Password Manager Project

Aim

To create a client side password manager which can store passwords of multiple users securely.

Main features

- **Create and delete users**
- **Add and delete passwords**
- **Show passwords**
  - **Show all passwords**
  - **Search by**
    - **User name**
    - **Service name**
    - **Both**

Security techniques and sample usage

Creating user

A user can be create using a user id and a master password.

Deleting user

A user can be deleted by entering the user id and entering the master password to confirm.

```
Enter user ID: user
Enter master password to confirm: password
User deleted successfully
```

Login system

The login for each user is stored as the SHA-256 digest, so its hard to distinguish a user, the sql table looks as follows:

```
+----------+------------+------+-----+---------+-------+
| Field    | Type       | Null | Key | Default | Extra |
+----------+------------+------+-----+---------+-------+
| userid   | binary(32) | NO   | PRI | NULL    |       |
```

```
| salt     | binary(16) | NO   |     | NULL    |       |
| password | binary(32) | NO   |     | NULL    |       |
+----------+------------+------+-----+---------+-------+
```

For example, a user named “user” with a password “password” will be stored as

```
userid: 0xC0D75E06C7E6870555800C88074596666C713D07579B216F8E3E90B35D60E223
salt: 0x7DAC9EA0113ED4B9B34B4D8A79308670
password:
0x8AAD6F5E5D715DD38E1358561C4AD33F2DC580EB4F1B5A140170C5EE6D8DD376
```

When a user logs in, the program will compute the hash with the salt and grant the access if it matches the database.

Password storage

The passwords are encrypted using AES-128 and are stored in a database, the database structure is as followed:

```
+------------+---------------+------+-----+---------+-------+
| Field      | Type          | Null | Key | Default | Extra |
+------------+---------------+------+-----+---------+-------+
| masteruser | binary(32)    | NO   |     | NULL    |       |
| service    | varchar(64)   | NO   |     | NULL    |       |
| userid     | varchar(64)   | NO   |     | NULL    |       |
| salt       | binary(16)    | NO   |     | NULL    |       |
| password   | varbinary(64) | YES  |     | NULL    |       |
+------------+---------------+------+-----+---------+-------+
```

`masteruser` **is used to identify passwords of the user currently logged in**

The key for encryption is generated using the master password string and a key deriving function called Scrypt

```python
from Crypto.Protocol.KDF import scrypt

def KDF(password, salt):
    key = scrypt(password, salt, 16, N=2**14, r=8, p=1)
    return key
```

The salt is randomlly generated and stored

Suppose an user wants to store the following

- **Service: Gmail**
- **Userid: user@gmail.com**
- **Password: password123**

The following will be stored

```
masteruser:
0x6E465FA2EA70AA8B1964D99D1A5666022083F7606B682CEAA05ABBF1277451FB
service: Gmail
userid: user@gmail.com
salt: 0xFAB11FE9E2905A60FDA31A3CF011CDD2
password: 0xC3E5AFAC26423717548AA3B585A22D14
```

Searching

The user can search the passwords in 3 ways:

- **By username**
- **By service**
- **By both username and service**

Example shows searching by both:

```
Enter user id: user
Enter service name: Gmail

Service Name:Gmail
User Id:user@gmail.com
Password:password123
```

User can also view all the stored passwords:

```
Choose an option:
 1. Save Password
 2. View all Passwords
 3. Search Password
 4. Delete Password
 5. Exit
Enter choice: 2

Service Name:Gmail
User Id:user@gmail.com
Password:password123
```

```
Service Name:Gmail
User Id:user2@gmail.com
Password:password456
```

Deleting

The user can use any of the above mentioned search techniques to look for passwords and can choose a number to select which password to delete

```
Choose an option:
 1. Search by username
 2. Search by service
 3. Search by both username and service
Enter choice: 1
Enter user id: user

1
Service Name:Gmail
User Id:user@gmail.com
Password:password123

2
Service Name:Gmail
User Id:user2@gmail.com
Password:password456

Enter number of the password to be deleted or 0 to cancel: 2
```

How to run the code

Pre-requisites

- **Python interpreter**
- **Command line**
- **MySQL**

Setting up

Creating a virtual environment

- **Create a folder in which you want to create a virtual environment.**
- **Open the terminal in the folder.**

- For MAC or Linux:
  - Run the commands:
    - `python3 -m venv env` : To create an environment
    - `source env/bin/activate` : To activate the environment
- Fow Windows:
  - Run the commands:
    - `python -m venv env` : To create an environment
    - `env/bin/activate` : To activate the environment

Running the code

- Extract the `zip` file in the folder where you created for the virtual environment.
- In the terminal run the command `pip install -r requirements.txt`, this will install all the dependencies.

Creating database

- Open MySQL on the command line
- Create a database using `CREATE DATABASE password_manager;`
  - Select the database using `USE password_manager;`
- Create the table for master passwords using

```
CREATE TABLE master (
    userid BINARY(32) NOT NULL PRIMARY KEY,
    salt BINARY(16) NOT NULL,
    password BINARY(32) NOT NULL
);
```

- Create the table for passwords using

```
CREATE TABLE passwords (
    masteruser BINARY(32) NOT NULL,
    service VARCHAR(64) NOT NULL,
    userid VARCHAR(64) NOT NULL,
    salt BINARY(16) NOT NULL,
    password VARBINARY(64) DEFAULT NULL
);
```

- Modify the database details in the `main.py` file

```
db = mysql.connector.connect(
    host="localhost",
    user="username",
    password="password",
```

```
    database = "password_manager"
    )
```

Running the code

- **Run the `main.py` file**
- **Follow the prompts**