

PASSWORD MANAGER WITH ENCRYPTION

Name :Tejasree Kilari

Kent ID:811241145

Email ID:tkilari@kent.edu

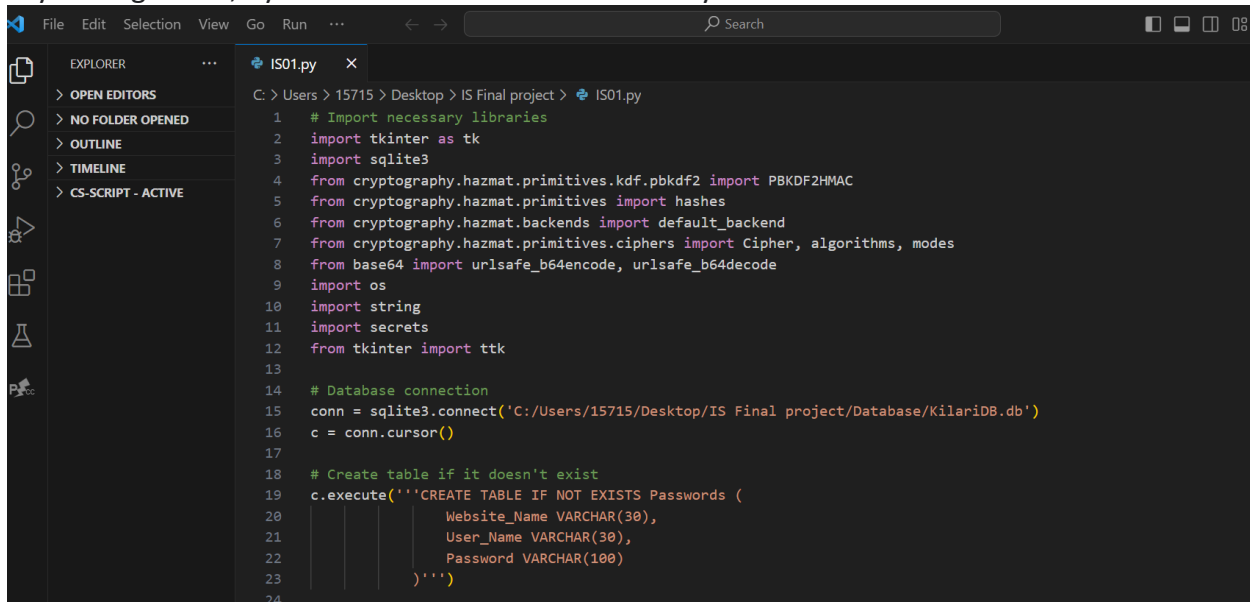
Importance of Password Manager:

A password manager (or a web browser) can store all your passwords **securely**, so you don't have to worry about remembering them. This allows you to use unique, strong passwords for all your important accounts (rather than using the same password for all of them, which you should *never* do). A password manager is created in python using tkinter and sqlite database to manage passwords for several applications. Passwords can be stored for various applications and details required for maintaining the passwords are application name, url, email id and password.

Project Implementation:

For Password manager with encryption Project, I had used Visual code studio For Python Programming Language, Sqlite3 for creating the database.

Reason for Using Python Programming Language: Python is a great choice for cryptography projects due to the wide range of libraries and modules available. Whether you're looking to implement basic encryption techniques or advanced applications such as digital signatures and key management, Python has the tools to make it easy.



```
1  # Import necessary libraries
2  import tkinter as tk
3  import sqlite3
4  from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
5  from cryptography.hazmat.primitives import hashes
6  from cryptography.hazmat.backends import default_backend
7  from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
8  from base64 import urlsafe_b64encode, urlsafe_b64decode
9  import os
10 import string
11 import secrets
12 from tkinter import ttk
13
14 # Database connection
15 conn = sqlite3.connect('C:/Users/15715/Desktop/IS Final project/Database/KilariDB.db')
16 c = conn.cursor()
17
18 # Create table if it doesn't exist
19 c.execute('''CREATE TABLE IF NOT EXISTS Passwords (
20             Website_Name VARCHAR(30),
21             User_Name VARCHAR(30),
22             Password VARCHAR(100)
23             )''')
24
```

Reason for Using Sqlite3: As an embedded database, SQLite allows developers to store data without needing a separate database server. It is helpful for small data needs, such as mobile devices and other applications that require low-memory footprint systems.

```
Command Prompt - sqlite3 K x + v
Microsoft Windows [Version 10.0.22631.2715]
(c) Microsoft Corporation. All rights reserved.

C:\Users\15715>cd Desktop

C:\Users\15715\Desktop>dir
Volume in drive C is Windows-SSD
Volume Serial Number is 0AFE-3B78

Directory of C:\Users\15715\Desktop

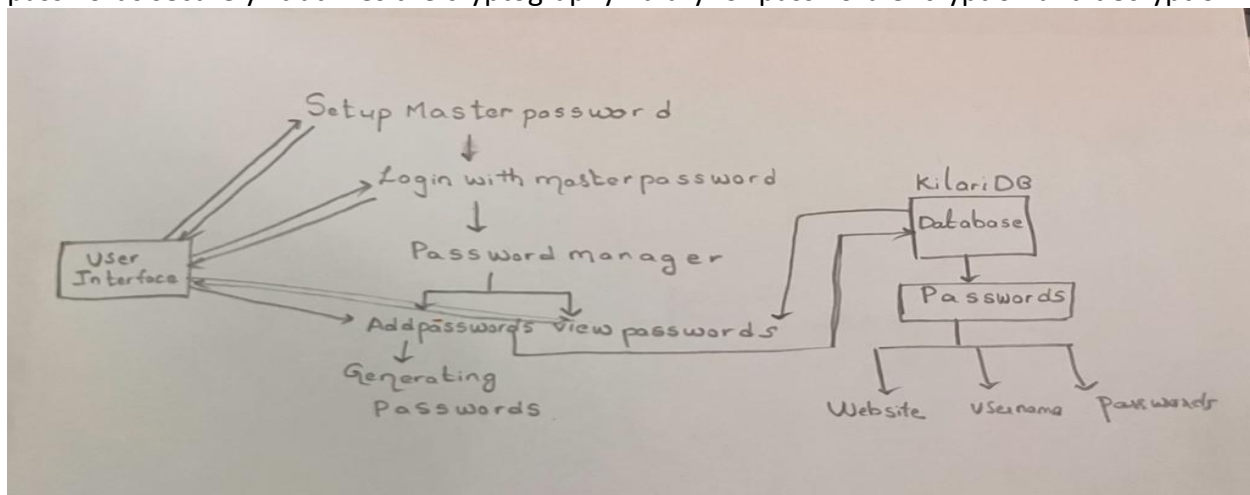
12/03/2023  11:39 PM  <DIR>      .
12/03/2023  02:31 PM  <DIR>      ..
12/03/2023  09:51 PM  <DIR>      Assignment
12/03/2023  11:45 PM  <DIR>      Database
12/03/2023  09:46 AM                1,411 Visual Studio Code.lnk
               1 File(s)              1,411 bytes
               4 Dir(s)  341,552,939,008 bytes free

C:\Users\15715\Desktop>cd Database

C:\Users\15715\Desktop\Database>sqlite3 KilariDB
SQLite version 3.44.2 2023-11-24 11:41:44 (UTF-16 console I/O)
Enter ".help" for usage hints.
sqlite> .databases
main: C:\Users\15715\Desktop\Database\KilariDB r/w
sqlite> |
```

Architecture of the Project:

The code implements a basic password manager application using the Tkinter library for the graphical user interface (GUI) and SQLite for database storage. The application allows users to set a master password, login, and manage passwords securely. It utilizes the cryptography library for password encryption and decryption.



The script begins by importing necessary libraries, including Tkinter for GUI, SQLite for database management, and cryptography for encryption and key derivation. A SQLite database named "KilariDB.db" is created, and a table named "Passwords" is defined to store website names, usernames, and encrypted passwords.

```
IS01.py X
C: > Users > 15715 > Desktop > IS Final project > IS01.py > generate_random_password

1  # Import necessary libraries
2  import tkinter as tk
3  import sqlite3
4  from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
5  from cryptography.hazmat.primitives import hashes
6  from cryptography.hazmat.backends import default_backend
7  from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
8  from base64 import urlsafe_b64encode, urlsafe_b64decode
9  import os
10 import string
11 import secrets
12 from tkinter import ttk
13
14 # Database connection
15 conn = sqlite3.connect('C:/Users/15715/Desktop/IS Final project/Database/KilariDB.db')
16 c = conn.cursor()
17
18 # Create table if it doesn't exist
19 c.execute('''CREATE TABLE IF NOT EXISTS Passwords (
20 |         |         |         |         Website_Name VARCHAR(30),
21 |         |         |         |         User_Name VARCHAR(30),
22 |         |         |         |         Password VARCHAR(100)
23 |         |         |         |     )''')
24
```

The script is a Python-based password manager with a Tkinter GUI and SQLite database integration. Its functionalities can be outlined as follows:

Random Password Generation: The script includes a feature to create random passwords. This is likely utilized when adding new entries to the password manager or when generating strong, random passwords for different purposes.

```
# Function to generate a random password
def generate_random_password():
    """Generate a random password with 15 characters."""
    alphabet = string.ascii_letters + string.digits + string.punctuation
    password = ''.join(secrets.choice(alphabet) for i in range(15))
    return password
```

Key Derivation: Another function employs the Password-Based Key Derivation Function 2 (PBKDF2) with HMAC to derive a key from the master password. PBKDF2HMAC adds an extra layer of security by making brute-force attacks computationally expensive.

```
# Function to derive a key from the master password using PBKDF2HMAC
def derive_key_from_master_password(master_password):
    """Derive a key from the master password using PBKDF2HMAC."""
    password_bytes = master_password.encode('utf-8')
    salt = os.urandom(16)
    iterations = 100000
    length = 32

    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=length,
        salt=salt,
```

Encryption and Decryption: AES in Cipher Feedback (CFB) mode is used for encrypting and decrypting passwords. AES is a widely adopted symmetric encryption algorithm, and CFB is a mode of operation for secure block-based data handling.

```
# Function to encrypt a password using AES in CFB mode
def encrypt_password(password, key, salt):
    """Encrypt a password using AES in CFB mode."""
    iv = os.urandom(16)
    cipher = Cipher(algorithms.AES(key), modes.CFB(iv), backend=default_backend())
    encryptor = cipher.encryptor()

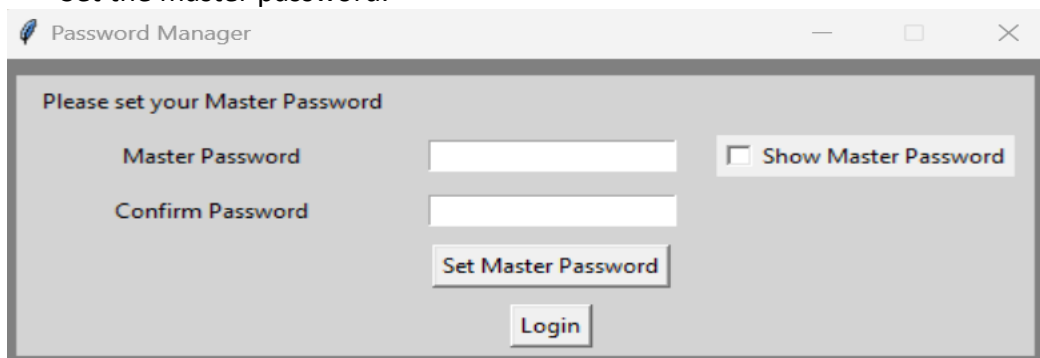
    encrypted_password = iv + encryptor.update(password.encode('utf-8')) + encryptor.finalize()
    return urlsafe_b64encode(encrypted_password), salt

# Function to decrypt a password
def decrypt_password(encrypted_password, key, salt):
    """Decrypt a password."""
    iv = encrypted_password[:16]
    encrypted_password = urlsafe_b64decode(encrypted_password)
    cipher = Cipher(algorithms.AES(key), modes.CFB(iv), backend=default_backend())
    decryptor = cipher.decryptor()

    decrypted_password = decryptor.update(encrypted_password[16:]) + decryptor.finalize()
    return decrypted_password.decode('utf-8')
```

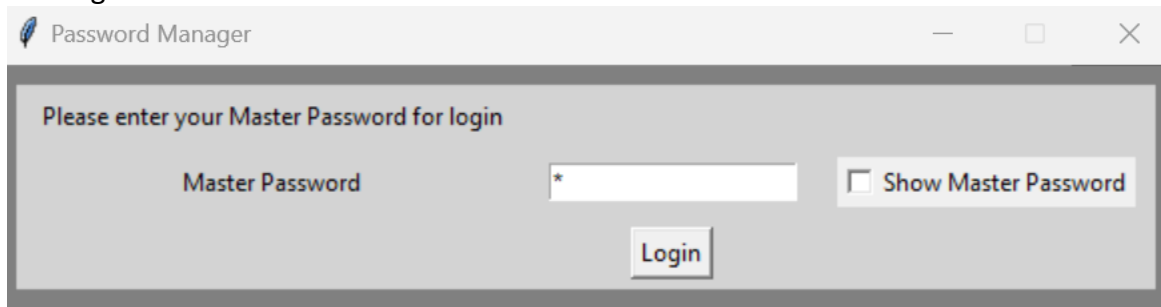
GUI Interactions: The primary GUI, built with Tkinter, allows users to:

- Set the master password.



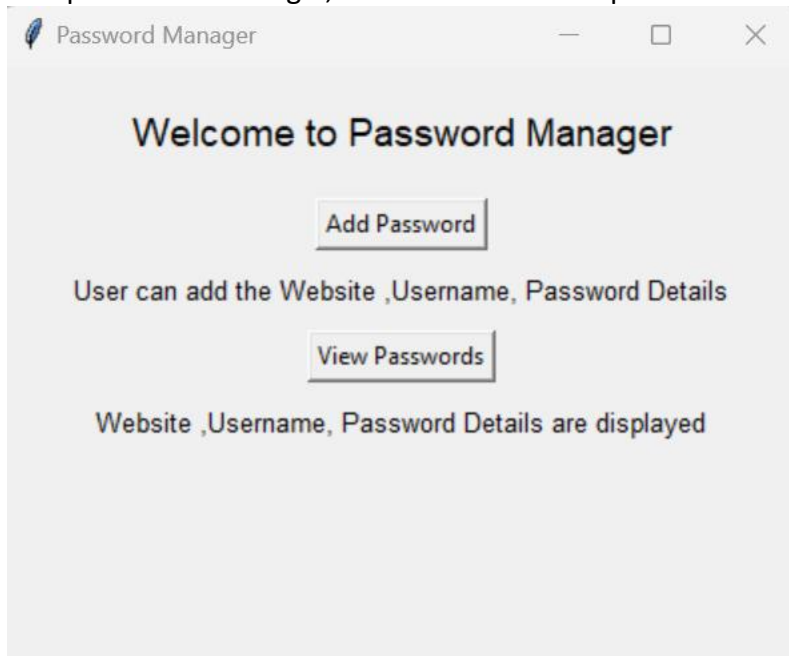
The screenshot shows a window titled "Password Manager" with a standard macOS-style title bar. Inside the window, there is a section titled "Please set your Master Password". Below this title, there are two text input fields: "Master Password" and "Confirm Password". To the right of the "Master Password" field is a checkbox labeled "Show Master Password". Below the input fields is a button labeled "Set Master Password". At the bottom center of the window is a button labeled "Login".

- Toggle visibility of the master password during input.
- Set up the password.
- Log in.



The screenshot shows a window titled "Password Manager" with standard window controls (minimize, maximize, close). The main content area has a light gray background and contains the text "Please enter your Master Password for login". Below this text, there is a label "Master Password" followed by a text input field containing a single asterisk (*). To the right of the input field is a checkbox labeled "Show Master Password". Below the input field and checkbox is a "Login" button.

- Upon successful login, users can access the password manager window, enabling them to:



The screenshot shows the same "Password Manager" window after a successful login. The title bar remains the same. The main content area now displays "Welcome to Password Manager" in a large, bold font. Below the welcome message is an "Add Password" button. Underneath the button, the text "User can add the Website ,Username, Password Details" is displayed. Below this text is a "View Passwords" button. At the bottom of the visible content, the text "Website ,Username, Password Details are displayed" is shown.

- Add new passwords.

Add_...

Please enter your passwords

Website:

Please enter your Website Name

Username:

Please enter your Username

Password:

Please enter your password

Generate Password

Random password created

☐ Show Password

Submit

- Generate random passwords.
- View all stored passwords.

Website	Username	Password
Linkedin	Tejasree Kilari	QQ_WKw4y6xPw6dQyPlcUAZfFBCM
Facebook	Tejasree	6qZIT6NdnITQ9-IKXvJMsRrRsRmllw=
Amazon	Teju	yIWLi_sgnALT9j2qBfr0y_q3zcHbWA=
Shein	Tejasree	WTtnLDUcr1Hf2s4PgDR1uWwnnrjfH
Python	Tejasree	vvrGjxxodeBx_NL7pigv0hOoHINxUjfc
Github	TEjasree	B0VmWsXav-gxyknYA28trf6zz4kVq-!
Canvas	Tejasree	iWDACuCx6NbsjwYVQdBqALdOY7)
Flashline	Kilari	WH01DnRRvdh1Ydql3Q6tqlldTzzS1C

Main Event Loop: The script initiates the Tkinter main event loop, crucial for the GUI to respond to user actions and events.

Database Connection Management: Database operations, such as initializing connections, executing queries, and closing connections when the application concludes, are handled. SQLite serves as a lightweight and embedded database, suitable for secure password management.

Security Emphasis: The script emphasizes the critical importance of password manager security. Users are strongly advised to adhere to best practices, including selecting robust master passwords, maintaining software updates, and handling sensitive data cautiously.

Security Measures implemented in the project:

Password Encryption: The user's master password is used to derive a key using PBKDF2HMAC (Password-Based Key Derivation Function 2 with Hash-based Message Authentication Code). The derived key is used for encrypting and decrypting passwords.

Random Password Generation: The code includes a function (`generate_random_password()`) to generate a random password with 15 characters, which can enhance the security of generated passwords.

Secure Password Storage: The passwords stored in the database are encrypted using the AES (Advanced Encryption Standard) algorithm in CFB (Cipher Feedback) mode. Each password is encrypted with a unique Initialization Vector (IV), and the IV is stored along with the encrypted password.

Salt for Key Derivation: A random salt is generated for key derivation using PBKDF2HMAC. The salt is unique for each user and is stored alongside the derived key.

Secure Database Connection: The code uses SQLite to store passwords, and SQL queries are parameterized to prevent SQL injection attacks.

GUI Security Measures: The GUI uses asterisks to hide the master password, and there's an option to toggle the visibility of the master password for the user's convenience.

Use of Cryptography Libraries: The code uses the `cryptography` library for cryptographic operations, which is a reputable library for handling cryptographic tasks.

Secure Random Number Generation: The `secrets` module is used for secure random number generation.

Potential Vulnerabilities and Mitigations:

- Since the files are stored locally, they can easily be deleted without needing to enter any credentials and consequently all stored passwords are gone along with other data
- The hardcoded database connection string, including the full path, poses a risk and should be stored in a secure configuration file.
- The handling of the master password, stored in memory during runtime, needs improvement to ensure sensitive information is adequately protected .
- The user interface lacks proper input validation and sanitation, opening the door to potential code injection or manipulation attacks. Encryption parameters should be stored as constants and regularly reviewed for optimal security.
- Error handling is minimal, and the code does not address potential cross-site scripting (XSS) vulnerabilities. Implementing encrypted connections for database communication is recommended to safeguard data in transit.
- Regularly updating dependencies and following security best practices are crucial for maintaining a resilient and secure application.

Future Enhancements:

- GUI can be enhanced in a better way to display the results.
- Encryption can be added to encrypt the stored information.
- Search feature can be added.
- User authentication can also be added.
- Better encrypted algorithm can be used.