**DATA PRIVACY PRACTICALS**

**Name - Vidya Vipul**

**College Roll No. - 4026**

**Q1. Write a program to perform encryption and decryption using Caesar cipher**

**(substitutional cipher).**

**CODE:**

**def caesar\_cipher(text, shift, mode):**

**result = ''**

**for char in text:**

**if char.isalpha():**

**start = ord('a') if char.islower() else ord('A')**

**shifted\_char = chr((ord(char) - start + shift) % 26 + start)**

**elif char.isdigit():**

**shifted\_char = str((int(char) + shift) % 10)**

**else:**

**shifted\_char = char**

**result += shifted\_char**

**return result**

**# Example usage**

**text = "DATA PRIVACY 000"**

**key = 3**

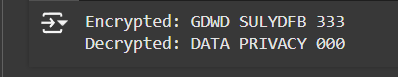
**encrypted\_text = caesar\_cipher(text, key, 'encrypt')**

**print("Encrypted:", encrypted\_text)**

**decrypted\_text = caesar\_cipher(encrypted\_text, -key, 'decrypt')**

**print("Decrypted:", decrypted\_text)**

**OUTPUT :**

****

**Q2. Write a program to perform encryption and decryption using Rail Fence Cipher**

**(transpositional cipher)**

**CODE:**

**def rail\_fence\_encrypt(plaintext, rails):**

**"""Encrypts the plaintext using the Rail Fence Cipher."""**

**# Create an empty list of rails**

**fence = [['' for \_ in range(len(plaintext))] for \_ in range(rails)]**

**rail = 0**

**direction = 1 # 1 for moving down, -1 for moving up**

**# Place characters in the zigzag pattern**

**for i, char in enumerate(plaintext):**

**fence[rail][i] = char**

**rail += direction**

**if rail == 0 or rail == rails - 1:**

**direction \*= -1**

**# Read row by row to get the ciphertext**

**ciphertext = ''.join(''.join(row) for row in fence if ''.join(row))**

**return ciphertext**

**def rail\_fence\_decrypt(ciphertext, rails):**

**"""Decrypts the ciphertext using the Rail Fence Cipher."""**

**# Create an empty list of rails**

**fence = [['' for \_ in range(len(ciphertext))] for \_ in range(rails)]**

**rail = 0**

**direction = 1 # 1 for moving down, -1 for moving up**

**# Mark the zigzag pattern**

**for i in range(len(ciphertext)):**

**fence[rail][i] = '\*'**

**rail += direction**

**if rail == 0 or rail == rails - 1:**

**direction \*= -1**

**# Fill the marked pattern with the ciphertext**

**index = 0**

**for r in range(rails):**

**for c in range(len(ciphertext)):**

**if fence[r][c] == '\*' and index < len(ciphertext):**

**fence[r][c] = ciphertext[index]**

**index += 1**

**# Read the zigzag pattern to reconstruct plaintext**

**rail = 0**

**direction = 1**

**plaintext = []**

**for i in range(len(ciphertext)):**

**plaintext.append(fence[rail][i])**

**rail += direction**

**if rail == 0 or rail == rails - 1:**

**direction \*= -1**

**return ''.join(plaintext)**

**# Example usage**

**if \_\_name\_\_ == "\_\_main\_\_":**

**plaintext = "HELLORAILFENCE"**

**rails = 3**

**print("Original Text:", plaintext)**

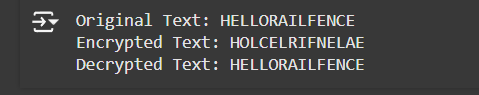
**ciphertext = rail\_fence\_encrypt(plaintext, rails)**

**print("Encrypted Text:", ciphertext)**

**decrypted\_text = rail\_fence\_decrypt(ciphertext, rails)**

**print("Decrypted Text:", decrypted\_text)**

**OUTPUT -**

****

**Q3.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.**

**CODE:**

**import hashlib**

**def hash\_password(password):**

**"""**

**Hashes a password using SHA-256 and returns the hexadecimal representation.**

**Args:**

**password (str): The password string to be hashed.**

**Returns:**

**str: The SHA-256 hash of the password in hexadecimal format.**

**"""**

**# Ensure the password is a string and encode it to bytes**

**password\_bytes = password.encode('utf-8')**

**# Create a SHA-256 hash object**

**sha256\_hash = hashlib.sha256()**

**# Update the hash object with the password bytes**

**sha256\_hash.update(password\_bytes)**

**# Return the hexadecimal digest**

**return sha256\_hash.hexdigest()**

**# Example usage**

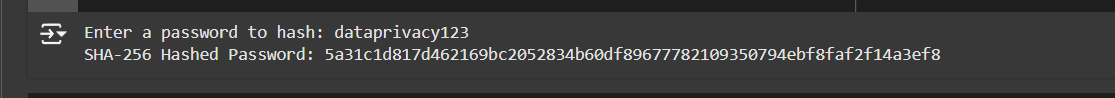
**if \_\_name\_\_ == "\_\_main\_\_":**

**password = input("Enter a password to hash: ")**

**hashed\_password = hash\_password(password)**

**print("SHA-256 Hashed Password:", hashed\_password)**

**OUTPUT -**

****

**Q4. 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.**

**CODE:**

# Opening file and writing usernames and passwords in it. with open('info.txt', 'w') as f:

data = [('user1', 'password'), ('user2', 'pass1234'), ('user3', 'djfhdgaohfoanf')]

for username, password in data:

f.write(f"{username},{password}\n")

with open('info.txt', 'r') as f:

print(f.read())

# Importing necessary libraries

import hashlib

import requests

# Opening file for reading data

with open('info.txt', 'r') as f:

# Iterating over each username and password for line in f:

# Splitting the username and password into two parts user, password = line.strip().split(',')

# Converting the password to hash and hexadecimal in uppercase using SHA1, hexdigest and upper functions

pass\_hash = hashlib.sha1(password.encode('utf-8')).hexdigest().upper()

# Calling API to get the leaked passwords by matching the first 5 letters of password hash

response =

requests.get(f'https://api.pwnedpasswords.com/range/{pass\_hash[:5]}')

# Checking status of response for the password(returns 200 if response is successful)

if response.status\_code == 200:

# Splitting the matching hashes

hashes = response.text.splitlines()

count = 0 # Keeping track how many times the password is leaked # Iterating through all matching hashes for hash\_suffix in hashes:

# Password is leaked if the password hash matches with response hash completely

if hash\_suffix.split(':')[0] == pass\_hash[5:]: count += 1

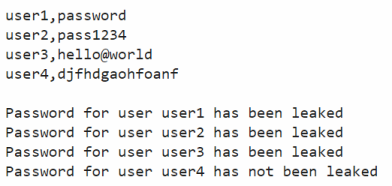
print(f'Password for user {user} has been leaked') break

if count == 0:

print(f'Password for user {user} has not been leaked') # Cannot search the password if response status is not 200 else:

print("Cannot search for password")

OUTPUT:



**Q5.Write a Python program that generates a password using a random combination of**

**words from a dictionary file.**

**CODE:**

**import random**

**words = []**

**for i in range(26):**

**words.append(chr(65 + i))**

**words.append(chr(97 + i))**

**def generate\_password(length, words):**

**password = ""**

**for i in range(length):**

**index = random.randint(0, 51)**

**password += words[index]**

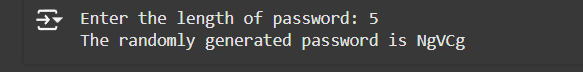
**return password**

**length = int(input("Enter the length of password: "))**

**password = generate\_password(length, words)**

**print(f'The randomly generated password is {password}')**

**OUTPUT -**

****

**Q6. Write a Python program that simulates a brute-force attack on a password by trying out**

**all possible character combinations.**

**CODE:**

**import itertools**

**import hashlib**

**def brute\_force\_attack(hashed\_password, max\_length=4):**

**"""**

**Performs a brute-force attack on a hashed password.**

**Args:**

**hashed\_password (str): The SHA-256 hash of the password to crack.**

**max\_length (int): The maximum length of the password to try.**

**Returns:**

**str or None: The cracked password if found, otherwise None.**

**"""**

**characters = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&\*()" # Add more characters as needed**

**for length in range(1, max\_length + 1):**

**for password\_attempt in itertools.product(characters, repeat=length):**

**password = "".join(password\_attempt)**

**hashed\_attempt = hashlib.sha256(password.encode()).hexdigest()**

**if hashed\_attempt == hashed\_password:**

**return password**

**return None # Password not found within the given constraints**

**if \_\_name\_\_ == "\_\_main\_\_":**

**target\_password = "password123" # Example password**

**hashed\_password = hashlib.sha256(target\_password.encode()).hexdigest()**

**print(f"Hashed Password: {hashed\_password}")**

**cracked\_password = brute\_force\_attack(hashed\_password)**

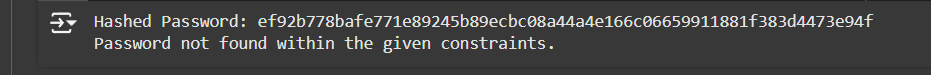
**if cracked\_password:**

**print(f"Cracked Password: {cracked\_password}")**

**else:**

**print("Password not found within the given constraints.")**

**OUTPUT -**

****