DATA PRIVACY
PRACTICAL FILE

SUBMITTED BY :TAMANNA AHUJA
ROLL NO. : 16119
CLASS : B.Sc. (Hons.) Computer Science
SEC : B
SEM : Vth
SUBMITTED TO : Ms. Nidhi Passi

INDEX

QUESTION	PAGE NO.
1. Write a program to perform encryption and decryption using Caesar cipher (substitutional cipher).	3
2. Write a program to perform encryption and decryption using Rail Fence Cipher (transpositional cipher)	5
3. Write a Python program that defines a function and takes a password string as input and returns its SHA-256 hashed representation as a hexadecimal string.	8
4. Write a Python program that reads a file containing a list of usernames and passwords, one pair per line (separated by a comma). It checks each password to see if it has been leaked in a data breach. You can use the "Have I Been Pwned" API (https://haveibeenpwned.com/API/v3) to check if a password has been leaked.	9
5. Write a Python program that generates a password using a random combination of words from a dictionary file.	10
6. Write a Python program that simulates a brute-force attack on a password by trying out all possible character combinations.	11
7. Demonstrate the usage/sending of a digitally signed document.	14

Ques 1. Write a program to perform encryption and decryption using Caesar cipher (substitutional cipher).

Input and Output :

In [1]:
```
def encrypt(text, s):
    result = ""
    for i in range(len(text)):
        char = text[i]
        if char.isupper():
            result += chr((ord(char) + s - 65) % 26 + 65)
        else :
            result += chr((ord(char) + s - 97) % 26 + 97)
    return result

def decrypt(text, s):
    result = ""
    for i in range(len(text)):
        char = text[i]
        if char.isupper():
            result += chr((ord(char) - s - 65) % 26 + 65)
        else :
            result += chr((ord(char) - s - 97) % 26 + 97)
    return result

def menu():
    while True:
        print("\nMenu:")
        print("1. Encrypt")
        print("2. Decrypt")
        print("3. Exit")

        choice = input("Enter choice (1/2/3): ")

        if choice == '1':
            text = input("Enter text to encrypt: ")
            s = int(input("Enter shift value: "))
            print("Encrypted text:", encrypt(text, s))
        elif choice == '2':
            text = input("Enter text to decrypt: ")
            s = int(input("Enter shift value: "))
            print("Decrypted text:", decrypt(text, s))
        elif choice == '3':
            print("Exiting...")
```

```
            print("Exiting...")
            break
        else:
            print("Invalid choice, please try again.")

if __name__ == "__main__":
    menu()
```

```
Menu:
1. Encrypt
2. Decrypt
3. Exit
Enter choice (1/2/3): 1
Enter text to encrypt: dataprivacy
Enter shift value: 3
Encrypted text: gdwdsulydfb

Menu:
1. Encrypt
2. Decrypt
3. Exit
Enter choice (1/2/3): 2
Enter text to decrypt: gdwdsulydfb
Enter shift value: 3
Decrypted text: dataprivacy

Menu:
1. Encrypt
2. Decrypt
3. Exit
Enter choice (1/2/3): 3
Exiting...
```

Ques 2. Write a program to perform encryption and decryption using Rail Fence Cipher (transpositional cipher).

Input and Output :

```
def encryptRailFence(message, levels):
    fence = [['\n' for _ in range(len(message))] for _ in range(levels)]
    going_down = False
    r, c = 0, 0

    for char in message:
        if r == 0 or r == levels - 1:
            going_down = not going_down
        fence[r][c] = char
        c += 1
        r += 1 if going_down else -1

    encrypted_text = []
    for i in range(levels):
        for j in range(len(message)):
            if fence[i][j] != '\n':
                encrypted_text.append(fence[i][j])
    return "".join(encrypted_text)

def decryptRailFence(encrypted_message, levels):
    fence = [['\n' for _ in range(len(encrypted_message))] for _ in range(levels)]
    going_down = None
    r, c = 0, 0

    for char in encrypted_message:
        if r == 0:
            going_down = True
        if r == levels - 1:
            going_down = False
        fence[r][c] = '*'
        c += 1
        r += 1 if going_down else -1

    index = 0
    for i in range(levels):
        for j in range(len(encrypted_message)):
            if fence[i][j] == '*' and index < len(encrypted_message):
                fence[i][j] = encrypted_message[index]
                index += 1
```

```
    decrypted_text = []
    r, c = 0, 0
    for char in encrypted_message:
        if r == 0:
            going_down = True
        if r == levels - 1:
            going_down = False
        if fence[r][c] != '*':
            decrypted_text.append(fence[r][c])
            c += 1
        r += 1 if going_down else -1
    return "".join(decrypted_text)

def menu():
    while True:
        print("\nRail Fence Cipher")
        print("1. Encrypt")
        print("2. Decrypt")
        print("3. Exit")
        choice = input("Enter your choice: ")

        if choice == '1':
            message = input("Enter the message to encrypt: ")
            levels = int(input("Enter the number of levels (key): "))
            encrypted_message = encryptRailFence(message, levels)
            print("Encrypted Message:", encrypted_message)

        elif choice == '2':
            encrypted_message = input("Enter the message to decrypt: ")
            levels = int(input("Enter the number of levels (key): "))
            decrypted_message = decryptRailFence(encrypted_message, levels)
            print("Decrypted Message:", decrypted_message)

        elif choice == '3':
            print("Exiting the program.")
            break

        else:
```

```
        elif choice == '3':
            print("Exiting the program.")
            break

        else:
            print("Invalid choice, please try again.")

if __name__ == "__main__":
    menu()
```

```
Rail Fence Cipher
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 1
Enter the message to encrypt: thankyouverymuch
Enter the number of levels (key): 3
Encrypted Message: tkvmhnyueyuhaorc

Rail Fence Cipher
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 2
Enter the message to decrypt: tkvmhnyueyuhaorc
Enter the number of levels (key): 3
Decrypted Message: thankyouverymuch

Rail Fence Cipher
1. Encrypt
2. Decrypt
3. Exit
Enter your choice: 3
Exiting the program.
```

Ques 3. Write a Python program that defines a function and takes a password string as input and returns its SHA-256 hashed representation as a hexadecimal string.

Input and Output :

```
import hashlib

def hash_password(password):
    sha256_hash = hashlib.sha256(password.encode('utf-8')).hexdigest()
    return sha256_hash

password = input("Enter the password: ")
hashed_password = hash_password(password)
print("SHA-256 Hashed Password:", hashed_password)
```

```
Enter the password: dataprivacy
SHA-256 Hashed Password: eabfbe2eedd1bb0a2b8b114146a1ec5cc4673318e8b644586084e29dea2964dc
```

Ques 4. Write a Python program that reads a file containing a list of usernames and passwords, one pair per line (separated by a comma). It checks each password to see if it has been leaked in a data breach. You can use the "Have I Been Pwned" API (https://haveibeenpwned.com/API/v3) to check if a password has been leaked.

Input and Output :

```
import hashlib
import requests

def get_sha1_hash(password):
    return hashlib.sha1(password.encode('utf-8')).hexdigest().upper()

def check_pwned_password(username, password):
    sha1_password = get_sha1_hash(password)
    prefix, suffix = sha1_password[:5], sha1_password[5:]
    url = f"https://api.pwnedpasswords.com/range/{prefix}"
    response = requests.get(url)

    if response.status_code != 200:
        raise RuntimeError(f"Error fetching data: {response.status_code}")

    hashes = (line.split(':') for line in response.text.splitlines())

    for hash_suffix, count in hashes:
        if hash_suffix == suffix:
            return f"Password for '{username}' found {count} times in breaches!"

    return f"Password for '{username}' not found in any breaches."

def check_credentials_from_file(filename):
    try:
        with open(filename, 'r') as file:
            credentials = file.readlines()
    except FileNotFoundError:
        print(f"File {filename} not found.")
        return

    for line in credentials:
        line = line.strip()
        if line:
            username, password = line.split(":")
            result = check_pwned_password(username, password)
            print(result)
```

```
filename = "credentials.txt"
check_credentials_from_file(filename)
```

```
Password for 'Ben' found 233673 times in breaches!
Password for 'Helen' found 294857 times in breaches!
Password for 'Kim' not found in any breaches.
```

Ques 5. Write a Python program that generates a password using a random combination of words from a dictionary file.

Input and Output :

```
import random
import string

def generate_password(num_letters):
    try:
        with open('dictionary.txt', 'r') as file:
            words = [line.strip() for line in file.readlines() if line.strip()]
    except FileNotFoundError:
        print("Dictionary file not found!")
        return None

    password = ''

    while len(password) < num_letters:
        word = random.choice(words)
        if len(password) + len(word) <= num_letters:
            password += word

    password = password[:num_letters]

    return password

def main():
    while True:
        try:
            num_letters = int(input("Enter the number of letters for your password (max 15): "))
            if num_letters <= 15:
                break
            else:
                print("Password length must not exceed 15 characters. Try again.")
        except ValueError:
            print("Please enter a valid number.")

    password = generate_password(num_letters)
    if password:
        print(f"Generated password: {password}")

if __name__ == '__main__':
    main()
```

```
Enter the number of letters for your password (max 15): 10
Generated password: 3B@qdNqzcC
```

Ques 6. Write a Python program that simulates a brute-force attack on a password by trying out all possible character combinations.

Input and Output :

```
import itertools
import string
import time

def brute_force_attack(target_password):
    characters = string.ascii_lowercase
    password_length = 1

    while True:
        for guess in itertools.product(characters, repeat=password_length):
            guess = ''.join(guess)
            print(f"Trying: {guess}")

            if guess == target_password:
                return guess

        password_length += 1

target_password = "abc"

start_time = time.time()

found_password = brute_force_attack(target_password)

end_time = time.time()

print(f"Password '{found_password}' found in {end_time - start_time:.2f} seconds.")
```

```
Trying: a
Trying: b
Trying: c
Trying: d
Trying: e
Trying: f
Trying: g
Trying: h
Trying: i
Trying: j
Trying: k
Trying: l
Trying: m
Trying: n
Trying: o
Trying: p
Trying: q
Trying: r
Trying: s
Trying: t
Trying: u
Trying: v
Trying: w
Trying: x
Trying: y
Trying: z
Trying: aa
Trying: ab
Trying: ac
Trying: ad
Trying: ae
Trying: af
Trying: ag
Trying: ah
Trying: ai
```

```
Trying: vh
Trying: vi
Trying: vj
Trying: vk
Trying: vl
Trying: vm
Trying: vn
Trying: vo
Trying: vp
Trying: vq
Trying: vr
Trying: vs
Trying: vt
Trying: vu
Trying: vv
Trying: vw
Trying: vx
Trying: vy
Trying: vz
Trying: wa
Trying: wb
Trying: wc
Trying: wd
Trying: we
Trying: wf
Trying: wg
Trying: wh
Trying: wi
Trying: wj
Trying: wk
Trying: wl
Trying: wm
Trying: wn
Trying: wo
Trying: wp
Trying: wq
Trying: wr
Trying: ws
Trying: wt
```

```
Trying: zs
Trying: zt
Trying: zu
Trying: zv
Trying: zw
Trying: zx
Trying: zy
Trying: zz
Trying: aaa
Trying: aab
Trying: aac
Trying: aad
Trying: aae
Trying: aaf
Trying: aag
Trying: aah
Trying: aai
Trying: aaj
Trying: aak
Trying: aal
Trying: aam
Trying: aan
Trying: aao
Trying: aap
Trying: aaq
Trying: aar
Trying: aas
Trying: aat
Trying: aau
Trying: aav
Trying: aaw
Trying: aax
Trying: aay
Trying: aaz
Trying: aba
Trying: abb
Trying: abc
Password 'abc' found in 3.03 seconds.
```

Ques 7. Demonstrate the usage/sending of a digitally signed document.

Input and Output :

```
import hashlib
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import hashes

document_text = input("Enter the document text: ")
document_hash = hashlib.sha256(document_text.encode()).hexdigest()
print("Document Hash:", document_hash)

private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048,
)

signature = private_key.sign(
    document_text.encode(),
    padding.PSS(
        mgf=padding.MGF1(hashes.SHA256()),
        salt_length=padding.PSS.MAX_LENGTH
    ),
    hashes.SHA256()
)
print("Digital Signature:", signature)

public_key = private_key.public_key()

try:
    public_key.verify(
        signature,
        document_text.encode(),
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH
        ),
        hashes.SHA256()
    )
    print("The signature is valid. The document is authentic.")
except Exception as e:
    print("Signature verification failed:", e)
```

```
========== RESTART: D:/My Documents/Desktop/Data_Privacy/q7.py ==========
Enter the document text: kkkakbb
Document Hash: 107536d4c2af451311759043a6ef4fb0013396d1162c2d9e34806b9a01a73801
Digital Signature: b'e=\xd7\x1f\x05r\xb3j\xc9_;T\xe9\xe5f\x90\xcc\xb6\xb2>\xdc\x
8b\x06\x17?\x07\xdfrFH\xea\xc8\x00a\x94\xa5\xf8$Y<\xea\x7f\xf5\xc8\xdc Z;\x03\x9
9\x15=\xf8\x13E\x8b\x9f8\x14h\x1c\xad)\xbas\xa1U\x82\x12\x1d\x12\x00E\xdf\xd2_\x
9b\x91\xa9\xa6\xc9&\xc8\x07\x13z\xef_\x90\x95\xeb\x85kj\x0fX\x90H\x1ee\xd8\xbc\x
0c\xafc\x9a\xb8\xc1\xb7\x9b\xe4P\xeb\xbc&sx]E\xf1\xde+.\xfe\x06CeS\xbd\xa1\x0c\\
\xe7\x98\xce.\x86\x14\xe1$\x10\xce\xdd\xc2\x95m\x08\xc1\xecT\x91\xa1E\xc3\xd2\xf
a\t\t\xefs\x07(Mnc\x1f\x83\x1e9\xb7\x02\t/v\xdeA\x02Fvi\xdbR0\xac\x06\xb2E~k\x82
\x18\xc5\xbb}H/h\xa5\x96:\x0c\xdd-\x87\x7fH\xa9\xceh\x8d\xdf\x9d\xb0\x94\xaa!\xb
2W\x184\x12E6\xc0\xbbV\xd2\xc61\x96j+\x1f,\xf7\x03\xd6\xd4\xbb\x84\nhs\x1b%\x12A
(\xd3\xe9\xf3\x16T\xbcV\xd3'
The signature is valid. The document is authentic.
```