

信息与软件工程学院上机实验报告

(第 2 次)

一、实验名称

实验二：散列算法的应用

二、实验目的及要求

1. 实验目的

通过编程实现 MD5/SHA 算法应用于系统登录界面中用户口令的安全认证，加深对散列算法及其应用的了解。

2. 实验要求

设计一个模拟的系统登录的界面程序，要求采用 MD5/SHA 算法实现用户登录口令的安全认证。程序具有注册和登录两个功能，用户注册功能提示用户输入用户名和口令，存储用户名和用户口令的 MD5/SHA 散列值。用户登录功能，提示用户输入用户名和口令，用户输入正确口令后，显示其 MD5/SHA 散列值，并与存储的 MD5/SHA 散列值比对，得出用户合法或非法的提示。

三、实验环境

1. 操作系统：macOS / Linux
2. 编程语言：Python 3.13
3. Web 框架：Flask
4. 环境管理：uv
5. 开发工具：Visual Studio Code

四、实验设计

1. 实验步骤

程序采用单页式 Web 交互流程：浏览器打开首页后，左侧完成注册，右侧完成登录，页面下方用于展示已注册用户信息和保存的散列值。

1.1 设计思想

在此实验中为了控制实现复杂度，我没有引入数据库、复杂用户会话等额外功能，而是采用一个 HTML 页面配合一个本地 JSON 文件来完成实验要求。

具体设计思想如下：

1. 页面尽量简单，在一个页面中同时放置注册区和登录区，方便演示与截图。
2. 后端逻辑集中在 `main.py` 中，便于直接查看程序入口、注册逻辑和登录逻辑。
3. 注册时只保存用户名、散列算法和口令散列值，不保存口令明文，从而体现散列算法在认证中的基本作用。
4. 登录时不直接比较口令字符串，而是重新计算散列值后再比较，以符合口令认证的基本原理。

1.2 实现流程

程序的整体执行流程如图所示。

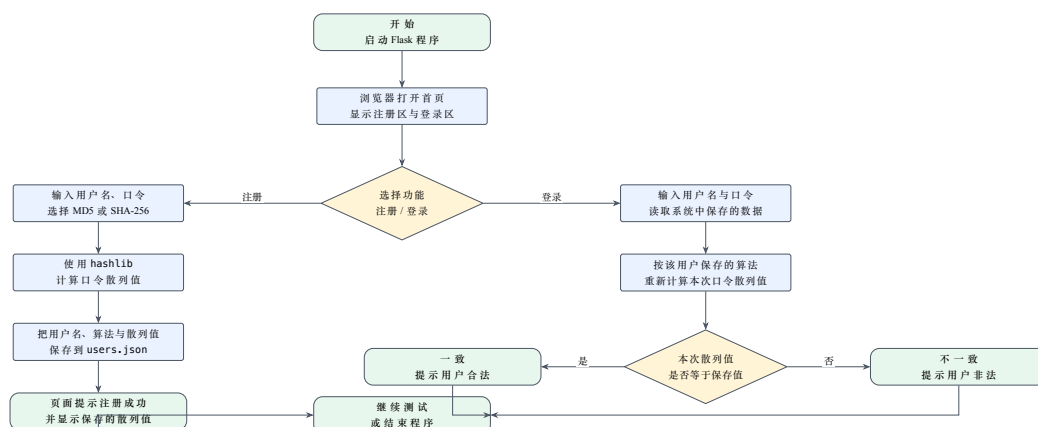


图 1: 散列算法登录认证程序实现流程图

1.3 核心算法说明

程序中最关键的三部分逻辑如下：

1. 数据读取与保存：程序使用 `users.json` 作为本地用户数据文件。注册

前先读取已有用户数据，注册成功后再把更新后的内容写回文件。

2. 散列值计算：程序通过 `hash_password()` 函数统一计算口令散列值。若算法为 md5，调用 `hashlib.md5()`；若算法为 sha256，调用 `hashlib.sha256()`；最后用 `hexdigest()` 得到十六进制字符串。
3. 登录校验：登录时先根据用户名查找对应用户，再按该用户保存的算法对本次输入口令重新计算散列值，并与系统中保存的散列值进行比较。若二者一致，则认证通过；否则认证失败。

其中，MD5 的散列结果通常为 32 位十六进制字符串，而 SHA-256 的散列结果通常为 64 位十六进制字符串。从实验运行结果中也可以直接观察到这一点。

2. 调试过程及实验结果

调试时，我主要关注三类问题：第一，注册时是否真的保存的是散列值而不是明文口令；第二，登录成功时本次散列值是否和保存值完全一致；第三，登录失败时程序是否能正确显示不一致结果并提示用户非法。

2.1 测试样例

为了验证程序是否正确实现了题目要求，本实验从 MD5 和 SHA-256 两种算法出发，分别对注册、登录成功和登录失败三种情况进行测试。

本次实验中使用的测试用户如下：

1. 用户 **test**：用于演示 MD5 注册、MD5 登录成功和 MD5 登录失败。
2. 用户 ：用于演示 SHA-256 注册、SHA-256 登录成功和 SHA-256 登录失败。

这样的测试方式可以同时覆盖：

1. 两种散列算法是否都能正常工作；
2. 注册功能是否能正确保存散列值；
3. 登录功能在口令正确和口令错误两种情况下是否都能给出正确提示。

2.2 运行结果

实验运行结果如表所示。

表 1: 实验二测试样例与运行结果汇总

编号	测试项目	输入说明	运行结果
1	MD5 注册测试	用户名 <code>test</code> ，选择 MD5 完成注册	程序提示注册成功，并保存 MD5 散列值 <code>e10adc3949ba59ab...</code> 。
2	MD5 登录成功测试	用户名 <code>test</code> ，输入与注册时相同的口令	本次输入口令的 MD5 值与系统保存值一致，程序提示用户合法。
3	MD5 登录失败测试	用户名 <code>test</code> ，故意输入错误口令	本次输入口令的 MD5 值与系统保存值不一致，程序提示用户非法。
4	SHA-256 注册测试	用户名 ，选择 SHA-256 完成注册	程序提示注册成功，并保存 SHA-256 散列值 <code>384fde3636e6e01e...</code> 。
5	SHA-256 登录成功测试	用户名 ，输入与注册时相同的口令	本次输入口令的 SHA-256 值与系统保存值一致，程序提示用户合法。
6	SHA-256 登录失败测试	用户名 ，故意输入错误口令	本次输入口令的 SHA-256 值与系统保存值不一致，程序提示用户非法。

2.3 运行截图

为直观展示程序运行效果，下面给出 MD5 和 SHA-256 两种算法下的注册、登录成功与登录失败截图。

用户注册

用户名

test

口令

散列算法

MD5

注册

注册成功，系统已经保存口令散列值。

用户名: test

散列算法: MD5

保存的散列值:

e10adc3949ba59abbe56e057f20f883e

用户登录

用户名

口令

登录

已注册用户信息

用户名	散列算法	保存的散列值
	SHA-256	384fde3636e6e01e0194d2976d8f26410af3e846e573379cb1a09e2f0752d8cc
test	MD5	e10adc3949ba59abbe56e057f20f883e

图 2: MD5 注册运行结果

用户注册

用户名

口令

散列算法

SHA-256

注册

用户登录

用户名

test

口令

登录

口令正确，用户合法。

用户名: test

散列算法: MD5

本次输入口令的散列值:

e10adc3949ba59abbe56e057f20f883e

系统中保存的散列值:

e10adc3949ba59abbe56e057f20f883e

已注册用户信息

用户名	散列算法	保存的散列值
	SHA-256	384fde3636e6e01e0194d2976d8f26410af3e846e573379cb1a09e2f0752d8cc
test	MD5	e10adc3949ba59abbe56e057f20f883e

图 3: MD5 登录成功运行结果

用户注册

用户名

口令

散列算法

SHA-256

注册

用户登录

用户名

test

No items to show

+ New login

登录

口令错误, 用户非法。

用户名: test

散列算法: MD5

本次输入口令的散列值:

c4d038b4bed09fdb1471ef51ec3a32cd

系统中保存的散列值:

e10adc3949ba59abbe56e057f20f883e

已注册用户信息

用户名	散列算法	保存的散列值
	SHA-256	384fde3636e6e01e0194d2976d8f26410af3e846e573379cb1a09e2f0752d8cc
test	MD5	e10adc3949ba59abbe56e057f20f883e

图 4: MD5 登录失败运行结果

用户注册

用户名

shuzhanhua

+ New login

散列算法

SHA-256

注册

注册成功, 系统已经保存口令散列值。

用户名

散列算法: SHA-256

保存的散列值:

384fde3636e6e01e0194d2976d8f26410af3e846e573379cb1a09e2f0752d8cc

用户登录

用户名

口令

登录

已注册用户信息

用户名	散列算法	保存的散列值
	SHA-256	384fde3636e6e01e0194d2976d8f26410af3e846e573379cb1a09e2f0752d8cc

图 5: SHA-256 注册运行结果



图 6: SHA-256 登录成功运行结果

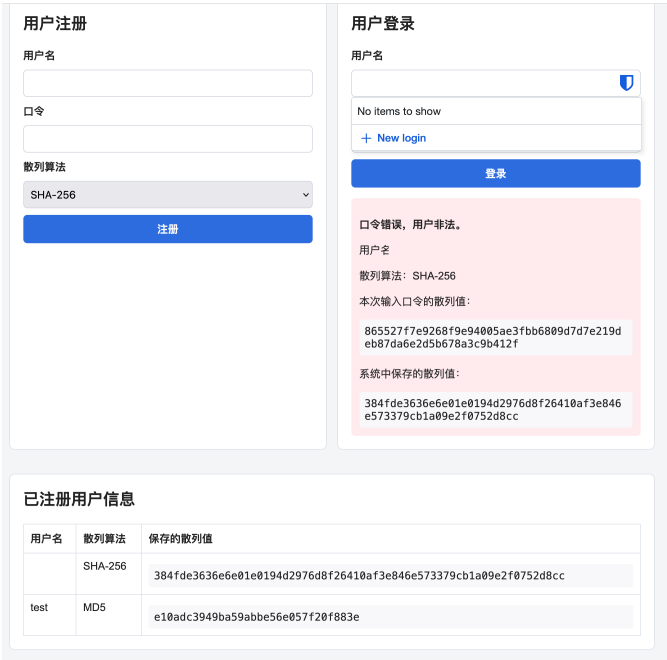


图 7: SHA-256 登录失败运行结果

2.4 结果分析

从运行结果可以看出，程序已经完整覆盖实验要求。首先，在注册阶段，系统保存的是口令散列值而不是口令明文本身，这体现了散列算法在认证中的基本应用。其次，在登录成功时，本次输入口令重新计算得到的散列值与系统保存值完全一致，程序能够正确判定用户合法。再次，在登录失败时，本次输入

口令计算出的散列值与保存值不同，程序能够明确提示用户非法。最后，通过 MD5 与 SHA-256 两种算法的对比，也可以直观看到二者输出长度不同，但整体认证流程是一致的。

3. 总结

3.1 对上机实践结果进行分析

本次实验完成了一个模拟系统登录界面程序，实现了注册与登录两个功能，并把 MD5 / SHA-256 散列算法应用到了用户口令认证中。从运行结果看，程序能够在注册时保存口令散列值，在登录时重新计算散列值并完成比对，同时正确给出用户合法或非法的提示，说明程序实现达到了实验要求。

3.2 上机的心得体会

通过这次上机，我对“口令为什么不能明文保存”这件事有了更具体的理解。以前只是知道真实系统里会把口令做散列处理，但真正自己写了注册和登录流程之后，才发现这个过程其实非常清晰：注册时先把口令变成散列值再保存，登录时再算一次散列值和保存结果比较。这样做既能完成身份验证，也避免了直接保存明文口令。

3.3 改进意见

可以增加更完善的异常输入提示；也可以在真实应用场景中进一步加入盐值 (salt)，或者改用更适合口令保护的算法，以提高安全性。

4. 附录

为了控制篇幅，附录中只保留与实验原理直接相关的核心实现代码。

4.1 散列认证核心程序 main.py

```
1 def load_users() -> dict[str, dict[str, str]]:
2     """读取已注册用户。"""
3     if not DATA_FILE.exists():
4         return {}
5
6     content = DATA_FILE.read_text(encoding="utf-8").strip()
7     if not content:
8         return {}
9
```

```
10     return json.loads(content)
11
12
13 def save_users(users: dict[str, dict[str, str]]) -> None:
14     """把用户数据保存到本地 JSON 文件。"""
15     DATA_FILE.parent.mkdir(parents=True, exist_ok=True)
16     DATA_FILE.write_text(
17         json.dumps(users, ensure_ascii=False, indent=2),
18         encoding="utf-8",
19     )
20
21
22 def hash_password(password: str, algorithm: str) -> str:
23     """
24     计算口令散列值。
25
26     这里直接使用 Python 标准库 hashlib。
27     因为本实验重点是“散列算法怎么用于登录认证”，
28     不是底层手写 MD5 或 SHA 的实现细节。
29     """
30     password_bytes = password.encode("utf-8")
31
32     if algorithm == "md5":
33         return hashlib.md5(password_bytes).hexdigest()
34
35     return hashlib.sha256(password_bytes).hexdigest()
36
37
38 def render_page(
39     register_result: dict[str, object] | None = None,
40     login_result: dict[str, object] | None = None,
41     form_data: dict[str, str] | None = None,
42 ) -> str:
43     """统一渲染页面，避免重复写 render_template。"""
44     defaults = {
45         "register_username": "",
46         "register_algorithm": "sha256",
47         "login_username": "",
48     }
49     if form_data:
50         defaults.update(form_data)
51
52     return render_template(
53         "index.html",
54         algorithms=SUPPORTED_ALGORITHMS,
```

```
55     users=load_users(),
56     register_result=register_result,
57     login_result=login_result,
58     form_data=defaults,
59 )
60
61
62 @app.get("/")
63 def index() -> str:
64     """显示首页。"""
65     return render_page()
66
67
68 @app.post("/register")
69 def register() -> str:
70     """处理注册请求。"""
71     username = request.form.get("username", "").strip()
72     password = request.form.get("password", "")
73     algorithm = request.form.get("algorithm", "sha256").strip().
lower()
74
75     if not username:
76         return render_page(
77             register_result={"success": False, "message": "用户名
不能为空。"},
78             form_data={
79                 "register_username": username,
80                 "register_algorithm": algorithm,
81             },
82         )
83
84     if not password.strip():
85         return render_page(
86             register_result={"success": False, "message": "口令不
能为空。"},
87             form_data={
88                 "register_username": username,
89                 "register_algorithm": algorithm,
90             },
91         )
92
93     if algorithm not in SUPPORTED_ALGORITHMS:
94         return render_page(
95             register_result={"success": False, "message": "请选择
正确的散列算法。"},
```

```
96         form_data={
97             "register_username": username,
98             "register_algorithm": "sha256",
99         },
100     )
101
102     users = load_users()
103     if username in users:
104         return render_page(
105             register_result={"success": False, "message": "该用户
106 名已经存在, 请重新注册。"},
107             form_data={
108                 "register_username": username,
109                 "register_algorithm": algorithm,
110             },
111         )
112
113     password_hash = hash_password(password, algorithm)
114     users[username] = {
115         "algorithm": algorithm,
116         "password_hash": password_hash,
117     }
118     save_users(users)
119
120     return render_page(
121         register_result={
122             "success": True,
123             "message": "注册成功, 系统已经保存口令散列值。",
124             "username": username,
125             "algorithm_name": SUPPORTED_ALGORITHMS[algorithm],
126             "password_hash": password_hash,
127         },
128         form_data={
129             "register_username": username,
130             "register_algorithm": algorithm,
131         },
132     )
133
134 @app.post("/login")
135 def login() -> str:
136     """处理登录请求。"""
137     username = request.form.get("username", "").strip()
138     password = request.form.get("password", "")
139
```

```
140     if not username:
141         return render_page(
142             login_result={"success": False, "message": "用户名不能
143             为空。"},
144             form_data={"login_username": username},
145         )
146
147     if not password.strip():
148         return render_page(
149             login_result={"success": False, "message": "口令不能为
150             空。"},
151             form_data={"login_username": username},
152         )
153
154     users = load_users()
155     if username not in users:
156         return render_page(
157             login_result={"success": False, "message": "该用户不存
158             在，请先注册。"},
159             form_data={"login_username": username},
160         )
161
162     stored_user = users[username]
163     algorithm = stored_user["algorithm"]
164     computed_hash = hash_password(password, algorithm)
165     stored_hash = stored_user["password_hash"]
166     is_valid = computed_hash == stored_hash
167
168     return render_page(
169         login_result={
170             "success": is_valid,
171             "message": "口令正确，用户合法。" if is_valid else "口令错
172             误，用户非法。",
173             "username": username,
174             "algorithm_name": SUPPORTED_ALGORITHMS[algorithm],
175             "computed_hash": computed_hash,
176             "stored_hash": stored_hash,
177         },
178         form_data={"login_username": username},
179     )
```