杭州电子科技大学
网络空间安全专业生产实习报告

项目名称：密码管理器系统设计与实现

项目组负责人：刘超

2024 — 2025 学年 第 一 学期

目录

1 项目概述

1.1 概述

本文档概述了我们小组所开展的生产实习项目——密码管理器系统设计与实现。文档主要内容包括项目任务需求、设计理念、实际运行效果以及项目代码实现等。通过本文档，我们旨在展示项目从构思到实施的全过程，以及团队在确保用户密码安全性和实现多端同步方面的努力与成果。

本项目使用 Python 语言编写，图形化界面使用 Flet 框架，实现了 Windows 和 Web 平台的多端跨平台特性，后续还会增加更多的平台。本项目使用的 Python 第三方库包括：

- Flet，用于实现图形化界面
- Pyperclip，用于实现 Windows 平台的复制功能
- Gvcode，用于实现验证码的生成
- Pycryptodomex，用于实现多种加密算法
- Mysql-connector，用于连接数据库并执行数据库操作
- Pysm4，用于实现 SM4 国密加密算法

1.2 术语

术语/缩略词	解释
One-time Pad (OTP)	一次性密码本，一种理论上不可破解的加密算法
AES-256	高级加密标准，256 位密钥
ChaCha20	一种高效安全的流加密算法，256 位密钥
Xchacha20	ChaCha20 的一个变种，增加了密钥流的安全性
SM4	一种高安全性和高效率的国密对称加密算法
PBKDF2	生成加密密钥的密码派生函数，通过多轮迭代哈希实现

ECB	一种块加密模式，每个块的加密都是独立的

1.3 项目人员组成

姓名	学号	项目分工/角色
刘超	21010219	程序界面/整体设计/文档编写
纪凤	21036103	加解密模块/接口设计/文档撰写
周知	21271109	数据库模块/接口设计/文档撰写
张丽	21271108	数据库模块/接口设计/文档撰写

2 项目任务需求

2.1 安全性需求

1. **主密钥哈希**：采用 SHA3-256 算法对主密码进行 Hash 操作，确保主密码的安全性。
2. **数据加密**：使用 AES-256、ChaCha20、Xchacha20 以及 SM4 算法对存储的密码数据进行加密，确保数据的机密性和完整性。
3. **密钥生成**：加密密钥为主密钥通过 SHA-256 加盐 Hash 算法利用 PBKDF2 迭代生成，并混合随机数，保证每次加密都是 One-time Pad 模式，提高密钥的安全性。
4. **国密算法集成**：在符合国家密码管理局要求的前提下，支持 SM4 国密算法，后续还支持 SM3 作为哈希函数。

2.2 功能性需求

1. **软件界面**：提供一个直观易用的用户界面，支持用户注册、登录、密码管理等功能。
2. **数据条目管理**：实现对密码库中的数据条目（如网站、银行卡、信用卡等）进行增加、删除和搜索的功能。

3. **数据分类：**支持用户自定义数据条目的备注，便于管理和检索。
4. **数据同步：**实现跨设备的数据同步功能，确保用户在不同设备上能够访问到最新的数据。
5. **密码生成器：**提供密码生成器，帮助用户创建复杂且安全的密码。
6. **数据加密：**实现密码的加密和解密功能，确保用户密码以加密形式存储和传输。
7. **数据库设计：**设计并实现数据库结构，以安全地存储用户密码信息和同步数据。

2.3 性能需求

1. **响应时间：**确保在正常网络条件下，用户操作的响应时间不超过 2 秒。
2. **数据同步效率：**数据同步操作应在 2 秒内完成，不影响用户体验。
3. **资源消耗：**应用程序应优化资源使用，避免过度消耗系统资源。

2.4 可用性需求

1. **多平台支持：**至少在一种平台上实现基本功能，如网页、小程序、浏览器插件、iOS、Android、Windows 等。
2. **用户界面：**提供直观、易用的用户界面，确保用户能够轻松管理和使用密码库。
3. **软件稳定：**提供稳定的服务，确保高可用性，减少系统故障和停机时间。

3 概要设计

3.1 项目系统架构

- **项目部署图：**

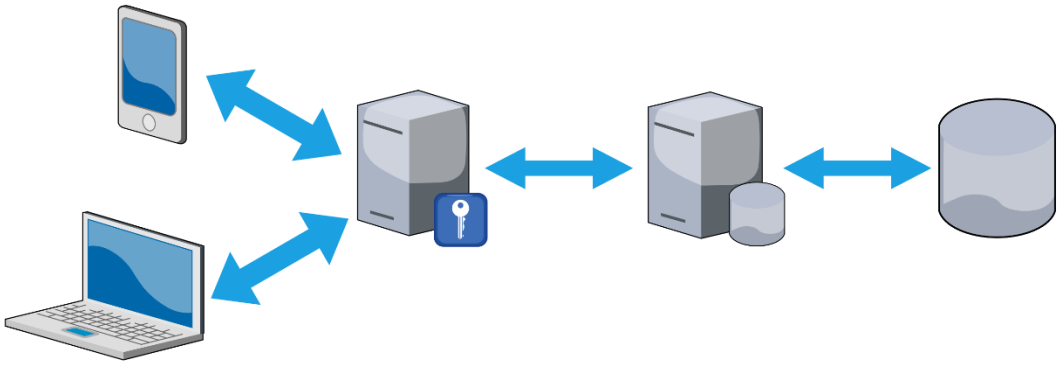

图 3-1 项目部署图

- **模块划分：**

表格 3-1 模块划分表

模块	功能
前端页面	登录/注册界面
	密码管理界面
	密码搜索界面
	密码信息展示界面
后端处理	用户认证服务
	密码管理服务
	数据加密服务
	数据库操作服务
数据库	用户信息表
	密码信息表

- **系统架构图：**

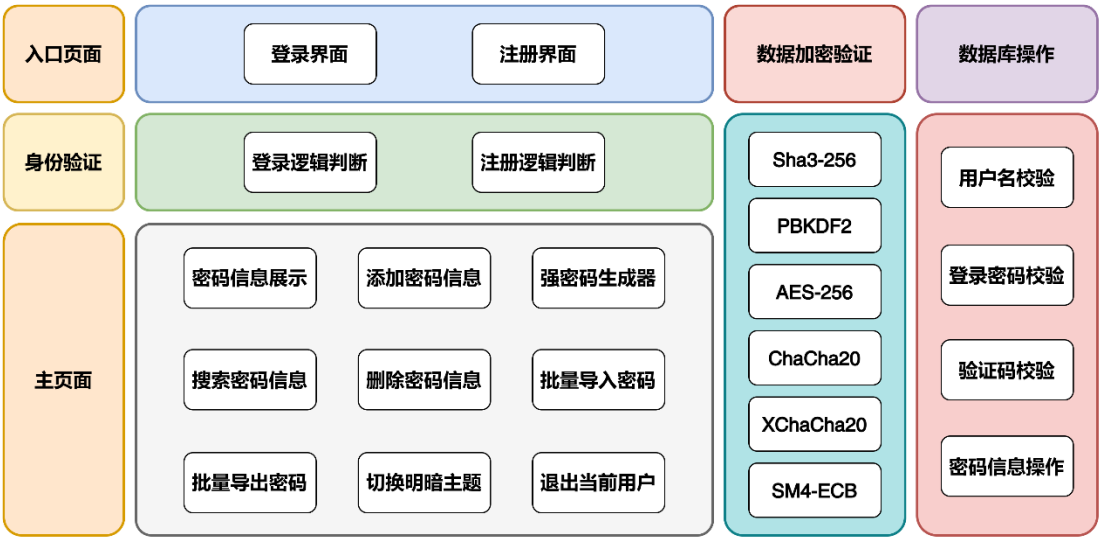

图 3-2 系统架构图

- **工作流程：**

1. 用户通过入口页面发起登录或注册请求。

2. 经过数据加密验证模块验证用户信息，与数据库交互进行身份验证。

3. 认证成功后，用户进入主界面可以管理密码，包括添加、查询和删除密码信息。同时还支持强密码生成，批量导入导出密码以及切换明暗主题的功能。

4. 密码信息在数据加密模块通过用户自行选择的加密方式进行加密，加密后的密码信息存储在数据库中。

5. 用户查询密码时，服务从数据库中检索并解密密码信息，返回给前端页面。

3.2 项目接口设计

本项目的接口设计主要分为三大模块，分别是前端页面接口，数据加密接口以及数据库操作接口。

- **前端页面接口：**

1. 登录事件接口

```
def login_click(self, e): → page.update(), database_op
```

2. 注册事件接口

```
def register_click(self, e): → page.update(), database_op
```

3. 验证码生成接口

```
def gen_vfcode(): → vfcode.save()
```

4. 添加密码事件接口

```
def add_pwn_dlg_save(self, e): → page.update(), database_op
```

5. 随机生成强密码接口

```
def add_random_pwd(self, e): → page.update()
```

6. 搜索密码事件接口

```
def search_box_submit(self, e): → page.update(), database_op
```

7. 删除密码事件接口

```
def delete_pwd_row(self, pwd_row): → page.update(), database_op
```

8. 复制单元格事件接口

```
def copy_cell(self, e): → page.update()
```

9. 导入密码事件接口

```
def import_pwd_dlg_save(self, e): → page.update(), database_op
```

10. 导出密码事件接口

```
def export_pwd_dlg_save(self, e): → page.update(), database_op
```

- **数据加密接口：**

1. Sha3-256 哈希

```
def sha3_256(data: {encode}) -> str
```

2. PBKDF2-sha 密码派生

```
def pbkdf2(data: {encode}) -> str
```

3. AES 加密

```
def aes_encrypt(data: {encode},key: Any) -> str
```

4. AES 解密

```
def aes_decrypt(data: Any,key: Any) -> str
```

5. ChaCha20 加密

```
def chacha20_encrypt(data: {encode},key: Any) -> str
```

6. ChaCha20 解密

```
def chacha20_decrypt(data: Any,key: Any) -> str
```

7. Xchacha20 加密

```
def xchacha20_encrypt(data: {encode},key: Any) -> str
```

8. Xchacha20 解密

```
def xchacha20_decrypt(data: Any,key: Any) -> str
```

9. SM4-ECB 加密

```
def sm4_encrypt(data: Any,key: {__len__}) -> str
```

10. SM4-ECB 解密

```
def sm4_decrypt(data: Any,key: {__len__}) -> str
```

11. 导出数据为 JSON

```
def export_password_to_json(data:list,username: Any,key: Any,dir_path: Any)
 -> int
```

12. 导出数据为 CSV

```
def export_password_to_csv(data:list,username: Any,key: Any,dir_path: Any)
-> int
```

13. 导出数据为 TXT

```
def export_password_to_txt(data:list,username: Any,key: Any,dir_path: Any)
-> int
```

14. 从 JSON 导入数据

```
def import_password_from_json(username: {__ne__},file_path: Any) -> list[di
ct[str, Any]] | int
```

15. 从 CSV 导入数据

```
def import_password_from_csv(username: {__ne__},file_path: Any) -> list[dic
t[str, str]] | int
```

16. 从 TXT 导入数据

```
def import_password_from_txt(username: Any,file_path: Any) -> list[dict[st
r, str]] | int
```

17. 随机生成强密码

```
def random_password(length: Any) -> str
```

- **数据库操作接口：**

1. 创建数据库连接

```
def create_connection(host_name: Any,
                      user_name: Any,
                      user_password: Any,
                      db_name: Any) -> MySQLFabricConnection | CMySQLConnect
ion | MySQLConnection | None
```

2. 关闭数据库连接

```
def close_connection(connection: Any) -> None
```

3. 用户表查询

```
def query_user(connection: {cursor},
               username: Any) -> int | None
```

4. 用户表插入

```
def insert_user(connection: {cursor, commit},
```

```
                username: Any,
                password_hash: Any) -> int
```

5. 密码表查询

```
def query_password(connection: {cursor},
                   username: Any) -> None | list[dict[str, Any]] | int
```

6. 密码表插入

```
def insert_password(connection: {cursor, commit},
                    username: Any,
                    website_name: Any,
                    website: Any,
                    account: Any,
                    password_encrypted: Any,
                    encrypted_method: Any,
                    note: Any) -> int
```

7. 查询密码创建时间

```
def query_password_created_at(connection: {cursor},
                              username: Any,
                              website_name: Any,
                              account: Any) -> int | None
```

8. 密码表删除

```
def delete_password(connection: {cursor, commit},
                    username: Any,
                    website_name: Any,
                    account: Any) -> int
```

9. 按关键字查询密码表

```
def search_password(connection: {cursor},
                    username: Any,
                    keywords: {__eq__}) -> None | list[dict[str, Any]] | int
 | list[dict[str, Any]]
```

3.3 数据结构设计

- **关键数据结构：**

1. 全局系统变量

```
# 全局主题变量
BGCOLOR = '#f0f0f0'
THEME = ft.ThemeMode.LIGHT
# 状态码
OK = 1
```

```
ERROR = -1
```

2. 登录页面类

```
class LoginPage(ft.Column)
```

3. 注册页面类

```
class RegisterPage(ft.Column)
```

4. 添加密码信息弹窗类

```
class AddPwdDialog(ft.AlertDialog)
```

5. 密码信息类

```
class PwdRow(ft.DataRow)
```

6. 主页面类

```
class MainPage(ft.Column)
```

7. 页面配置

```
def main_page(page: {clean, window, add, update, on_disconnect},
              current_user: Any,
              current_key: Any) -> None
```

- **数据库设计：**

1. 用户表设计

```
username VARCHAR(255) NOT NULL PRIMARY KEY,
password_hash VARCHAR(255) NOT NULL,
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

2. 密码信息表设计

```
username VARCHAR(255) NOT NULL,
website_name VARCHAR(150) NOT NULL,
website VARCHAR(150),
account VARCHAR(150) NOT NULL,
password_encrypted VARCHAR(255) NOT NULL,
encrypted_method VARCHAR(50) NOT NULL,
note TEXT,
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY (username, website_name, account),
FOREIGN KEY (username) REFERENCES user(username)
```

4 详细设计

4.1 图形界面模块设计

● 模块概述

在这一部分，主要是详细描述密码管理器应用程序的图形用户界面（GUI）设计，包括模块内详细算法设计、伪代码、流程图和详细接口。

我们设计这个图形化界面的原则是用户友好的，界面简洁直观，易于新用户学习和使用。

● 界面设计

1. 主界面

- 背景颜色：可切换的明暗主题。
- 窗口尺寸：宽度 1200px，高度 700px，可居中显示。
- 标题栏：显示应用程序名称“密码管理器”。
- 功能按钮：
  - 添加密码按钮：图标表示，提示为“添加新的密码信息”。
  - 搜索框：允许输入关键字搜索密码信息。
  - 批量导入按钮：图标表示，提示为“批量导入密码信息”。
  - 批量导出按钮：图标表示，提示为“批量导出密码信息”。
  - 退出按钮：图标表示，提示为“退出当前用户”。
  - 主题切换按钮：图标表示，提示为“切换明暗模式”。

2. 登录界面

- 背景颜色：与主界面一致。
- 窗口尺寸：宽度 600px，高度 500px。
- 标题：显示“密码管理器”。
- 输入框：
  - 用户名输入框：提示为“请输入你的用户名”。
  - 密码输入框：提示为“请输入你的主密码”，密码隐藏。

- 按钮：
  - 登录按钮：文本为“登入”，点击触发登录验证。
  - 注册按钮：文本为“注册”，点击跳转到注册界面。
- 错误提示：当用户名或密码错误时，显示弹窗提示。

3. 注册界面

- 背景颜色：与主界面一致。
- 窗口尺寸：宽度 600px，高度 500px。
- 标题：显示“注册用户”。
- 输入框：
  - 用户名输入框：提示为“请输入你的用户名”。
  - 密码输入框：提示为“请输入你的主密码”，密码隐藏。
  - 验证码输入框：与验证码图片并排，用于验证用户操作。
- 按钮：
  - 注册按钮：文本为“注册”，点击触发注册流程。
  - 取消按钮：文本为“取消”，点击返回主界面。
- 错误提示：包括用户名已存在、验证码错误、输入为空等提示。

4. 添加密码信息弹窗

- 标题：显示“添加新密码信息”。
- 输入框：包括名称、账号、密码、网址、备注、加密方式等字段。
- 功能按钮：
  - 随机密码按钮：点击生成随机强密码。
  - 保存按钮：点击保存密码信息到数据库。
  - 取消按钮：点击关闭弹窗。

5. 密码信息列表

- 显示格式：表格形式，包括名称、账号、密码、网址、备注、创建日期、删除按钮。
- 操作：
  - 点击密码单元格：复制密码到剪贴板。

- 点击删除按钮：弹出确认删除弹窗。

6. 导入导出数据弹窗

- 输入框：
  - 导入弹窗：选择文件路径输入框。
  - 导出弹窗：选择目录路径输入框和导出格式选择。
- 按钮：
  - 导入/导出按钮：点击执行导入/导出操作。
  - 取消按钮：点击关闭弹窗。

● 设计要求

- 弹窗：在需要用户确认或额外信息时使用，如错误提示、操作确认等。
- 数据验证：在用户输入信息后，系统进行验证，如检查格式、是否已存在等。
- 前端框架：使用 flet 库进行 GUI 开发。
- 后端服务：数据库操作，包括用户验证、密码信息存储等。
- 安全性：所有密码信息必须加密存储和传输。

● 伪代码

1. 登录页面类

```
# 登录页面类
LoginPage:
    - 初始化：设置标题、用户名和密码输入框、登录和注册按钮、错误弹窗
    - 登录按钮点击事件：检查用户名和密码，验证成功后进入主页面，失败则显示错误弹窗
- 注册按钮点击事件：跳转到注册页面
```

2. 注册页面类

```
# 注册页面类
RegisterPage:
    - 初始化：设置标题、用户名和密码输入框、验证码、注册和取消按钮、各种错误弹窗和成功弹窗
    - 注册按钮点击事件：验证输入信息和验证码，成功后添加用户到数据库并跳转到登录页面
    - 取消按钮点击事件：返回到主页面
```

3. 添加密码信息弹窗类

```
# 添加密码信息弹窗类
AddPwdDialog:
    - 初始化：设置标题、密码信息输入框、保存和取消按钮
    - 随机生成密码：生成一个强密码
    - 保存按钮点击事件：验证输入信息，加密密码后保存到数据库，更新密码列表
    - 取消按钮点击事件：关闭弹窗
```

4. 密码信息类

```
# 密码信息类
PwdRow:
    - 初始化：设置密码信息的显示
    - 删除按钮点击事件：显示删除确认弹窗
    - 删除确认弹窗确定按钮：从数据库和界面上删除密码信息
    - 删除确认弹窗取消按钮：关闭删除确认弹窗
```

5. 主页面类

```
# 主页面类
MainPage:
    - 初始化：设置添加密码按钮、搜索框、批量导入导出按钮、密码信息表
    - 添加密码按钮点击事件：打开添加密码信息弹窗
    - 搜索框提交事件：根据关键词搜索密码信息并更新列表
    - 删除密码表格信息：从数据库和界面上删除密码信息
    - 复制单元格内容：复制选中的单元格内容到剪贴板
    - 导出密码点击事件：打开导出密码信息弹窗
    - 导入密码点击事件：打开导入密码信息弹窗
```

6. 切换明暗主题函数

```
# 切换明暗主题函数
change_theme:
    - 切换应用程序的主题模式和背景颜色
```

7. 主页面函数

```
# 主页面函数
main_page:
    - 设置页面属性，如大小、位置
    - 添加主页面对象到页面
    - 更新页面
```

8. 注册页面函数

```
# 注册页面函数
Register_page:
    - 设置页面属性，如大小、位置
    - 添加注册页面对象到页面
    - 更新页面
```

9. 登录页面函数

```
# 登录页面函数
main:
    - 设置页面属性，如大小、位置、标题
    - 添加登录页面对象到页面
    - 更新页面
```

● 各部分流程图

1. 登录逻辑流程图

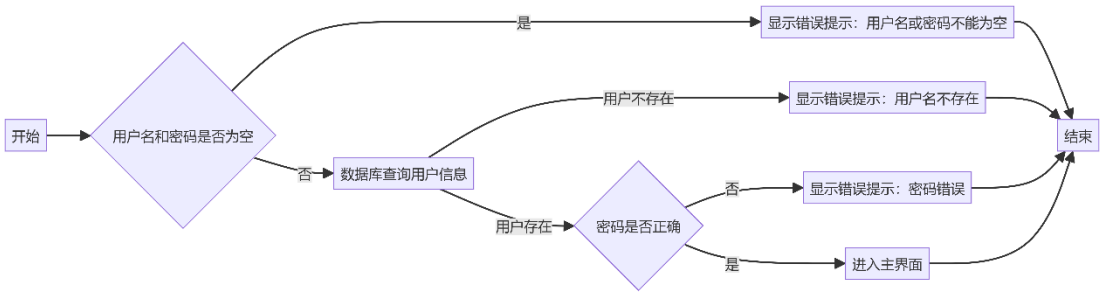

图 4-1 用户登录流程图

2. 注册逻辑流程图

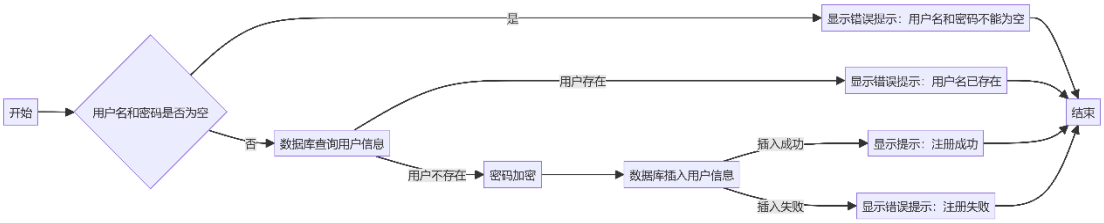

图 4-2 用户注册流程图

3. 添加密码信息流程图

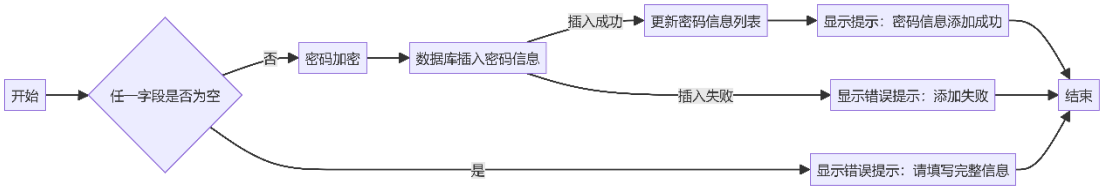

图 4-3 添加密码流程图

4. 删除密码信息流程图

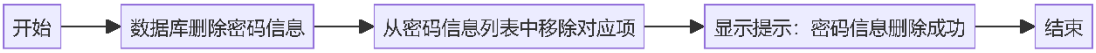

图 4-4 删除密码流程图

5. 导入数据流程图

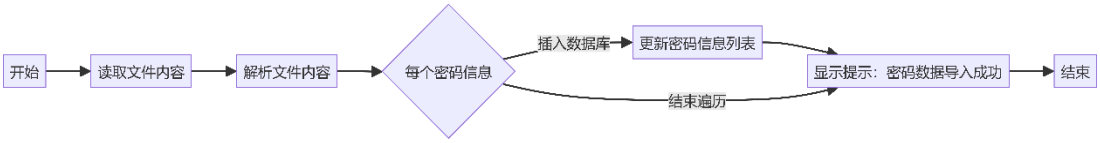

图 4-5 导入密码流程图

6. 导出数据流程图

图 4-6 导出密码流程图

4.2 数据加密模块设计

- **模块概述**

密码模块是密码管理器的核心部分，负责加密、解密、生成和验证密码。该模块确保用户数据的安全性和隐私性。

- **算法设计**

本项目的密码模块支持多种加密算法，同时还集成由国密算法，并且实现了哈希函数以及密钥派生函数，使得整个系统具有十分高的安全性。

支持的密码算法：AES-256、ChaCha20、Xchacha20、SM4-ECB。

支持的哈希和派生算法：SHA-256、SHA3-256、PBKDF2。

后续还会支持更多的密码学算法，以进一步加强软件的安全性。

在这个模块中我们把用户登录软件的密钥作为整个密码存储库的主密钥，所有的密码信息都是靠这个主密钥进行加解密，这样使用户在使用我们软件的时候对于后端的数据加解密操作是无感的，因为他登录进软件后，不需要再次输入加解密的密钥，这也符合我们软件的设计原则：用户友好的。

为了解决整个密码库都使用同一个主密钥使得安全性不高的问题，我们进行了如下的设计：

1. 用户的主密钥使用 SHA3-256 哈希处理后放入数据库中，保护主密钥。
2. 用于加密密码信息的密钥是通过主密钥使用 PBKDF2 密钥派生函数，以加盐的 SHA-256 作为其中的哈希函数，迭代 10000 轮后生成的。
3. 在本项目支持的密码算法中，AES-256、ChaCha20 和 Xchacha20 在实

际过程中会混合进随机数，相当于每次加密都使用不同的密钥，因此即使是使用相同明文加密多次，每次生成的密文都会不同，通过这种方式，我们软件实现了接近一次一密(One-time Pad)的加密方式，使得整个系统的安全性大幅提升。

4. 用户可以自定义选择密码信息使用的加密算法。但我们更推荐 AES-256、ChaCha20 和 Xchacha20 作为加密算法。

● 伪代码

1. SHA3-256

```
# 使用 SHA3-256 哈希算法对数据进行哈希
def sha3_256(data):
    return data 的 SHA3-256 哈希值
```

2. PKBDF2

```
# 使用 PBKDF2 算法和 SHA256 哈希函数生成密钥
def pbkdf2(data):
    salt = 固定的盐值
    return data 使用 SHA256 和 salt 进行 PBKDF2 迭代 10000 次的哈希值
```

3. AES-256

```
# 使用 AES-256 算法加密数据
def aes_encrypt(data, key):
    key = 将 key 转换为字节
    cipher = 创建 AES 加密器
    nonce = cipher 的随机数
    ciphertext, tag = cipher 加密 data
    return nonce + tag + ciphertext 的十六进制字符串

# 使用 AES-256 算法解密数据
def aes_decrypt(data, key):
    key = 将 key 转换为字节
    nonce = 从 data 中提取 nonce
    tag = 从 data 中提取 tag
    ciphertext = 从 data 中提取 ciphertext
    cipher = 创建 AES 解密器
    return cipher 解密 ciphertext 并验证 tag 得到的原始数据
```

4. ChaCha20

```
# 使用 ChaCha20 算法加密数据
def chacha20_encrypt(data, key):
    key = 将 key 转换为字节
    nonce = 生成 12 字节随机数
```

```
    cipher = 创建 ChaCha20 加密器
    ciphertext = cipher 加密 data
    return nonce + ciphertext 的十六进制字符串

# 使用 ChaCha20 算法解密数据
def chacha20_decrypt(data, key):
    key = 将 key 转换为字节
    nonce = 从 data 中提取 nonce
    ciphertext = 从 data 中提取 ciphertext
    cipher = 创建 ChaCha20 解密器
    return cipher 解密 ciphertext 得到的原始数据
```

5. Xchacha20

```
# 使用 XChaCha20 算法加密数据（与 ChaCha20 类似，但使用 24 字节的 nonce）
def xchacha20_encrypt(data, key):
    ...

# 使用 XChaCha20 算法解密数据（与 ChaCha20 类似，但使用 24 字节的 nonce）
def xchacha20_decrypt(data, key):
    ...
```

6. SM4-ECB

```
# 使用 SM4 算法加密数据
def sm4_encrypt(data, key):
    如果 key 超过 16 字节，则截断
    return 使用 ECB 模式加密 data 得到的十六进制字符串

# 使用 SM4 算法解密数据
def sm4_decrypt(data, key):
    如果 key 超过 16 字节，则截断
return 使用 ECB 模式解密 data 得到的原始数据
```

7. 随机生成强密码

```
# 随机生成强密码
def random_password(length):
    alphabet = 字母+数字+特殊字符
    return 随机选择 alphabet 中的字符生成指定长度的密码
```

8. 导入数据

```
# 从 JSON 文件导入密码信息
def import_password_from_json(username, file_path):
    尝试从 JSON 文件读取数据并过滤出 username 匹配的数据

# 从 CSV 文件导入密码信息
def import_password_from_csv(username, file_path):
```

```
    尝试从 CSV 文件读取数据并过滤出 username 匹配的数据

# 从 TXT 文件导入密码信息
def import_password_from_txt(username, file_path):
    尝试从 TXT 文件读取数据并过滤出 username 匹配的数据
```

9. 导出数据

```
# 将密码信息导出为 JSON 格式
def export_password_to_json(data, username, key, dir_path):
    key = 使用 PBKDF2 算法生成密钥
    尝试将 data 转换为 JSON 格式并保存到文件

# 将密码信息导出为 CSV 格式
def export_password_to_csv(data, username, key, dir_path):
    key = 使用 PBKDF2 算法生成密钥
    尝试将 data 转换为 CSV 格式并保存到文件

# 将密码信息导出为 TXT 格式
def export_password_to_txt(data, username, key, dir_path):
    key = 使用 PBKDF2 算法生成密钥
    尝试将 data 转换为 TXT 格式并保存到文件
```

- **流程图**

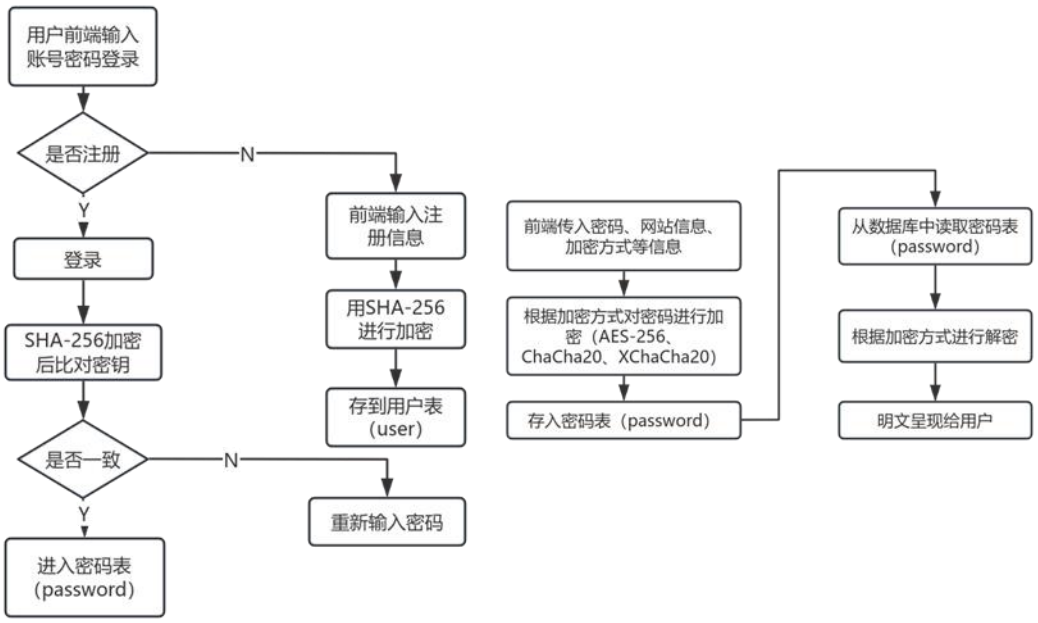

图 4-7 数据加密验证流程图

- **详细接口**

1. 加密接口

```
# 数据加密接口
def data_encrypt(data, key, algorithm):
  if algorithm == 'AES-256':
    return aes_encrypt(data, key)
  elif algorithm == 'ChaCha20':
    return chacha20_encrypt(data, key)
  elif algorithm == 'XChaCha20':
    return xchacha20_encrypt(data, key)
  elif algorithm == 'SM4-ECB':
    return sm4_encrypt(data, key)
  else:
    return None
```

2. 解密接口

```
# 数据解密接口
def data_decrypt(data, key, algorithm):
  if algorithm == 'AES-256':
    return aes_decrypt(data, key)
  elif algorithm == 'ChaCha20':
    return chacha20_decrypt(data, key)
  elif algorithm == 'XChaCha20':
    return xchacha20_decrypt(data, key)
  elif algorithm == 'SM4-ECB':
    return sm4_decrypt(data, key)
  else:
    return None
```

3. 导出接口

```
# 导出数据接口
def export_password(data: list, username, key, dir_path, export_format):
  if export_format == 'JSON':
    return export_password_to_json(data, username, key, dir_path)
  elif export_format == 'CSV':
    return export_password_to_csv(data, username, key, dir_path)
  elif export_format == 'TXT':
    return export_password_to_txt(data, username, key, dir_path)
  else:
    return ERROR
```

4. 导入接口

```
# 导入数据接口
def import_password(username, file_path, import_format):
```

```
if import_format == 'JSON':
  return import_password_from_json(username, file_path)
elif import_format == 'CSV':
  return import_password_from_csv(username, file_path)
elif import_format == 'TXT':
  return import_password_from_txt(username, file_path)
else:
  return ERROR
```

4.2.8 问题及解决方案

➢ 选择合适的加密算法

问题：确定哪种加密算法最适合项目需求，同时保证安全性和性能。

解决方案：对不同的加密算法（如 AES、ChaCha20、XChaCha20）进行研究和比较，考虑它们的加密强度、速度和资源消耗。最终选择了 AES-256、ChaCha20 和 XChaCha20，因为它们在安全性和性能上都有良好的表现，并且适用于不同的使用场景。

➢ 密钥管理

问题：如何安全地生成、存储和分发密钥。

解决方案：使用 SHA-256 哈希算法从用户的主密码生成密钥，确保密钥的随机性和不可预测性。密钥不在任何地方明文存储，而是通过哈希函数在需要时动态生成。

➢ 加密和解密的效率

问题：加密和解密过程需要快速且高效，以免影响用户体验。

解决方案：优化代码，减少不必要的计算和内存使用。同时，选择了性能较好的加密库，如 Cryptodome，来执行加密和解密操作。

➢ 数据格式的兼容性

问题：加密和解密过程中需要确保数据格式的兼容性，以便于存储和传输。

解决方案：定义了统一的数据格式，如使用十六进制字符串来表示加密后的数据。在加密和解密过程中，确保数据的格式转换正确无误。

4.3 数据库操作模块设计

- **模块概述**

数据库模块负责安全存储用户的敏感信息，用户名、密码信息。在本项目中，数据库模块的操作较少，相较数据加密模块来说更为简单，但是为了实现跨平台的多端同步，我们将数据库部署在远程服务器上，因此需要注意数据库的本身的防护，设置强密码，开启权限控制，提高数据库的安全性。而在数据库之内，由于数据都是加密处理过后的，因此数据都比较安全，数据库模块设计的重点在于数据库表的设计，以及数据库的操作。

● 数据表设计

1. 用户表

表格 4-1 用户表设计

字段	类型	说明
username	varchar(255)	主键，不能为空
password_hash	varchar(255)	不能为空
created_at	datetime	自动填入当前时间

SQL：

```
username VARCHAR(255) NOT NULL PRIMARY KEY,
password_hash VARCHAR(255) NOT NULL,
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

2. 密码表

表格 4-2 密码表设计

字段	类型	说明
username	varchar(255)	外键连接用户表，不能为空
website_name	varchar(255)	不能为空
website	varchar(255)	
account	varchar(255)	不能为空
password_encrypted	varchar(255)	不能为空
encrypted_method	varchar(50)	不能为空
note	text	
created_at	datetime	自动填入当前时间
(username, website_name, account)为主键		

SQL：

```
username VARCHAR(255) NOT NULL,
website_name VARCHAR(255) NOT NULL,
website VARCHAR(255),
account VARCHAR(255) NOT NULL,
password_encrypted VARCHAR(255) NOT NULL,
encrypted_method VARCHAR(50) NOT NULL,
note TEXT,
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY (username, website_name, account),
FOREIGN KEY (username) REFERENCES user(username)
```

● 详细接口与伪代码

1. 创建数据库连接

```
def create_connection(HOST_NAME, USER_NAME, USER_PASSWORD, DB_NAME)
    尝试连接到数据库
    如果连接成功
        记录日志信息 "Connection to DB successful"
    否则
        记录错误日志 "Create connection error: {错误信息}"
    返回连接对象
```

2. 关闭数据库连接

```
def close_connection(连接对象)
    如果连接对象不为空
        关闭连接
        记录日志信息 "Connection closed"
```

3. 根据用户名查询用户信息

```
def query_user(连接对象, 用户名)
    创建游标
    尝试执行查询语句
    如果查询结果为空
        返回 None
    否则
        返回查询到的第一个结果的密码
    捕获异常并记录错误日志
    关闭游标
```

4. 插入用户信息

```
def insert_user(连接对象, 用户名, 密码哈希)
    创建游标
```

```
    尝试执行插入语句
    提交事务
    如果成功
        返回 OK
    否则
        记录错误日志
        返回 ERROR
    关闭游标
```

5. 查询用户的密码信息

```
def query_password(连接对象, 用户名)
    创建游标
    尝试执行查询语句
    如果查询结果为空
        返回 None
    否则
        将查询结果转换为字典列表
        返回字典列表
    捕获异常并记录错误日志
    关闭游标
```

6. 插入用户的密码信息

```
def insert_password(连接对象, 用户名, 网站名称, 网站, 账号, 加密密码, 加密方法, 备注)
    创建游标
    尝试执行插入语句
    提交事务
    如果成功
        返回 OK
    否则
        记录错误日志
        返回 ERROR
    关闭游标
```

7. 根据用户名、网站名和账号查询密码创建时间

```
def query_password_created_at(连接对象, 用户名, 网站名称, 账号)
    创建游标
    尝试执行查询语句
    如果查询结果为空
        返回 None
    否则
        返回查询到的第一个结果的创建时间
    捕获异常并记录错误日志
    关闭游标
```

8. 删除用户的密码信息

```
def delete_password(连接对象，用户名，网站名称，账号)
    创建游标
    尝试执行删除语句
    提交事务
    如果成功
        返回 OK
    否则
        记录错误日志
        返回 ERROR
    关闭游标
```

9. 根据关键词搜索密码信息

```
def search_password(连接对象，用户名，关键词)
    如果关键词为空
        调用 query_password 函数
    否则
        分割关键词
        构建 SQL 查询语句
        执行查询
        如果查询结果为空
            返回 None
        否则
            将查询结果转换为字典列表
            返回字典列表
    捕获异常并记录错误日志
```

- **安全性考虑**

- 异常处理：在每个数据库操作函数中，使用了 try-except 块来捕获 mysql.connector.Error 异常，并在发生错误时记录错误信息。有助于防止程序因未处理的异常而崩溃，并且可以提供错误诊断信息。
- 日志记录：使用了 logging 模块来记录操作日志，包括成功连接数据库和发生错误时的情况。有助于监控系统状态和排查问题。
- 资源管理：在每个数据库操作函数中，在 finally 块中关闭了数据库游标，可以防止资源泄漏。
- 敏感信息保护：没有在代码中硬编码数据库的用户名和密码，而是将其作为参数传递给函数，可以避免在代码库中暴露敏感信息。

5 实验结果与截图

5.1 实验配置

- **Windows 平台**
  - 机器型号：CPU AMD Ryzen 7 5700U with Radeon Graphics 1.80 GHz, RAM 16GB
  - 操作系统：Windows 10 64 位 企业版 22H2 19045.4894
- **Web 平台**
  - 机器型号：阿里云 ECS CPU 2 核心 RAM 2GB
  - 操作系统：Debian 11.9 64 位
- **远程数据库**
  - 10.5.23-MariaDB-0+deb11u1

5.2 功能测试

- **Windows 平台**

图 5-1 软件详细信息

图 5-2 登录界面

图 5-3 注册界面

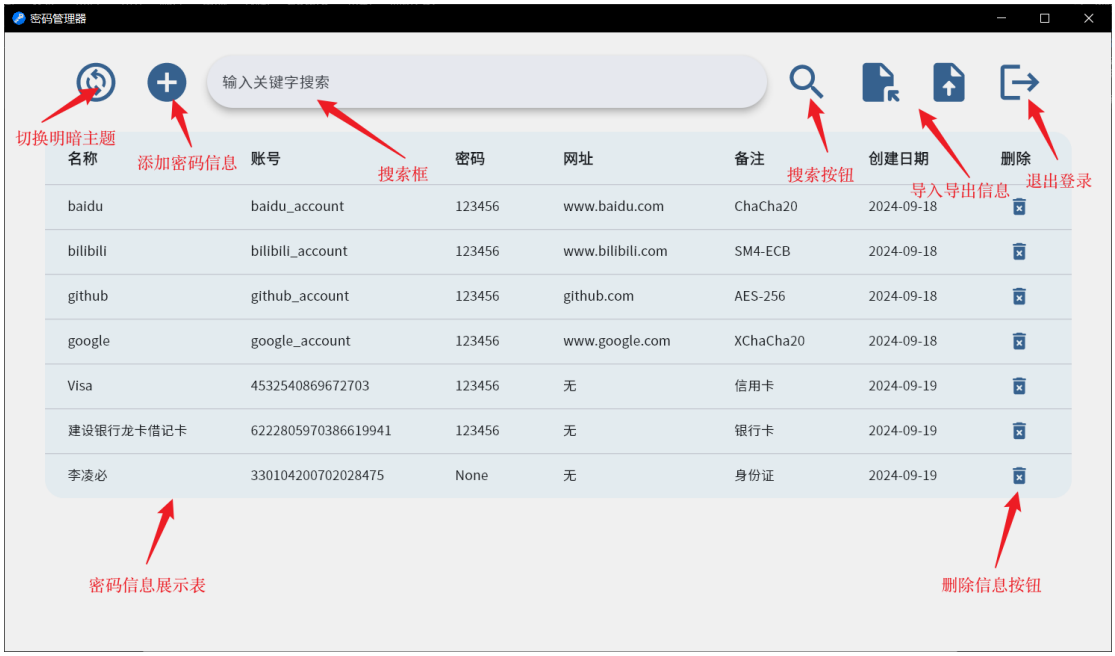

图 5-4 主界面

图 5-5 暗色主题

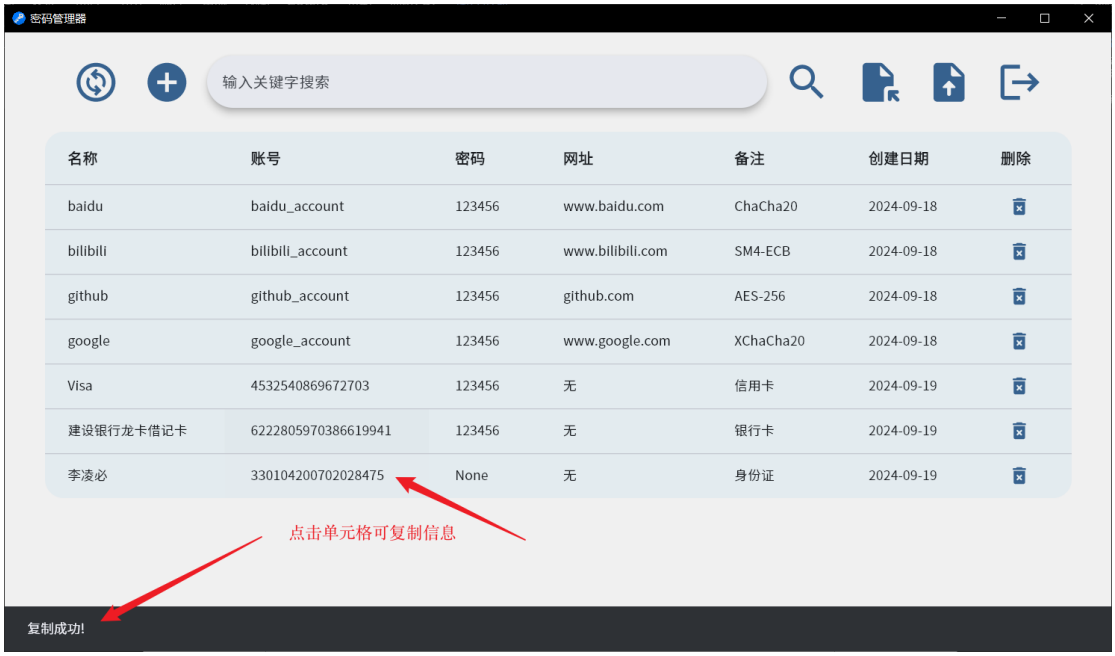

图 5-6 点击单元格复制信息

图 5-7 添加密码信息界面

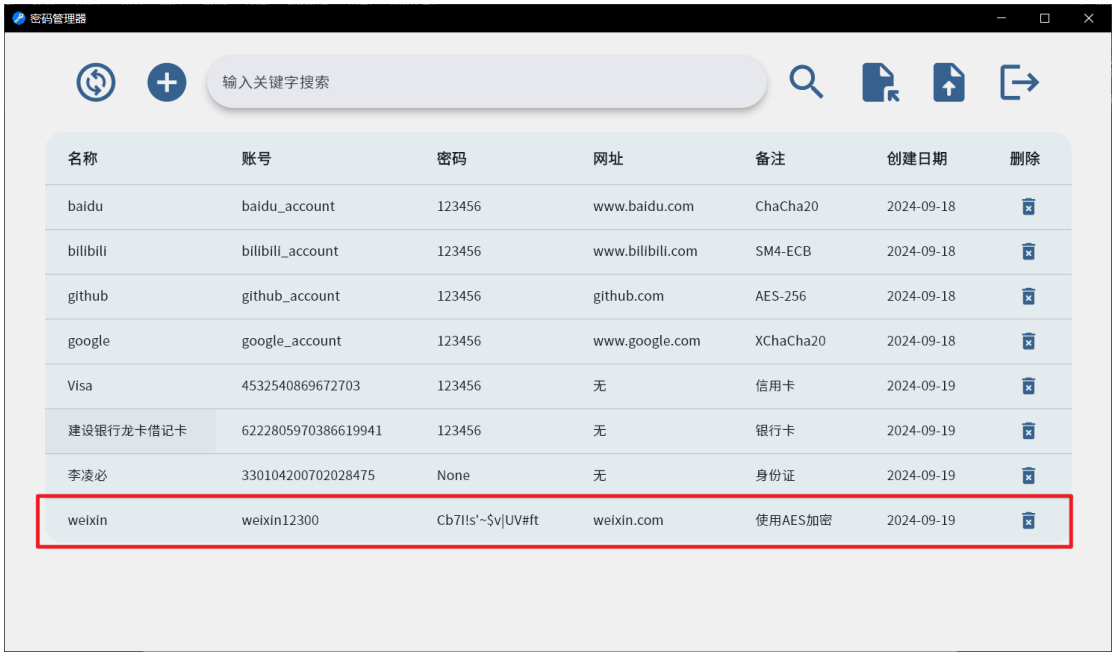

图 5-8 新添加的密码信息

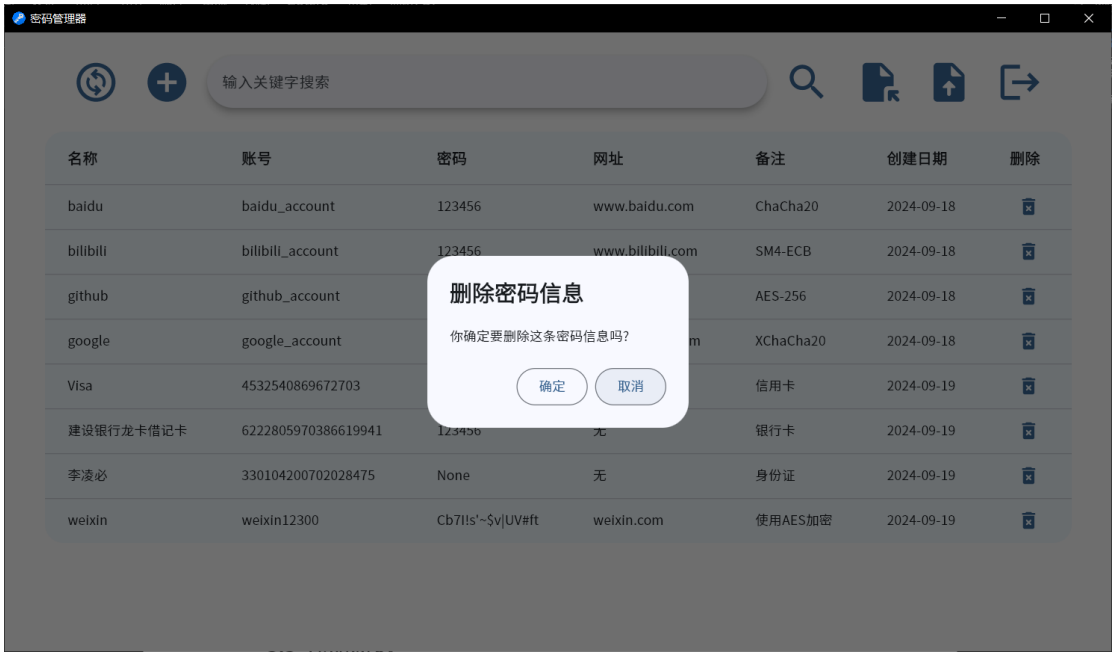

图 5-9 删除密码信息弹窗

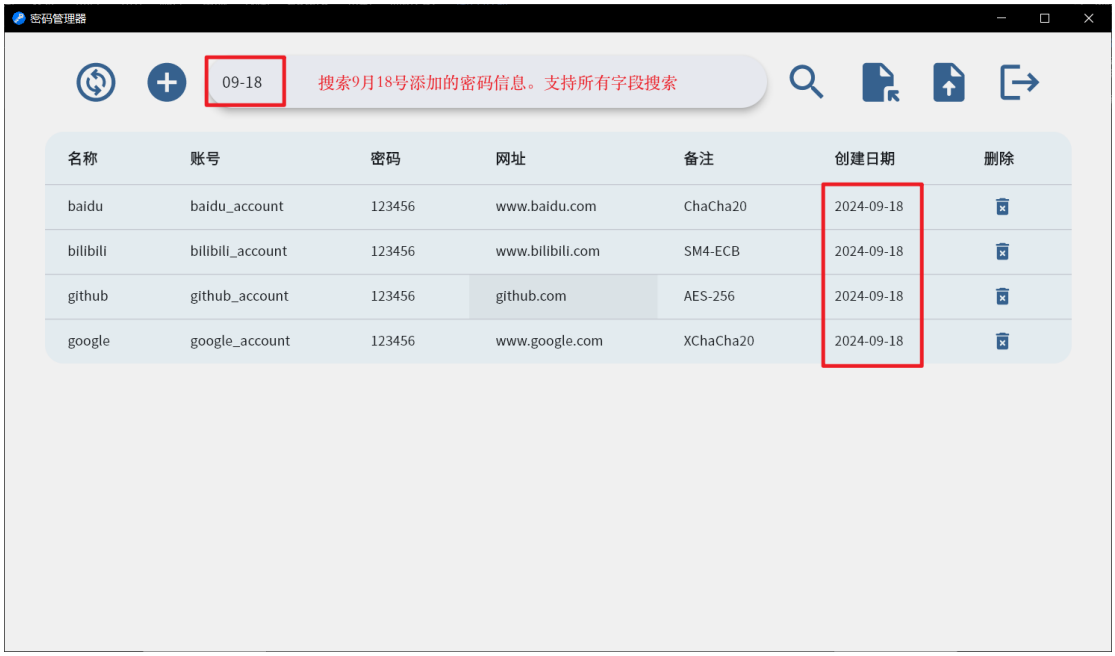

图 5-10 搜索功能展示

图 5-11 导出功能弹窗

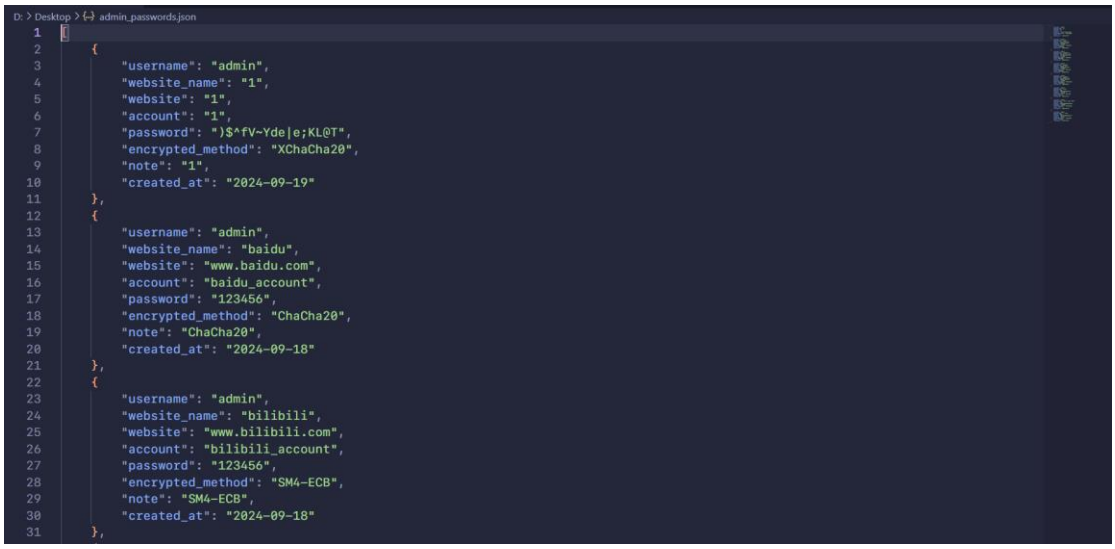

图 5-12 导出 JSON 格式的数据

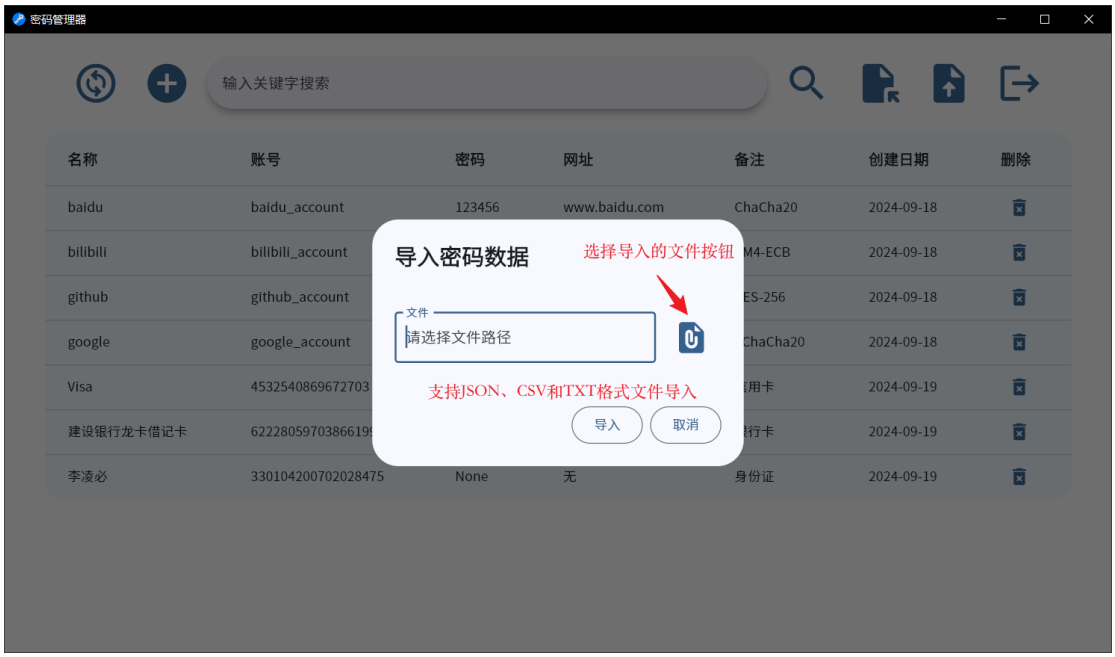

图 5-13 导入功能弹窗

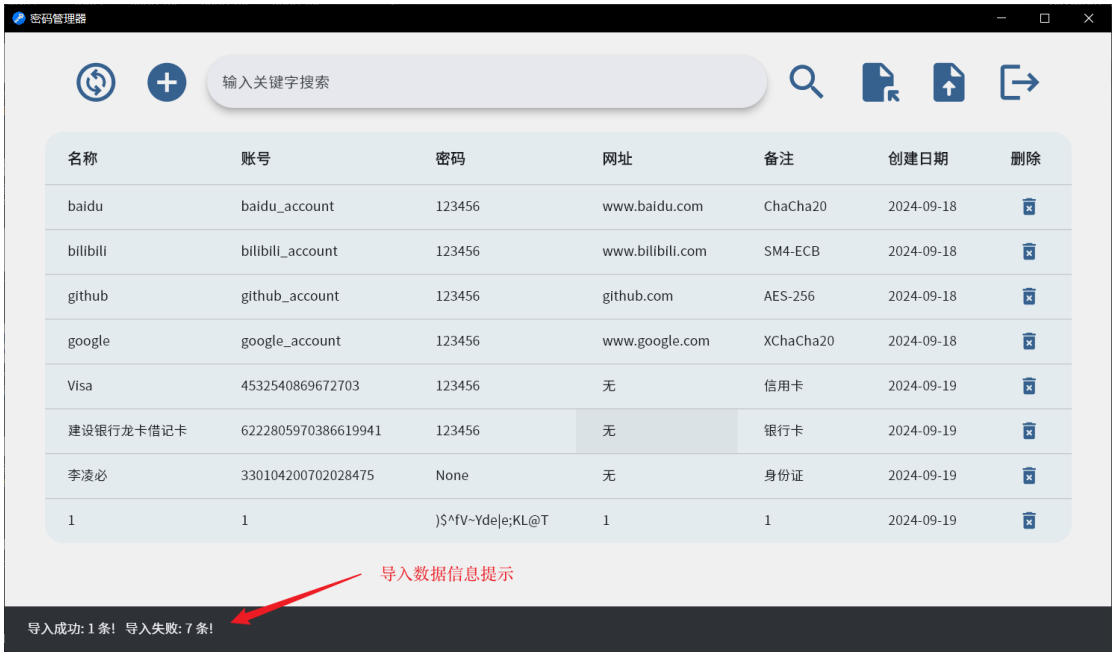

图 5-14 导入信息提示

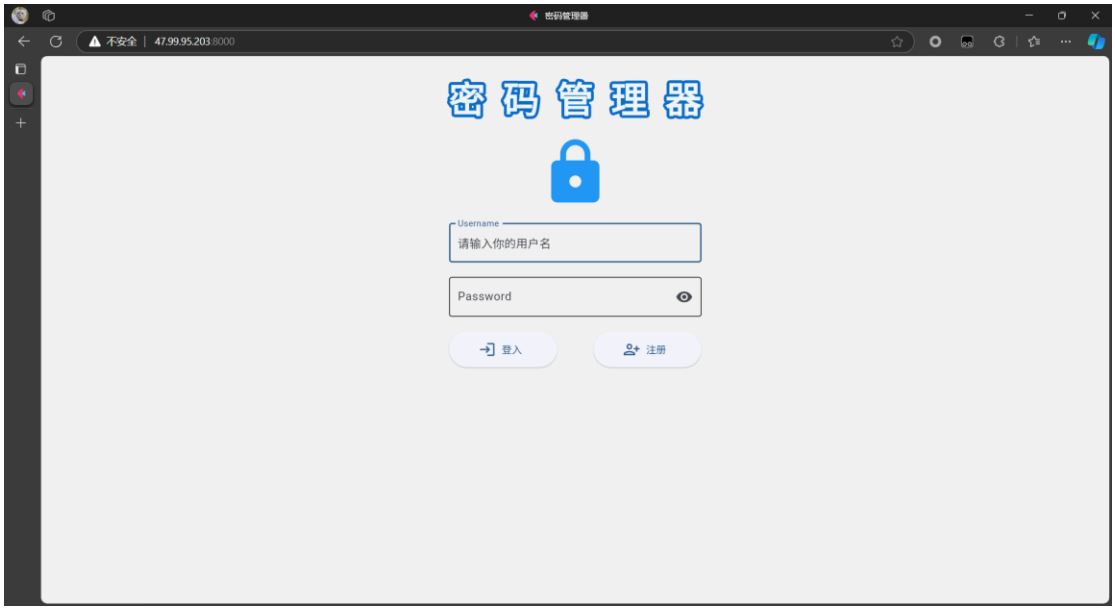

图 5-15 网页端登录界面

图 5-16 网页端注册界面

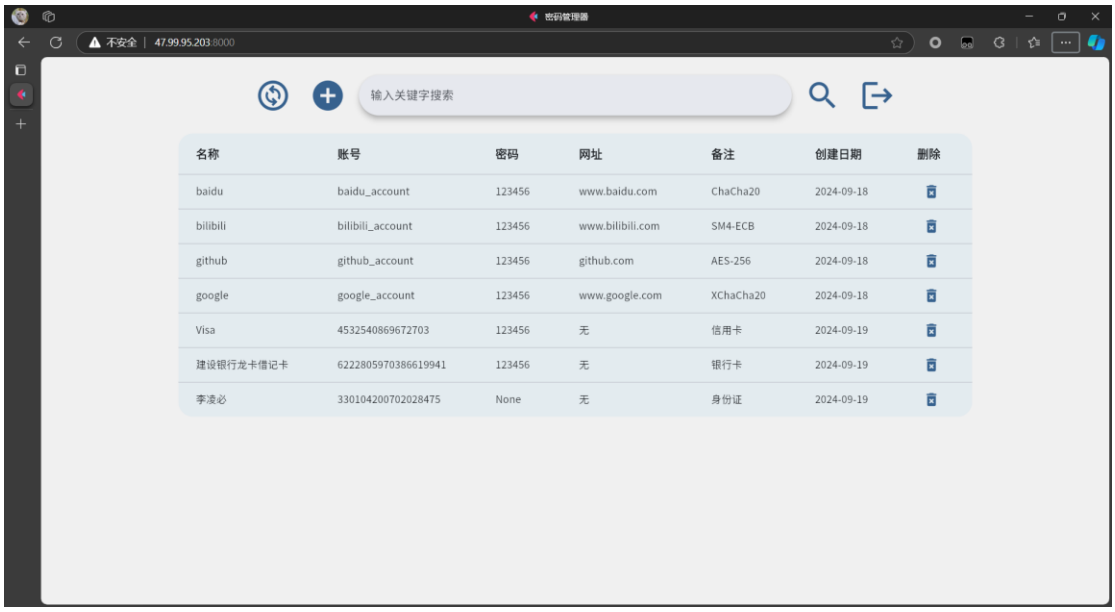

图 5-17 网页端主界面

网页端的所有功能均与 Windows 平台的功能完全一致。这里不再进行详细的展示和阐述。

5.3 项目测试

- **性能测试**

完整测试代码：

```python
import data_encrypt as de
import database_op as db
import time

# 各部分性能测试

# 数据加密性能测试
# SHA3-256
time_start = time.time()
hash_sha3 = de.sha3_256('123456')
time_end = time.time()
print(f'SHA3-256 time: {time_end - time_start}')

# PBKDF2
time_start = time.time()
hash_pbkdf2 = de.pbkdf2('123456')
time_end = time.time()
print(f'PBKDF2 time: {time_end - time_start}')

# AES-256
time_start = time.time()
aes_encrypted = de.data_encrypt('123456', de.pbkdf2('123456'), 'AES-256')
time_end = time.time()
print(f'AES-256 encrypto time: {time_end - time_start}')
time_start = time.time()
aes_decrypted = de.data_decrypt(aes_encrypted, de.pbkdf2('123456'), 'AES-256
')
time_end = time.time()
print(f'AES-256 decrypto time: {time_end - time_start}')

# ChaCha20
time_start = time.time()
chacha20_encrypted = de.data_encrypt('123456', de.pbkdf2('123456'), 'ChaCha2
0')
time_end = time.time()
print(f'ChaCha20 encrypto time: {time_end - time_start}')
time_start = time.time()
chacha20_decrypted = de.data_decrypt(chacha20_encrypted, de.pbkdf2('123456
'), 'ChaCha20')
time_end = time.time()
print(f'ChaCha20 decrypto time: {time_end - time_start}')

# XChaCha20
time_start = time.time()
```

```
xchacha20_encrypted = de.data_encrypt('123456', de.pbkdf2('123456'), 'XChaCh
a20')
time_end = time.time()
print(f'XChaCha20 encrypto time: {time_end - time_start}')
time_start = time.time()
xchacha20_decrypted = de.data_decrypt(xchacha20_encrypted, de.pbkdf2('123456
'), 'XChaCha20')
time_end = time.time()
print(f'XChaCha20 decrypto time: {time_end - time_start}')

# SM4-ECB
time_start = time.time()
sm4_encrypted = de.data_encrypt('123456', de.pbkdf2('123456'), 'SM4-ECB')
time_end = time.time()
print(f'SM4-ECB encrypto time: {time_end - time_start}')
time_start = time.time()
sm4_decrypted = de.data_decrypt(sm4_encrypted, de.pbkdf2('123456'), 'SM4-ECB
')
time_end = time.time()
print(f'SM4-ECB decrypto time: {time_end - time_start}')

# 随机生成强密码
time_start = time.time()
strong_password = de.random_password(16)
time_end = time.time()
print(f'Strong password generation time: {time_end - time_start}')

# 数据库操作性能测试
# 查询用户表
time_start = time.time()
db.query_user(db.conn, 'admin')
time_end = time.time()
print(f'Query user time: {time_end - time_start}')

# 插入用户表
time_start = time.time()
db.insert_user(db.conn, 'test1', 'test1')
time_end = time.time()
print(f'Insert user time: {time_end - time_start}')

# 查询密码表
time_start = time.time()
db.query_password(db.conn, 'admin')
time_end = time.time()
```

```
print(f'Query password time: {time_end - time_start}')

# 插入密码表
time_start = time.time()
db.insert_password(db.conn, 'test1', 'test1', 'test1', 'test1', 'test1', 'te
st1', 'test1')
time_end = time.time()
print(f'Insert password time: {time_end - time_start}')

# 查询密码创建时间
time_start = time.time()
db.query_password_created_at(db.conn, 'test1', 'test1', 'test1')
time_end = time.time()
print(f'Query password created time: {time_end - time_start}')

# 删除密码
time_start = time.time()
db.delete_password(db.conn, 'test1', 'test1', 'test1')
time_end = time.time()
print(f'Delete password time: {time_end - time_start}')

# 搜索关键字
time_start = time.time()
db.search_password(db.conn, 'admin', 'baidu')
time_end = time.time()
print(f'Search password time: {time_end - time_start}')
```

测试结果汇总：

```
# 数据加密性能测试结果
SHA3-256 time: 0.0  # 精度不够
PBKDF2 time: 0.005513906478881836
AES-256 encrypto time: 0.005999326705932617
AES-256 decrypto time: 0.004008054733276367
ChaCha20 encrypto time: 0.00400090217590332
ChaCha20 decrypto time: 0.0040073394775390625
XChaCha20 encrypto time: 0.003996133804321289
XChaCha20 decrypto time: 0.004012107849121094
SM4-ECB encrypto time: 0.004988908767700195
SM4-ECB decrypto time: 0.004006624221801758
Strong password generation time: 0.0  # 精度不够

# 数据库操作性能测试结果
Query user time: 0.13013291358947754
Insert user time: 0.17148613929748535
Query password time: 0.11891531944274902
```

```
Insert password time: 0.16904258728027344
Query password created time: 0.12987828254699707
Delete password time: 0.1889970302581787
Search password time: 0.14607882499694824
```

- 安全性测试

username varchar(25	password_hash varchar(255)	created_at timestamp
admin	fb001dfcffd1c899f3297871406242f097aecf1a5342cc	2024-09-18 22:43:39
test	36f028580bb02cc8272a9a020f4200e346e276ae664e	2024-09-19 00:55:51

图 5-18 加密后的用户表

username varchar(255)	website_name varchar(150)	website varchar(150)	account varchar(150)	password_encrypted varchar(255)	encrypted_method varchar(50)	note text	created_at timestamp
admin	baidu	www.baidu.com	baidu_account	9358b86f9f28785c62e8252552a39bdc27b4	ChaCha20	ChaCha20	2024-09-18 22:4
admin	bilibili	www.bilibili.com	bilibili_account	rL959Ro7xjFlpLacaKFzoA==	SM4-ECB	SM4-ECB	2024-09-18 22:4
admin	github	github.com	github_account	c405475fa0dc6c2219a6bba76735d472b5f6bbf	AES-256	AES-256	2024-09-18 22:4
admin	google	www.google.com	google_account	ae99169a796dac2744c480b1aa11541878363d2	XChaCha20	XChaCha20	2024-09-18 22:4
admin	Visa	无	4532540869672703	9031ef4776221da8ef2c491f9e8b17555a99284(	AES-256	信用卡	2024-09-19 09:2
admin	建设银行龙卡借记卡	无	6222805970386619	864255f5f643ab1ae19a5d1eb3bf8ed0c0e0462(	XChaCha20	银行卡	2024-09-19 09:2
admin	李凌必	无	3301042007020284	137c78d0f2965f1ea5506a7b93c2cbc6	ChaCha20	身份证	2024-09-19 09:3

图 5-19 加密后的密码表

6 项目总结

经过三周的努力，我们小组的密码管理器系统设计与实现项目已经圆满完成。我们的项目旨在开发一款安全、高效、用户友好的密码管理工具，以帮助用户更好地管理和保护他们的密码信息。在这个项目中，我们有许多的创新点和优势，比如：

- **多算法的支持：**我们实现了包括 AES-256、ChaCha20、XChaCha20 以及 SM4-ECB 在内的多种加密算法，为用户提供了灵活的加密选项，以满足不同安全需求。同时使用 SHA3-256 对用户主密钥进行哈希操作，更安全。
- **动态密钥生成：**我们通过 PBKDF2 算法和 SHA-256 哈希函数，我们确保了密钥的安全性和随机性，每次加密都使用了接近一次一次（One-time Pad）的策略，极大提升了安全性。
- **多平台兼容性：** 我们的系统支持 Windows 和 Web 平台，未来计划扩展到 iOS、Android 以及 Linux 等更多平台，实现了跨平台的数据同步。
- **用户界面友好：** 我们设计了直观易用的用户界面，包括明暗主题切换、密码信息的批量导入导出等功能，提升了用户体验。

在本次的项目中，数据加解密模块是整个密码管理器系统的核心部分，它直接影响到用户数据的安全性和可信度。本模块采用了 SHA-256、AES-256、ChaCha20 以及 XChaCha20 等多种加密算法，为用户提供多样化的加密策略，确保了密码信息在存储和传输过程中的保密性。同时，模块支持将数据导出为 JSON、CSV 和 TXT 格式，满足用户在不同应用场景下的便捷操作需求。在整个设计和实施过程中，我们严格遵循行业安全标准，致力于打造既安全又用户友好的接口，以简化操作流程并提升用户体验。

数据库模块也是密码管理器系统的重要组成部分，负责安全存储用户账户和加密后的密码数据。它通过参数化查询防止 SQL 注入，并在数据传输中采用加密措施，保障用户敏感信息的安全。由于使用云端数据库，使得我们的项目支持多端同步，确保用户在不同设备上访问一致的数据。我们还增加了高效的查询功能，如模糊搜索和字段查询，提升了操作的便捷性。此模块设计灵活，支持未来扩展，确保系统的高安全性和稳定性。

此外，我们也进行了整个项目的测试，其中包含性能测试和安全性测试，从测试结果来看，我们的项目响应迅速，系统在正常网络条件下，用户操作的响应时间均不超过 2 秒，保证了流畅的用户体验。同时我们的应用程序优化了资源使用，避免了系统资源的过度消耗。我们的项目也通过安全性测试，我们验证了加密算法的有效性，确保了数据在存储和传输过程中的安全性。我们对数据库进行了加固，包括使用强密码、开启权限控制等措施，防止了未授权访问，避免我们的核心数据库被黑客攻击。

最后，对于未来升级改造，我们计划引入更多的加密算法，以提供更全面的安全保护。并且我们将继续优化用户界面，如在 Android 平台上增加生物识别验证等新功能。我们最终的目标是将系统部署到更多的操作系统和设备上，实现更广泛的覆盖，让用户随时随地安全高效地管理自己的密码信息。