VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CRYPTOGRAPGY LAB

ASSESSMENT

NAME:SHIVANK PANDEY

REG NUMBER:21BKT0021

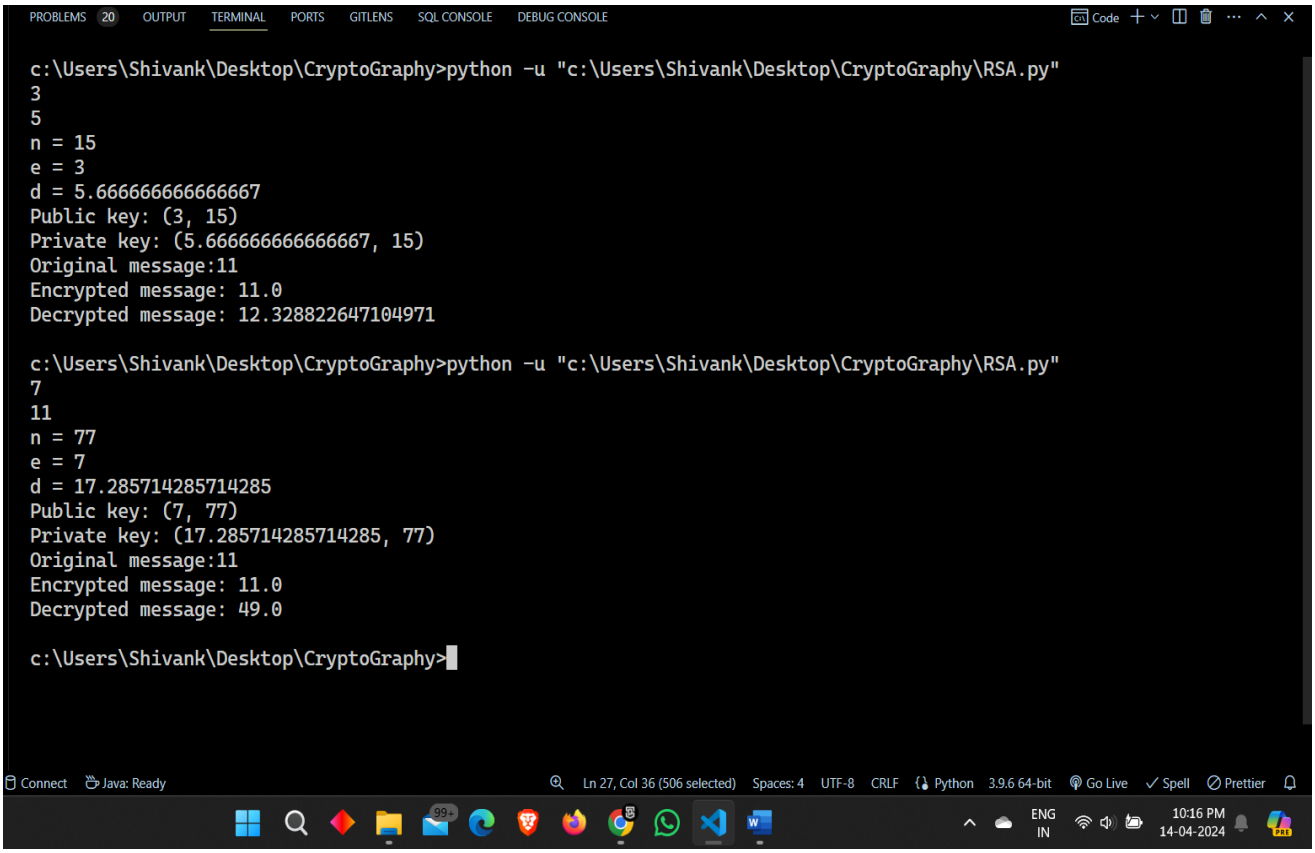
1)RSA ALGO IN PYTHON

CODE-

```
import math
p = eval(input());
q = eval(input());
n = p*q
print("n =", n)
phi = (p-1)*(q-1)
e = 2
while(e<phi):
    if (math.gcd(e, phi) == 1):
        break
    else:
        e += 1

print("e =", e)
k = 2
d = ((k*phi)+1)/e
print("d =", d)
print(f'Public key: {e, n}')
print(f'Private key: {d, n}')
msg = 11
print(f'Original message:{msg}')
C = pow(msg, e)
C = math.fmod(C, n)
print(f'Encrypted message: {C}')
M = pow(C, d)
M = math.fmod(M, n)
print(f'Decrypted message: {M}')
```

OUTPUT-



2)ECC ALGO IN PYTHON

```
from tinyec import registry
from Crypto.Cipher import AES
import hashlib, secrets, binascii
```

```
def encrypt_AES_GCM(msg, secretKey):
    aesCipher = AES.new(secretKey, AES.MODE_GCM)
    ciphertext, authTag = aesCipher.encrypt_and_digest(msg)
    return (ciphertext, aesCipher.nonce, authTag)

def decrypt_AES_GCM(ciphertext, nonce, authTag, secretKey):
    aesCipher = AES.new(secretKey, AES.MODE_GCM, nonce)
    plaintext = aesCipher.decrypt_and_verify(ciphertext, authTag)
    return plaintext

def ecc_point_to_256_bit_key(point):
    sha = hashlib.sha256(int.to_bytes(point.x, 32, 'big'))
    sha.update(int.to_bytes(point.y, 32, 'big'))
    return sha.digest()

curve = registry.get_curve('brainpoolP256r1')

def encrypt_ECC(msg, pubKey):
    ciphertextPrivKey = secrets.randbelow(curve.field.n)
    sharedECCKey = ciphertextPrivKey * pubKey
    secretKey = ecc_point_to_256_bit_key(sharedECCKey)
    ciphertext, nonce, authTag = encrypt_AES_GCM(msg, secretKey)
    ciphertextPubKey = ciphertextPrivKey * curve.g
    return (ciphertext, nonce, authTag, ciphertextPubKey)

def decrypt_ECC(encryptedMsg, privKey):
    (ciphertext, nonce, authTag, ciphertextPubKey) = encryptedMsg
    sharedECCKey = privKey * ciphertextPubKey
    secretKey = ecc_point_to_256_bit_key(sharedECCKey)
    plaintext = decrypt_AES_GCM(ciphertext, nonce, authTag, secretKey)
    return plaintext

msg = b'Text to be encrypted by ECC public key and ' \
      b'decrypted by its corresponding ECC private key'
print("original msg:", msg)
privKey = secrets.randbelow(curve.field.n)
pubKey = privKey * curve.g

encryptedMsg = encrypt_ECC(msg, pubKey)
encryptedMsgObj = {
    'ciphertext': binascii.hexlify(encryptedMsg[0]),
    'nonce': binascii.hexlify(encryptedMsg[1]),
    'authTag': binascii.hexlify(encryptedMsg[2]),
    'ciphertextPubKey': hex(encryptedMsg[3].x) + hex(encryptedMsg[3].y %
2)[2:]
}
print("encrypted msg:", encryptedMsgObj)

decryptedMsg = decrypt_ECC(encryptedMsg, privKey)
```

```
print("decrypted msg:", decryptedMsg)
```

OUTPUT-