صبا سحبان ۹۹۳۱۰۹۶ – گزارش تمرین عملی ۱ امنیت

فاز ۱) قسمت اول
خروجی:

```
Enter your password: password
Weak: Password is in the common passwords dataset.
Try again
Enter your password: p
Weak: Password length should be at least 8 characters.
Try again
Enter your password: aosdjfpaosdjfpaisjdpaiosjdfpaoji
Weak: Password length exceeds the maximum limit of 15 characters.
Try again
Enter your password: sabasabasaba
Weak: Password is easily guessable and part of the blacklist.
Try again
Enter your password: sabauuuu
Weak: Password should include at least one digit, one uppercase letter, and one special character.
Try again
Enter your password: aaaa2*aa
Weak: Password should include at least one digit, one uppercase letter, and one special character.
Try again
Enter your password: aAa2*a3;
Password is strong:)
```

تابع check_password_strength:

(شروط بررسی قوت پسوورد از سند استاندارد NIST استخراج شده است)

- این تابع با گرفتن یک رمز عبور و یک DataFrame از رمزهای عبور متداول، قوت رمز عبور را ارزیابی می‌کند.
- ابتدا بررسی می‌شود که آیا رمز عبور در لیست رمزهای عبور متداول قرار دارد یا خیر. اگر در لیست باشد، رمز عبور به عنوان ضعیف اعلام می‌شود.
- سپس طول حداقل و حداکثر رمز عبور چک می‌شود. اگر طول رمز عبور کمتر از 8 کاراکتر یا بیشتر از 15 کاراکتر باشد، رمز عبور به عنوان ضعیف اعلام می‌شود.
- یک لیست سیاه (blacklist) از رمزهای عبور معین نیز وجود دارد. اگر رمز عبور در این لیست قرار داشته باشد، به عنوان ضعیف شناخته می‌شود.
- پیچیدگی رمز عبور با بررسی وجود حداقل یک رقم، یک حرف بزرگ و یک کاراکتر خاص ارزیابی می‌شود.
- همچنین، تصادفی بودن رمز عبور با بررسی تعداد حروف یکتا نسبت به تعداد کل حروف چک می‌شود.
- نتیجه ارزیابی به صورت متنی ("ضعیف" یا "قوی") برگردانده می‌شود.

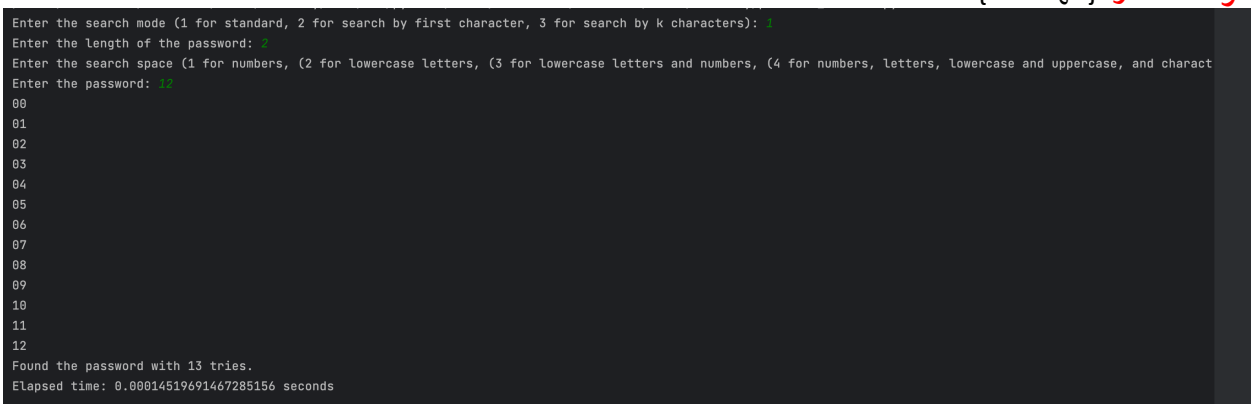
```python
def check_password_strength(password, common_passwords_df):
    # Check common password
    if password in common_passwords_df['password'].values:
        return "Weak: Password is in the common passwords dataset."

    # Check minimum length
    if len(password) < 8:
        return "Weak: Password length should be at least 8 characters."

    # Check maximum length
```

```
    max_length = 15
    if len(password) > max_length:
        return f"Weak: Password length exceeds the maximum limit of {max_length} characters."

    # Check blacklist
    blacklist = ["password", "123456", "1234", "admin", ”sabasabasaba”]
    if password.lower() in blacklist:
        return "Weak: Password is easily guessable and part of the blacklist."

    # Check complexity
    if not re.search(r"\d", password) or not re.search(r"[A-Z]", password) or not
re.search(r"[!@#$%^&*(),.?\":{}|<>]",
                                                        password):
        return "Weak: Password should include at least one digit, one uppercase letter, and one special character."

    # Check randomness
    if len(set(password)) < len(password) / 2:
        return "Strong: Password appears to be randomly chosen."

    return "Strong: Password meets recommended security guidelines."
```

فاز ۱) قسمت دوم

خروجی:

مود استاندارد {تنها اعداد}:



مود دانستن کاراکتر اول{تنها اعداد}:

```
Enter the search mode (1 for standard, 2 for search by first character, 3 for search by k characters): 2
Enter the first character of the password: 1
Enter the length of the password: 2
Enter the search space (1 for numbers, (2 for lowercase letters, (3 for lowercase letters and numbers, (4 fo
Enter the password: 12
10
11
12
Found the password with 3 tries.
Elapsed time: 0.00010704994201660156 seconds
```

مود دانستن k کاراکتر{تنها اعداد}:

```
Enter the search mode (1 for standard, 2 for search by first character, 3 for search by k characters): 3
Enter the length of the password: 2
Enter the search space (1 for numbers, (2 for lowercase letters, (3 for lowercase letters and numbers, (4 fo
Enter the known characters: 12
Enter the password: 12
12
Found the password with 1 tries.
Elapsed time: 0.000102996826171875 seconds
```

مود استاندارد {تنها حروف کوچک}:

```
Enter the search mode (1 for standard, 2 for search by first character, 3 for search by k characters): 1
Enter the length of the password: 3
Enter the search space (1 for numbers, (2 for lowercase letters, (3 for lowercase letters and numbers, (4 for
Enter the password: abc

aba
abb
abc
Found the password with 29 tries.
Elapsed time: 0.004427194595336914 seconds
```

مود دانستن کاراکتر اول{تنها حروف کوچک}:

```
Enter the search mode (1 for standard, 2 for search by first character, 3 for search by k characters): 2
Enter the first character of the password: b
Enter the length of the password: 3
Enter the search space (1 for numbers, (2 for lowercase letters, (3 for lowercase letters and numbers, (4 for n
Enter the password: bca
baa
bbz
bca
Found the password with 53 tries.
Elapsed time: 0.0003631114959716797 seconds
```

مود دانستن k کاراکتر{تنها حروف کوچک}؛

```
Enter the search mode (1 for standard, 2 for search by first character,
Enter the length of the password: 3
Enter the search space (1 for numbers, (2 for lowercase letters, (3 for
Enter the known characters: ac
Enter the password: abc
acb
abc
Found the password with 2 tries.
Elapsed time: 0.0003559589385986328 seconds
```

مود استاندارد {اعداد و حروف کوچک}:

```
Enter the search mode (1 for standard, 2 for search by first character, 3 for search by k characters): 1
Enter the length of the password: 3
Enter the search space (1 for numbers, (2 for lowercase letters, (3 for lowercase letters and numbers, (4 for n
Enter the password: 2aa
aaa
2aa
Found the password with 36289 tries.
Elapsed time: 0.09813213348388672 seconds
```

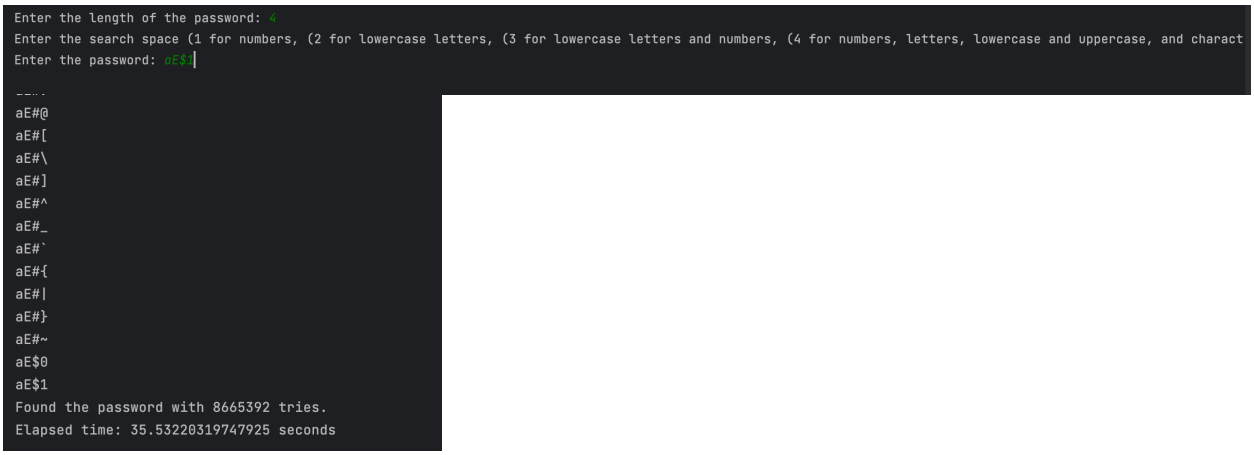
مود دانستن کاراکتر اول{اعداد و حروف کوچک}:

```
Enter the search mode (1 for standard, 2 for search by first character,
Enter the first character of the password: 2
Enter the length of the password: 3
Enter the search space (1 for numbers, (2 for lowercase letters, (3 for
Enter the password: 2aa
2aa
Found the password with 1 tries.
Elapsed time: 0.0010039806365966797 seconds
```

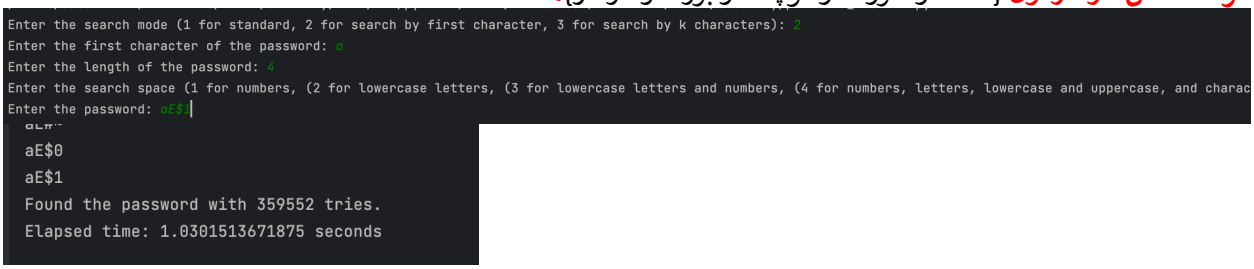
مود دانستن k کاراکتر{اعداد و حروف کوچک}:

```
Enter the search mode (1 for standard, 2 for search by first character, 3 for search by k characters): 3
Enter the length of the password: 3
Enter the search space (1 for numbers, (2 for lowercase letters, (3 for lowercase letters and numbers, (4 for nu
Enter the known characters: b2
Enter the password: ab2
b2a
ba2
2ab
ab2
Found the password with 4 tries.
Elapsed time: 0.0004088878631591797 seconds
```

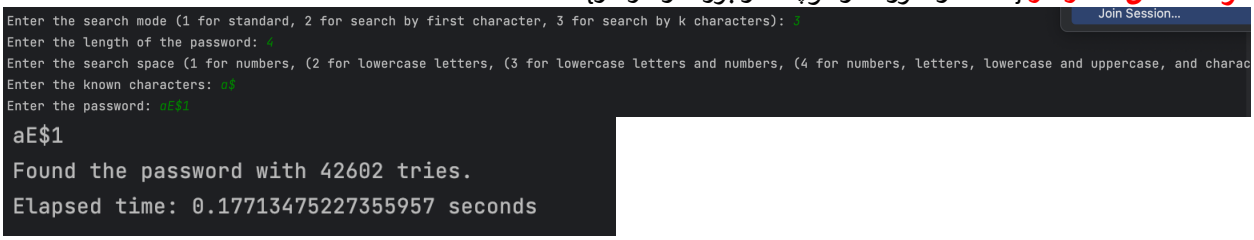
مود استاندارد {اعداد و حروف و کوچک و بزرگ و کاراکتر}**:**



مود دانستن کاراکتر اول {اعداد و حروف و کوچک و بزرگ و کاراکتر}**:**



مود دانستن k کاراکتر {اعداد و حروف و کوچک و بزرگ و کاراکتر}**:**



در این بخش کاربر یک ورودی پسوورد به برنامه میدهد و برنامه با استفاده از روش brute force و بررسی تمام حالات پسوورد کاربر را به دست میاورد و گزارش میدهد چند حدس زده و چقدر طول کشیده تا پسوورد به دست بیاید.
این برنامه سه مد دارد:
مد ۱) خود پسوورد را میدهد با طول پسوورد
مد ۲) کاراکتر اول پسوورد و پسوورد را میدهد با طول پسوورد
مد ۳)k کاراکتر از پسوورد و پسوورد را میدهد با طول پسوورد

همچنین کاربر باید search space را تعیین کند که حالات زیر است:
{تنها اعداد}
{اعداد و حروف کوچک}
{تنها حروف کوچک}
{اعداد و حروف و کوچک و بزرگ و کاراکتر}

حالت استاندارد:

```
def standard_mode():
    password_length = int(input("Enter the length of the password: "))
    search_space = input("Enter the search space"
                         " (1 for numbers, (2 for lowercase letters,"
                         " (3 for lowercase letters and numbers,"
                         " (4 for numbers, letters, lowercase and uppercase, and characters")
    if search_space == "1":
        search_space = string.digits
    elif search_space == "2":
        search_space = string.ascii_lowercase
    elif search_space == "3":
        search_space = string.ascii_lowercase + string.digits
    elif search_space == "4":
        search_space = string.digits + string.ascii_letters + string.punctuation
    else:
        print("Invalid search space.")
        return

    password = input("Enter the password: ")

    if len(password) != password_length or not all(char in search_space for char in password):
        print("Invalid password.")
        return

    start_time = time.time()

    possible_strings = [''.join(p) for p in itertools.product(search_space, repeat=password_length)]
    find_password_in_generated_passwords(possible_strings, password, "")

    end_time = time.time()
    elapsed_time = end_time - start_time
    print(f"Elapsed time: {elapsed_time} seconds")

    return possible_strings
```

در این حالت استاندارد، کاربر برنامه ابتدا طول مورد نظر برای گذرواژه و فضای جستجو را مشخص می‌کند. سپس با توجه به فضای جستجو، برنامه تمام ترکیب‌های ممکن از پسوورد‌ها را ایجاد می‌کند.

```
def find_password_in_generated_passwords(generated_strings, target_string, first_letter):
    for i, generated_string in enumerate(generated_strings):
        print(first_letter + generated_string)
        if generated_string == target_string:
            print(f"Found the password with {i + 1} tries.")
            return
```

سپس برنامه روی حالت های مختلف پیمایش میکند و اگر پسوورد را پیدا کرد تعداد پیمایش و زمان صرف شده را چاپ میکند.

حالت جست وجو با دانستن حرف اول:

- مشابه حالت استاندارد است، اما کاربر همچنین اولین حرف گذرواژه را مشخص می‌کند.
- برنامه رشته‌های ممکنه را از دومین حرف به بعد تولید می‌کند و زمان گذشته برای یافتن گذرواژه را اندازه‌گیری می‌کند.

```
def search_by_first_char():
    first_char = input("Enter the first character of the password: ")
    password_length = int(input("Enter the length of the password: "))
    search_space = input("Enter the search space"
                         " (1 for numbers, (2 for lowercase letters,"
                         " (3 for lowercase letters and numbers,"
                         " (4 for numbers, letters, lowercase and uppercase, and characters")
    if search_space == "1":
        search_space = string.digits
    elif search_space == "2":
        search_space = string.ascii_lowercase
    elif search_space == "3":
        search_space = string.ascii_letters + string.digits
    elif search_space == "4":
        search_space = string.digits + string.ascii_letters + string.punctuation
    else:
        print("Invalid search space.")
        return

    password = input("Enter the password: ")

    if len(password) != password_length or not all(char in search_space for char in password):
        print("Invalid password.")
        return

    password = password[1:]

    start_time = time.time()

    possible_strings = [''.join(p) for p in itertools.product(search_space, repeat=password_length - 1)]
    find_password_in_generated_passwords(possible_strings, password, first_char)

    end_time = time.time()
    elapsed_time = end_time - start_time
    print(f"Elapsed time: {elapsed_time} seconds")

    return possible_strings
```

این حالت کاملا مانند حالت قبلی است ولی حرف اول پسوورد از فضای تمام حالات خارج میشود و باید جایگشت های حرف دوم تا حرف آخر بررسی شود تا پسوورد پیدا شود.

همینطور که مشاهده میشود فضای حالت جایگشت پسوورد ها طولش یکی کمتر شده:

```
possible_strings = [''.join(p) for p in itertools.product(search_space, repeat=password_length - 1)]
```

برای پرینت کردن حالات حرف اول را اضافه میکنیم:

```
        print(first_letter + generated_string)
```

حالت جست وجو با دانستن حرف اول:

```
def search_by_k_chars():
    password_length = int(input("Enter the length of the password: "))
    search_space = input("Enter the search space"
                         " (1 for numbers, (2 for lowercase letters,"
                         " (3 for lowercase letters and numbers,"
                         " (4 for numbers, letters, lowercase and uppercase, and characters")
    known_characters = input("Enter the known characters: ")

    if search_space == "1":
        search_space = string.digits
    elif search_space == "2":
        search_space = string.ascii_lowercase
    elif search_space == "3":
        search_space = string.ascii_lowercase + string.digits
    elif search_space == "4":
        search_space = string.digits + string.ascii_letters + string.punctuation
    else:
        print("Invalid search space.")
        return

    password = input("Enter the password: ")

    if len(password) != password_length or not all(char in search_space for char in password):
        print("Invalid password.")
        return

    start_time = time.time()

    possible_strings = generate_permutations_with_known_chars(password_length, known_characters,
search_space)

    find_password_in_generated_passwords(possible_strings, password, "")

    end_time = time.time()
    elapsed_time = end_time - start_time
    print(f"Elapsed time: {elapsed_time} seconds")

    return possible_strings
```

- در این حالت، کاربر طول گذرواژه، فضای جستجو، تعدادی از حروف شناخته‌شده و گذرواژه هدف را مشخص می‌کند.
- برنامه جایگشت‌هایی با حروف شناخته‌شده تولید می‌کند و زمان گذشته برای یافتن گذرواژه را اندازه‌گیری می‌کند.

باید جایگشتی از فضای حالت مانند حالت‌های قبلی ایجاد شود با این تفاوت که در این جایگشت‌ها حتما باید حروف شناخته شده وجود داشته باشد. برای ایجاد این جایگشت از کد زیر استفاده میکنیم:

```
def generate_permutations_with_known_chars(length, known_chars, search_space):
    remaining_chars = [char for char in search_space if char not in known_chars]
```

```
permutations_remaining = list(itertools.permutations(remaining_chars, length - len(known_chars)))

result_permutations = []
for perm in permutations_remaining:
    for known_char_positions in itertools.permutations(range(length), len(known_chars)):
        current_permutation = list(perm)
        for i, pos in enumerate(known_char_positions):
            current_permutation.insert(pos, known_chars[i])
        result_permutations.append(''.join(current_permutation))

return result_permutations
```

این تابع جایگشت‌هایی با حروف شناخته‌شده را تولید می‌کند با در نظر گرفتن حروف باقیمانده در فضای جستجو.

این تابع برای حل مسئله تولید ترکیب‌های ممکن از حروف فضای جستجو با طول مشخص و حروف شناخته شده مورد استفاده قرار می‌گیرد. با استفاده از این تابع، می‌توانیم تمام ترکیب‌های ممکن از حروف فضای جستجو را به دست آوریم که حروف مشخص نیز در جایگاه‌های خاصی در آن‌ها قرار دارند.

توضیح برای هر قسمت از تابع:

تعیین حروف باقی‌مانده:

- ابتدا از فضای حالات کاراکتر های مشخص شده کم میشود.

```
remaining_chars = [char for char in search_space if char not in known_chars]
```

تولید ترکیب‌های باقی‌مانده:

- از itertools.permutations برای تولید ترکیب‌های ممکن از حروف باقی‌مانده استفاده می‌شود. این ترکیب‌ها در لیست permutations_remaining ذخیره می‌شوند.

```
permutations_remaining = list(itertools.permutations(remaining_chars, length -
len(known_chars)))
```

ساخت ترکیب‌های نهایی:

```
for perm in permutations_remaining:
    for known_char_positions in itertools.permutations(range(length), len(known_chars)):
        current_permutation = list(perm)
        for i, pos in enumerate(known_char_positions):
            current_permutation.insert(pos, known_chars[i])
        result_permutations.append(''.join(current_permutation))
```

- با یک حلقه for بر رویpermutations_remaining، برای هر ترکیب از حروف باقی‌مانده، یک حلقه for دیگر بر روی ترتیب‌های ممکن حروف شناخته شده اجرا می‌شود.
- در هر دور از حلقه داخلی، حروف شناخته شده در مکان‌های مشخص در ترکیب افزوده می‌شوند.
- ترکیب نهایی با افزودن حروف شناخته شده در مکان‌های خاص، در لیست result_permutations قرار می‌گیرد.

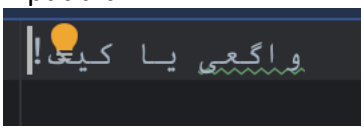
- در نهایت، لیست نهایی از ترکیب‌ها به عنوان خروجی تابع برگردانده می‌شود.

فاز ۲)

خروجی:

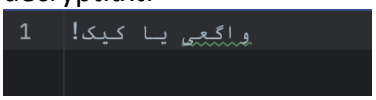
```
(venv) sabasahban@sabas-MacBook-Pro security % python file_cryptography.py encrypt input.txt encrypt.txt -p your_password
File encrypted successfully.
(venv) sabasahban@sabas-MacBook-Pro security % python file_cryptography.py decrypt encrypt.txt decrypt.txt -p your_password
File decrypted successfully.
```

input.txt:



encrypt.txt:

decrypt.txt:



اگر پسوورد اشتباه باشد decrypt اشتباه میشود:

```
(venv) sabasahban@sabas-MacBook-Pro security % python file_cryptography.py decrypt encrypt.txt decrypt.txt -p your_password1
```

decrypt.txt:

تابع key_derivation:

```python
def key_derivation(password, salt):
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        iterations=100000,
        length=32,
        salt=salt,
        backend=default_backend()
    )
    return kdf.derive(password.encode())
```

این تابع یک گذرواژه کاربر و یک salt که به صورت تصادفی تولید شده را به عنوان ورودی می‌گیرد، سپس از الگوریتم HMAC-SHA256 برای تولید کردن یک کلید امن استفاده می‌کند. کلید تولید می‌تواند برای رمزنگاری و رمزگشایی در ادامه استفاده شود.

در این قسمت، دو مود وجود دارد :

1 - مود رمزنگاری: در این مود برنامه یک فایل را به‌عنوان ورودی گرفته و پس از رمز کردن آن را به‌صورت رمز شده در سیستم ذخیره میکند .

توضیح تابع encryption:

```python
def encrypt_file(input_file, output_file, password):
    salt = os.urandom(16)
    key = key_derivation(password, salt)
    iv = os.urandom(16)

    with open(input_file, 'rb') as f:
        plaintext = f.read()

    cipher = Cipher(algorithms.AES(key), modes.CFB(iv), backend=default_backend())
```

```
    encryptor = cipher.encryptor()
    ciphertext = encryptor.update(plaintext) + encryptor.finalize()

    with open(output_file, 'wb') as f:
        f.write(salt + iv + ciphertext)
```

ورودی‌ها:

```
def encrypt_file(input_file, output_file, password):
```

input_file - نام فایل ورودی که باید رمزنگاری شود.
output_file - نام فایل خروجی که در آن متن رمزنگاری شده ذخیره می‌شود.
password - گذرواژه‌ای که برای تولید کلید و استفاده در عملیات رمزنگاری استفاده می‌شود.

2. تولید salt

```
salt = os.urandom(16)
```

- یک مقدار salt به صورت تصادفی با طول 16 بایت با استفاده از تابع `os.urandom` تولید می‌شود.

3. تولید کلید

```
key = key_derivation(password, salt)
```

- از تابع `key_derivation` برای تولید کلید مشتق از گذرواژه و salt استفاده می‌شود.

4. تولید بردار اولویت(IV)

```
iv = os.urandom(16)
```

- یک بردار اولویت به صورت تصادفی با طول 16 بایت تولید می‌شود.

5رمزنگاری فایل:

```
cipher = Cipher(algorithms.AES(key), modes.CFB(iv), backend=default_backend())
encryptor = cipher.encryptor()
ciphertext = encryptor.update(plaintext) + encryptor.finalize()
```

- محتوای فایل ورودی به صورت باینری خوانده می‌شود.
- از الگوریتم رمزنگاری AES با حالت CFB (Cipher Feedback) و کلید تولید شده به همراه بردار اولویت استفاده می‌شود.
- متن رمزنگاری شده با استفاده از update و finalize تولید می‌شود.

6. نوشتن به فایل خروجی

```
with open(output_file, 'wb') as f:
    f.write(salt + iv + ciphertext)
```

بردار اولویت و متن رمزنگاری شده به صورت پشت‌سرهم در فایل خروجی نوشته می‌شوند.

به طور خلاصه، این تابع یک فایل را با استفاده از الگوریتم رمزنگاری AES و یک گذرواژه به صورت امن رمزنگاری می‌کند و نتیجه را در یک فایل خروجی ذخیره می‌کند.

2 - مود رمزگشایی: در این مود برنامه یک فایل رمز شده را به عنوان ورودی گرفته و پس از رمزگشایی آن، فایل را در سیستم ذخیره میکند.

```
def decrypt_file(input_file, output_file, password):
    with open(input_file, 'rb') as f:
        data = f.read()
```

```
    salt = data[:16]
    iv = data[16:32]
    ciphertext = data[32:]

key = key_derivation(password, salt)

cipher = Cipher(algorithms.AES(key), modes.CFB(iv), backend=default_backend())
decryptor = cipher.decryptor()
plaintext = decryptor.update(ciphertext) + decryptor.finalize()

with open(output_file, 'wb') as f:
    f.write(plaintext)
```

در تابع decrypt_file رمزگشایی یک فایل به وسیله پسوورد اجرا می‌شود. این تابع از الگوریتم رمزنگاری AES با حالت CFBبرای رمزگشایی استفاده می‌کند.

ورودی‌ها:

```
def decrypt_file(input_file, output_file, password):
```

input_file : مسیر فایل رمز شده که باید رمزگشایی شود.
output_file: مسیر فایل خروجی برای ذخیره محتوای رمزگشایی شده.
password: گذرواژه مورد استفاده برای رمزگشایی فایل.

2 .خواندن داده‌ها از فایل

```
with open(input_file, 'rb') as f:
    data = f.read()
```

- تابع فایل رمز شده و تمام داده‌ها را به صورت خوانده و در متغیر data ذخیره می‌کند.

3 .استخراج اجزای مهم

```
salt = data[:16]
```

salt: - اولین 16 بایت از داده‌ها که برای افزایش امنیت در تولید کلید استفاده می‌شود.

```
iv = data[16:32]
```

`iv` (Initialization Vector): - ۱۶بایت بعدی که برای استفاده در الگوریتم CFB مورد نیاز است.

```
ciphertext = data[32:]
```

ciphertext: - بقیه بخش داده‌ها که باید رمزگشایی شود.

4 .تولید کلید

```
key = key_derivation(password, salt)
```

- 5 .رمزگشایی

```
cipher = Cipher(algorithms.AES(key), modes.CFB(iv), backend=default_backend())
```

- از الگوریتم AES به همراه حالت CFB برای رمزگشایی استفاده می‌شود.

```
decryptor = cipher.decryptor()
plaintext = decryptor.update(ciphertext) + decryptor.finalize()
```

decryptor - برای انجام عملیات رمزگشایی ایجاد می‌شود و سپس با استفاده از update وfinalize ، محتوای رمزگشایی شده به دست می‌آید.

6 .نوشتن محتوای رمزگشایی شده در فایل خروجی

```
with open(output_file, 'wb') as f:
    f.write(plaintext)
```

- محتوای رمزگشایی شده در فایل خروجی ذخیره می‌شود.