COMP8760 Assessment 2: Practical Report

Week 16 Class

Week 16 was about working on some password cracking exercises to gain a better understanding of password security and usability.

The following exercises were attempted: -

- Password hashing and salting
- General dictionary attacks
- Personalised dictionary attack
- Breaking a graphical password (an Android unlock pattern)

How did I do the exercises and what did I learn about?

All of these exercises were attempted and completed through implementation of the same through Python 3.x.

Exercise 1 was done by implementing 2 important functions: one for user registration and the other for user login. Another function was implemented to perform necessary multiple SHA256 hashing on the password. User Registration took username, password and profile as parameters, concatenated the password and a randomly generated salt, this string was hashed using the multiple hashing function. The username, hashed value and profile are written and stored in a file. User Login takes username and password as input, searches the stored file for the username, reads the salt and does multiple hashing the string combining the password and salt, reads the stored hash value and compares with the calculated hash to see if they match. If they match, user authentication is successful else failed. Learnt how hashing with salt is useful to secure passwords and how it can be authenticated.

Exercise 2 was done by implementing a system that could crack a password based on a given hash value. Tried single to multiple hashing at most 4 times. The given data was from a leaked password database called phpbb. All values were hit with SHA256 hashing from 1 to 4 times and one of the hashes matched with the given hash and the password was found. Learnt how to crack a password from a leaked database given the hash value.

Exercise 3 was done by implementing a password cracking system which followed multiple hashing similar to exercise 2 and the passwords tried were based on keywords of a given information about the user, plus added with given salt and hashed to check for which combination of guessed passwords were the hash values matched for the given hash and bonus hash values. Learnt to try different permutation and combination of passwords based on information leaked from a user and to try hashing with salt to crack the passwords.

Exercise 4 was done by trying different permutations and combinations of the letters a,b,c,d,e,f,g,h and i with a length of 9 characters and no letters repeating to crack the alphabetical passcode of a user's phone given the hash value. Matched hash value with the hashes of different possible passcode combinations. Hashing was done using SHA1 hash function once. Learnt how to generate different combinations of an alphabetical passcode and hashed to crack the passcode of the user.

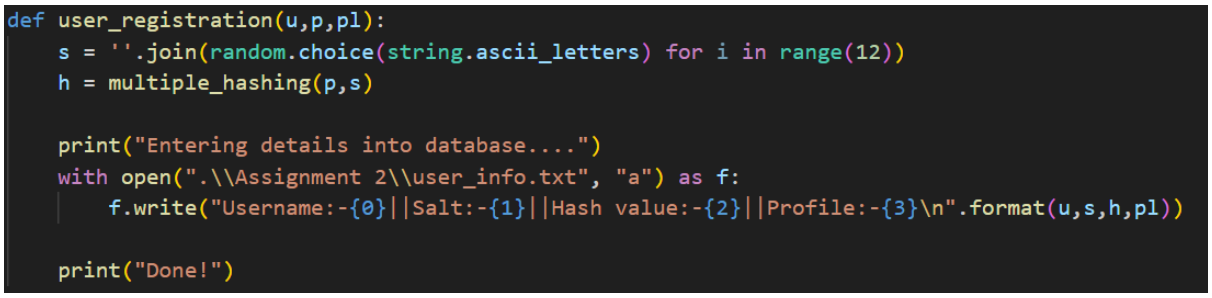
What difficulties / problems did I encounter? / What observations and/or thoughts did you have on the exercises?

Other than some syntax errors, file path, file read and write errors, did not face much challenges in exercises 1,2 and 4. Exercise 3 however was quite cumbersome, especially in coming up with keywords and combining all those in all ways plus considering lower and camel cases. Despite so many keywords, had a hard time cracking the passwords at first. Then realised there was a space in the salt added by mistake and removed it. Was able to finally crack the passwords then.

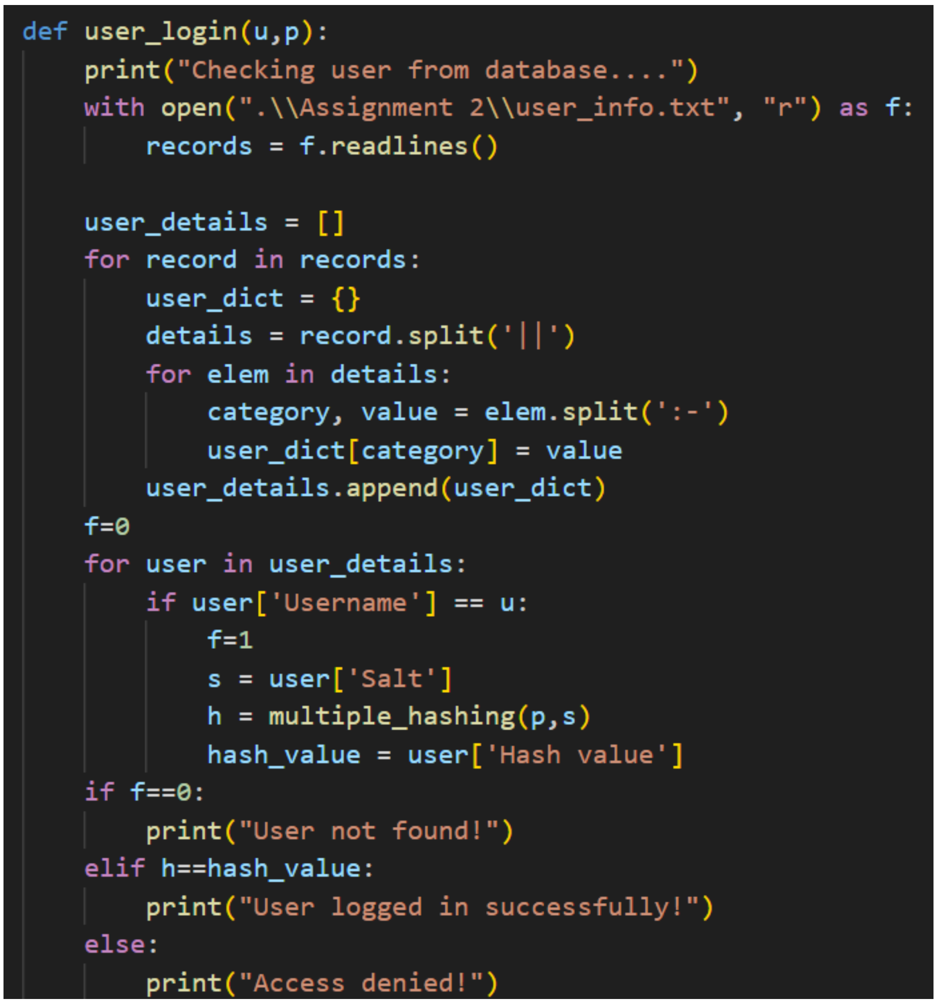
My observations were that these exercises could suggest a maximum limit on trying multiple hashing, because for password I started with 10 times hashing then gradually reduced it to a few. Also, the 4th exercise confused me a bit with the letter-number code and I thought the hashing was to be done on the corresponding numbers instead of the letters which was of course a futile effort.

Snippets of Code and Output

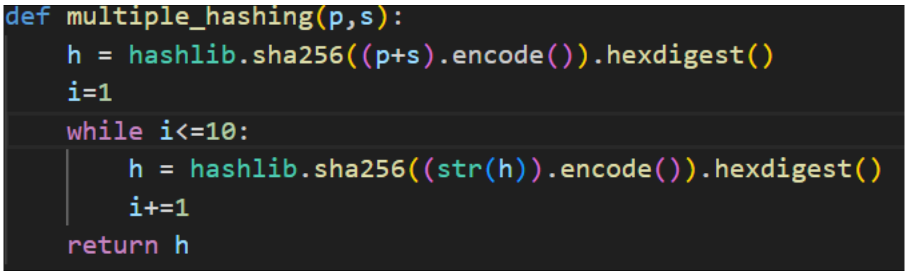
Exercise 1



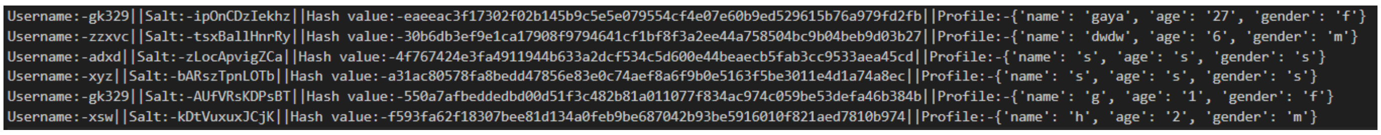
```python
def user_registration(u,p,pl):
    s = ''.join(random.choice(string.ascii_letters) for i in range(12))
    h = multiple_hashing(p,s)

    print("Entering details into database....")
    with open(".\\Assignment 2\\user_info.txt", "a") as f:
        f.write("Username:-{0}||Salt:-{1}||Hash value:-{2}||Profile:-{3}\n".format(u,s,h,pl))

    print("Done!")
```



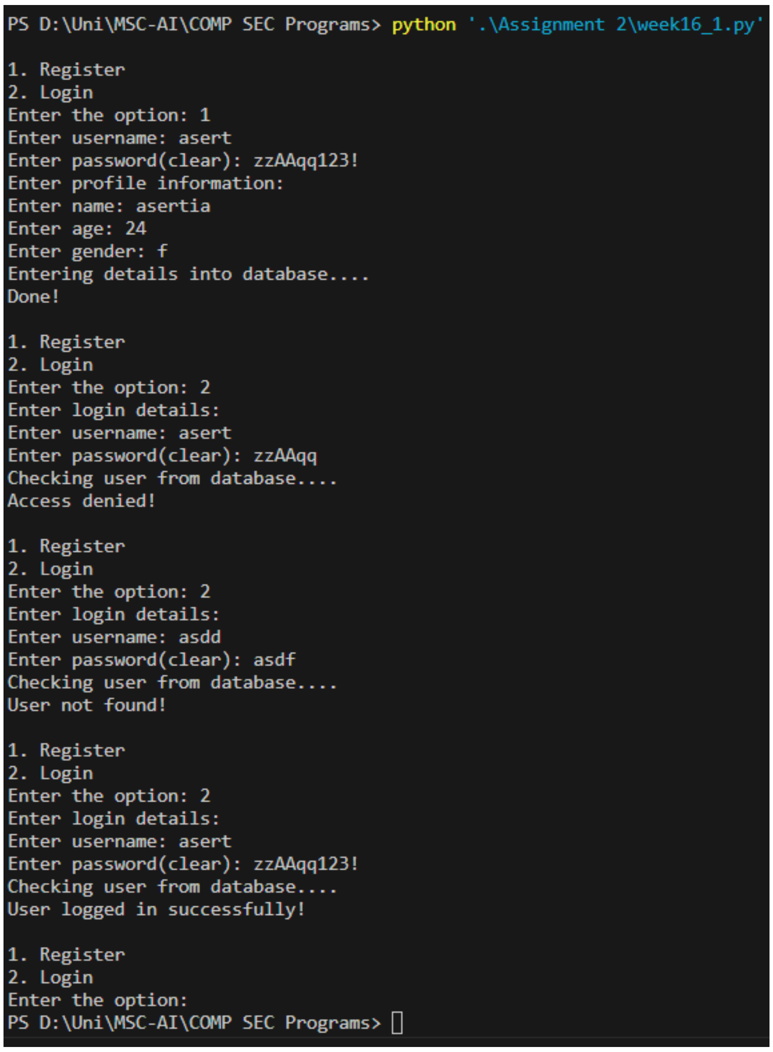
```python
def user_login(u,p):
    print("Checking user from database....")
    with open(".\\Assignment 2\\user_info.txt", "r") as f:
        records = f.readlines()

    user_details = []
    for record in records:
        user_dict = {}
        details = record.split('||')
        for elem in details:
            category, value = elem.split(':-')
            user_dict[category] = value
        user_details.append(user_dict)
    f=0
    for user in user_details:
        if user['Username'] == u:
            f=1
            s = user['Salt']
            h = multiple_hashing(p,s)
            hash_value = user['Hash value']
    if f==0:
        print("User not found!")
    elif h==hash_value:
        print("User logged in successfully!")
    else:
        print("Access denied!")
```



```python
def multiple_hashing(p,s):
    h = hashlib.sha256((p+s).encode()).hexdigest()
    i=1
    while i<=10:
        h = hashlib.sha256((str(h)).encode()).hexdigest()
        i+=1
    return h
```

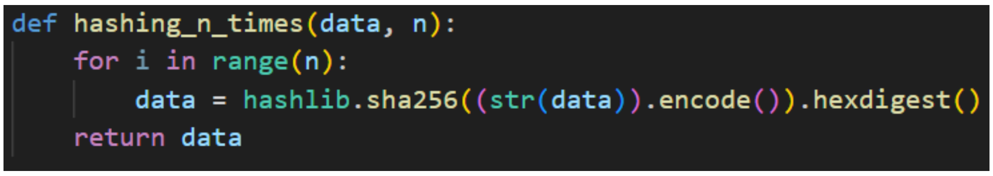


```
Username:-gk329||Salt:-ipOnCDzIekhz||Hash value:-eaeeac3f17302f02b145b9c5e5e079554cf4e07e60b9ed529615b76a979fd2fb||Profile:-{'name': 'gaya', 'age': '27', 'gender': 'f'}
Username:-zzxvc||Salt:-tsxBallHnrRy||Hash value:-30b6db3ef9e1ca17908f9794641cf1bf8f3a2ee44a758504bc9b04beb9d03b27||Profile:-{'name': 'dwdw', 'age': '6', 'gender': 'm'}
Username:-adxd||Salt:-zLocApvigZCa||Hash value:-4f767424e3fa4911944b633a2dcf534c5d600e44beaecb5fab3cc9533aea45cd||Profile:-{'name': 's', 'age': 's', 'gender': 's'}
Username:-xyz||Salt:-bARszTpnLOTb||Hash value:-a31ac80578fa8bedd47856e83e0c74aef8a6f9b0e5163f5be3011e4d1a74a8ec||Profile:-{'name': 's', 'age': 's', 'gender': 's'}
Username:-gk329||Salt:-AUfVRsKDPsBT||Hash value:-550a7afbeddedbd00d51f3c482b81a011077f834ac974c059be53defa46b384b||Profile:-{'name': 'g', 'age': '1', 'gender': 'f'}
Username:-xsw||Salt:-kDtVuxuxJCjK||Hash value:-f593fa62f18307bee81d134a0feb9be687042b93be5916010f821aed7810b974||Profile:-{'name': 'h', 'age': '2', 'gender': 'm'}
```

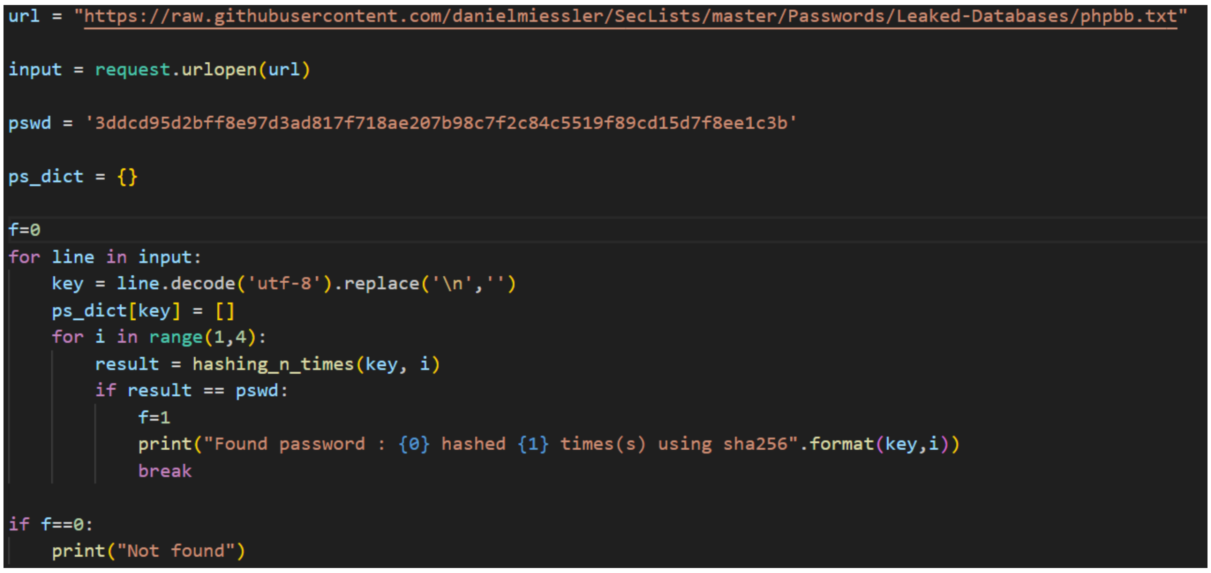


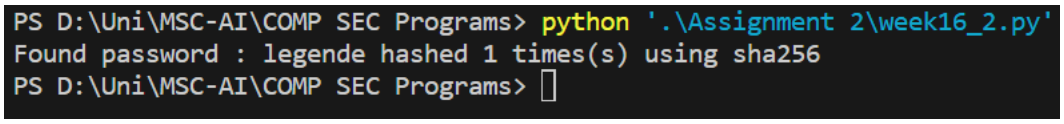
```
PS D:\Uni\MSC-AI\COMP SEC Programs> python '.\Assignment 2\week16_1.py'

1. Register
2. Login
Enter the option: 1
Enter username: asert
Enter password(clear): zzAAqq123!
Enter profile information:
Enter name: asertia
Enter age: 24
Enter gender: f
Entering details into database....
Done!

1. Register
2. Login
Enter the option: 2
Enter login details:
Enter username: asert
Enter password(clear): zzAAqq
Checking user from database....
Access denied!

1. Register
2. Login
Enter the option: 2
Enter login details:
Enter username: asdd
Enter password(clear): asdf
Checking user from database....
User not found!

1. Register
2. Login
Enter the option: 2
Enter login details:
Enter username: asert
Enter password(clear): zzAAqq123!
Checking user from database....
User logged in successfully!

1. Register
2. Login
Enter the option:
PS D:\Uni\MSC-AI\COMP SEC Programs> 
```

Exercise 2

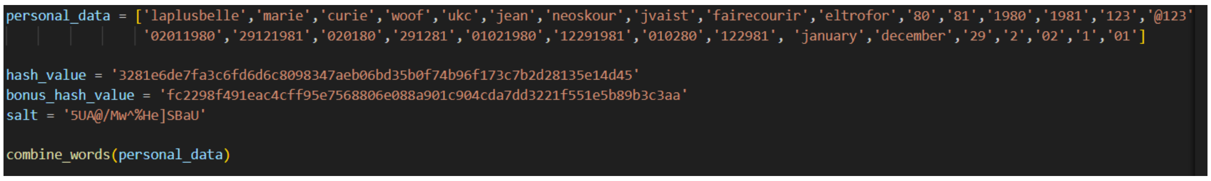


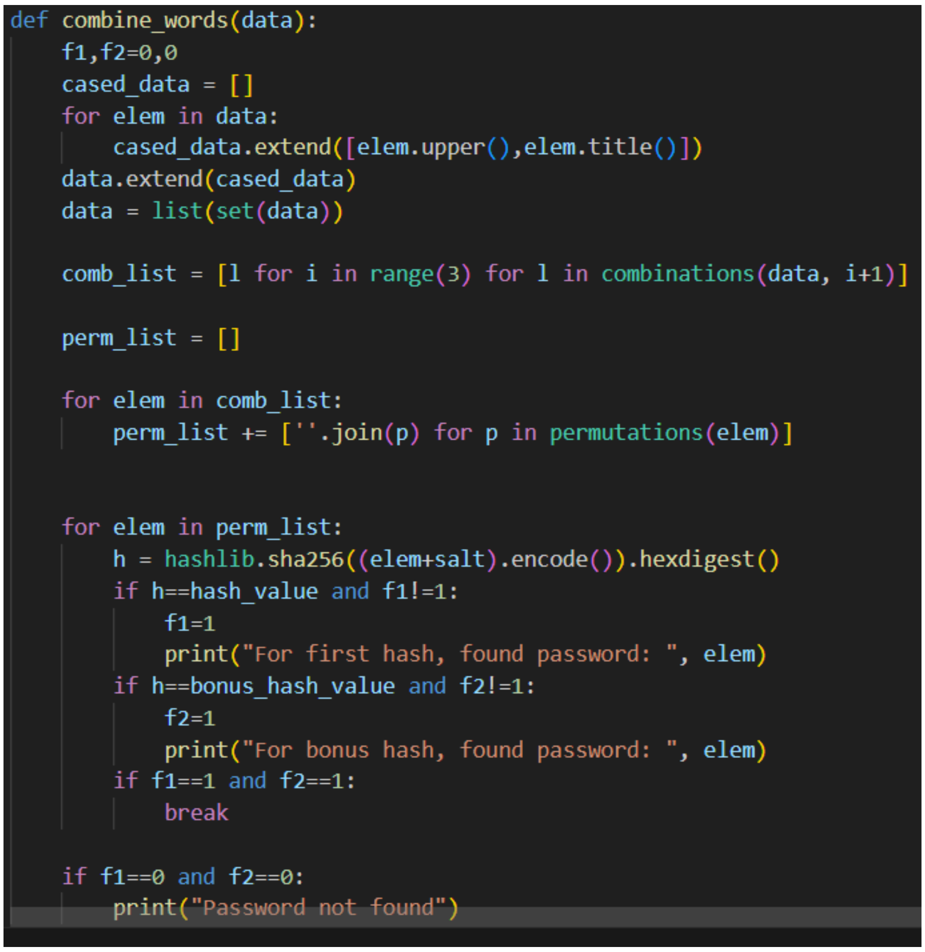
```python
def hashing_n_times(data, n):
    for i in range(n):
        data = hashlib.sha256((str(data)).encode()).hexdigest()
    return data
```


```
url = "https://raw.githubusercontent.com/danielmiessler/SecLists/master/Passwords/Leaked-Databases/phpbb.txt"

input = request.urlopen(url)

pswd = '3ddcd95d2bff8e97d3ad817f718ae207b98c7f2c84c5519f89cd15d7f8ee1c3b'

ps_dict = {}

f=0
for line in input:
    key = line.decode('utf-8').replace('\n','')
    ps_dict[key] = []
    for i in range(1,4):
        result = hashing_n_times(key, i)
        if result == pswd:
            f=1
            print("Found password : {0} hashed {1} times(s) using sha256".format(key,i))
            break

if f==0:
    print("Not found")
```

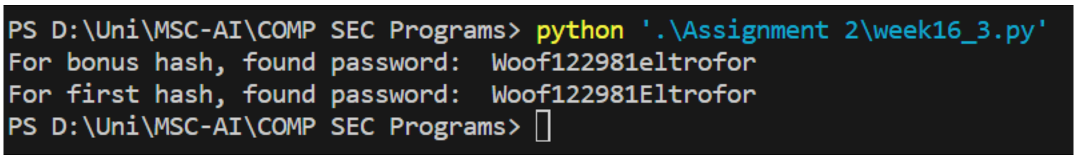

```
PS D:\Uni\MSC-AI\COMP SEC Programs> python '.\Assignment 2\week16_2.py'
Found password : legende hashed 1 times(s) using sha256
PS D:\Uni\MSC-AI\COMP SEC Programs> 
```

Exercise 3


```
personal_data = ['laplusbelle','marie','curie','woof','ukc','jean','neoskour','jvaist','fairecourir','eltrofor','80','81','1980','1981','123','@123'
                 '02011980','29121981','020180','291281','01021980','12291981','010280','122981', 'january','december','29','2','02','1','01']

hash_value = '3281e6de7fa3c6fd6d6c8098347aeb06bd35b0f74b96f173c7b2d28135e14d45'
bonus_hash_value = 'fc2298f491eac4cff95e7568806e088a901c904cda7dd3221f551e5b89b3c3aa'
salt = '5UA@/Mw^%He]SBaU'

combine_words(personal_data)
```

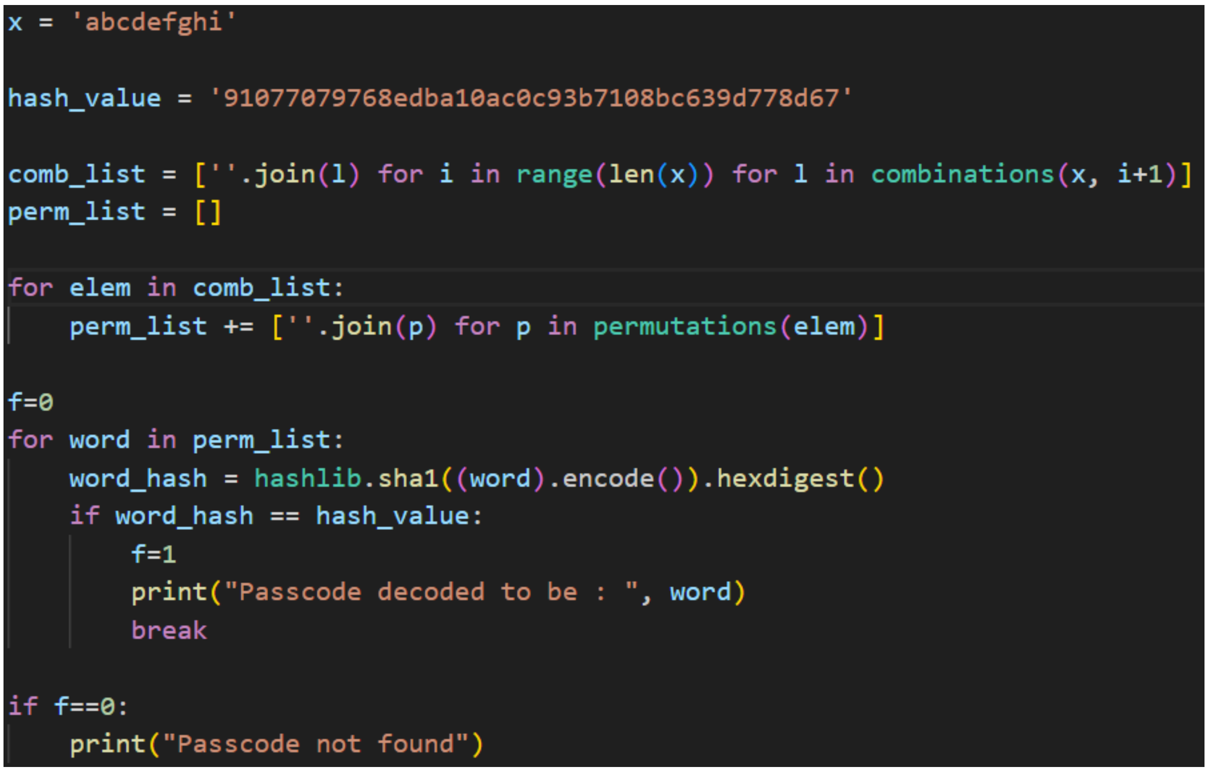


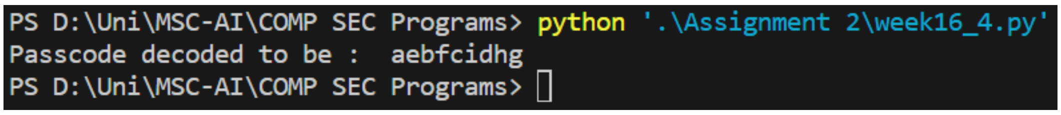
```python
def combine_words(data):
    f1,f2=0,0
    cased_data = []
    for elem in data:
        cased_data.extend([elem.upper(),elem.title()])
    data.extend(cased_data)
    data = list(set(data))

    comb_list = [l for i in range(3) for l in combinations(data, i+1)]

    perm_list = []

    for elem in comb_list:
        perm_list += [''.join(p) for p in permutations(elem)]


    for elem in perm_list:
        h = hashlib.sha256((elem+salt).encode()).hexdigest()
        if h==hash_value and f1!=1:
            f1=1
            print("For first hash, found password: ", elem)
        if h==bonus_hash_value and f2!=1:
            f2=1
            print("For bonus hash, found password: ", elem)
        if f1==1 and f2==1:
            break

    if f1==0 and f2==0:
        print("Password not found")
```



```
PS D:\Uni\MSC-AI\COMP SEC Programs> python '.\Assignment 2\week16_3.py'
For bonus hash, found password:  Woof122981eltrofor
For first hash, found password:  Woof122981Eltrofor
PS D:\Uni\MSC-AI\COMP SEC Programs> 
```

Exercise 4


```
x = 'abcdefghi'

hash_value = '91077079768edba10ac0c93b7108bc639d778d67'

comb_list = [''.join(l) for i in range(len(x)) for l in combinations(x, i+1)]
perm_list = []

for elem in comb_list:
    perm_list += [''.join(p) for p in permutations(elem)]

f=0
for word in perm_list:
    word_hash = hashlib.sha1((word).encode()).hexdigest()
    if word_hash == hash_value:
        f=1
        print("Passcode decoded to be : ", word)
        break

if f==0:
    print("Passcode not found")
```


```
PS D:\Uni\MSC-AI\COMP SEC Programs> python '.\Assignment 2\week16_4.py'
Passcode decoded to be :  aebfcidhg
PS D:\Uni\MSC-AI\COMP SEC Programs> 
```

Week 17 Class

Week 17 was about working on some exercises beyond just cracking passwords to gain a better understanding of user authentication as a whole.

The following exercises were attempted: -

- Password with emojis
- Password as a file on your computer
- A toy one-time-password (OTP) generator
- Evaluating performance of a mock biometric-based authentication system

How did I do the exercises and what did I learn about?

All of these exercises were attempted and completed through implementation of the same through Python 3.x.

Exercise 1 was done by using the same implementation system as that of week 16’s first exercise. It was able to take in passwords with emojis for user registration as well as user login as I had made the practice of encoding the password+salt before hashing in the same exercise which helped to incorporate the emojis. Learnt that we can use passwords with one or more combinations of emojis and hash and store the values.

Exercise 2 was done by implementing a system similar to exercise 1 but it asks for file name as input for file password. The corresponding file was read, its contents stored in a variable and then hashed and stored. When user logs in, the given file for password is read and hashed with the salt and compared to the stored hash value. Learnt that even files can be stored as passwords.

Exercise 3 was done by implementing previous exercise's system but one that makes sure during user login - to add the user login time to the existing hash value password of the user, hashing it again using SHA256 and storing the new hash in the user details file. The last 6 bytes of this hash is then sent as OTP. When user enters the correct OTP, authentication is successful. Learnt that hash values need not be constant till password changes, can use login datetime as a factor and understood a method of OTP generation and verification.

Exercise 4 was done by implementing a system to read the provided data of similarities of biometric templates of a user and attacker, and come up with different threshold values of similarities to check for which thresholds to accept user Alice with a small false reject rate and reject attacker Eve with a small false accept rate. Threshold values were chosen based on Alice's and Eve's average values from the given data. Learnt how to analyse performance of similarity checking and suggest thresholds from provided datasets.

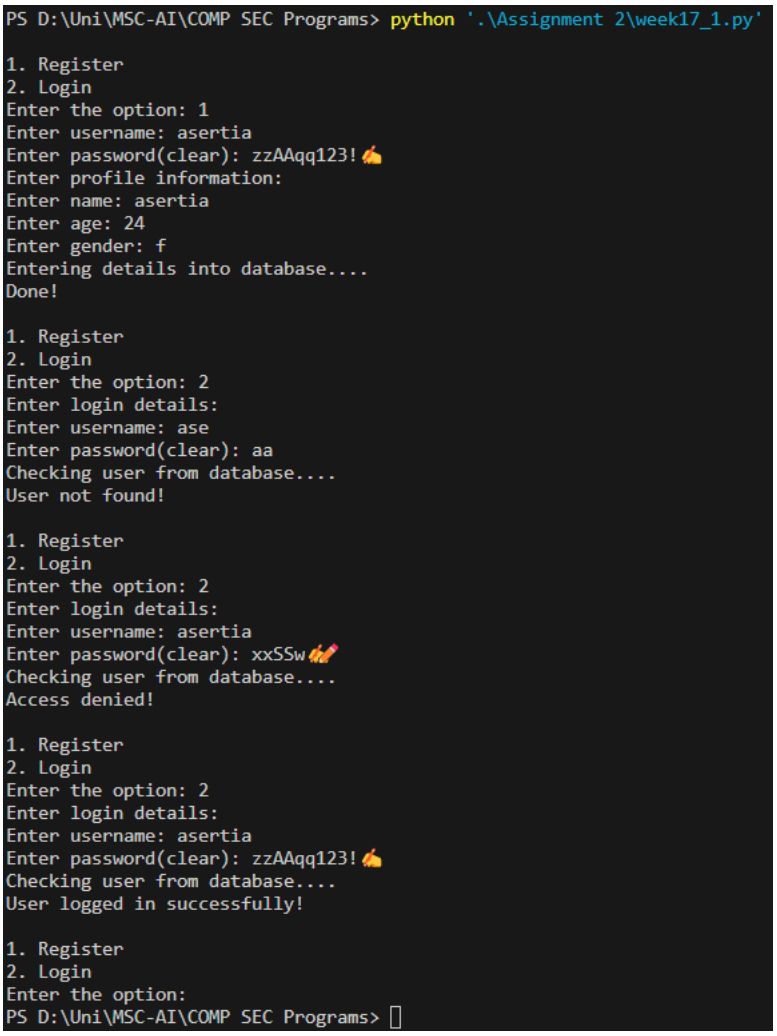
What difficulties / problems did I encounter? / What observations and/or thoughts did you have on the exercises?

Other than some syntax errors, file path, file read and write errors, did not face much challenges in these. Exercise 3 was a bit challenging, with coming up on how to compare password with hash value after each login and to decide whether to use password along with OTP for user login or just login with OTP and the latter was decided.

My observations were that the second exercise could have specified on how to input the password: whether to pass file name or create a UI to upload the file. I have implemented based on the latter.

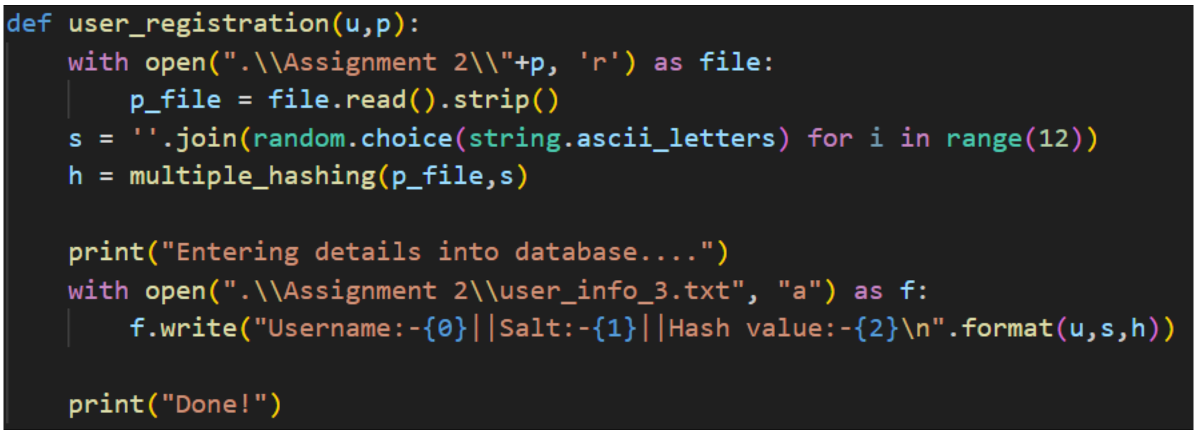
Snippets of Code and Output

Exercise 1

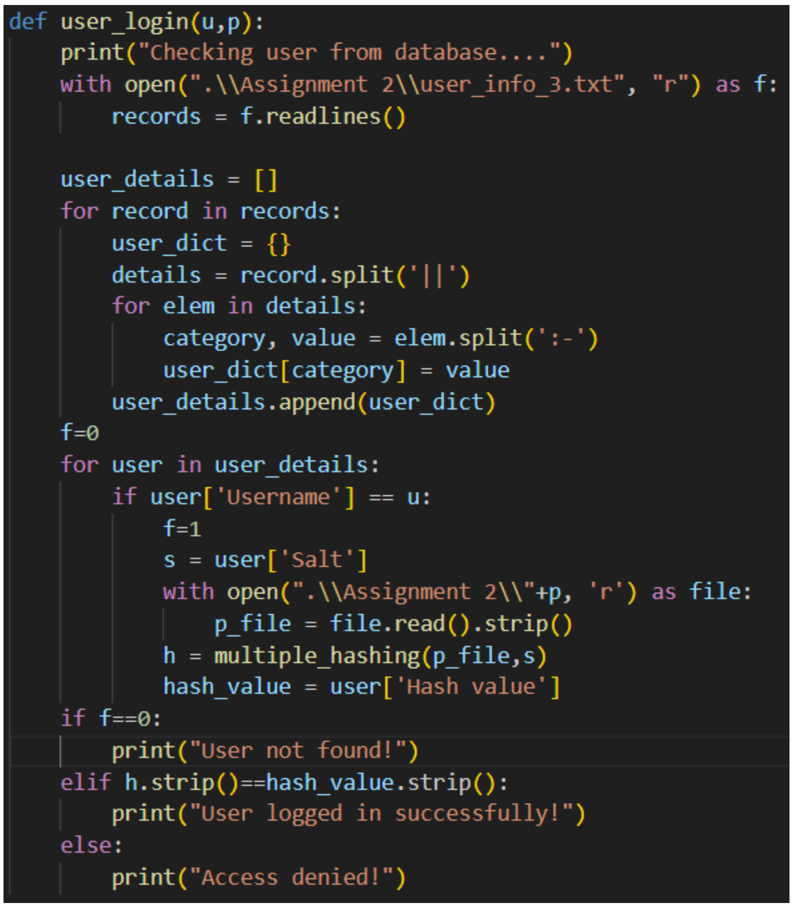


```
PS D:\Uni\MSC-AI\COMP SEC Programs> python '.\Assignment 2\week17_1.py'

1. Register
2. Login
Enter the option: 1
Enter username: asertia
Enter password(clear): zzAAqq123!✍
Enter profile information:
Enter name: asertia
Enter age: 24
Enter gender: f
Entering details into database....
Done!

1. Register
2. Login
Enter the option: 2
Enter login details:
Enter username: ase
Enter password(clear): aa
Checking user from database....
User not found!

1. Register
2. Login
Enter the option: 2
Enter login details:
Enter username: asertia
Enter password(clear): xxSSw✍✏
Checking user from database....
Access denied!

1. Register
2. Login
Enter the option: 2
Enter login details:
Enter username: asertia
Enter password(clear): zzAAqq123!✍
Checking user from database....
User logged in successfully!

1. Register
2. Login
Enter the option:
PS D:\Uni\MSC-AI\COMP SEC Programs> 
```

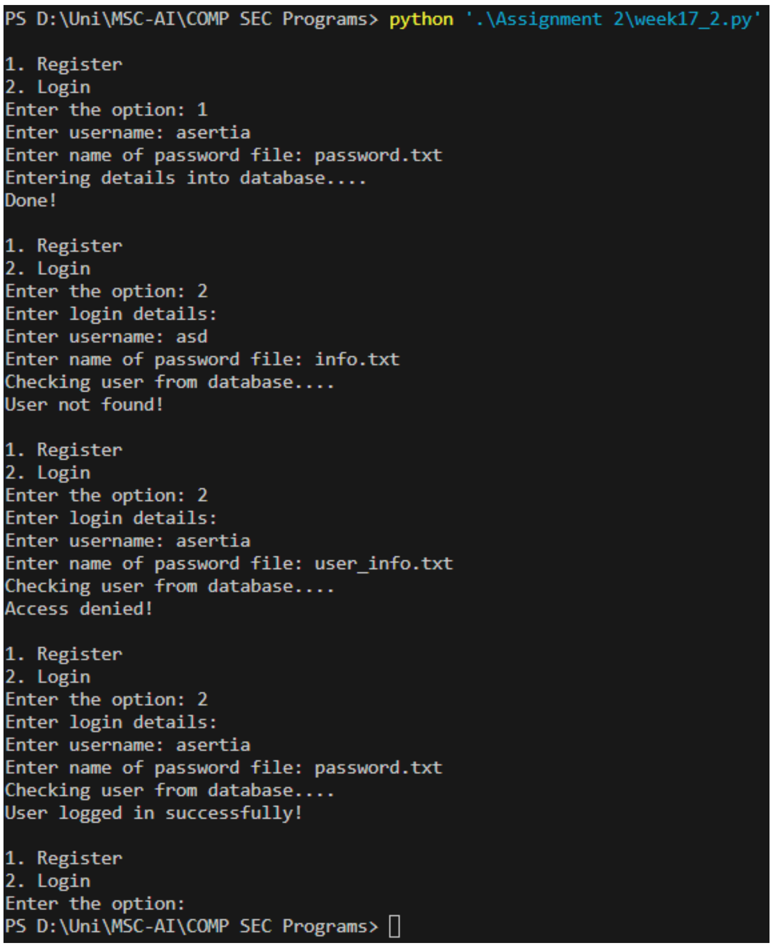
Exercise 2



```python
def user_registration(u,p):
    with open(".\\Assignment 2\\"+p, 'r') as file:
        p_file = file.read().strip()
    s = ''.join(random.choice(string.ascii_letters) for i in range(12))
    h = multiple_hashing(p_file,s)

    print("Entering details into database....")
    with open(".\\Assignment 2\\user_info_3.txt", "a") as f:
        f.write("Username:-{0}||Salt:-{1}||Hash value:-{2}\n".format(u,s,h))

    print("Done!")
```

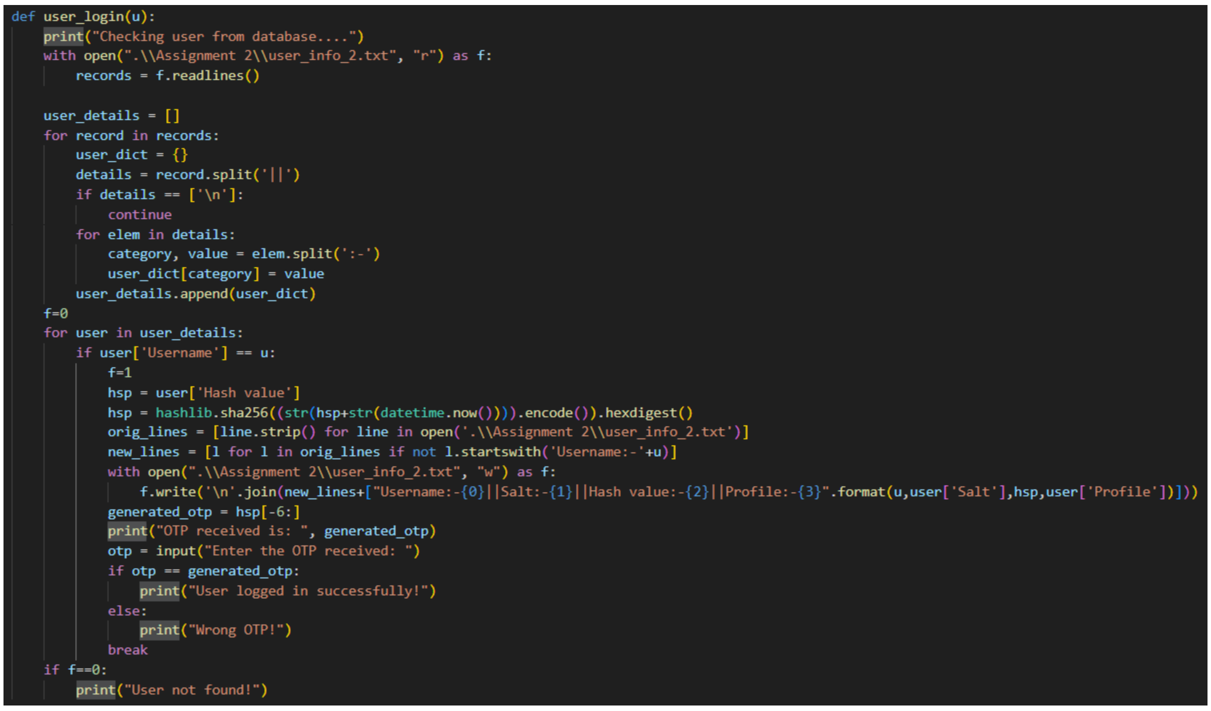


```python
def user_login(u,p):
    print("Checking user from database....")
    with open(".\\Assignment 2\\user_info_3.txt", "r") as f:
        records = f.readlines()

    user_details = []
    for record in records:
        user_dict = {}
        details = record.split('||')
        for elem in details:
            category, value = elem.split(':-')
            user_dict[category] = value
        user_details.append(user_dict)
    f=0
    for user in user_details:
        if user['Username'] == u:
            f=1
            s = user['Salt']
            with open(".\\Assignment 2\\"+p, 'r') as file:
                p_file = file.read().strip()
            h = multiple_hashing(p_file,s)
            hash_value = user['Hash value']
    if f==0:
        print("User not found!")
    elif h.strip()==hash_value.strip():
        print("User logged in successfully!")
    else:
        print("Access denied!")
```

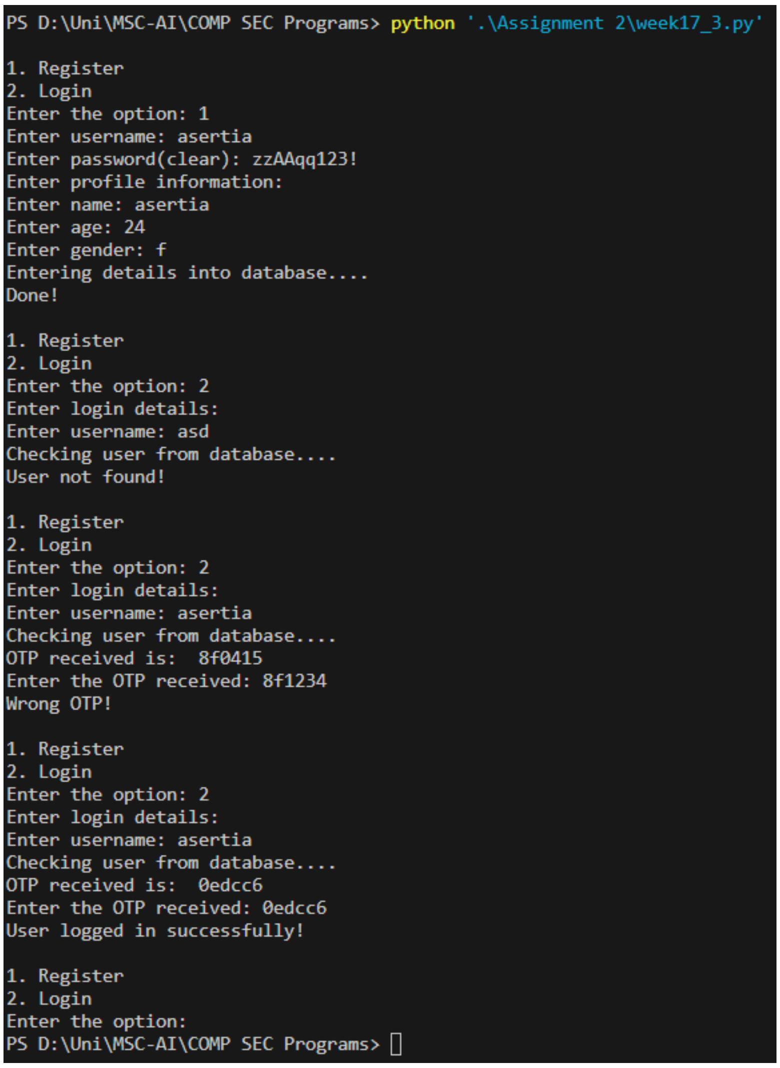


```
PS D:\Uni\MSC-AI\COMP SEC Programs> python '.\Assignment 2\week17_2.py'

1. Register
2. Login
Enter the option: 1
Enter username: asertia
Enter name of password file: password.txt
Entering details into database....
Done!

1. Register
2. Login
Enter the option: 2
Enter login details:
Enter username: asd
Enter name of password file: info.txt
Checking user from database....
User not found!

1. Register
2. Login
Enter the option: 2
Enter login details:
Enter username: asertia
Enter name of password file: user_info.txt
Checking user from database....
Access denied!

1. Register
2. Login
Enter the option: 2
Enter login details:
Enter username: asertia
Enter name of password file: password.txt
Checking user from database....
User logged in successfully!

1. Register
2. Login
Enter the option:
PS D:\Uni\MSC-AI\COMP SEC Programs> 
```

Exercise 3

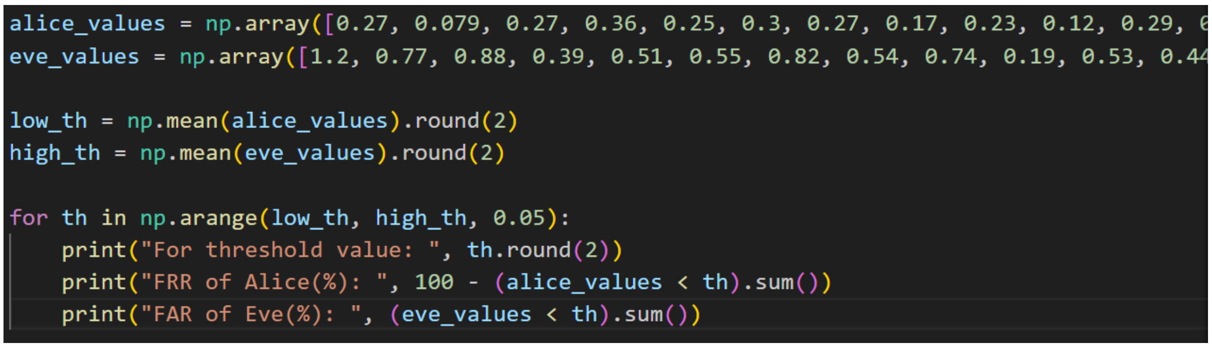


```python
def user_login(u):
    print("Checking user from database....")
    with open(".\\Assignment 2\\user_info_2.txt", "r") as f:
        records = f.readlines()

    user_details = []
    for record in records:
        user_dict = {}
        details = record.split('||')
        if details == ['\n']:
            continue
        for elem in details:
            category, value = elem.split(':-')
            user_dict[category] = value
        user_details.append(user_dict)
    f=0
    for user in user_details:
        if user['Username'] == u:
            f=1
            hsp = user['Hash value']
            hsp = hashlib.sha256((str(hsp+str(datetime.now()))).encode()).hexdigest()
            orig_lines = [line.strip() for line in open('.\\Assignment 2\\user_info_2.txt')]
            new_lines = [l for l in orig_lines if not l.startswith('Username:-'+u)]
            with open(".\\Assignment 2\\user_info_2.txt", "w") as f:
                f.write('\n'.join(new_lines+["Username:-{0}||Salt:-{1}||Hash value:-{2}||Profile:-{3}".format(u,user['Salt'],hsp,user['Profile'])]))
            generated_otp = hsp[-6:]
            print("OTP received is: ", generated_otp)
            otp = input("Enter the OTP received: ")
            if otp == generated_otp:
                print("User logged in successfully!")
            else:
                print("Wrong OTP!")
            break
    if f==0:
        print("User not found!")
```

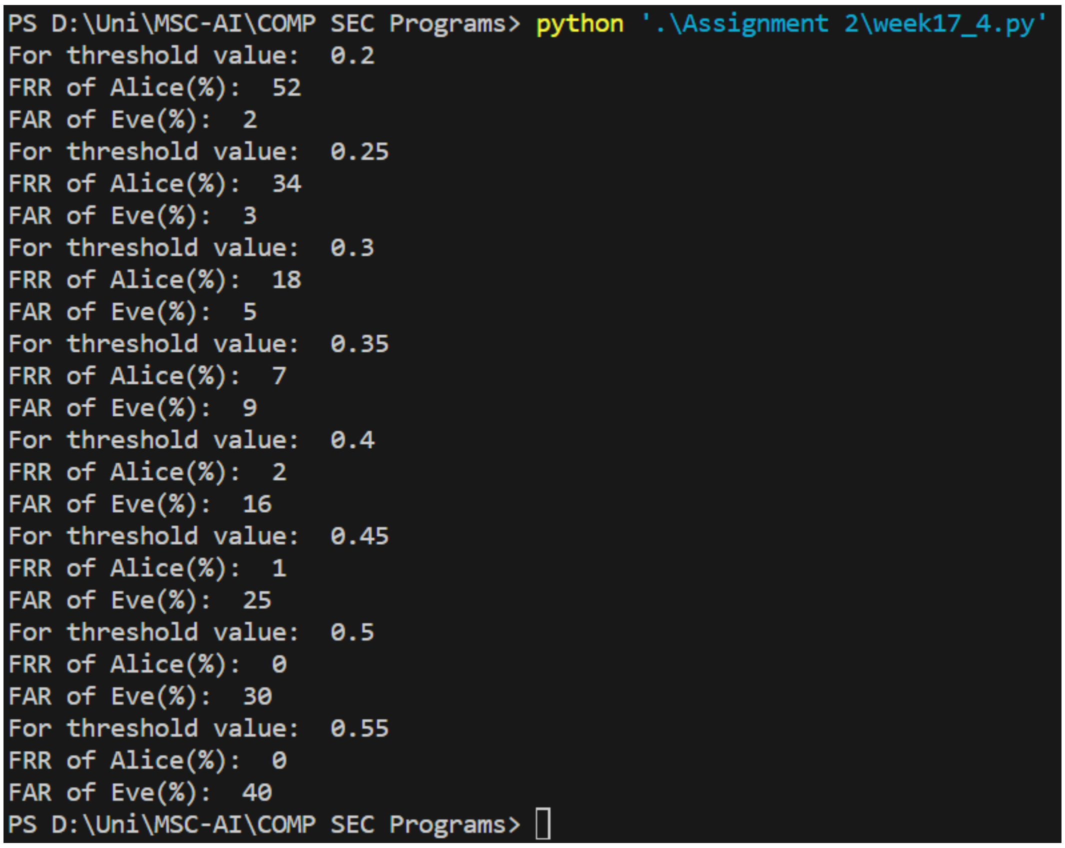


```
PS D:\Uni\MSC-AI\COMP SEC Programs> python '.\Assignment 2\week17_3.py'

1. Register
2. Login
Enter the option: 1
Enter username: asertia
Enter password(clear): zzAAqq123!
Enter profile information:
Enter name: asertia
Enter age: 24
Enter gender: f
Entering details into database....
Done!

1. Register
2. Login
Enter the option: 2
Enter login details:
Enter username: asd
Checking user from database....
User not found!

1. Register
2. Login
Enter the option: 2
Enter login details:
Enter username: asertia
Checking user from database....
OTP received is:  8f0415
Enter the OTP received: 8f1234
Wrong OTP!

1. Register
2. Login
Enter the option: 2
Enter login details:
Enter username: asertia
Checking user from database....
OTP received is:  0edcc6
Enter the OTP received: 0edcc6
User logged in successfully!

1. Register
2. Login
Enter the option:
PS D:\Uni\MSC-AI\COMP SEC Programs> 
```

Exercise 4



```python
alice_values = np.array([0.27, 0.079, 0.27, 0.36, 0.25, 0.3, 0.27, 0.17, 0.23, 0.12, 0.29, 0
eve_values = np.array([1.2, 0.77, 0.88, 0.39, 0.51, 0.55, 0.82, 0.54, 0.74, 0.19, 0.53, 0.44

low_th = np.mean(alice_values).round(2)
high_th = np.mean(eve_values).round(2)

for th in np.arange(low_th, high_th, 0.05):
    print("For threshold value: ", th.round(2))
    print("FRR of Alice(%): ", 100 - (alice_values < th).sum())
    print("FAR of Eve(%): ", (eve_values < th).sum())
```



```
PS D:\Uni\MSC-AI\COMP SEC Programs> python '.\Assignment 2\week17_4.py'
For threshold value:  0.2
FRR of Alice(%):  52
FAR of Eve(%):  2
For threshold value:  0.25
FRR of Alice(%):  34
FAR of Eve(%):  3
For threshold value:  0.3
FRR of Alice(%):  18
FAR of Eve(%):  5
For threshold value:  0.35
FRR of Alice(%):  7
FAR of Eve(%):  9
For threshold value:  0.4
FRR of Alice(%):  2
FAR of Eve(%):  16
For threshold value:  0.45
FRR of Alice(%):  1
FAR of Eve(%):  25
For threshold value:  0.5
FRR of Alice(%):  0
FAR of Eve(%):  30
For threshold value:  0.55
FRR of Alice(%):  0
FAR of Eve(%):  40
PS D:\Uni\MSC-AI\COMP SEC Programs>
```

Week 18 Class

Week 18 was about working on some exercises beyond user authentication to gain a better understanding of authentication in the wider sense.

The following exercises were attempted: -

- Message authentication with HMAC
- Needham-Schroeder protocol
- Attacking Needham-Schroeder protocol

How did I do the exercises and what did I learn about?

All of these exercises were attempted and completed through implementation of the same through Python 3.x.

Exercise 1 was done by implementing a system to generate 16-bit HMAC to authenticate between user and banking system. Also implemented a method for an attacker to manipulate the message between user and server depending on different values of money to be transferred and crack which for which value server accepts the message. Learnt a way of cracking authentication key as an attacker.

Exercise 2 was done by implementing a simulation of the Needham-Schroeder protocol between 2 users, Alice and Bob and a Server. Implementations were done to define a random key generator using SHA256 hashing, encryption and decryption of messages with these keys and steps to show how the protocol works. Learnt how 2 users do encryption and decryption of data and how the server generates a key for them to use to communicate with each other.

Exercise 3 was done by implementing a simulation of how someone can attack the Needham-Schroeder protocol, provided they recorded the K_AB and Message 3 from a previous session of the protocol. Implementations were done to define encryption and decryption of messages with given keys and steps to show how the protocol gets attacked. Learnt how an attacker can manipulate a user to communicate with them thinking it is another authorised user.

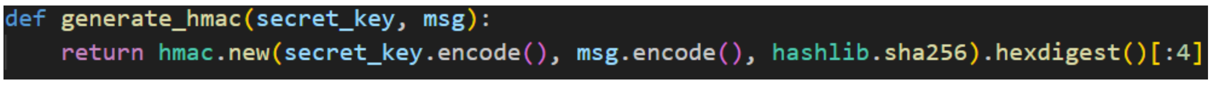
What difficulties / problems did I encounter? / What observations and/or thoughts did you have on the exercises?

Other than some syntax errors and the time-consuming efforts of observing and coding each and every step, did not face much challenges in these.

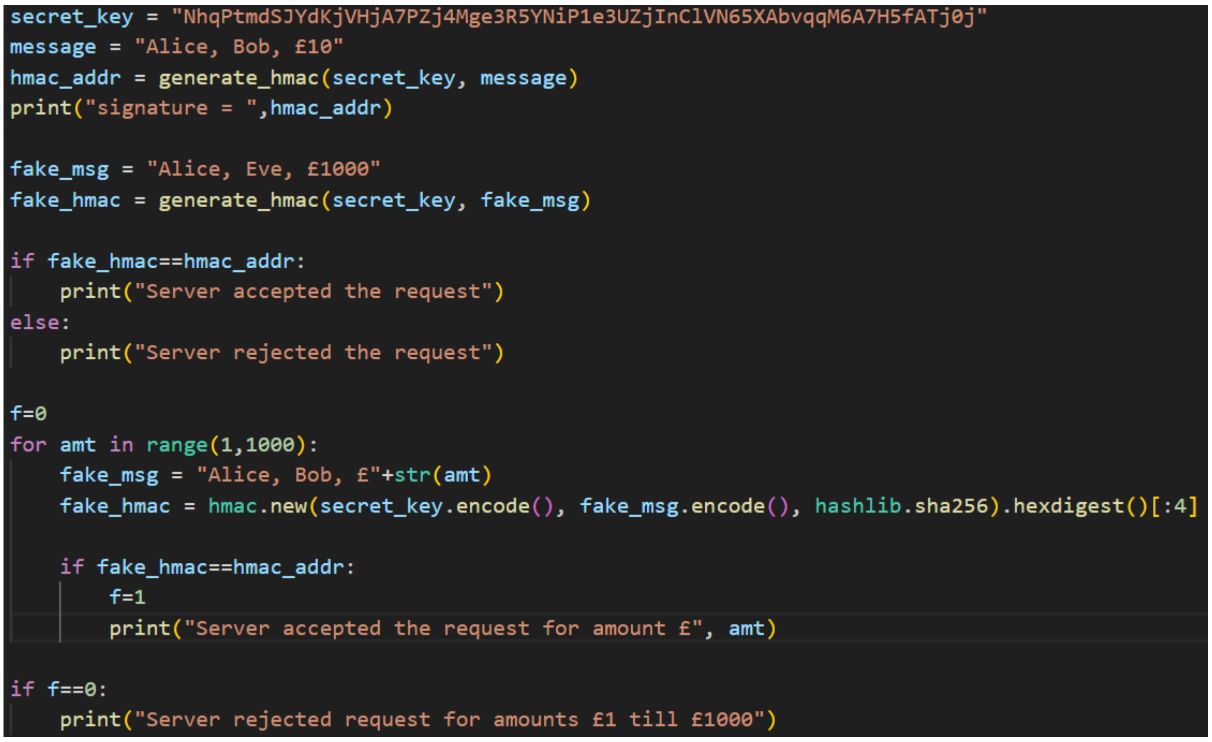
My thoughts were that these were interesting exercises to demonstrate HMAC message authentication and the working and limitations of Needham-Schroeder protocol.

Snippets of Code and Output

Exercise 1

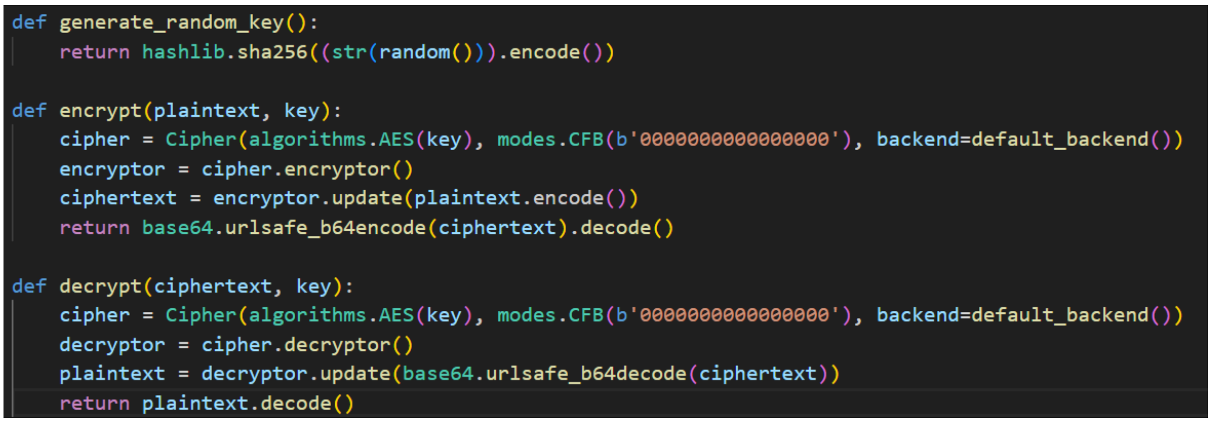


```
def generate_hmac(secret_key, msg):
    return hmac.new(secret_key.encode(), msg.encode(), hashlib.sha256).hexdigest()[:4]
```

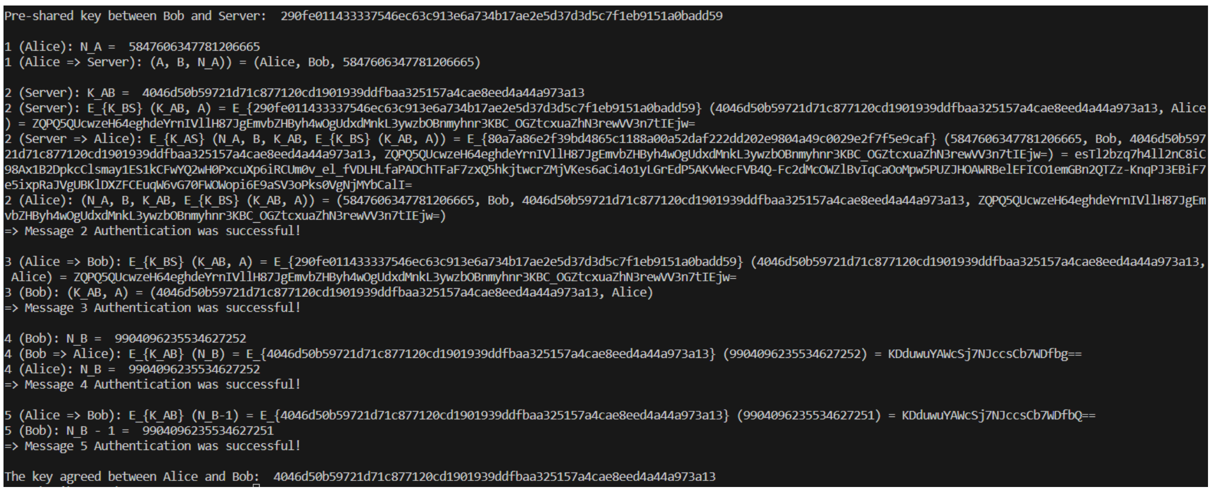


```
secret_key = "NhqPtmdSJYdKjVHjA7PZj4Mge3R5YNiP1e3UZjInClVN65XAbvqqM6A7H5fATj0j"
message = "Alice, Bob, £10"
hmac_addr = generate_hmac(secret_key, message)
print("signature = ",hmac_addr)

fake_msg = "Alice, Eve, £1000"
fake_hmac = generate_hmac(secret_key, fake_msg)

if fake_hmac==hmac_addr:
    print("Server accepted the request")
else:
    print("Server rejected the request")

f=0
for amt in range(1,1000):
    fake_msg = "Alice, Bob, £"+str(amt)
    fake_hmac = hmac.new(secret_key.encode(), fake_msg.encode(), hashlib.sha256).hexdigest()[:4]

    if fake_hmac==hmac_addr:
        f=1
        print("Server accepted the request for amount £", amt)

if f==0:
    print("Server rejected request for amounts £1 till £1000")
```

Exercise 2

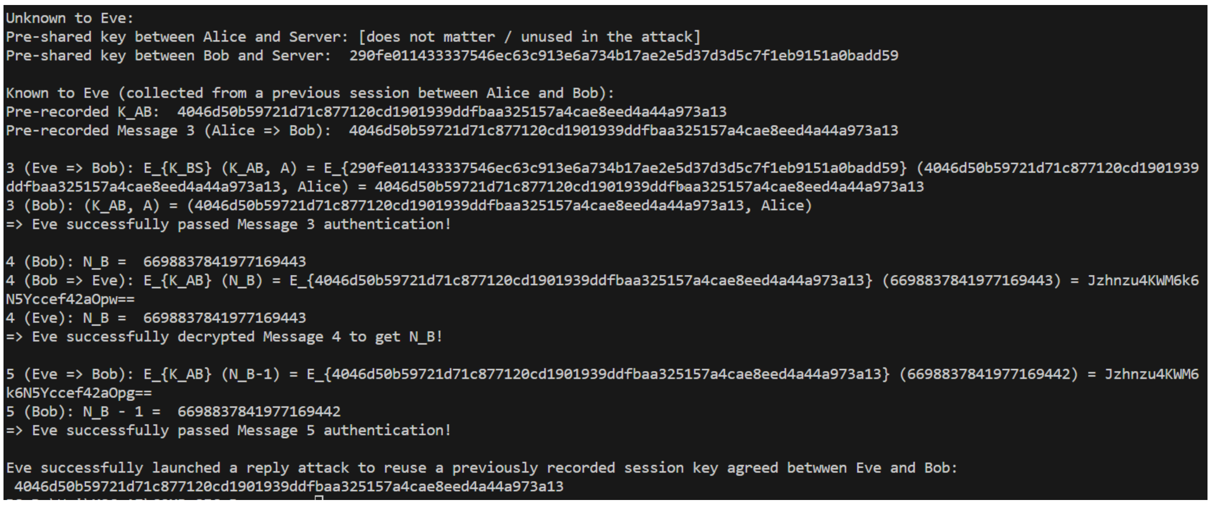


```
def generate_random_key():
    return hashlib.sha256((str(random())).encode())

def encrypt(plaintext, key):
    cipher = Cipher(algorithms.AES(key), modes.CFB(b'0000000000000000'), backend=default_backend())
    encryptor = cipher.encryptor()
    ciphertext = encryptor.update(plaintext.encode())
    return base64.urlsafe_b64encode(ciphertext).decode()

def decrypt(ciphertext, key):
    cipher = Cipher(algorithms.AES(key), modes.CFB(b'0000000000000000'), backend=default_backend())
    decryptor = cipher.decryptor()
    plaintext = decryptor.update(base64.urlsafe_b64decode(ciphertext))
    return plaintext.decode()
```



```
Pre-shared key between Bob and Server:  290fe011433337546ec63c913e6a734b17ae2e5d37d3d5c7f1eb9151a0badd59

1 (Alice): N_A =  5847606347781206665
1 (Alice => Server): (A, B, N_A)) = (Alice, Bob, 5847606347781206665)

2 (Server): K_AB =  4046d50b59721d71c877120cd1901939ddfbaa325157a4cae8eed4a44a973a13
2 (Server): E_{K_BS} (K_AB, A) = E_{290fe011433337546ec63c913e6a734b17ae2e5d37d3d5c7f1eb9151a0badd59} (4046d50b59721d71c877120cd1901939ddfbaa325157a4cae8eed4a44a973a13, Alice
) = ZQPQ5QUcwzeH64eghdeYrnIVllH87JgEmvbZHByh4wOgUdxdMnkL3ywzbOBnmyhnr3KBC_OGZtcxuaZhN3rewVV3n7tIEjw=
2 (Server => Alice): E_{K_AS} (N_A, B, K_AB, E_{K_BS} (K_AB, A)) = E_{80a7a86e2f39bd4865c1188a00a52daf222dd202e9804a49c0029e2f7f5e9caf} (5847606347781206665, Bob, 4046d50b597
21d71c877120cd1901939ddfbaa325157a4cae8eed4a44a973a13, ZQPQ5QUcwzeH64eghdeYrnIVllH87JgEmvbZHByh4wOgUdxdMnkL3ywzbOBnmyhnr3KBC_OGZtcxuaZhN3rewVV3n7tIEjw=) = esTl2bzq7h4ll2nC8iC
98Ax1B2DpkcClsmay1ES1kCFwYQ2wH0PxcuXp6iRCUm0v_el_fVDLHLfaPADChTFaF7zxQ5hkjtwcrZMjVKes6aCi4o1yLGrEdP5AKvWecFVB4Q-Fc2dMcOWZlBvIqCaOoMpw5PUZJHOAWRBelEFICO1emGBn2QTZz-KnqPJ3EBiF7
e5ixpRaJVgUBKlDXZFCEuqW6vG70FWOWopi6E9aSV3oPks0VgNjMYbCalI=
2 (Alice): (N_A, B, K_AB, E_{K_BS} (K_AB, A)) = (5847606347781206665, Bob, 4046d50b59721d71c877120cd1901939ddfbaa325157a4cae8eed4a44a973a13, ZQPQ5QUcwzeH64eghdeYrnIVllH87JgEm
vbZHByh4wOgUdxdMnkL3ywzbOBnmyhnr3KBC_OGZtcxuaZhN3rewVV3n7tIEjw=)
=> Message 2 Authentication was successful!

3 (Alice => Bob): E_{K_BS} (K_AB, A) = E_{290fe011433337546ec63c913e6a734b17ae2e5d37d3d5c7f1eb9151a0badd59} (4046d50b59721d71c877120cd1901939ddfbaa325157a4cae8eed4a44a973a13,
 Alice) = ZQPQ5QUcwzeH64eghdeYrnIVllH87JgEmvbZHByh4wOgUdxdMnkL3ywzbOBnmyhnr3KBC_OGZtcxuaZhN3rewVV3n7tIEjw=
3 (Bob): (K_AB, A) = (4046d50b59721d71c877120cd1901939ddfbaa325157a4cae8eed4a44a973a13, Alice)
=> Message 3 Authentication was successful!

4 (Bob): N_B =  9904096235534627252
4 (Bob => Alice): E_{K_AB} (N_B) = E_{4046d50b59721d71c877120cd1901939ddfbaa325157a4cae8eed4a44a973a13} (9904096235534627252) = KDduwuYAWcSj7NJccsCb7WDfbg==
4 (Alice): N_B =  9904096235534627252
=> Message 4 Authentication was successful!

5 (Alice => Bob): E_{K_AB} (N_B-1) = E_{4046d50b59721d71c877120cd1901939ddfbaa325157a4cae8eed4a44a973a13} (9904096235534627251) = KDduwuYAWcSj7NJccsCb7WDfbQ==
5 (Bob): N_B - 1 =  9904096235534627251
=> Message 5 Authentication was successful!

The key agreed between Alice and Bob:  4046d50b59721d71c877120cd1901939ddfbaa325157a4cae8eed4a44a973a13
```

Exercise 3



```
Unknown to Eve:
Pre-shared key between Alice and Server: [does not matter / unused in the attack]
Pre-shared key between Bob and Server:  290fe011433337546ec63c913e6a734b17ae2e5d37d3d5c7f1eb9151a0badd59

Known to Eve (collected from a previous session between Alice and Bob):
Pre-recorded K_AB:  4046d50b59721d71c877120cd1901939ddfbaa325157a4cae8eed4a44a973a13
Pre-recorded Message 3 (Alice => Bob):  4046d50b59721d71c877120cd1901939ddfbaa325157a4cae8eed4a44a973a13

3 (Eve => Bob): E_{K_BS} (K_AB, A) = E_{290fe011433337546ec63c913e6a734b17ae2e5d37d3d5c7f1eb9151a0badd59} (4046d50b59721d71c877120cd1901939
ddfbaa325157a4cae8eed4a44a973a13, Alice) = 4046d50b59721d71c877120cd1901939ddfbaa325157a4cae8eed4a44a973a13
3 (Bob): (K_AB, A) = (4046d50b59721d71c877120cd1901939ddfbaa325157a4cae8eed4a44a973a13, Alice)
=> Eve successfully passed Message 3 authentication!

4 (Bob): N_B =  6698837841977169443
4 (Bob => Eve): E_{K_AB} (N_B) = E_{4046d50b59721d71c877120cd1901939ddfbaa325157a4cae8eed4a44a973a13} (6698837841977169443) = Jzhnzu4KWM6k6
N5Yccef42aOpw==
4 (Eve): N_B =  6698837841977169443
=> Eve successfully decrypted Message 4 to get N_B!

5 (Eve => Bob): E_{K_AB} (N_B-1) = E_{4046d50b59721d71c877120cd1901939ddfbaa325157a4cae8eed4a44a973a13} (6698837841977169442) = Jzhnzu4KWM6
k6N5Yccef42aOpg==
5 (Bob): N_B - 1 =  6698837841977169442
=> Eve successfully passed Message 5 authentication!

Eve successfully launched a reply attack to reuse a previously recorded session key agreed betwwen Eve and Bob:
 4046d50b59721d71c877120cd1901939ddfbaa325157a4cae8eed4a44a973a13
```

Week 19 Class

Week 18 was about work on some exercises beyond user authentication to gain a better understanding of authentication in the wider sense.

The following exercises were attempted: -

- Setting Linux file and folder permissions
- Investigating a real-world example of role-based access control
- Learning about same-origin policies (SOP) and cross-site scripting (XSS) attacks
- Playing with a sandbox
- Examining a real-world federated identity management (FIM) system

How did I do the exercises and what did I learn about?

Exercise 1 was attempted and completed through implementation of the same through Python 3.x. Exercises 2,3,4 and 5 were attempted and completed through research on windows system application and other applications mentioned to use.

Exercise 1 was done by implementing a system that converts Linux based file permission code into its octal representation. It understands if the size of the code is longer/shorter than the actual format and if it uses wrong characters or wrong order in the format.

Exercise 2 was done by exploring various kinds of RBAC in Windows system such as Windows User Management System, User Account Settings, Users in Task Manager and Device Security. Learnt the different kinds of roles and their permissions.

Exercise 3 was done by playing a game of how in real life XSS attacks can be done to affect the SOPs of a target website and observing various factors. Learnt different ways of attacking by cross site scripting and the vulnerabilities of SOPs.

Exercise 4 was done by investigating a Docker Container Sandbox, examining the security settings and restrictions on access control. Learnt more about the importance of security settings and access control and how to enforce the same.

Exercise 5 was done by examining the implementation of an FIM system called Tripwire. Learnt about the configuration, monitoring, hashing, architecture, etc. of the same.

What difficulties / problems did I encounter? / What observations and/or thoughts did you have on the exercises?

Other than some syntax errors did not face much challenges in Exercise 1. Exercises 2-5 was able to attempt but there were some limitations within the system while doing research.

For Exercise 1, my thoughts were on how well Linux permission codes need to be generated without faults and on converting to octal representation.

For Exercise 2, in family and other user accounts, found these types of roles and their features: -

1. Standard User:
   - These users have read and execute permissions for most files and folders usually. Inside their profile folders they can modify, delete and create files withput much issues.
   - Usually restricted from critical system settings and should be able to customize personal preferences.
   - Don’t have permission to install or uninstall system level applications.

2. Administrator:
   - Full control over file system. (read, write, modify, execute and delete)
   - Can install drivers, configure settings and modify system configuration.
   - Can do system wide software installations/uninstallations.

3. Organiser:
   - Role depends on implementation of system.
   - Permissions defined by the system

In Device Security some of the key security configurations are: -

- Virus and Threat Protection
- App and Browser Control
- Device Performance and Health
- Firewall and Network Protection
- Windows Security Dashboard

For Exercise 3, here are some of the ways by which XSS can attack SOPs: -

- Bypassing Restrictions:
    - The injected script has the same level of access to functionalities and resources as authorised scripts hence can bypass restrictions and the attacker can communicate with same origin resources.
- Injecting Scripts:
    - Attacker can input malicious scripts into form fields, URLs and even content of the webpage.
- Accessing sensitive information:
    - Can steal cookies, credential information and other user related information stored.
- Executing in Victim's Browser:
    - Browser will unknowingly execute the injected scripts when a user opens the web page.
- Actions on user's behalf
    - The attacker's scripts can make requests on the web application through the victim's credentials, modify user settings and even initiate financial transactions.

For Exercise 4, Docker containers are a kind of Sandbox, as they allow applications to run inside an isolated environment.

Security Settings:

- Namespace Isolation: Each docker container has its own namespaces to prevent conflict between containers.
- Cgroups: Limits and controls resource utilization by containers.
- Read-Only File system: Restricts applications inside the containers to modify system files.
- Seccomp Profiles: Restrict system calls a container can make, making it less vulnerable to attacks.

Enforcement:

- Container runtimes enforce security settings
- Authorised docker images that contain security settings are enforced by registries.
- Docker Daemon enforces security policies to limit which users can interact with it.

Docker has faced vulnerabilities before and security patches are done regularly to resolve the same. It is advised to keep software up to date, use official docker images and reducing the vulnerable areas.

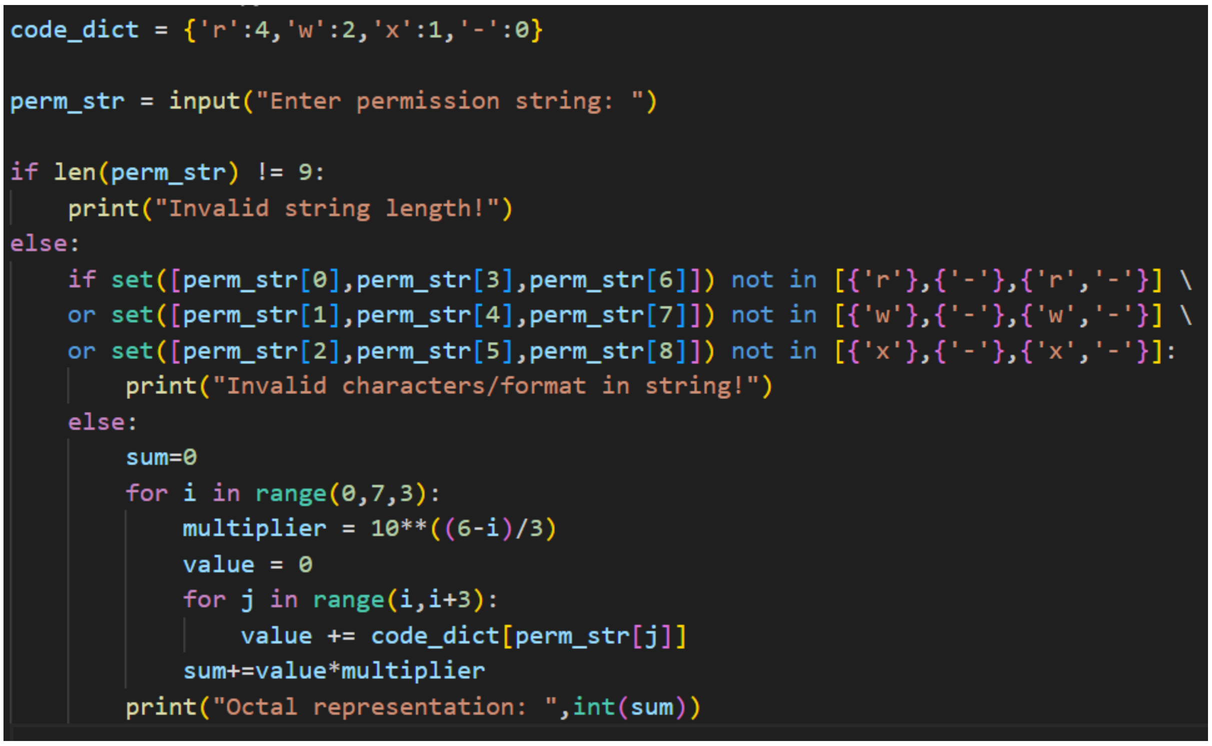
For Exercise 5, here is an overview of the implementation an FIM called Tripwire: -

- An agent-based architecture where agents are kept at end points for file monitoring and system configurations.
- Monitoring is done continuously by the agents to detect any changes in

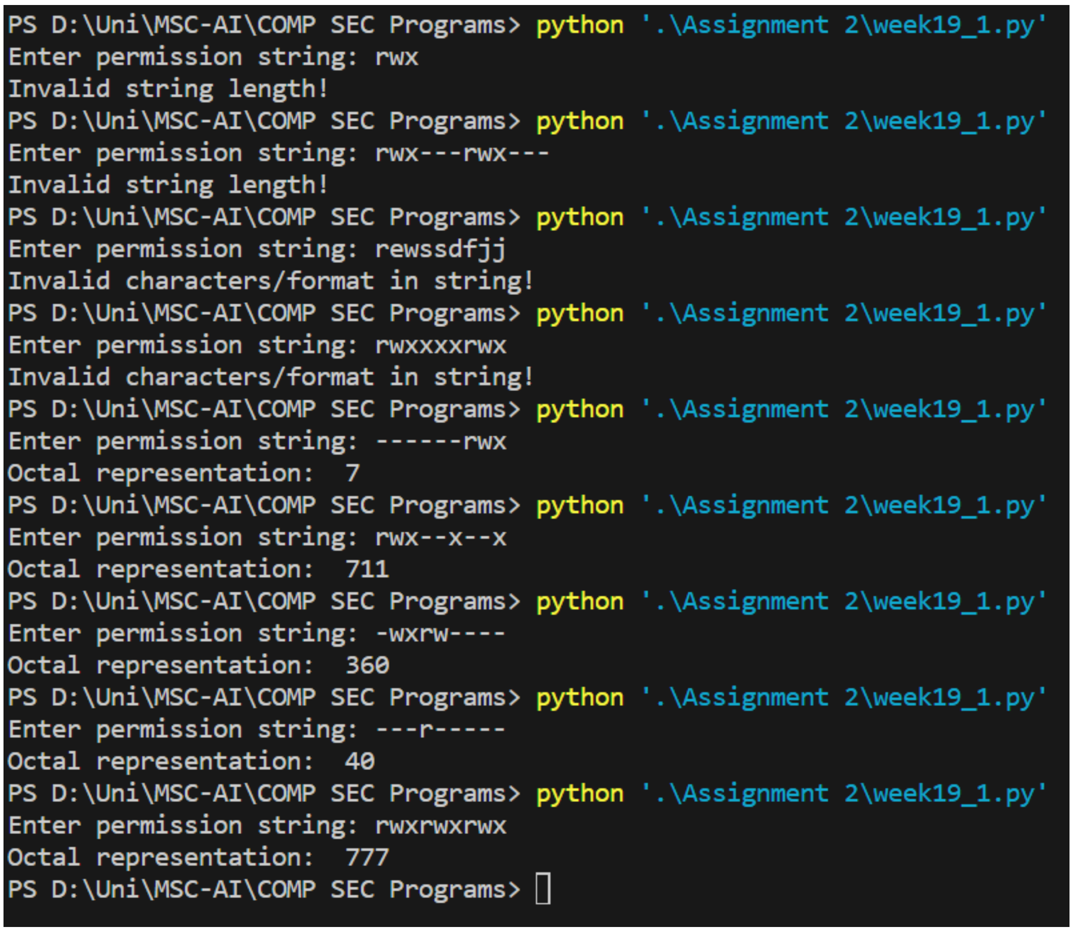
files, directories and system configurations such as file addition, deletion or modification.

- Tripwire uses SHA-256 hashing to generate checksums or hashes for files which are compared against baseline to detect alterations.
- Administrators can define policies on which files to monitor, which can have modify permissions and what actions to take upon unauthorized updates.
- Keeps an initial snapshot of the monitored files and configurations which is then used as reference baseline to detect changes.
- Generates alerts upon unauthorized operations which can be triggered as notifications to email, SNMP, etc. Also generates reports on file integrity and system changes.
- Meets regulatory compliance requirements, helps with monitoring and documenting changes to critical files and configurations.

Snippets of Code and Output of Exercise 1



```python
code_dict = {'r':4,'w':2,'x':1,'-':0}

perm_str = input("Enter permission string: ")

if len(perm_str) != 9:
    print("Invalid string length!")
else:
    if set([perm_str[0],perm_str[3],perm_str[6]]) not in [{'r'},{'-'},{'r','-'}] \
    or set([perm_str[1],perm_str[4],perm_str[7]]) not in [{'w'},{'-'},{'w','-'}] \
    or set([perm_str[2],perm_str[5],perm_str[8]]) not in [{'x'},{'-'},{'x','-'}]:
        print("Invalid characters/format in string!")
    else:
        sum=0
        for i in range(0,7,3):
            multiplier = 10**((6-i)/3)
            value = 0
            for j in range(i,i+3):
                value += code_dict[perm_str[j]]
            sum+=value*multiplier
        print("Octal representation: ",int(sum))
```



```
PS D:\Uni\MSC-AI\COMP SEC Programs> python '.\Assignment 2\week19_1.py'
Enter permission string: rwx
Invalid string length!
PS D:\Uni\MSC-AI\COMP SEC Programs> python '.\Assignment 2\week19_1.py'
Enter permission string: rwx---rwx---
Invalid string length!
PS D:\Uni\MSC-AI\COMP SEC Programs> python '.\Assignment 2\week19_1.py'
Enter permission string: rewssdfjj
Invalid characters/format in string!
PS D:\Uni\MSC-AI\COMP SEC Programs> python '.\Assignment 2\week19_1.py'
Enter permission string: rwxxxxrwx
Invalid characters/format in string!
PS D:\Uni\MSC-AI\COMP SEC Programs> python '.\Assignment 2\week19_1.py'
Enter permission string: ------rwx
Octal representation:  7
PS D:\Uni\MSC-AI\COMP SEC Programs> python '.\Assignment 2\week19_1.py'
Enter permission string: rwx--x--x
Octal representation:  711
PS D:\Uni\MSC-AI\COMP SEC Programs> python '.\Assignment 2\week19_1.py'
Enter permission string: -wxrw----
Octal representation:  360
PS D:\Uni\MSC-AI\COMP SEC Programs> python '.\Assignment 2\week19_1.py'
Enter permission string: ---r-----
Octal representation:  40
PS D:\Uni\MSC-AI\COMP SEC Programs> python '.\Assignment 2\week19_1.py'
Enter permission string: rwxrwxrwx
Octal representation:  777
PS D:\Uni\MSC-AI\COMP SEC Programs> 
```