwords are securely stored in a designated file, enhancing confidentiality. This project lays a foundation for a more comprehensive password management solution, adaptable to diverse security needs and extensions.

## 1.2 Objective:

The objective of a password management project is to create a secure and user-friendly system for storing, organizing, and accessing passwords. This entails implementing encryption techniques to protect sensitive data, providing a simple interface for users to add, retrieve, and manage passwords efficiently, and ensuring compatibility across devices for seamless access. By prioritizing security, ease of use, and accessibility, the project aims to enhance password security, streamline password management processes, and mitigate the risks associated with unauthorized access or data breaches.

## 1.3 Requirements:

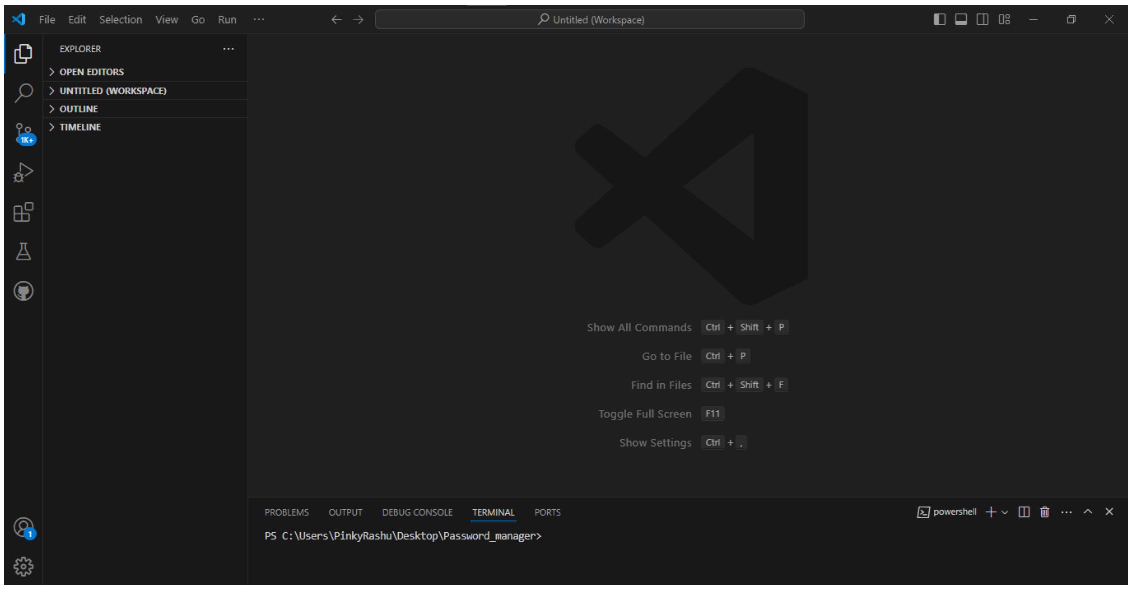
The requirements of Vulnerability Assessment and Penetration Testing are as follows:

- The VS Code
- Python IDE
- Cryptography

## 2. High-Level Summary:

A password management system is a tool designed to securely store and organize passwords for various accounts and services. It typically involves creating a master password, which grants access to an encrypted database containing all other passwords. Users can generate strong, unique passwords for each account and let the system handle the complexity of remembering them. Advanced systems may offer features like auto-fill for login forms, multi-factor authentication, and syncing across devices. The primary goal is to enhance security by promoting the use of strong, unique passwords while reducing the cognitive burden of managing them.

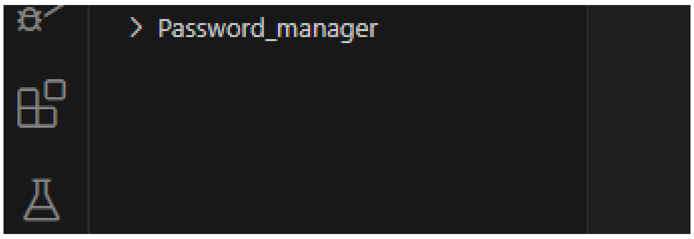
## 3. Set Up Visual Studio



```
PS C:\Users\PinkyRashu\Desktop\Password_manager>
```

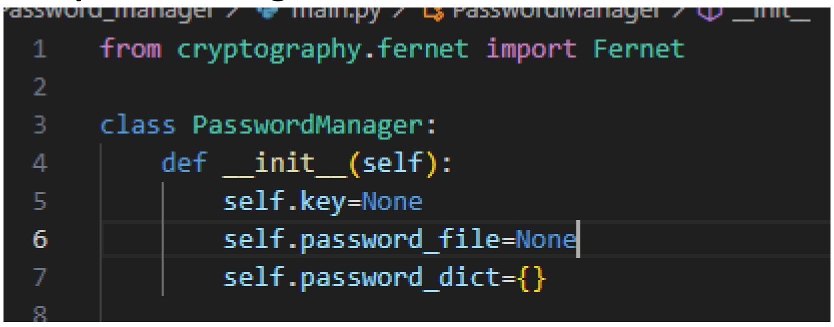
- ➢ To set up the VS Code install it in the OS and add python extension to execute the program.

### 3.1. Create a folder



- ➢ Create a new folder for the project and add that folder in the VS Code.

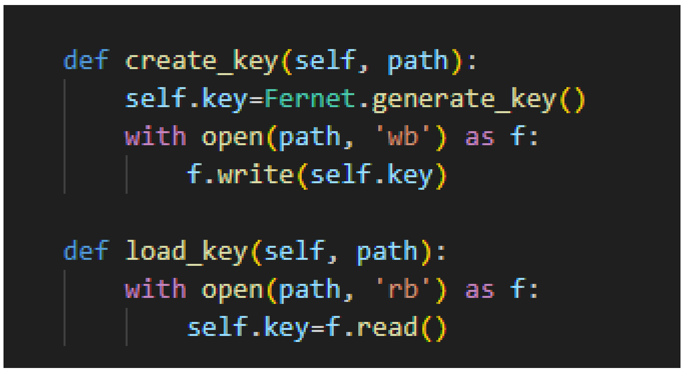
## 4. Compose the Program



```python
from cryptography.fernet import Fernet

class PasswordManager:
    def __init__(self):
        self.key=None
        self.password_file=None
        self.password_dict={}
```

- ➢ The statement imports the Fernet module from the cryptography library, enabling the use of Fernet symmetric encryption for secure communication and data protection.

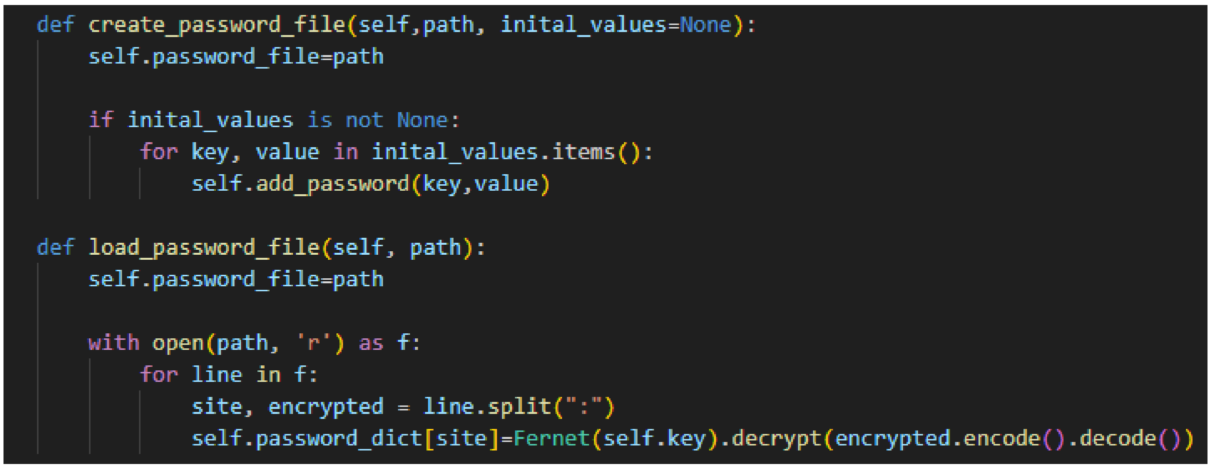
- ➢ The **PasswordManager class** initializes with attributes for encryption key, password file, and an empty dictionary to store passwords securely.



```python
def create_key(self, path):
    self.key=Fernet.generate_key()
    with open(path, 'wb') as f:
        f.write(self.key)

def load_key(self, path):
    with open(path, 'rb') as f:
        self.key=f.read()
```

- The `**create_key**` method generates a Fernet encryption key and saves it to a file specified by the `path` parameter.

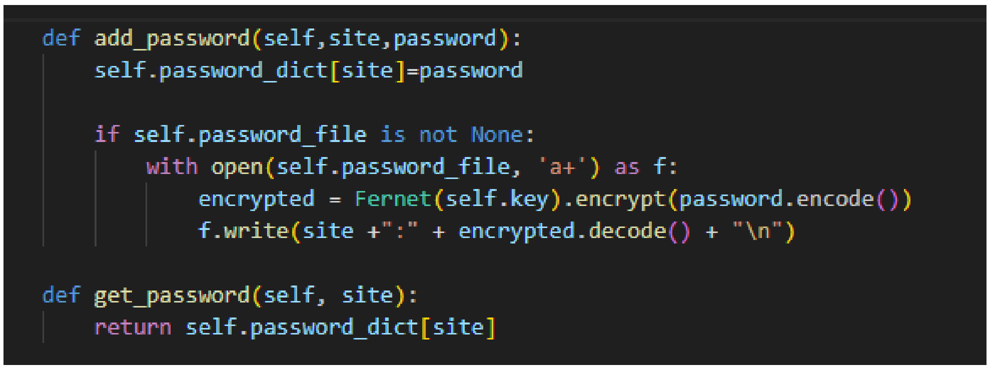
- The `**load_key**` method reads a Fernet encryption key from a file specified by the `path` parameter and stores it.



```python
def create_password_file(self,path, inital_values=None):
    self.password_file=path

    if inital_values is not None:
        for key, value in inital_values.items():
            self.add_password(key,value)

def load_password_file(self, path):
    self.password_file=path

    with open(path, 'r') as f:
        for line in f:
            site, encrypted = line.split(":")
            self.password_dict[site]=Fernet(self.key).decrypt(encrypted.encode().decode())
```

- `**create_password_file**` method sets the password file path attribute. If initial values are provided, it iterates through them, adding each key-value pair using `**add_password**` method.

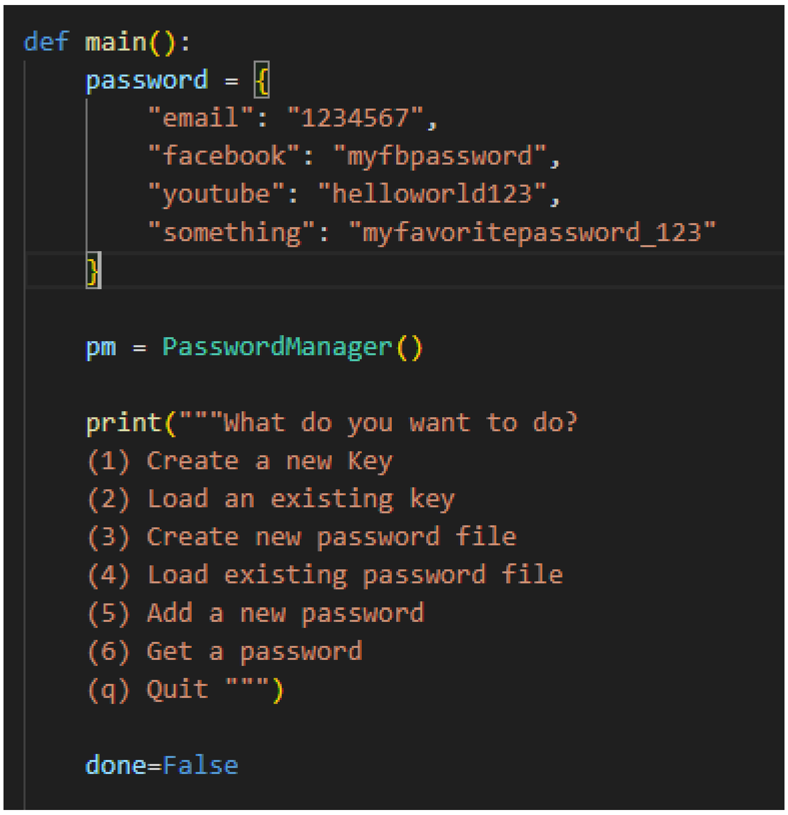
- `**load_password_file**` method sets the password file path, reads each line, splits it into site and encrypted password, decrypts it using Fernet encryption with the stored key, and stores it in the password dictionary.



```python
def add_password(self,site,password):
    self.password_dict[site]=password

    if self.password_file is not None:
        with open(self.password_file, 'a+') as f:
            encrypted = Fernet(self.key).encrypt(password.encode())
            f.write(site +":" + encrypted.decode() + "\n")

def get_password(self, site):
    return self.password_dict[site]
```

- `**add_password**` method adds a password for a site to the password dictionary. If a password file is specified, it encrypts the password using Fernet encryption, appends it to the file along with the site, and saves it.

- `**get_password**` method retrieves the password for a specific site from the password dictionary and returns it.



```
def main():
    password = {
        "email": "1234567",
        "facebook": "myfbpassword",
        "youtube": "helloworld123",
        "something": "myfavoritepassword_123"
    }

    pm = PasswordManager()

    print("""What do you want to do?
    (1) Create a new Key
    (2) Load an existing key
    (3) Create new password file
    (4) Load existing password file
    (5) Add a new password
    (6) Get a password
    (q) Quit """)

    done=False
```

**Program Initialization:**

- Defines a dictionary `password` containing pre-existing site-password pairs.
- Instantiates a `PasswordManager` object.
- Prints a menu displaying various password management options.

**Main Loop Initialization**:

- Initializes a boolean variable `done` to False, indicating the program is not yet finished.
- This variable controls the main loop of the program.

**User Interaction:**

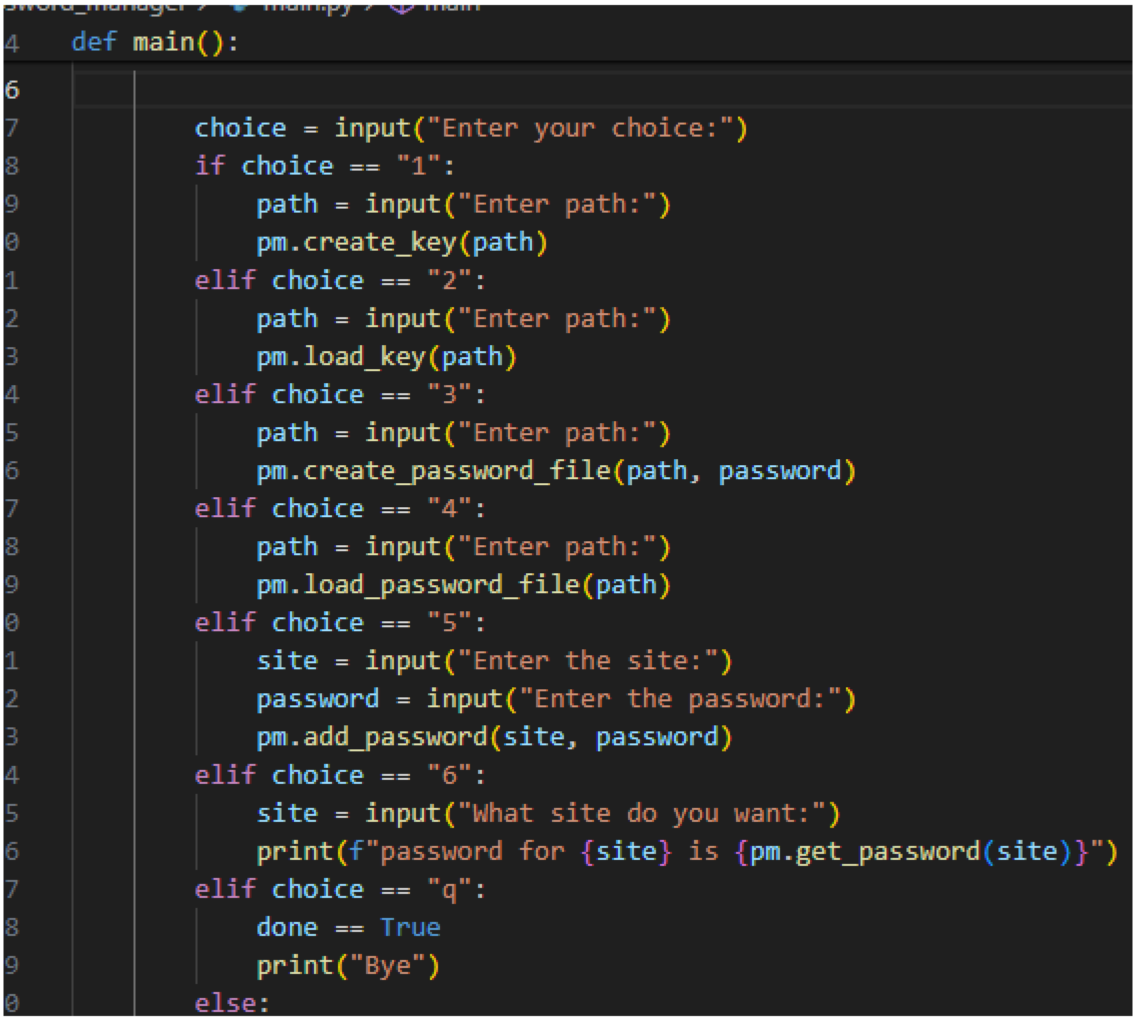
- Prompts the user to select an action from the menu.
- Offers options for creating or loading encryption keys, creating or loading password files, adding new passwords, retrieving passwords, or quitting the program.

**Execution and User Input Handling:**

- Executes corresponding actions based on the user's selection.
- The program performs tasks like generating or loading encryption keys, managing password files, adding new passwords, or retrieving existing ones.

**Program Termination:**

- The main loop continues until the user chooses to quit by selecting 'q'.
- Upon quitting, sets the `done` variable to True, ending the loop and terminating the program.



```python
def main():

        choice = input("Enter your choice:")
        if choice == "1":
            path = input("Enter path:")
            pm.create_key(path)
        elif choice == "2":
            path = input("Enter path:")
            pm.load_key(path)
        elif choice == "3":
            path = input("Enter path:")
            pm.create_password_file(path, password)
        elif choice == "4":
            path = input("Enter path:")
            pm.load_password_file(path)
        elif choice == "5":
            site = input("Enter the site:")
            password = input("Enter the password:")
            pm.add_password(site, password)
        elif choice == "6":
            site = input("What site do you want:")
            print(f"password for {site} is {pm.get_password(site)}")
        elif choice == "q":
            done == True
            print("Bye")
        else:
```

**User Interaction Loop:**

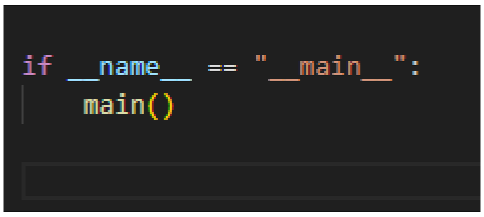
- The program enters a loop that continues until the `done` variable becomes True, indicating the user's choice to quit.
- Within this loop, the program prompts the user for input, requesting their choice from the menu.

**Handling User Choices:**

- Based on the user's input, the program executes different actions.
- **If the user selects:**
- "**1**": Prompts for a file path and creates a new encryption key using `create_key` method.
- "**2**": Prompts for a file path and loads an existing encryption key using `load_key` method.
- "**3**": Prompts for a file path and creates a new password file, populating it with initial passwords from the `password` dictionary using `create_password_file` method.
- "**4**": Prompts for a file path and loads an existing password file using `load_password_file` method.
- "**5**": Prompts for a site and password, then adds the password to the password manager using `add_password` method.
- "**6**": Prompts for a site and retrieves the corresponding password using `get_password` method, then prints it.
- "**q**": Sets `done` to True, indicating the user's intention to quit the program, and prints a farewell message.
- Any other input: Prints an error message indicating an invalid choice.

**Program Termination:**

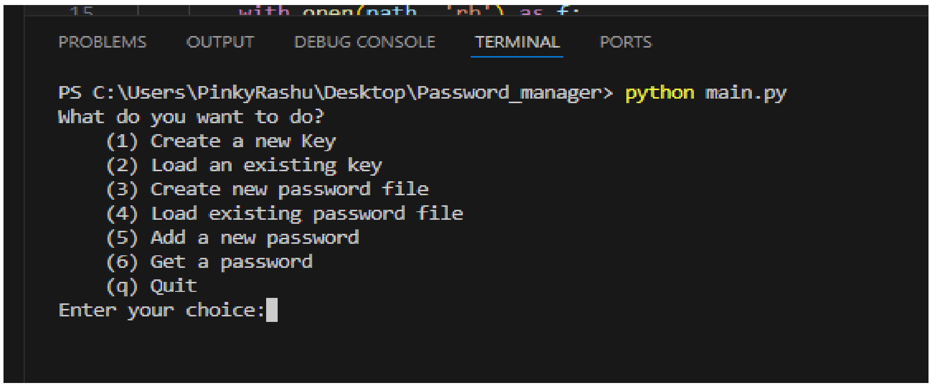
- Upon the user's selection of 'q', the loop ends, and the program terminates.
- Otherwise, the loop continues, prompting the user for further input.



```python
if __name__ == "__main__":
    main()
```

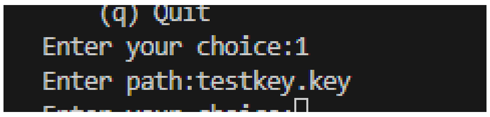
- The `**main()**` function is executed only if the script is run directly, not if it's imported as a module into another script.

## 5. Execute the Program

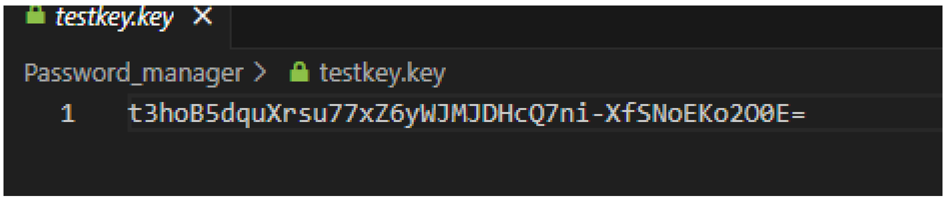


```
PS C:\Users\PinkyRashu\Desktop\Password_manager> python main.py
What do you want to do?
    (1) Create a new Key
    (2) Load an existing key
    (3) Create new password file
    (4) Load existing password file
    (5) Add a new password
    (6) Get a password
    (q) Quit
Enter your choice:
```

- The program presents a menu of options numbered from 1 to 6 and 'q' for quitting, allowing users to choose various password management tasks.
- Users input their choice corresponding to the desired action, such as creating a new key, loading an existing key, etc.

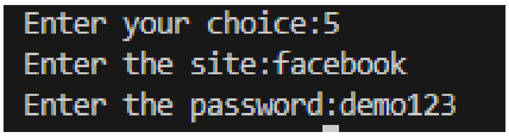


```
      (q) Quit
Enter your choice:1
Enter path:testkey.key
Enter your choice:
```



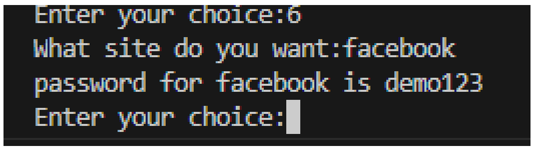
```
testkey.key
Password_manager > testkey.key
t3hoB5dquXrsu77xZ6yWJMJDHcQ7ni-XfSNoEKo2O0E=
```

- By clicking choice 1 allows to create a path for the encrypted key and a new path can be created as shown above



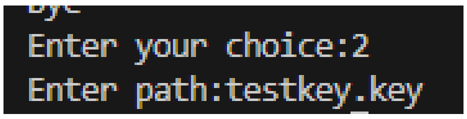
```
Enter your choice:5
Enter the site:facebook
Enter the password:demo123
```

- By selecting choice 5 we can choose any existing file and change or create a new password for it.



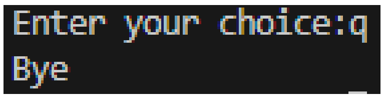
```
Enter your choice:6
What site do you want:facebook
password for facebook is demo123
Enter your choice:
```

- For choice 6 we can get a password for the particular site as in the above picture the site 'facebook' password is determined.



```
Bye
Enter your choice:2
Enter path:testkey.key
```

- The choice 2 gives us to load an existing key



```
Enter your choice:q
Bye
```

- By selecting choice q we can quit the execution of the program.

## 6. Output Summary:

**User Interaction:**

- Users are presented with a menu of options for password management tasks.
- They can create or load encryption keys, create or load password files, add new passwords, retrieve passwords, or quit.

**Execution Flow:**

- Users input their choice corresponding to desired actions.
- The program executes the chosen action, such as generating or loading keys, managing password files, adding or retrieving passwords.

**Termination**:

- Users can quit the program by selecting 'q'.
- Upon quitting, the program ends, displaying a farewell message.