## **Aim 1 - WAP to perform encryption and decryption of string.**

**from cryptography.fernet import Fernet  
message = "hello world"  
key = Fernet.generate\_key()  
fernet = Fernet(key)  
# Encrypt the message (convert to bytes first)  
encMessage = fernet.encrypt(message.encode())  
print("Original string:", message)  
print("Encrypted string:", encMessage)  
# Decrypt the encrypted message  
decMessage = fernet.decrypt(encMessage).decode()  
print("Decrypted string:", decMessage)**

## **Aim 2 - WAP to perform encryption and decryption using Caesar cipher & Subs algorithm.**

**# Caesar Cipher Technique  
def caesar\_encrypt(text, s):  
 result = ""  
 for char in text:  
 if char.isupper():  
 result += chr((ord(char) + s - 65) % 26 + 65)  
 elif char.islower():  
 result += chr((ord(char) + s - 97) % 26 + 97)  
 else:  
 result += char   
 return result  
  
# Substitution Cipher Technique  
def substitution\_encrypt(text, key):  
 result = ""  
 alphabet = 'abcdefghijklmnopqrstuvwxyz'  
 for char in text:  
 if char.islower():  
 index = alphabet.index(char)  
 result += key[index]   
 elif char.isupper():  
 index = alphabet.index(char.lower())  
 result += key[index].upper()   
 else:  
 result += char   
 return result  
  
# Testing Caesar Cipher  
caesar\_text = "ATTACKATONCE"  
caesar\_shift = 4  
print("Caesar Cipher")  
print("Text : " + caesar\_text)  
print("Shift : " + str(caesar\_shift))  
print("Cipher: " + caesar\_encrypt(caesar\_text, caesar\_shift))  
print("\r")  
  
# Testing Substitution Cipher  
substitution\_text = "HELLO WORLD"  
substitution\_key = "qazwsxderfvctgbhnujmikolp"  
print("Substitution Cipher")  
print("Text : " + substitution\_text)  
print("Key : " + substitution\_key)  
print("Cipher: " + substitution\_encrypt(substitution\_text, substitution\_key))**

## **Aim 3 - Implementation of stegnography.**

**from PIL import Image  
import stepic  
  
#ENCRYPTION  
original\_img=Image.open('img1.jpg')  
encoded\_img=stepic.encode(original\_img,b'hello this is ns practical')  
encoded\_img.save('img2.png')  
encoded\_img=Image.open('img2.png')  
encoded\_img.show()  
  
#DECRYPTION  
decoded\_img=stepic.decode(encoded\_img)  
print("the decoded message is >>", decoded\_img)**

## **Aim 4 - Implementation of hash function.**

**import hashlib  
  
# SHA256  
str1 = "Network Security"  
result = hashlib.sha256(str1.encode())  
print("The hexadecimal equivalent of SHA256 is :")  
print(result.hexdigest())  
print("\r")  
  
# SHA384  
str2 = "Cryptography"  
result = hashlib.sha384(str2.encode())  
print("The hexadecimal equivalent of SHA384 is :")  
print(result.hexdigest())  
print("\r")  
  
# SHA224  
str3 = "Cybersecurity"  
result = hashlib.sha224(str3.encode())  
print("The hexadecimal equivalent of SHA224 is :")  
print(result.hexdigest())  
print("\r")  
  
# SHA512  
str4 = "Hello World"  
result = hashlib.sha512(str4.encode())  
print("The hexadecimal equivalent of SHA512 is :")  
print(result.hexdigest())  
print("\r")**

## **Aim 5 - Implementation of symemetric & Asymmetric cryptography.**

**import os  
from Crypto.Cipher import AES  
from cryptography.hazmat.primitives.asymmetric import rsa, padding  
from cryptography.hazmat.primitives import hashes  
  
# Symmetric Encryption (AES)  
def symmetric\_encryption():  
 key = os.urandom(32)  
 plaintext = b'This is a secret message'  
   
 cipher = AES.new(key, AES.MODE\_EAX)  
 ciphertext, tag = cipher.encrypt\_and\_digest(plaintext)  
  
 cipher\_decrypt = AES.new(key, AES.MODE\_EAX, nonce=cipher.nonce)  
 decrypted\_message = cipher\_decrypt.decrypt(ciphertext)  
  
 print("\n[Symmetric AES Encryption]")  
 print("Original Message:", plaintext.decode())  
 print("Encrypted Message:", ciphertext)  
 print("Decrypted Message:", decrypted\_message.decode())  
  
# Asymmetric Encryption (RSA)  
def asymmetric\_encryption():  
 private\_key = rsa.generate\_private\_key(public\_exponent=65537, key\_size=2048)  
 public\_key = private\_key.public\_key()  
  
 message = b"This is an RSA encrypted message"  
 ciphertext = public\_key.encrypt(  
 message,  
 padding.OAEP(mgf=padding.MGF1(algorithm=hashes.SHA256()),  
 algorithm=hashes.SHA256(), label=None)  
 )  
  
 decrypted\_message = private\_key.decrypt(  
 ciphertext,  
 padding.OAEP(mgf=padding.MGF1(algorithm=hashes.SHA256()),  
 algorithm=hashes.SHA256(), label=None)  
 )  
  
 print("\n[Asymmetric RSA Encryption]")  
 print("Original Message:", message.decode())  
 print("Encrypted Message:", ciphertext)  
 print("Decrypted Message:", decrypted\_message.decode())  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 symmetric\_encryption()  
 asymmetric\_encryption()**

## **Aim 6 - Implementation of default password generator.**

**import random  
import string  
  
def generate\_password(length):  
 if length < 4:  
 print("Password length should be at least 4 for better security.")  
 return None  
   
 lowercase\_letters = string.ascii\_lowercase  
 uppercase\_letters = string.ascii\_uppercase  
 digits = string.digits  
 special\_characters = string.punctuation  
  
 all\_characters = lowercase\_letters + uppercase\_letters + digits + special\_characters  
  
 password = [  
 random.choice(lowercase\_letters),  
 random.choice(uppercase\_letters),  
 random.choice(digits),  
 random.choice(special\_characters)  
 ]  
 password += random.choices(all\_characters, k=length - 4)  
 random.shuffle(password)  
  
 return ''.join(password)  
  
def main():  
 print("welcome to the Password Generator!")  
  
 while True:  
 try:  
 length = int(input("enter the desired length of the password: "))  
 if length < 4:  
 print("password length should be at least 4. Please try again.")  
 else:  
 break  
 except ValueError:  
 print("invalid input. Please enter a numeric value.")  
  
 password = generate\_password(length)  
 if password:  
 print("generated password:", password)  
 print("keep it secure and don't share it with anyone!")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()**