تمرین عملی سوم

صبا سحبان ۹۹۳۱۰۹۶

بخش ۱)

تست خروجی:

اضافه کردن دو پسوورد

```
(venv) sabasahban@sabas-MacBook-Pro Security3 % python passmanager.py --newpass bankmelli -c "bank password" --key 12simple
Password for 'bankmelli' added.
```

```
(venv) sabasahban@sabas-MacBook-Pro Security3 % python passmanager.py --newpass portal -c uniportal --key 12simple
Password for 'portal' added.
```

نشان دادن پسوورد ها:(طول پسوورد ها متفاوت است، در فاز بعدی مشخص تر میشود)

```
Name: portal, Password: OjfY0WkVJtmZ0u1n4, Comment: uniportal
Name: bankmelli, Password: f8N5fNqeUMlU/wfR, Comment: bank password
```

سلکت کردن پسوورد:

```
(venv) sabasahban@sabas-MacBook-Pro Security3 % python passmanager.py --sel portal --key 12simple

Password: OjfY0WkVJtmZ0u1n4, Comment: uniportal
```

آپدیت کردن پسوورد:

```
(venv) sabasahban@sabas-MacBook-Pro Security3 % python passmanager.py --update portal --key 12simple
Password for 'portal' updated.
(venv) sabasahban@sabas-MacBook-Pro Security3 % python passmanager.py --showpass --key 12simple
Name: portal, Password: w60DjBKFOGHDBdLT77Vp2, Comment: uniportal
Name: bankmelli, Password: f8N5fNqeUMlU/wfR, Comment: bank password
```

دیلیت کردن پسوورد:

```
(venv) sabasahban@sabas-MacBook-Pro Security3 % python passmanager.py --delete portal --key 12simple
Password for 'portal' deleted.
(venv) sabasahban@sabas-MacBook-Pro Security3 % python passmanager.py --showpass --key 12simple
Name: bankmelli, Password: f8N5fNqeUMlU/wfR, Comment: bank password
```

توضیح کد:

1. derive_key (استخراج کلید):

```python
def derive_key(password: str, salt: bytes) -> bytes:
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=salt,
        iterations=10000,
        backend=default_backend()
    )
    return kdf.derive(password.encode())
```

- این تابع برای استخراج یک کلید رمزنگاری از رمز عبور وارد شده و یک نمک (salt) استفاده می‌شود.
- `PBKDF2HMAC` یک تابع کلید مشتق‌ساز است که با استفاده از الگوریتم هش SHA256، تعداد تکرار (iterations) مشخص، و نمک، یک کلید 32 بایتی ایجاد می‌کند.
- این کلید برای رمزنگاری و رمزگشایی داده‌ها مورد استفاده قرار می‌گیرد.

2. encrypt_data (رمزنگاری داده‌ها):

```python
def encrypt_data(key: bytes, data: str) -> bytes:
    padder = padding.PKCS7(128).padder()
    padded_data = padder.update(data.encode()) + padder.finalize()

    iv = os.urandom(16)
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
    encryptor = cipher.encryptor()
    encrypted_data = encryptor.update(padded_data) + encryptor.finalize()

    return base64.b64encode(iv + encrypted_data)
```

- برای رمزنگاری داده‌ها با استفاده از AES در حالت CBC (Block Cipher Chaining) استفاده می‌شود.
- ابتدا داده‌ها با استفاده از PKCS7 padding به یک طول مناسب برای بلوک‌های AES رسانده می‌شوند.
- یک IV (Initialization Vector) تصادفی 16 بایتی ایجاد و برای افزایش امنیت رمزنگاری استفاده می‌شود.
- داده‌های رمزنگاری شده به همراه IV با استفاده از Base64 کدگذاری می‌شوند تا در فایل ذخیره شوند.

3. decrypt_data (رمزگشایی داده‌ها):

```python
def decrypt_data(key: bytes, encrypted_data: bytes) -> bytes:
    encrypted_data = base64.b64decode(encrypted_data)
    iv = encrypted_data[:16]
    encrypted_data = encrypted_data[16:]

    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
    decryptor = cipher.decryptor()
    decrypted_data = decryptor.update(encrypted_data) + decryptor.finalize()

    unpadder = padding.PKCS7(128).unpadder()
    return unpadder.update(decrypted_data) + unpadder.finalize()
```

- ابتدا داده‌های رمزنگاری شده که با Base64 کدگذاری شده‌اند، رمزگشایی می‌شوند.
- IV که در 16 بایت اول قرار دارد جدا شده و برای رمزگشایی استفاده می‌شود.
- داده‌ها با استفاده از کلید و IV رمزگشایی شده و سپس از padding خارج می‌شوند تا داده‌های اصلی بازیابی شوند.

4. generate_complex_password (تولید رمز عبور پیچیده):

```python
def generate_complex_password(simple_password: str, name: str) -> str:
    salt = os.urandom(16)
    key = derive_key(simple_password, salt)
    return encrypt_password(key, name).decode()
```

- این تابع یک رمز عبور پیچیده را با استفاده از یک رمز عبور ساده و نامی که به عنوان داده وارد می‌شود، تولید می‌کند.
- ابتدا با استفاده از `derive_key` یک کلید بر اساس رمز عبور ساده و یک نمک تصادفی استخراج می‌شود.
- سپس این کلید برای رمزنگاری نام با استفاده از تابع `encrypt_password` به کار گرفته می‌شود.

5. encrypt_password (رمزنگاری رمز عبور):

```python
def encrypt_password(key: bytes, data: str) -> bytes:
    padder = padding.PKCS7(128).padder()
    padded_data = padder.update(data.encode()) + padder.finalize()

    iv = os.urandom(16)
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
    encryptor = cipher.encryptor()
    encrypted_data = encryptor.update(padded_data) + encryptor.finalize()

    # Generate a random length that is at least as long as the IV
    random_length = random.randint(len(iv), len(iv) + len(encrypted_data))
    return base64.b64encode(iv + encrypted_data)[:random_length]
```

- این تابع مشابه `encrypt_data` عمل می‌کند، با این تفاوت که طول خروجی رمزنگاری شده به صورت تصادفی محدود می‌شود.
- این ویژگی امنیت اضافی به رمز عبور اضافه می‌کند زیرا طول خروجی پیش‌بینی‌ناپذیر است.

6. load_passwords (بارگذاری رمزهای عبور):

```python
def load_passwords(simple_password: str) -> dict:
    if not os.path.exists(PASSWORD_FILE):
        return {}

    with open(PASSWORD_FILE, 'rb') as file:
        encrypted_data = file.read()
        salt = encrypted_data[:16]
        key = derive_key(simple_password, salt)
        try:
            data = decrypt_data(key, encrypted_data[16:])
            return json.loads(data)
        except Exception:
            raise ValueError("Incorrect password or corrupted file.")
```

- این تابع فایل رمزنگاری شده حاوی رمزهای عبور را بارگذاری می‌کند.
- اگر فایل وجود داشته باشد، داده‌ها خوانده شده و با استفاده از کلید مشتق شده از رمز عبور وارد شده و نمک (که در ابتدای فایل قرار دارد) رمزگشایی می‌شوند.
- در صورت خطا در رمزگشایی، یک خطای "رمز عبور نادرست یا فایل فاسد" نمایش داده می‌شود.

7. save_passwords (ذخیره رمزهای عبور):

```python
def save_passwords(simple_password: str, passwords: dict):
    salt = os.urandom(16)
    key = derive_key(simple_password, salt)
    data = json.dumps(passwords)
    encrypted_data = encrypt_data(key, data)

    with open(PASSWORD_FILE, 'wb') as file:
        file.write(salt + encrypted_data)
```

- این تابع دیکشنری رمزهای عبور را به فرمت JSON تبدیل کرده و سپس رمزنگاری می‌کند.
- داده‌های رمزنگاری شده به همراه یک نمک تصادفی جدید در ابتدای فایل `passwords.enc` ذخیره می‌شوند.

توابع مرتبط با دستورات خط فرمان

- add_new_password, show_passwords, select_password, update_password, delete_password:

- این توابع برای اجرای عملیات‌های مختلف مدیریت رمز عبور مانند اضافه کردن، نمایش، انتخاب، به‌روزرسانی و حذف یک رمز عبور خاص استفاده می‌شوند.
- هر کدام از این توابع ابتدا رمزهای عبور را با استفاده از `load_passwords` بارگذاری کرده و سپس عملیات مورد نظر را انجام می‌دهند.
- در نهایت، تغییرات در فایل ذخیره می‌شوند (به جز در عملیات نمایش).

1. add_new_password (اضافه کردن رمز عبور جدید):

```python
def add_new_password(args):
    passwords = load_passwords(args.key)
    complex_password = generate_complex_password(args.key, args.newpass)
    passwords[args.newpass] = {'password': complex_password, 'comment':
args.c}
    save_passwords(args.key, passwords)
    print(f"Password for '{args.newpass}' added.")
```

- این تابع برای افزودن یک رمز عبور جدید به مجموعه رمزهای عبور استفاده می‌شود.
- ابتدا، با استفاده از تابع `load_passwords`، تمام رمزهای عبور موجود بارگذاری می‌شوند.
- سپس، یک رمز عبور پیچیده جدید با استفاده از تابع `generate_complex_password` تولید می‌شود.
- این رمز عبور جدید به همراه نام و توضیحات (اگر وجود داشته باشد) به دیکشنری اضافه شده و با استفاده از `save_passwords` در فایل ذخیره می‌شود.

2. show_passwords (نمایش رمزهای عبور):

```python
def show_passwords(args):
    passwords = load_passwords(args.key)
    for name, details in passwords.items():
```

```
        print(f"Name: {name}, Password: {details['password']}, Comment:
{details['comment']}")
```

- این تابع لیست تمام رمزهای عبور ذخیره‌شده را نمایش می‌دهد.
- با استفاده از `load_passwords`، تمام رمزهای عبور بارگذاری شده و سپس به صورت فرمت‌بندی شده چاپ می‌شوند.
- این تابع تغییری در فایل ایجاد نمی‌کند و صرفاً برای نمایش اطلاعات است.

3. select_password (انتخاب رمز عبور):

```
def select_password(args):
    passwords = load_passwords(args.key)
    if args.sel in passwords:
        details = passwords[args.sel]
        print(f"Password: {details['password']}, Comment:
{details['comment']}")
    else:
        print("Password not found.")
```

- این تابع برای بازیابی جزئیات یک رمز عبور خاص از میان رمزهای عبور ذخیره‌شده استفاده می‌شود.
- پس از بارگذاری رمزهای عبور، اگر نام رمز عبور مورد نظر در مجموعه وجود داشته باشد، جزئیات آن چاپ می‌شوند.

4. update_password (به‌روزرسانی رمز عبور):

```
def update_password(args):
    passwords = load_passwords(args.key)
    if args.update in passwords:
        complex_password = generate_complex_password(args.key, args.update)
        passwords[args.update]['password'] = complex_password
        save_passwords(args.key, passwords)
        print(f"Password for '{args.update}' updated.")
    else:
        print("Password not found.")
```

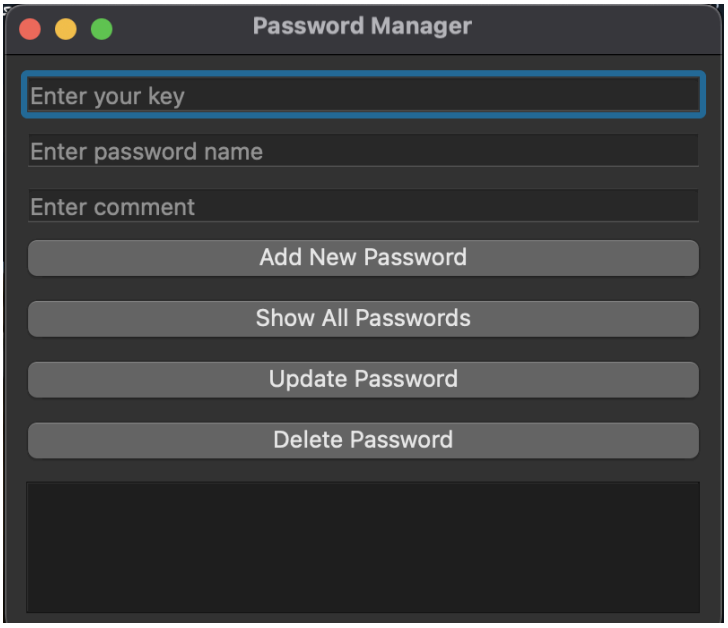
- این تابع برای به‌روزرسانی یک رمز عبور موجود استفاده می‌شود.
- پس از بارگذاری رمزهای عبور، اگر رمز عبور مورد نظر وجود داشته باشد، یک رمز عبور جدید و پیچیده با استفاده از `generate_complex_password` تولید می‌شود.
- رمز عبور قدیمی با رمز عبور جدید جایگزین شده و تغییرات در فایل ذخیره می‌شوند.

5. delete_password (حذف رمز عبور):

```
def delete_password(args):
    passwords = load_passwords(args.key)
    if args.delete in passwords:
        del passwords[args.delete]
        save_passwords(args.key, passwords)
        print(f"Password for '{args.delete}' deleted.")
    else:
        print("Password not found.")
```

- این تابع برای حذف یک رمز عبور از مجموعه استفاده می‌شود.
- پس از بارگذاری رمزهای عبور، اگر رمز عبور مورد نظر وجود داشته باشد، از دیکشنری حذف شده و تغییرات در فایل ذخیره می‌شوند.

همین فرایند با GUI نیز قابل انجام است که خروجی ای مشابه دارد:



بخش دوم:

توابع اصلی

1. generate_random_names (تولید رمز عبور تصادفی):

```
def generate_random_names(length=8):
    characters = string.ascii_letters + string.digits + string.punctuation
    return ''.join(random.choice(characters) for _ in range(length))
```

این تابع اسم های رندوم برای استفاده از تابع generate_strong_password بخش قبل استفاده میشود.

2. generate_and_save_passwords (تولید و ذخیره رمزهای عبور):

```
def generate_and_save_passwords():
    simple_password = "0000"
    with open("test.txt", "w") as file:
        for _ in range(10000):
            random_password = generate_random_names()
            complex_password = generate_complex_password(simple_password, random_password)
            file.write(complex_password + "\n")
```

- این تابع برای تولید 10,000 رمز عبور پیچیده و ذخیره آن‌ها در فایل `test.txt` استفاده می‌شود.
- برای هر رمز عبور، ابتدا یک نام تصادفی با استفاده از `generate_random_name` تولید می‌شود.
- سپس، این رمز عبور تصادفی به عنوان ورودی به `generate_complex_name` داده می‌شود تا یک رمز عبور پیچیده‌تر تولید شود. این تابع از رمز عبور ساده "0000" به عنوان کلید استفاده می‌کند.
- رمزهای عبور پیچیده تولید شده در فایل `test.txt` ذخیره می‌شوند، هر کدام در یک خط جداگانه.

فایل test.txt را به statsgen برای تحلیل میدهیم:

تحلیل:

```
[*] Analyzing passwords in [test.txt]
[+] Analyzing 100% (10000/10000) of passwords
    NOTE: Statistics below is relative to the number of analyzed passwords, not total number of passwords

[*] Length:
[+]                         25: 06% (628)
[+]                         19: 06% (625)
[+]                         17: 06% (619)
[+]                         23: 06% (619)
[+]                         26: 06% (615)
[+]                         21: 05% (599)
[+]                         20: 05% (591)
[+]                         24: 05% (590)
[+]                         30: 05% (584)
[+]                         31: 05% (580)
[+]                         27: 05% (570)
[+]                         29: 05% (568)
[+]                         18: 05% (567)
[+]                         16: 05% (565)
[+]                         28: 05% (564)
[+]                         32: 05% (560)
[+]                         22: 05% (556)

[*] Character-set:
[+]                        all: 51% (5172)
[+]             mixedalphanum: 46% (4600)
[+]         mixedalphaspecial: 01% (123)
[+]                mixedalpha: 01% (104)
[+]             loweralphanum: 00% (1)

[*] Password complexity:
[+]                      digit: min(0) max(13)
[+]                      lower: min(1) max(24)
[+]                      upper: min(0) max(21)
[+]                    special: min(0) max(6)

[*] Simple Masks:
[+]                  othermask: 92% (9276)
[+]         stringdigitstring: 04% (462)
[+]                    string: 01% (104)
[+]       stringspecialstring: 00% (81)
[+]               stringdigit: 00% (32)
[+]               digitstring: 00% (24)
[+]        specialstringdigit: 00% (5)
[+]             stringspecial: 00% (4)
[+]          digitstringdigit: 00% (4)
[+]        specialdigitstring: 00% (3)
[+]             specialstring: 00% (3)
[+]        digitspecialstring: 00% (2)
```

تحلیل استفاده از StatsGen برای تجزیه و تحلیل رمزهای عبور

StatsGen یک ابزار تحلیلگر رمز عبور است که برای تجزیه و تحلیل ویژگی‌های مختلف رمزهای عبور طراحی شده است. این ابزار اطلاعات مفیدی در مورد ساختار و پیچیدگی رمزهای عبور فراهم می‌کند. در اینجا، StatsGen برای تجزیه و تحلیل 10,000 رمز عبور از فایل `test.txt` استفاده شده است.

تحلیل‌های کلیدی انجام شده توسط StatsGen:

1. طول رمز عبور:
- این بخش طول رمزهای عبور را نشان می‌دهد. برای مثال، 6% رمزهای عبور دارای طول 25 کاراکتر هستند. این اطلاعات نشان می‌دهد که رمزهای عبور تولید شده چه تنوع طولی دارند.

2. مجموعه کاراکتری:
- این بخش نوع کاراکترهای به کار رفته در رمزهای عبور را مشخص می‌کند. برای مثال، 51% رمزهای عبور شامل همه انواع کاراکترها (حروف بزرگ و کوچک، اعداد و نشانه‌ها) هستند.

3. پیچیدگی رمز عبور:
- این بخش حداقل و حداکثر تعداد انواع کاراکترهای مختلف (حروف کوچک، حروف بزرگ، اعداد و نشانه‌ها) به کار رفته در رمزهای عبور را نشان می‌دهد. این اطلاعات می‌تواند در تعیین میزان پیچیدگی رمزهای عبور مفید باشد.

4. ماسک‌های ساده:
- این بخش الگوهای مختلف به کار رفته در ساختار رمزهای عبور را نشان می‌دهد. برای مثال، 92% رمزهای عبور از الگوهای دیگر به جز الگوهای شناخته شده استفاده می‌کنند.

5. ماسک‌های پیشرفته:
- این بخش الگوهای پیچیده‌تر و دقیق‌تر رمزهای عبور را نمایش می‌دهد