Program30 C program for CBC MAC of a oneblock message X,

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

from cryptography.hazmat.primitives import padding

from cryptography.hazmat.backends import default\_backend

import os

def xor\_bytes(byte\_str1, byte\_str2):

return bytes([b1 ^ b2 for b1, b2 in zip(byte\_str1, byte\_str2)])

def cbc\_mac(key, message):

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

encryptor = cipher.encryptor()

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

padded\_message = padder.update(message) + padder.finalize()

iv = os.urandom(algorithms.AES.block\_size)

prev\_block = iv

for i in range(0, len(padded\_message), algorithms.AES.block\_size):

block = padded\_message[i:i+algorithms.AES.block\_size]

xor\_result = xor\_bytes(block, prev\_block)

prev\_block = encryptor.update(xor\_result)

return prev\_block

def main():

key = os.urandom(32) # Use 32 bytes for a 256-bit key

message = b"Hello, this is a one-block message."

t = cbc\_mac(key, message)

x\_xor\_t = xor\_bytes(message, t)

two\_block\_message = message + x\_xor\_t

cbc\_mac\_for\_two\_block = cbc\_mac(key, two\_block\_message)

print("Original T (MAC for one-block message):", t.hex())

print("Calculated CBC MAC for two-block message:", cbc\_mac\_for\_two\_block.hex())

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

main()

OUTPUT:

Original Message 1: 6b c1 be e2 2e 40 9f 96 e9 3d 7e 11 73 93 17 2a

MAC for Message 1: 40 bf ab f4 06 ee 4d 30 42 ca 6b 99 7a 5c 58 16

Original Message 2: 6b c1 be e2 2e 40 9f 96 e9 3d 7e 11 73 93 17 2a 2b 7e 15 16 28 ae d2 a6 ab

f7 15 88 09 cf 4f 3c