My programming work is a password manager which is software application designed to help users manage their passwords securely. The program stores passwords in an encrypted form, protecting the user's data from unauthorized access. The program has an intuitive user interface that makes it easy for users to create and manage their passwords.

Structure of the Program:.

The program is written in Python and uses the Tkinter module for the user interface. SQLite is used to store the data. The program consists of several modules that work together to provide the desired functionality. The main modules of the program are:

GUI.py: The PasswordManagerGUI class is a part of a GUI-based password manager application that allows users to create an account, login, store and retrieve their passwords securely.

The *__init__* method initializes the root window with a fixed size and title, and sets an icon for the application. It also calls the main_menu method to display the main menu.

The *main_menu* method creates a frame that contains buttons for creating an account, logging in and exiting the application. If a database is found, a message is indicating that an account already exists and prompts the user to log in. Otherwise, user is guided to create an account.

The *create_account_menu* method creates a frame for the user to create a master password, confirm the password and submit the account details. If any error occurs during account creation, the *submit_account* method displays an error message indicating the nature of the error.

The *submit_account* method retrieves the passwords from the entries and creates a database with the master password. If any error occurs during database creation, an error message is displayed.

The *login_menu* method creates a frame that allows the user to enter their master password and log in to their account. The password is verified with *verify_login* method which also has prevention against brute force attacks. User has three attempts to write the correct password, after which account is locked for 5 seconds. After every three wrong attempts the waiting time is doubled. Attempts and lockout time is stored to the database so closing the program does not bypass the brute force prevention.

The *add_password_menu* method creates a frame that allows the user to store a new password with an application name/sitename, URL, email, and username. There are limitations for the length of all fields expect password and spaces are not allowed in any of the fields. Formats of URL and email are checked. User is notified if there are any problems.

The *passwords_menu* method creates a frame that displays a list of stored passwords, and allows user to copy the password to the clipboard or delete it from database. There are also buttons to add a new password, log out and delete the account entirely.

retrieve.py: This module contains functions for retrieving data from the encrypted database, decrypting password so they can be copied to the clipboard and deleting passwords from the database. It includes functions for validating the master password, updating and retrieving the login attempts count and lockout time.

add.py: Is used to store data to the database. *computeMasterKey* computes a master key using the Password-Based Key Derivation Function 2 (PBKDF2) with the hashed master password (mp) and device secret (ds) as inputs. *addEntry* checks if inputs are valid and then stores them to the database.

config.py has *checkConfig* which checks if the password manager database has been already configured by verifying the existence of required tables in the SQLite database. *generateDeviceSecret* generates a random device secret (salt) of the specified length using a combination of uppercase letters and digits. *make* function creates the password manager database if it has not been configured already. It also performs validation on the provided master password at account creation, hashes the masterpassword and creates the necessary tables for storing secrets, entries, and variables in the database. *delete* function drops all the tables from the database and removes the database ("pm.db") file.

dbconfig.py includes a function for creating and connecting to an SQLite database.

aesutil.py includes functions for encrypting and decrypting data using the Advanced Encryption Standard (AES) algorithm in Cipher Block Chaining (CBC) mode.

Secure Programming Solutions:

The password manager has been developed using secure programming practices to ensure that the user's data is safe and secure. Following security features has been implemented:

Encryption: The program encrypts the user's data using AES-256 encryption, which is a widely used and trusted encryption algorithm.

Password Policy: The program enforces a strong password policy, which requires the use of complex and unique mater password.

Secure Design: The interface of the application is very simple, so it helps narrowing possible flaws in security.

Login Attempts: The program tracks the number of login attempts and locks the user's account if there are too many failed attempts. This prevents brute-force attacks.

Input Validation: The program validates all user inputs with parametrized queries to prevent SQL injection attacks. These are used in *addEntry* function in **add.py**, and in *make* function in **config.py**.

Secure Password Storage: The program stores passwords in an encrypted form, which protects them from unauthorized access.

Error Handling: The program has robust error handling to prevent information leakage. All error messages are generic and do not disclose any sensitive information.

Figure 1 is slide from presentation to give clear picture of how the password encryption works in the program.

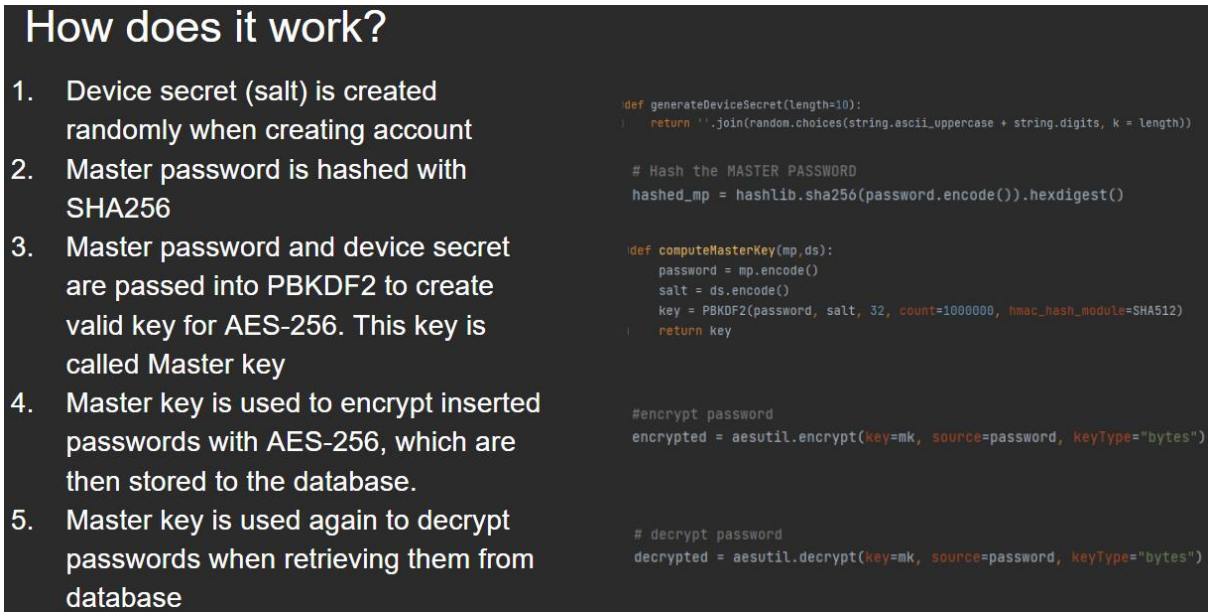

Figure 1: Slide from presentation to clarify password encryption

Tests:

With test_add_entry.py I found out that program does not accept URLs which begins with https://...

So I modified the code in *addEntry* from this:

```
if siteurl and not re.match(r'^[a-zA-Z0-9]+([-.][a-zA-Z0-9]+)*\.[a-zA-
Z]{2,}$', siteurl):
    raise ValueError("Invalid siteurl format.")
```

to this:

```
if siteurl and not re.match(r'^(?:http|ftp)s?://|www\.|[a-zA-Z0-9_.+-]+@[a-
zA-Z0-9-]+\.[a-zA-Z0-9-.]+|\w+\.\w+$',
                            siteurl):
    raise ValueError("Invalid siteurl format.")
```

During testing I also noticed that maybe I should not allow spaces in the fields other than password. So, I added condition for that too to *addEntry*.

I also noticed that if passwords in database are passing certain quantity, they hide the function buttons and the title shown in figure 2.

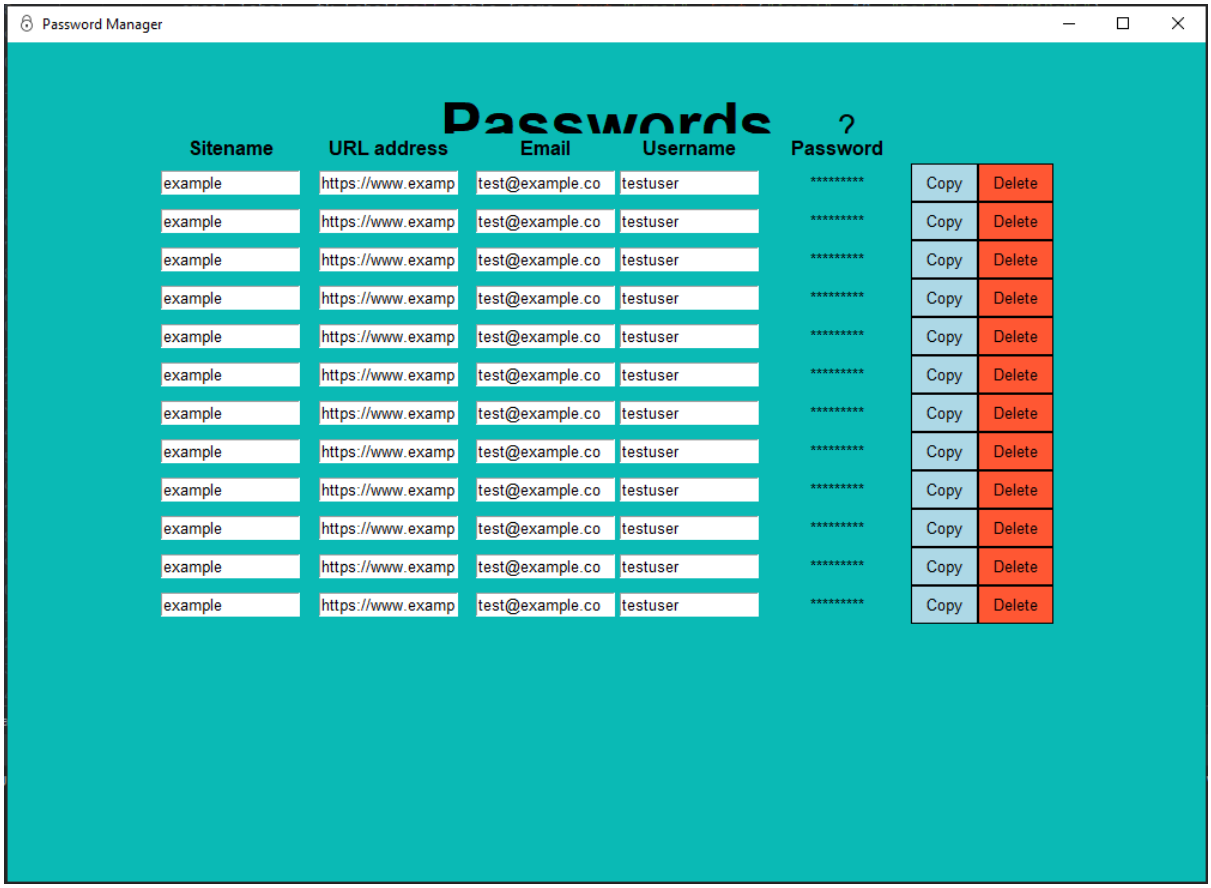

Figure 2: Wrong functionality of password menu

I did add a scrollbar, so the data does not cover the buttons or title anymore, but it does not work the I like. I could have fixed it to satisfy me but it is not the topic of this course so I let it pass.

Other functionalities I have tested by using the interface in many ways to test the correct functionality of the functions.

What could be done more?

There are no limits how far I could have developed this software but at some point I had to accept the program as it is. I could have added a log to keep note of logins. Also password generator was in my mind in some point but there was no time to implement it in this password manager.