**Cloud security assignment-2**

**3. Simple Encryption and Decryption Tool**

Python script for a **Simple Encryption and Decryption Tool** using **AES encryption** from the cryptography library. The script allows users to encrypt plaintext, decrypt ciphertext, and handle file operations for encrypted data.

Python code

from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes

from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC

from cryptography.hazmat.primitives import hashes

from cryptography.hazmat.primitives.padding import PKCS7

from cryptography.hazmat.backends import default\_backend

import os

import base64

def generate\_salt():

return os.urandom(16)

def derive\_key(password, salt):

kdf = PBKDF2HMAC(

algorithm=hashes.SHA256(),

length=32,

salt=salt,

iterations=100000,

backend=default\_backend()

)

return kdf.derive(password.encode())

def encrypt(plaintext, password):

salt = generate\_salt()

key = derive\_key(password, salt)

iv = os.urandom(16)

cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default\_backend())

encryptor = cipher.encryptor()

padder = PKCS7(algorithms.AES.block\_size).padder()

padded\_data = padder.update(plaintext.encode()) + padder.finalize()

ciphertext = encryptor.update(padded\_data) + encryptor.finalize()

return base64.b64encode(salt + iv + ciphertext).decode()

def decrypt(ciphertext, password):

decoded\_data = base64.b64decode(ciphertext)

salt = decoded\_data[:16]

iv = decoded\_data[16:32]

encrypted\_data = decoded\_data[32:]

key = derive\_key(password, salt)

cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default\_backend())

decryptor = cipher.decryptor()

padded\_plaintext = decryptor.update(encrypted\_data) + decryptor.finalize()

unpadder = PKCS7(algorithms.AES.block\_size).unpadder()

plaintext = unpadder.update(padded\_plaintext) + unpadder.finalize()

return plaintext.decode()

def save\_to\_file(filename, data):

with open(filename, 'w') as file:

file.write(data)

def load\_from\_file(filename):

with open(filename, 'r') as file:

return file.read()

def main():

print("Simple Encryption and Decryption Tool")

print("1. Encrypt")

print("2. Decrypt")

print("3. Exit")

choice = input("Choose an option (1/2/3): ")

if choice == "1":

plaintext = input("Enter plaintext to encrypt: ")

password = input("Enter a password: ")

encrypted\_data = encrypt(plaintext, password)

print(f"Encrypted Data: {encrypted\_data}")

save\_option = input("Save encrypted data to a file? (y/n): ").lower()

if save\_option == 'y':

filename = input("Enter the filename: ")

save\_to\_file(filename, encrypted\_data)

print(f"Encrypted data saved to {filename}")

elif choice == "2":

source\_option = input("Load encrypted data from a file? (y/n): ").lower()

if source\_option == 'y':

filename = input("Enter the filename: ")

encrypted\_data = load\_from\_file(filename)

else:

encrypted\_data = input("Enter encrypted data: ")

password = input("Enter the password: ")

try:

decrypted\_data = decrypt(encrypted\_data, password)

print(f"Decrypted Data: {decrypted\_data}")

except Exception as e:

print("Decryption failed. Ensure the password and data are correct.")

print(f"Error: {e}")

elif choice == "3":

print("Exiting...")

return

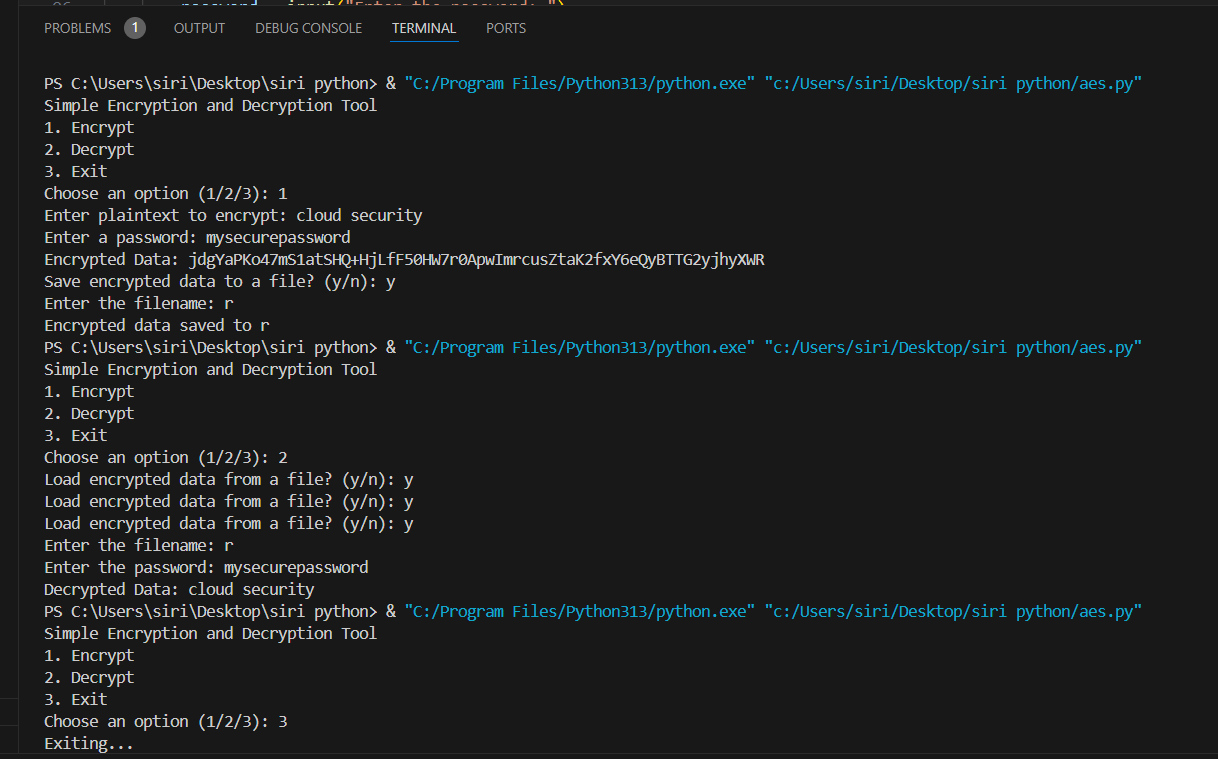
else:

print("Invalid choice. Please try again.")

main()

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

main()

output

Features of the Tool

1. Encryption and Decryption:
   * Encrypts plaintext using AES encryption in CBC mode with a randomly generated salt and IV.
   * Decrypts ciphertext using the derived key and removes padding.
2. Password-Based Encryption:
   * Derives a secure key from the user-provided password using PBKDF2 with SHA-256 and a random salt.
3. File Operations:
   * Save encrypted data to a file.
   * Load encrypted data from a file for decryption.
4. User Interaction:
   * Simple menu-driven interface for ease of use.

**4. Password Strength Checker**

**Password Strength Checker** implemented in Python. The tool evaluates the strength of passwords based on length, character variety, and presence of common words. It also suggests improvements for weak passwords.

Python code

import re

COMMON\_PASSWORDS = [

"password", "123456", "123456789", "qwerty", "abc123",

"password1", "12345678", "iloveyou", "admin", "welcome"

]

def check\_password\_strength(password):

suggestions = []

score = 0

if len(password) >= 12:

score += 1

else:

suggestions.append("Use at least 12 characters.")

if re.search(r"[a-z]", password):

score += 1

else:

suggestions.append("Include lowercase letters.")

if re.search(r"[A-Z]", password):

score += 1

else:

suggestions.append("Include uppercase letters.")

if re.search(r"[0-9]", password):

score += 1

else:

suggestions.append("Include numbers.")

if re.search(r"[!@#$%^&\*(),.?\":{}|<>]", password):

score += 1

else:

suggestions.append("Include special characters (e.g., !@#$%^&\*).")

if any(common.lower() in password.lower() for common in COMMON\_PASSWORDS):

suggestions.append("Avoid using common passwords or sequences (e.g., 'password', '123456').")

else:

score += 1

if score <= 2:

strength = "Weak"

elif score == 3 or score == 4:

strength = "Moderate"

else:

strength = "Strong"

return strength, suggestions

def main():

print("Password Strength Checker")

password = input("Enter a password to check: ")

strength, suggestions = check\_password\_strength(password)

print(f"Password Strength: {strength}")

if suggestions:

print("Suggestions to improve your password:")

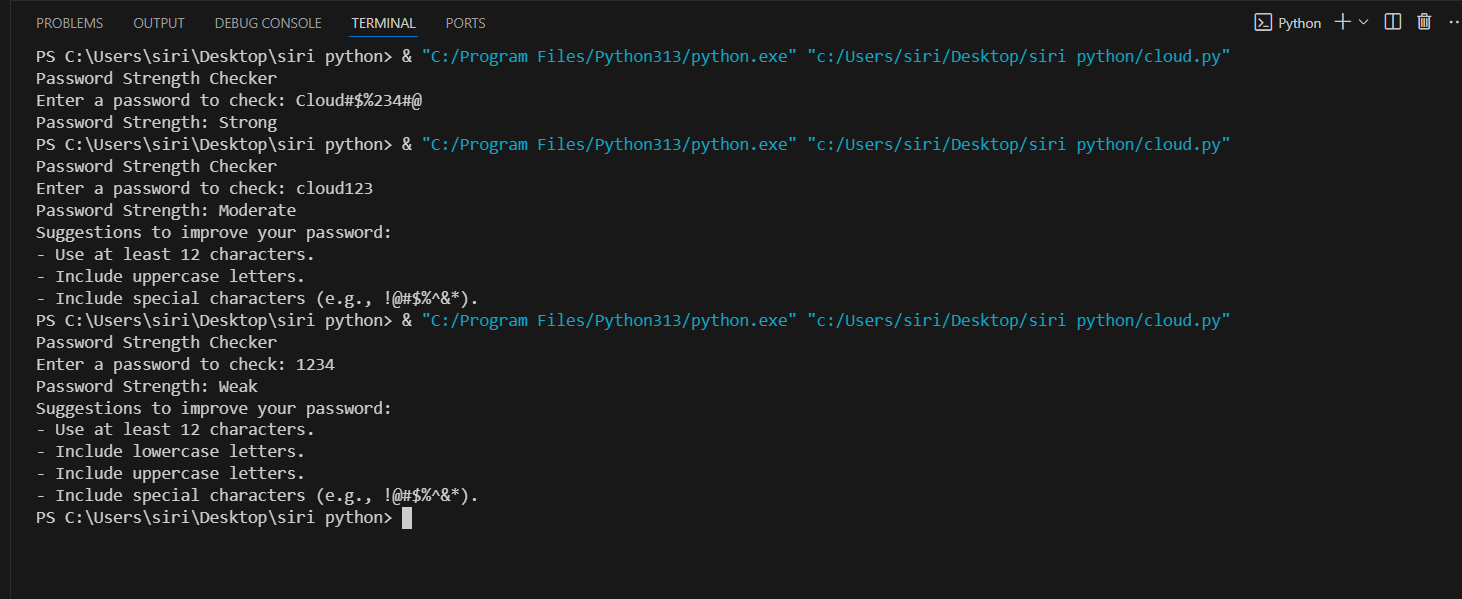
for suggestion in suggestions:

print(f"- {suggestion}")

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

main()

output



Features

1. Strength Criteria:
   * Length: Passwords should be at least 12 characters long.
   * Character Variety: Passwords should include lowercase letters, uppercase letters, numbers, and special characters.
   * Avoid Common Passwords: The tool checks for commonly used passwords and discourages their use.
2. Password Rating:
   * Weak: Score of 2 or less.
   * Moderate: Score of 3 or 4.
   * Strong: Score of 5.
3. Suggestions:
   * The tool provides actionable suggestions to improve password strength, such as adding character variety or avoiding common patterns.

**5.Basic Access Control System with User Roles**

Python implementation of a **Basic Access Control System with User Roles**. The script assigns roles to users and restricts access to specific resources based on the roles.

Python code

class AccessControlSystem:

def \_\_init\_\_(self):

self.roles\_permissions = {

"admin": ["view\_dashboard", "manage\_users", "access\_sensitive\_data"],

"user": ["view\_dashboard"],

"guest": []

}

self.users = {}

def add\_user(self, username, role):

if role not in self.roles\_permissions:

print(f"Error: Role '{role}' does not exist.")

return

self.users[username] = role

print(f"User '{username}' added with role '{role}'.")

def check\_access(self, username, action):

role = self.users.get(username)

if not role:

print(f"Access Denied: User '{username}' does not exist.")

return False

permissions = self.roles\_permissions.get(role, [])

if action in permissions:

print(f"Access Granted: User '{username}' ({role}) can perform '{action}'.")

return True

else:

print(f"Access Denied: User '{username}' ({role}) cannot perform '{action}'.")

return False

def list\_users(self):

print("\nCurrent Users:")

for username, role in self.users.items():

print(f"- {username}: {role}")

print()

def add\_role(self, role, permissions):

if role in self.roles\_permissions:

print(f"Error: Role '{role}' already exists.")

return

self.roles\_permissions[role] = permissions

print(f"Role '{role}' added with permissions: {permissions}")

def main():

system = AccessControlSystem()

print("Setting up the access control system...")

system.add\_role("manager", ["view\_dashboard", "manage\_users"])

system.add\_user("Alice", "admin")

system.add\_user("Bob", "user")

system.add\_user("Charlie", "guest")

system.add\_user("Dana", "manager")

system.list\_users()

print("\nAccess checks:")

system.check\_access("Alice", "view\_dashboard") # Allowed

system.check\_access("Bob", "manage\_users") # Denied

system.check\_access("Dana", "manage\_users") # Allowed

system.check\_access("Charlie", "view\_dashboard") # Denied

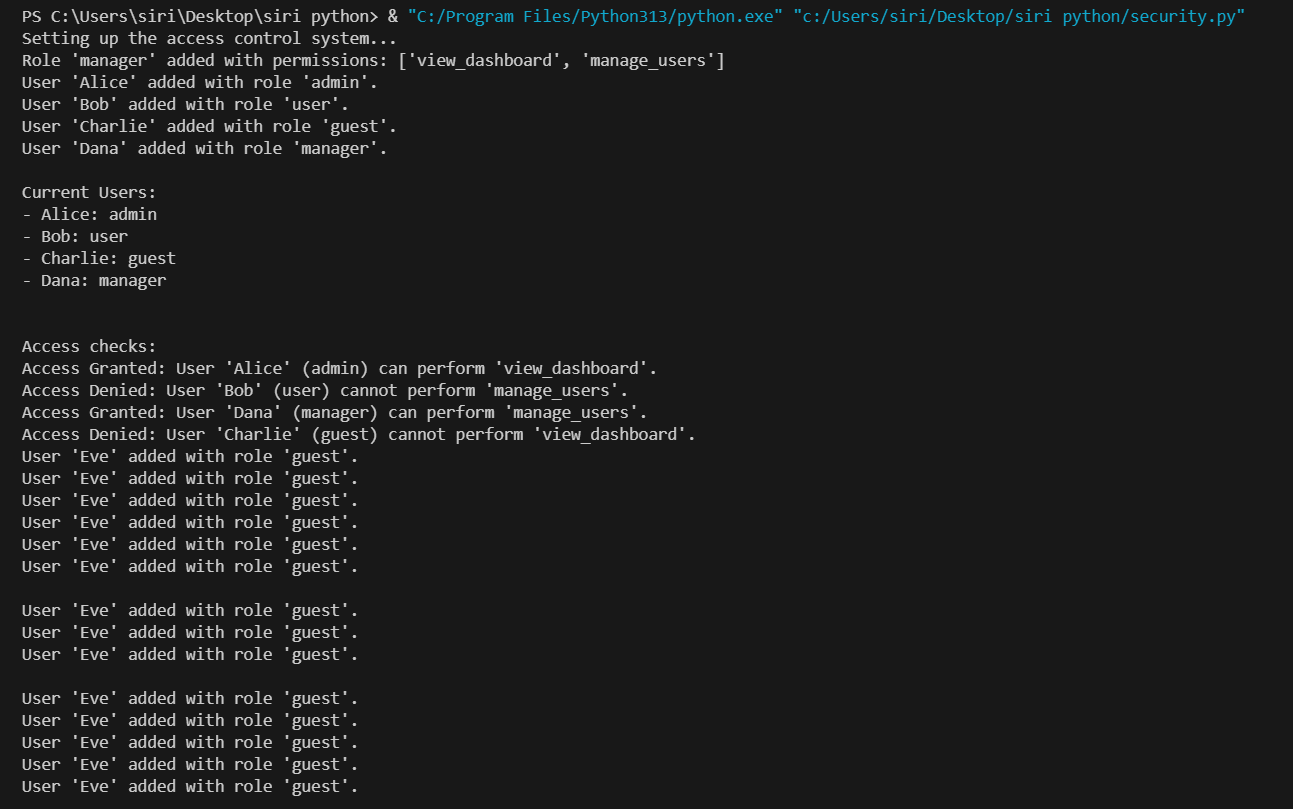
system.add\_user("Eve", "guest")

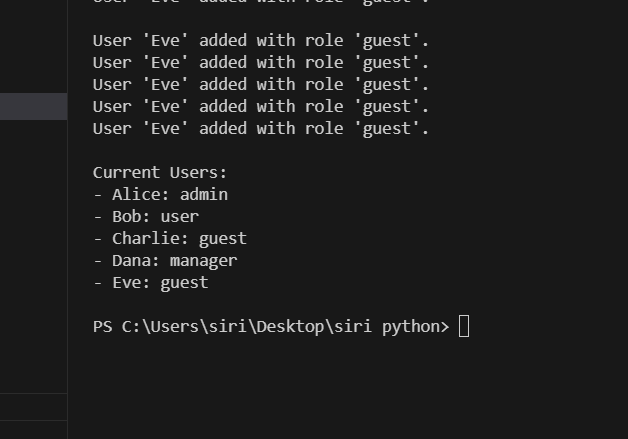
system.list\_users()

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

main()

output





Features

1. Role Definitions:
   * Predefined roles (admin, user, guest) with specific permissions.
   * Ability to add new roles dynamically with custom permissions.
2. User Management:
   * Add users with specific roles.
   * List all users and their associated roles.
3. Access Control:
   * Role-based access to actions such as view\_dashboard, manage\_users, and access\_sensitive\_data.
   * Dynamically checks if a user has permission for a given action.
4. Extensibility:
   * Easy to add new roles and permissions.
   * Supports additional functionalities as needed.