



University
of Windsor

Lab 5

Course: Networking and Data Security

COMP8677-1-R-2023F

Professor: Dr. Shaoquan Jiang

Prepared by

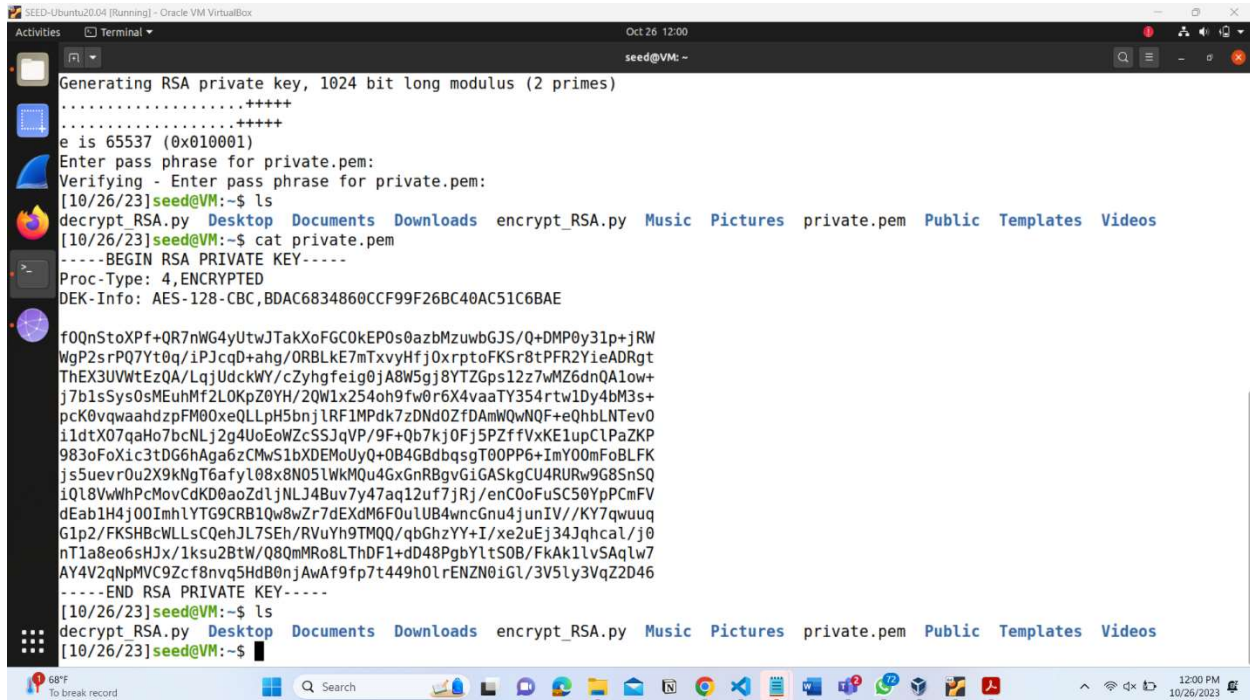
Harshil Hitendrabhai Panchal (110096129)

Due date: October 26, 2023

1. Use openssl to generate RSA public/private key

We can generate RSA private key (p, q, d) using openssl:

\$ openssl genrsa -aes128 -out private.pem 1024



```
SEED-Ubuntu20.04 [Running] - Oracle VM VirtualBox
Activities Terminal
Oct 26 12:00
seed@VM: ~
Generating RSA private key, 1024 bit long modulus (2 primes)
.....+++++
.....+++++
e is 65537 (0x010001)
Enter pass phrase for private.pem:
Verifying - Enter pass phrase for private.pem:
[10/26/23]seed@VM:~$ ls
decrypt_RSA.py Desktop Documents Downloads encrypt_RSA.py Music Pictures private.pem Public Templates Videos
[10/26/23]seed@VM:~$ cat private.pem
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-128-CBC,BDAC6834860CCF99F26BC40AC51C6BAE
f0QnStoXPf+QR7nWG4yUtwJTakXoFGC0kEP0s0azbMzuwbGJS/Q+DMP0y31p+jRW
WgP2srPQ7Yt0q/iPjCqD+ahg/ORBLkE7mTxvyHfj0xrptoFKSr8tPFR2YieADRgt
ThEX3UVWtEzQA/LqjUdckWY/cZyhgfeg0jA8W5gj8YTZGps12z7wMZ6dnQAlo+
j7b1sSys0sMEuhMf2L0KpZ0YH/2QW1x254oh9fw0r6X4vaaTY354rtw1Dy4bM3s+
pcK0vqwaahdzpFM00xeQLLpH5bnjLRF1MPdk7zDNd0ZfDAmWQwNQF+eQhblNTEv0
i1dtX07qaHo7bcNLj2g4UoEoWZcSSJqVP/9F+Qb7Kj0Fj5PZffVxKE1upClPaZKP
983oFoXic3tDG6hAga6zCMwS1bXDEMoUyQ+OB4GBdbqsgT00PP6+ImY00mFoBLFK
js5uevr0u2X9kNgT6afyL08x8N051WkMQ4GxGnRBgvGiGASkgCU4RURw9G8SnSQ
iQl8VwWhPcMocvCdK0aoZdljNLJ4Buv7y47aq12uf7Jrj/enC0oFuSC50YpPCmFV
dEab1H4j00ImhLYTG9CRB1Qw8wZr7dEXdM6F0uLUB4wncGnu4junIV//KY7qwuuq
G1p2/FKSHBcWLLsCQehJL7SEh/RVuYh9TMQ0/qbGhZYY+I/xe2uEj34Jghcal/j0
nT1a8eo6sHJx/1ksu2BtW/Q8QmMRo8LThDF1+dD48PgbYltS0B/FkAk1lvSAqLw7
AY4V2qNpMVC9Zcf8nvq5HdB0njAwAf9fp7t449h0lRENZN0iGL/3V5ly3VqZ2D46
-----END RSA PRIVATE KEY-----
[10/26/23]seed@VM:~$ ls
decrypt_RSA.py Desktop Documents Downloads encrypt_RSA.py Music Pictures private.pem Public Templates Videos
[10/26/23]seed@VM:~$
```

This will generate a rsa instance (p, q, d, e, n) with p, q of 1024 bits and to prevent leaking the private key, the output private.pem is encrypted by aes128 cipher with password you will be prompted to provide. Now use the above command to generate a rsa private key and save it in file private.pem. Then, extract the public key (e, n) in a file public.pem:

\$ openssl rsa -in private.pem -pubout >public.pem

```
SEED-Ubuntu20.04 [Running] - Oracle VM VirtualBox
Activities Terminal
Oct 26 12:01
seed@VM: ~
ThEX3UvWtEzQA/LqjUdckWY/cZyhgfeg0jA8W5gj8YTZGps12z7wM26dnQA1ow+
j7b1sSy0sMEuhMf2L0KpZ0YH/2QW1x254oh9fw0r6X4vaaTY354rtw1Dy4bM3s+
pcK0vqwaahdzpFM00xeQLpH5bnj1RF1MPdk7zDNd0ZfDAmWQwNQF+eQhbLNTev0
i1dtX07qaHo7bcNLj2g4UoEoWZcSSJqVP/9F+Qb7Kj0Fj5PZffVxKE1upCLPaZKP
983oFoXic3tDG6hAga6zCMwS1bXDEMoUy0+0B4GBdbqsgT00PP6+ImY00mFoBLFK
js5uevr0u2X9KngT6afyl08x8N05lWkMQu4GxGnRBgvGiGASkgCU4RURw9G8SnSQ
iQl8VWhPcMovCdKD0aoZdljNLJ4Buv7y47aq12uf7jRj/enC0oFuSC50YpPCmFV
dEab1H4j00ImhLYTG9CRB1Qw8wZr7dEXdM6F0uLUB4wncGnu4junIV//KY7qwuuq
G1p2/FKSHBcWLLsCQehJL7SEh/RVUyH9TMQq/qbGhzYY+I/xe2uEj34JqhcAl/j0
nT1a8eo6sHJx/1ksu2BtW/Q8QmMRo8LThDF1+dD48PgbYlt50B/FkAk1lvSAqlw7
AY4V2qNpMVC9Zcf8nvq5HdB0njAwAf9fp7t449h0LrENZN0iGL/3V5ly3VqZ2D46
-----END RSA PRIVATE KEY-----
[10/26/23]seed@VM:~$ ls
decrypt_RSA.py Desktop Documents Downloads encrypt_RSA.py Music Pictures private.pem Public Templates Videos
[10/26/23]seed@VM:~$ openssl rsa -in private.pem -pubout > public.pem
rsa: Use -help for summary.
[10/26/23]seed@VM:~$ openssl rsa -in private.pem -pubout > public.pem
Enter pass phrase for private.pem:
writing RSA key
[10/26/23]seed@VM:~$ ls
decrypt_RSA.py Documents encrypt_RSA.py Pictures Public Templates
Desktop Downloads Music private.pem public.pem Videos
[10/26/23]seed@VM:~$ cat public.pem
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQBAQUAA4GNADCBiQKBggQCC6LALxpJKBppEwjfMAvh0h9lMp
i00UzIoL12XDwh+gS/hk/gt1dzSjqYidz0U8Knrs5f+tg0VmIcdWiF+d/cUXYw
dk/eQfDYsa58QEBN7tePrqu0g53h5E0y4v7kux4eBXBzB+sLnuEQRYKYvOK8pX7
TvoPiacvAL/P8lRpywIDAQAB
-----END PUBLIC KEY-----
[10/26/23]seed@VM:~$
```

You can display private key using

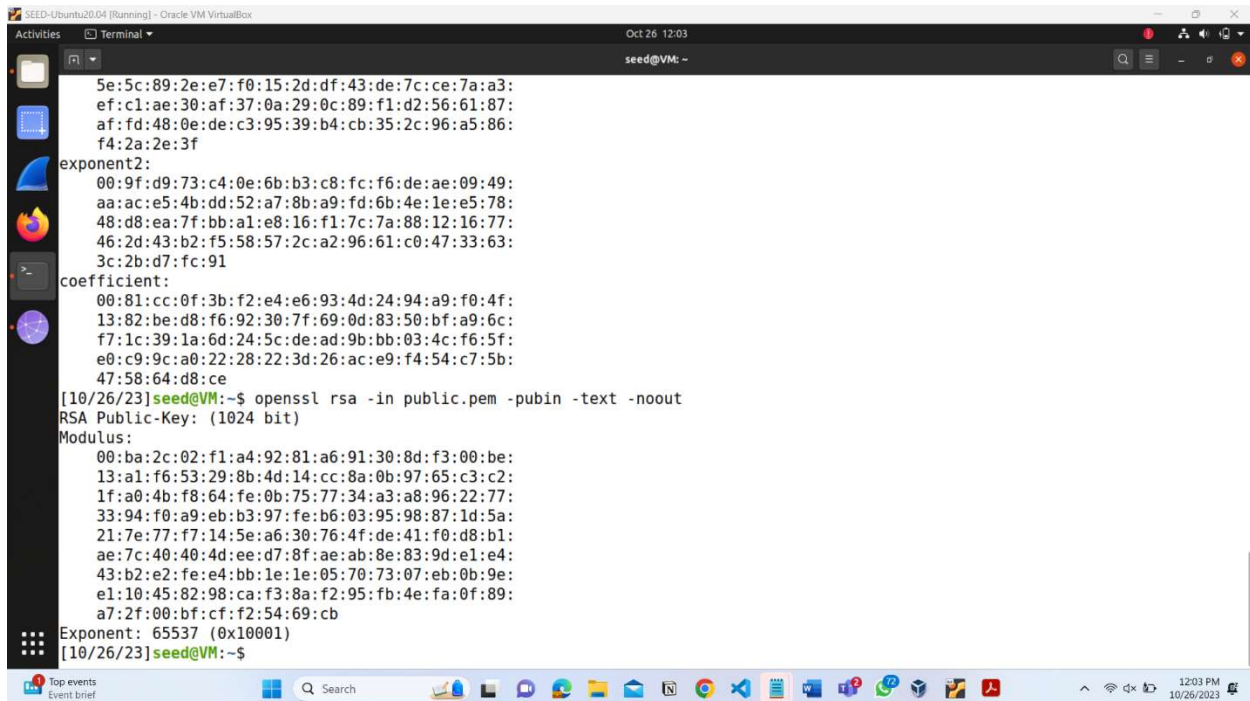
\$openssl rsa -in private.pem -text -noout

```
SEED-Ubuntu20.04 [Running] - Oracle VM VirtualBox
Activities Terminal
Oct 26 12:02
seed@VM: ~
dk/eQfDYsa58QEBN7tePrqu0g53h5E0y4v7kux4eBXBzB+sLnuEQRYKYvOK8pX7
TvoPiacvAL/P8lRpywIDAQAB
-----END PUBLIC KEY-----
[10/26/23]seed@VM:~$ openssl rsa -in private.pem -text -noout
rsa: Use -help for summary.
[10/26/23]seed@VM:~$ openssl rsa -in private.pem -text -noout
Enter pass phrase for private.pem:
RSA Private-Key: (1024 bit, 2 primes)
modulus:
 00:ba:2c:02:f1:a4:92:81:a6:91:30:8d:f3:00:be:
 13:a1:f6:53:29:8b:4d:14:cc:8a:0b:97:65:c3:c2:
 1f:a0:4b:f8:64:fe:0b:75:77:34:a3:a8:96:22:77:
 33:94:f0:a9:eb:b3:97:fe:b6:03:95:98:87:1d:5a:
 21:7e:77:f7:14:5e:a6:30:76:4f:de:41:f0:d8:b1:
 ae:7c:40:40:4d:ee:d7:8f:ae:ab:8e:83:9d:e1:e4:
 43:b2:e2:fe:e4:bb:1e:1e:05:70:73:07:eb:0b:9e:
 e1:10:45:82:98:ca:f3:8a:f2:95:fb:4e:fa:0f:89:
 a7:2f:00:bf:cf:f2:54:69:cb
publicExponent: 65537 (0x10001)
privateExponent:
 00:87:78:89:37:df:42:80:b7:7d:45:30:b5:d8:1f:
 78:57:cd:cf:dc:16:32:a3:e4:e6:ba:e2:93:39:ac:
 a3:a8:d8:3f:4a:f6:15:ce:87:4c:b5:9e:72:89:67:
 e6:10:06:44:0f:70:a3:34:c4:ab:bd:0a:bd:9f:1d:
 3b:ec:34:4d:84:b4:64:2f:4c:7f:09:86:7f:3a:6f:
 ad:5a:be:d1:27:03:c0:7d:58:39:b4:5c:98:57:c8:
 78:57:19:b6:3b:b1:28:e4:86:80:eb:6b:d3:ec:4a:
 f5:fa:af:59:dd:e8:43:98:b2:1c:b0:be:e7:92:1a:
 97:86:24:6e:e3:00:e4:cf:a1
prime1:
```

You also can display public key using

\$openssl rsa -in public.pem -pubin -text -noout

Take screen for the displays for these two files, as evidence of your work



```
SEED-Ubuntu20.04 [Running] - Oracle VM VirtualBox
Activities Terminal
Oct 26 12:03
seed@VM: ~
5e:5c:89:2e:e7:f0:15:2d:df:43:de:7c:ce:7a:a3:
ef:c1:ae:30:af:37:0a:29:0c:89:f1:d2:56:61:87:
af:fd:48:0e:de:c3:95:39:b4:cb:35:2c:96:a5:86:
f4:2a:2e:3f
exponent2:
00:9f:d9:73:c4:0e:6b:b3:c8:fc:f6:de:ae:09:49:
aa:ac:e5:4b:dd:52:a7:8b:a9:fd:6b:4e:1e:e5:78:
48:d8:ea:7f:bb:a1:e8:16:f1:7c:7a:88:12:16:77:
46:2d:43:b2:f5:58:57:2c:a2:96:61:c0:47:33:63:
3c:2b:d7:fc:91
coefficient:
00:81:cc:0f:3b:f2:e4:e6:93:4d:24:94:a9:f0:4f:
13:82:be:d8:f6:92:30:7f:69:0d:83:50:bf:a9:6c:
f7:1c:39:1a:6d:24:5c:de:ad:9b:bb:03:4c:f6:5f:
e0:c9:9c:a0:22:28:22:3d:26:ac:e9:f4:54:c7:5b:
47:58:64:d8:ce
[10/26/23]seed@VM:~$ openssl rsa -in public.pem -pubin -text -noout
RSA Public-Key: (1024 bit)
Modulus:
00:ba:2c:02:f1:a4:92:81:a6:91:30:8d:f3:00:be:
13:a1:f6:53:29:8b:4d:14:cc:8a:0b:97:65:c3:c2:
1f:a0:4b:f8:64:fe:0b:75:77:34:a3:a8:96:22:77:
33:94:f0:a9:eb:b3:97:fe:b6:03:95:98:87:1d:5a:
21:7e:77:f7:14:5e:a6:30:76:4f:de:41:f0:d8:b1:
ae:7c:40:40:4d:ee:d7:8f:ae:ab:8e:83:9d:e1:e4:
43:b2:e2:fe:e4:bb:1e:1e:05:70:73:07:eb:0b:9e:
e1:10:45:82:98:ca:f3:8a:f2:95:fb:4e:fa:0f:89:
a7:2f:00:bf:cf:f2:54:69:cb
Exponent: 65537 (0x10001)
[10/26/23]seed@VM:~$
```

2. In this problem, you need to practice RSA encryption and decryption.

- Encrypt messages using PKCS1_OAEP, which is an implementation of RSA. Use the key **RsaKey** derived above to do the encryption. The functions are described as follow.
 - Cipher=**PKCS1_OAEP.new**(RsaKey): ○ For the encryption, RsaKey is a public-key. Return an encryption object **Cipher**.
 - Cipher.**encrypt**(message): ○ This returns ciphertext of message (byte string) under encryption object **Cipher**.

```
SEED-Ubuntu20.04 [Running] - Oracle VM VirtualBox
Activities Terminal Oct 26 12:06 seed@VM: ~

13:a1:f6:53:29:8b:4d:14:cc:8a:0b:97:65:c3:c2:
1f:a0:4b:f8:64:fe:0b:75:77:34:a3:a8:96:22:77:
33:94:f0:a9:eb:b3:97:fe:b6:03:95:98:87:1d:5a:
21:7e:77:f7:14:5e:a6:30:76:4f:de:41:f0:d8:b1:
ae:7c:40:40:4d:ee:d7:8f:ae:ab:8e:83:9d:e1:e4:
43:b2:e2:fe:e4:bb:1e:1e:05:70:73:07:eb:0b:9e:
e1:10:45:82:98:ca:f3:8a:f2:95:fb:4e:fa:0f:89:
a7:2f:00:bf:cf:f2:54:69:cb
Exponent: 65537 (0x10001)
[10/26/23]seed@VM:~$ cat encrypt_RSA.py
#!/usr/bin/python3

from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA

message=b'Harshil Panchal 110096129\n'

key=RSA.import_key(open('public.pem').read())

cipher=PKCS1_OAEP.new(key)

ciphertext=cipher.encrypt(message)

f=open('ciphertext.bin', 'wb')

f.write(ciphertext)

f.close()

[10/26/23]seed@VM:~$
```

```
SEED-Ubuntu20.04 [Running] - Oracle VM VirtualBox
Activities Terminal Oct 26 12:08 seed@VM: ~

[10/26/23]seed@VM:~$ cat encrypt_RSA.py
#!/usr/bin/python3

from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA

message=b'Harshil Panchal 110096129\n'

key=RSA.import_key(open('public.pem').read())

cipher=PKCS1_OAEP.new(key)

ciphertext=cipher.encrypt(message)

f=open('ciphertext.bin', 'wb')

f.write(ciphertext)

f.close()

[10/26/23]seed@VM:~$ ls
decrypt_RSA.py  Documents  encrypt_RSA.py  Pictures  Public  Templates
Desktop         Downloads  Music          private.pem  public.pem  Videos
[10/26/23]seed@VM:~$ python3 encrypt_RSA.py
[10/26/23]seed@VM:~$ ls
ciphertext.bin  Desktop  Downloads  Music  private.pem  public.pem  Videos
decrypt_RSA.py  Documents  encrypt_RSA.py  Pictures  Public  Templates
[10/26/23]seed@VM:~$ cat ciphertext.bin
^Kk00/000100700b0Y000507E0f0S70`0000gj_+^Vl0g=!0U00B0000/NG0P0W0000E`r0N!] [y%00}20s0,k0l00K00w0'g000'>0v0N0[10/26/23]seed@VM:
~$
```

Encrypt message='your name and ID' and save ciphertext into a file. Take a screen shot for hexdump of your ciphertext (\$hexdump -C filename). Ref. encrypt_RSA.py.


```
SEED-Ubuntu20.04 [Running] - Oracle VM VirtualBox
Activities Terminal
Oct 26 12:07
seed@VM: ~

a7:2f:00:bf:cf:f2:54:69:cb
Exponent: 65537 (0x10001)
[10/26/23]seed@VM:~$ cat encrypt_RSA.py
#!/usr/bin/python3

from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA

message=b'Harshil Panchal 110096129\n'

key=RSA.import_key(open('public.pem').read())

cipher=PKCS1_OAEP.new(key)

ciphertext=cipher.encrypt(message)

f=open('ciphertext.bin', 'wb')

f.write(ciphertext)

f.close()

[10/26/23]seed@VM:~$ ls
decrypt_RSA.py  Documents  encrypt_RSA.py  Pictures  Public  Templates
Desktop        Downloads  Music          private.pem  public.pem  Videos
[10/26/23]seed@VM:~$ python3 encrypt_RSA.py
[10/26/23]seed@VM:~$ ls
ciphertext.bin  Desktop  Downloads  Music  private.pem  public.pem  Videos
decrypt_RSA.py  Documents  encrypt_RSA.py  Pictures  Public  Templates
[10/26/23]seed@VM:~$
```

```
SEED-Ubuntu20.04 [Running] - Oracle VM VirtualBox
Activities Terminal
Oct 26 12:09
seed@VM: ~

ciphertext=cipher.encrypt(message)

f=open('ciphertext.bin', 'wb')

f.write(ciphertext)

f.close()

[10/26/23]seed@VM:~$ ls
decrypt_RSA.py  Documents  encrypt_RSA.py  Pictures  Public  Templates
Desktop        Downloads  Music          private.pem  public.pem  Videos
[10/26/23]seed@VM:~$ python3 encrypt_RSA.py
[10/26/23]seed@VM:~$ ls
ciphertext.bin  Desktop  Downloads  Music  private.pem  public.pem  Videos
decrypt_RSA.py  Documents  encrypt_RSA.py  Pictures  Public  Templates
[10/26/23]seed@VM:~$ cat ciphertext.bin
^@k00/000i00700b0Y000507E0f0S?0 0000gj_+^Vl0g=!0U06B0000/NG0P0W000 0E`r0N!][y%00}20s0,k0l00K00w0^000'>0v0N0[10/26/23]seed@VM:
~$ ^C
[10/26/23]seed@VM:~$ hexdump -C ciphertext.bin
00000000  5e f1 6b 00 e7 d7 2f bb e2 02 f1 69 b3 ac 37 fc |^k.../...i..7.|
00000010  f4 62 b2 59 97 9b ec 24 b8 37 45 cf 66 ab 26 6f |.b.Y...$.7E.f.&o|
00000020  08 3f 9e 60 b8 91 9c b8 67 6a 5f 1c 2b 5e 7f 56 |.?.`....gj_+^V|
00000030  6c cb 67 3d 21 ca 55 84 c7 ad 42 da db f7 c4 2f |l.g=!..U...B.../|
00000040  4e 47 ac 50 80 57 ab 05 bf 51 ef 9c af 22 c6 45 |NG.P.W...Q...".E|
00000050  60 72 c4 01 4e 21 5d 5b 79 25 ec b7 f3 97 7d 32 |`r..N!][y%....}2|
00000060  d5 73 bb dd b8 6b 1c ce 6c ff 8d 4b df bb b8 77 |.s...k..l..K..w|
00000070  8d ca b8 c9 a2 ae d6 07 b8 27 3e c0 76 c3 4e ae |.....'>.v.N.|
00000080
[10/26/23]seed@VM:~$
```

- b. Decrypt the ciphertext in (a). The functions are described as follow.
- i. Cipher=PKCS1_OAEP.new(RsaKey):
 - o For the decryption, RsaKey is a private-key. Return an decryption object Cipher.

- ii. Cipher.decrypt(ctxt): ○ This returns message='your name and ID' under decryption object Cipher.

```
SEED-Ubuntu20.04 [Running] - Oracle VM VirtualBox
Activities Terminal
Oct 26 12:10
seed@VM: ~
ciphertext.bin Desktop Downloads Music private.pem public.pem Videos
decrypt RSA.py Documents encrypt RSA.py Pictures Public Templates
[10/26/23]seed@VM:~$ cat ciphertext.bin
^0k00/000100700b0Y000507E0f0S70'0000gj_+^Vl0g=!0U00B0000/NG0P0W000 0E' r0N![y%00}20s0,k0l00K00w0'c000'>0v0N0[10/26/23]seed@VM:
~$ ^C
[10/26/23]seed@VM:~$ hexdump -C ciphertext.bin
00000000 5e f1 6b 00 e7 d7 2f bb e2 02 f1 69 b3 ac 37 fc |^..k.../....i..7.|
00000010 f4 62 b2 59 97 9b ec 24 b8 37 45 cf 66 ab 26 6f |.b.Y...$.7E.f.&o|
00000020 08 3f 9e 60 b8 91 9c b8 67 6a 5f 1c 2b 5e 7f 56 |.?.`....gj_+^..V|
00000030 6c cb 67 3d 21 ca 55 84 c7 ad 42 da db f7 c4 2f |l.g=!..U...B....|
00000040 4e 47 ac 50 80 57 ab 05 bf 51 ef 9c af 22 c6 45 |NG.P.W...Q....E|
00000050 60 72 c4 01 4e 21 5d 5b 79 25 ec b7 f3 97 7d 32 |`r..N!][y%....}2|
00000060 d5 73 bb dd b8 6b 1c ce 6c ff 8d 4b df bb b8 77 |.s...k..l..K...w|
00000070 8d ca b8 c9 a2 ae d6 07 b8 27 3e c0 76 c3 4e ae |.....'>.v.N.|
00000080
[10/26/23]seed@VM:~$ python3 decrypt_RSA.py
Traceback (most recent call last):
  File "decrypt RSA.py", line 9, in <module>
    prikey = RSA.import_key(key_str, passphrase='harshil')
  File "/usr/local/lib/python3.8/dist-packages/Crypto/PublicKey/RSA.py", line 764, in import_key
    (der, marker, enc_flag) = PEM.decode(tostr(extern_key), passphrase)
  File "/usr/local/lib/python3.8/dist-packages/Crypto/IO/PEM.py", line 183, in decode
    data = unpad(objdec.decrypt(data), objdec.block_size)
  File "/usr/local/lib/python3.8/dist-packages/Crypto/Util/Padding.py", line 93, in unpad
    raise ValueError("PKCS#7 padding is incorrect.")
ValueError: PKCS#7 padding is incorrect.
[10/26/23]seed@VM:~$ gedit decrypt_RSA.py
[10/26/23]seed@VM:~$ python3 decrypt_RSA.py
b'Harshil Panchal 110096129\n'
[10/26/23]seed@VM:~$

SEED-Ubuntu20.04 [Running] - Oracle VM VirtualBox
Activities Terminal
Oct 26 12:11
seed@VM: ~
[10/26/23]seed@VM:~$ python3 decrypt_RSA.py
Traceback (most recent call last):
  File "decrypt RSA.py", line 9, in <module>
    prikey = RSA.import_key(key_str, passphrase='harshil')
  File "/usr/local/lib/python3.8/dist-packages/Crypto/PublicKey/RSA.py", line 764, in import_key
    (der, marker, enc_flag) = PEM.decode(tostr(extern_key), passphrase)
  File "/usr/local/lib/python3.8/dist-packages/Crypto/IO/PEM.py", line 183, in decode
    data = unpad(objdec.decrypt(data), objdec.block_size)
  File "/usr/local/lib/python3.8/dist-packages/Crypto/Util/Padding.py", line 93, in unpad
    raise ValueError("PKCS#7 padding is incorrect.")
ValueError: PKCS#7 padding is incorrect.
[10/26/23]seed@VM:~$ gedit decrypt_RSA.py
[10/26/23]seed@VM:~$ python3 decrypt_RSA.py
b'Harshil Panchal 110096129\n'
[10/26/23]seed@VM:~$ cat decrypt_RSA.py
#!/usr/bin/python3

from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA

ciphertext = open('ciphertext.bin', 'rb').read()

key_str = open('private.pem').read()
prikey = RSA.import_key(key_str, passphrase='dees')
cipher = PKCS1_OAEP.new(prikey)
message = cipher.decrypt(ciphertext)
print(message)

[10/26/23]seed@VM:~$
```

Take a screen shot for your decryption. Ref. decrypt_RSA.py.

3. In this problem, you practice RSA signature: generation and verification.

a. Generate RSA based signature. The functions are described as follows.

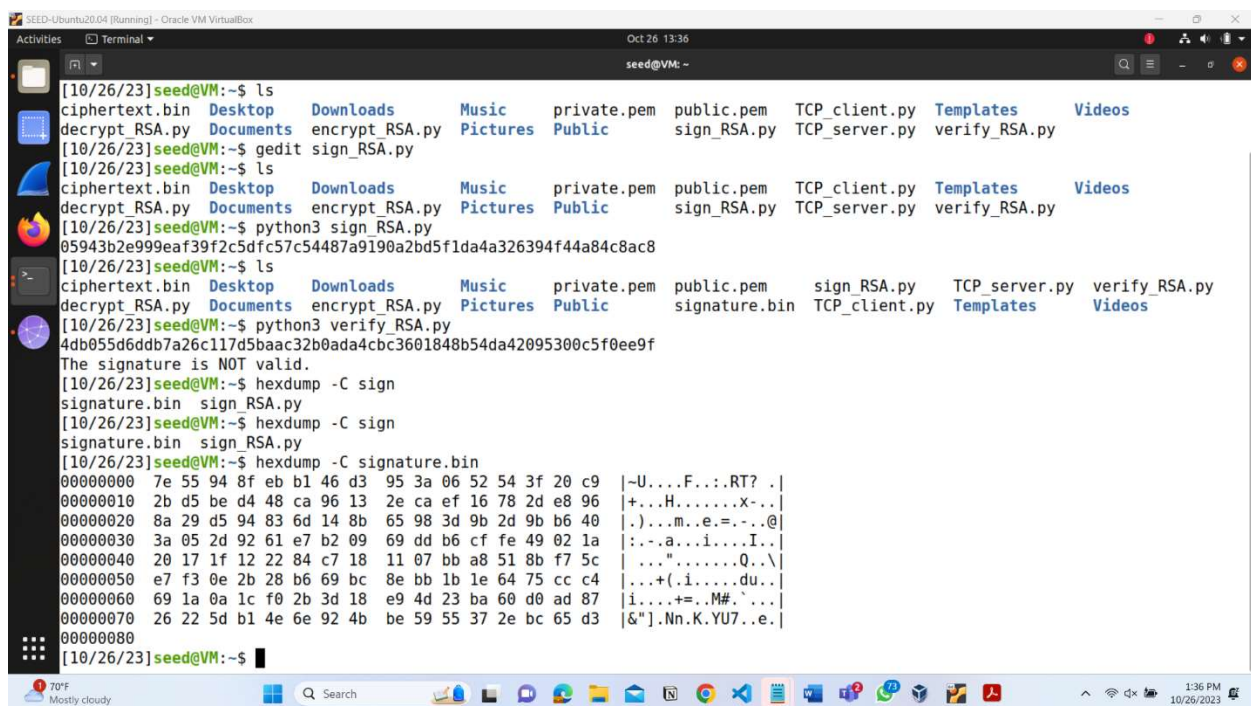
- `Signer=pss.new(RsaKey):`

- o This defines a signing object `signer` with `RsaKey` (imported from your RSA private key file).

- `Signer.sign(hashmessage):`

- o This generates the RSA signature of the hashed message. Here you can use SHA512 to generate the hash value of your message.

M = "I owe you \$2000". Change \$2000 to \$3000 and sign the modified message. Compare both signatures. Are they similar? Save your signature into a file. Take a screen shot for your file content (using hexdump). Ref. `sign_RSA.py`



```
[10/26/23]seed@VM:~$ ls
ciphertext.bin Desktop Downloads Music private.pem public.pem TCP_client.py Templates Videos
decrypt RSA.py Documents encrypt RSA.py Pictures Public sign_RSA.py TCP_server.py verify RSA.py
[10/26/23]seed@VM:~$ gedit sign_RSA.py
[10/26/23]seed@VM:~$ ls
ciphertext.bin Desktop Downloads Music private.pem public.pem TCP_client.py Templates Videos
decrypt RSA.py Documents encrypt RSA.py Pictures Public sign_RSA.py TCP_server.py verify RSA.py
[10/26/23]seed@VM:~$ python3 sign_RSA.py
05943b2e999eaf39f2c5dfc57c54487a9190a2bd5f1da4a326394f44a84c8ac8
[10/26/23]seed@VM:~$ ls
ciphertext.bin Desktop Downloads Music private.pem public.pem sign_RSA.py TCP_server.py verify RSA.py
decrypt RSA.py Documents encrypt RSA.py Pictures Public signature.bin TCP_client.py Templates Videos
[10/26/23]seed@VM:~$ python3 verify_RSA.py
4db055d6ddb7a26c117d5baac32b0ada4cbc3601848b54da42095300c5f0ee9f
The signature is NOT valid.
[10/26/23]seed@VM:~$ hexdump -C sign
signature.bin sign RSA.py
[10/26/23]seed@VM:~$ hexdump -C sign
signature.bin sign RSA.py
[10/26/23]seed@VM:~$ hexdump -C signature.bin
00000000 7e 55 94 8f eb b1 46 d3 95 3a 06 52 54 3f 20 c9 |~U....F...RT? ..|
00000010 2b d5 be d4 48 ca 96 13 2e ca ef 16 78 2d e8 96 |+...H.....x-..|
00000020 8a 29 d5 94 83 6d 14 8b 65 98 3d 9b 2d 9b b6 40 |.)...m..e.=...@|
00000030 3a 05 2d 92 61 e7 b2 09 69 dd b6 cf fe 49 02 1a |:-.a...i....I..|
00000040 20 17 1f 12 22 84 c7 18 11 07 bb a8 51 8b f7 5c |..."......Q..\|
00000050 e7 f3 0e 2b 28 b6 69 bc 8e bb 1b 1e 64 75 cc c4 |...+{i.....du..|
00000060 69 1a 0a 1c f0 2b 3d 18 e9 4d 23 ba 60 d0 ad 87 |i....+=..M#...'..|
00000070 26 22 5d b1 4e 6e 92 4b be 59 55 37 2e bc 65 d3 |&"]..Nn.K.YU7...e|
00000080
```



```
SEED-Ubuntu20.04 [Running] - Oracle VM VirtualBox
Activities Terminal Oct 26 13:39
seed@VM: ~
00000020 8a 29 d5 94 83 6d 14 8b 65 98 3d 9b 2d 9b b6 40 |.)...m..e.=.-..@|
00000030 3a 05 2d 92 61 e7 b2 09 69 dd b6 cf fe 49 02 1a |:.-.a...i....I..|
00000040 20 17 1f 12 22 84 c7 18 11 07 bb a8 51 8b f7 5c |...".....Q..\|
00000050 e7 f3 0e 2b 28 b6 69 bc 8e bb 1b 1e 64 75 cc c4 |...+ (.i.....du..|
00000060 69 1a 0a 1c f0 2b 3d 18 e9 4d 23 ba 60 d0 ad 87 |i....+=..M#.`...|
00000070 26 22 5d b1 4e 6e 92 4b be 59 55 37 2e bc 65 d3 |&"]..Nn.K.YU7..e.|
00000080
[10/26/23]seed@VM:~$ python3 sign_RSA.py
05943b2e999eaf39f2c5dfc57c54487a9190a2bd5f1da4a326394f44a84c8ac8
[10/26/23]seed@VM:~$ python3 verify_RSA.py
4db055d6ddb7a26c117d5baac32b0ada4cbc3601848b54da42095300c5f0ee9f
The signature is NOT valid.
[10/26/23]seed@VM:~$ gedit verify_RSA.py
[10/26/23]seed@VM:~$ python3 sign_RSA.py
05943b2e999eaf39f2c5dfc57c54487a9190a2bd5f1da4a326394f44a84c8ac8
[10/26/23]seed@VM:~$ gedit verify_RSA.py
[10/26/23]seed@VM:~$ python3 verify_RSA.py
8dcafff847a6ffc173913d1a723ddf0aee14ec6fd45bda39a96505c8ab2074f2
The signature is NOT valid.
[10/26/23]seed@VM:~$ gedit verify_RSA.py
[10/26/23]seed@VM:~$ python3 sign_RSA.py
05943b2e999eaf39f2c5dfc57c54487a9190a2bd5f1da4a326394f44a84c8ac8
[10/26/23]seed@VM:~$ gedit sign_RSA.py
[10/26/23]seed@VM:~$ gedit verify_RSA.py
[10/26/23]seed@VM:~$ python3 sign_RSA.py
05943b2e999eaf39f2c5dfc57c54487a9190a2bd5f1da4a326394f44a84c8ac8
[10/26/23]seed@VM:~$ python3 verify_RSA.py
05943b2e999eaf39f2c5dfc57c54487a9190a2bd5f1da4a326394f44a84c8ac8
The signature is valid.
[10/26/23]seed@VM:~$
```

4. In this problem, you will use Diffie-Hellman with authentication to protect the client-server communication. Implement the following functionalities.
- Create two files: TCP client and TCP server, capable to chat with each other using socket.

SERVER:

```
import socket

from Crypto.Random.random import getrandbits

from Crypto.PublicKey import DSA

from Crypto.Util.number import bytes_to_long, long_to_bytes

from Crypto.Hash import SHA256

p =
25822498780869085896559191720030118743297057928292235128306593565406476220168
41194629645353280137831435903171972747559779

g = 2

def compute_shared_key(private_key, other_public_key):
    return pow(other_public_key, private_key, p)
```

```

def hash_shared_key(shared_key):
    return SHA256.new(long_to_bytes(shared_key)).digest()

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
    server_socket.bind(('127.0.0.1', 12345))
    server_socket.listen()

    print("Server is listening...")
    conn, addr = server_socket.accept()
    with conn:
        print("Connected by", addr)
        y = getrandbits(400)
        server_public_key = pow(g, y, p)
        conn.sendall(long_to_bytes(server_public_key))

        client_public_key = bytes_to_long(conn.recv(1024))

        shared_key = compute_shared_key(y, client_public_key)
        sk = hash_shared_key(shared_key)
        print("Secret Key:", sk)

    while True:
        data = conn.recv(1024)
        if not data:
            break
        print("Client:", data.decode('utf-8'))
        message = input("Server: ")
        conn.sendall(message.encode('utf-8'))

```

CLIENT:

```

import socket

from Crypto.Random.random import getrandbits
from Crypto.PublicKey import DSA

```

```

from Crypto.Util.number import bytes_to_long, long_to_bytes
from Crypto.Hash import SHA256

p =
25822498780869085896559191720030118743297057928292235128306593565406476220168
41194629645353280137831435903171972747559779
g = 2

def compute_shared_key(private_key, other_public_key):
    return pow(other_public_key, private_key, p)

def hash_shared_key(shared_key):
    return SHA256.new(long_to_bytes(shared_key)).digest()

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
    client_socket.connect(('127.0.0.1', 12345))

    x = getrandbits(400)
    client_public_key = pow(g, x, p)
    client_socket.sendall(long_to_bytes(client_public_key))

    server_public_key = bytes_to_long(client_socket.recv(1024))

    shared_key = compute_shared_key(x, server_public_key)
    sk = hash_shared_key(shared_key)
    print("Secret Key:", sk)

while True:
    message = input("Client: ")
    client_socket.sendall(message.encode('utf-8'))
    data = client_socket.recv(1024)
    print("Server:", data.decode('utf-8'))

```

- b.** Client and Server execute Diffie-Hellman to generate a shared key and use sha256 to hash this shared key to 32-byte secret **sk**. Diffie-Hellman uses parameters:

p=2582249878086908589655919172003011874329705792829223512830659356540647622016841194629645353280137831435903171972747559779
g=2

```

File "/usr/local/lib/python3.8/dist-packages/Crypto/Cipher/AES.py", line 232, in new
    return _create_cipher(sys.modules[__name__], key, mode, *args, **kwargs)
File "/usr/local/lib/python3.8/dist-packages/Crypto/Cipher/_init_.py", line 79, in _create_cipher
    return modes[mode](factory, **kwargs)
File "/usr/local/lib/python3.8/dist-packages/Crypto/Cipher/mode_eax.py", line 408, in _create_eax_cipher
    return EaxMode(factory, key, nonce, mac_len, kwargs)
File "/usr/local/lib/python3.8/dist-packages/Crypto/Cipher/mode_eax.py", line 103, in _init
    raise ValueError("Nonce cannot be empty in EAX mode")
ValueError: Nonce cannot be empty in EAX mode
[10/26/23]seed@VM:~$ ds
^C[10/26/23]seed@VM:~$ gedit TCP_server.py
[10/26/23]seed@VM:~$ python3 TCP_server.py
Server listening on 127.0.0.1:12349
Connected to client: ('127.0.0.1', 38570)
Key:1666098727003379014253870953825229416295368995711165818718264802919785837856273516313483980129930438917192858027095297984
[10/26/23]seed@VM:~$ python3 TCP_client.py
Secret Key: 1666098727003379014253870953825229416295368995711165818718264802919785837856273516313483980129930438917192858027095297984
You: Harshil Panchal 110096129
\xcd\xdl\x1fg\xea+\xfc\x12'
Tag: b'\x1a\xf2E\x9fyg\xe9K\x1ey\x9eq\xa6Y\x99\xaa\xc3\x92\xb8P\x1c\x95\xdf\x9ar\xf6\xf3\xc5\x96)\xabx'
Message: b'\x00\x00\x00\x19\x00\x00\x00 RVLG}R\x84\xbf\xc2\xb3\x8e\xe8\x10K\xe0\xc5h\xcd\xdl\x1fg\xea+\xfc\x12\x1a\xf2E\x9fyg\xe9K\x1ey\x9eq\xa6Y\x99\xaa\xc3\x92\xb8P\x1c\x95\xdf\x9ar\xf6\xf3\xc5\x96)\xabx\xf6\xfa\x04\x1f\xc5\xffX:z}\x10\xe8\xfa\xae\xal\x03'
ds
^CTraceback (most recent call last):
  File "TCP_client.py", line 63, in <module>
    main()
  File "TCP_client.py", line 46, in main
    data = client_socket.recv(4096)
KeyboardInterrupt
[10/26/23]seed@VM:~$ python3 TCP_client.py
Secret Key: 1666098727003379014253870953825229416295368995711165818718264802919785837856273516313483980129930438917192858027095297984
You: Harshil Panchal 110096129

```

Note: x, y in Diffie-Hellman can be obtained with `Crypto.Random.random.getrandbits(400)`; see <https://pycryptodome.readthedocs.io/en/latest/src/random/random.html> if necessary.

- c. Sender (Client or Server) uses **sk** as a secret key of AES to encrypt your chat message in (a).

This results in ciphertext C and computes tag=sha256(C). In (a), sender sends (C, tag), instead of plain chat message.

SERVER:

```

import socket
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Random.random import getrandbits
from Crypto.Hash import SHA256

# Diffie-Hellman parameters
p =
2582249878086908589655919172003011874329705792829223512830659356540647622016841194629645353280137831435903171972747559779
g = 2

def generate_dh_key():
    private_key = getrandbits(400)
    public_key = pow(g, private_key, p)
    return private_key, public_key

def compute_shared_key(private_key, other_public_key):
    return pow(other_public_key, private_key, p)

def main():
    server_ip = '127.0.0.1'
    server_port = 12349

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((server_ip, server_port))

```



```

server_socket.listen(1)

print("Server listening on {}:{}".format(server_ip, server_port))

conn, addr = server_socket.accept()
print("Connected to client:", addr)

private_key, public_key = generate_dh_key()

# sending the public key
conn.send(str(public_key).encode())

# receive the clients public key
client_public_key = int(conn.recv(4096).decode())

# create a shared key on both the sides
shared_key = compute_shared_key(private_key, client_public_key)
print(f"Key:{shared_key}")
secret_key = SHA256.new(str(shared_key).encode()).digest()

aes_cipher = AES.new(secret_key, AES.MODE_EAX, nonce=b'\x00' * 16)

while True:
    data = conn.recv(4096)
    ciphertext_length = int.from_bytes(data[:4], 'big')
    tag_length = int.from_bytes(data[4:8], 'big')
    ciphertext = data[8:8+ciphertext_length]
    print(f"Cipher Text (C): {ciphertext}")
    received_tag =
data[8+ciphertext_length:8+ciphertext_length+tag_length]
    print(f"Tag: {received_tag}")
    nonce=data[8+ciphertext_length+tag_length:]
    print(f"secret_key (sk): {secret_key}")
    aes_cipher = AES.new(secret_key, AES.MODE_EAX, nonce=nonce)
    decrypted_message = aes_cipher.decrypt(ciphertext)
    print(f"Decrypted Message: {decrypted_message}")
    # Verify the tag
    new_tag = SHA256.new(ciphertext).digest()
    if new_tag != received_tag:
        print("Tag verification failed. Message might be tampered.")
    else:
        print("Client:", decrypted_message.decode()) # Convert to string
for display

if __name__ == "__main__":
    main()

```

CLIENT:

```

import socket
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Random.random import getrandbits
from Crypto.Hash import SHA256

# Diffie-Hellman parameters

```

```

p =
25822498780869085896559191720030118743297057928292235128306593565406476220168
41194629645353280137831435903171972747559779
g = 2

def generate_dh_key():
    private_key = getrandbits(400)
    public_key = pow(g, private_key, p)
    return private_key, public_key

def compute_shared_key(private_key, other_public_key):
    return pow(other_public_key, private_key, p)

def main():
    server_ip = '127.0.0.1'
    server_port = 12349

    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect((server_ip, server_port))

    private_key, public_key = generate_dh_key()
    client_socket.send(str(public_key).encode())

    server_public_key = int(client_socket.recv(4096).decode())
    shared_key = compute_shared_key(private_key, server_public_key)
    print(f"Secret Key: {shared_key}")
    secret_key = SHA256.new(str(shared_key).encode()).digest()
    aes_cipher = AES.new(secret_key, AES.MODE_EAX)

    while True:
        message = input("You: ")

        # Encrypt the message and compute the tag
        ciphertext = aes_cipher.encrypt(message.encode('utf-8'))
        print(f"Cipher Text (C): {ciphertext}")
        tag = SHA256.new(ciphertext).digest()
        message_to_send = len(ciphertext).to_bytes(4, 'big') +
len(tag).to_bytes(4, 'big') + ciphertext + tag + aes_cipher.nonce
        print(f"Tag: {tag}")
        client_socket.send(message_to_send)
        print(f"Message: {message_to_send}")
        data = client_socket.recv(4096)
        ciphertext_length = int.from_bytes(data[:4], 'big')
        tag_length = int.from_bytes(data[4:8], 'big')
        ciphertext = data[8:8+ciphertext_length]
        received_tag = data[8+ciphertext_length:]

        aes_cipher = AES.new(secret_key, AES.MODE_EAX,
nonce=aes_cipher.nonce)
        decrypted_message = aes_cipher.decrypt(ciphertext)

        # Verify the tag
        new_tag = SHA256.new(ciphertext).digest()
        if new_tag != received_tag:
            print("Tag verification failed. Message might be tampered.")
        else:

```

```

        print("Server:", decrypted_message.decode()) # Convert to string
for display

if __name__ == "__main__":
    main()

```

- d. At the receiver, when receiving (C, tag), verify whether tag=sha256(C) holds. If it fails, raise exception; otherwise, use sk as the AES secret to decrypt C. This will recover your chat message.
- Paste your client and server programs in your submission file. Print out sk, C, tag and decrypted *chat message* in (d) for one *chat message*.

```

File "/usr/local/lib/python3.8/dist-packages/Crypto/Cipher/AES.py", line 232, in new
    return _create_cipher(sys.modules[__name__], key, mode, *args, **kwargs)
File "/usr/local/lib/python3.8/dist-packages/Crypto/Cipher/_init_.py", line 79, in _create_cipher
    return modes[mode](factory, **kwargs)
File "/usr/local/lib/python3.8/dist-packages/Crypto/Cipher/mode_eax.py", line 408, in _create_eax_cipher
    return EaxMode(factory, key, nonce, mac_len, kwargs)
File "/usr/local/lib/python3.8/dist-packages/Crypto/Cipher/mode_eax.py", line 103, in __init__
    raise ValueError("Nonce cannot be empty in EAX mode")
ValueError: Nonce cannot be empty in EAX mode
[10/26/23]seed@VM:~$ ds
^C[10/26/23]seed@VM:~$ gedit TCP_server.py
[10/26/23]seed@VM:~$ python3 TCP_server.py
Server listening on 127.0.0.1:12349
Connected to client: ('127.0.0.1', 38570)
Key:166609872700337901425387095382522941629536899571165818718264802919785837856273516313483980129930438917192858027095297984
Cipher Text (C): b'q{x\xd81G\xe4[P\xb5S\xd0\xa90\x9e\x04u\xca\x9e\xe6\xd3\xd4|\r\xb3'
Tag: b"\x08\xdc\x9b0\xa2e3\xf2U^\x82\xe9(\xaa\x1a(\xaf%n-2'\xd8d\x1d\xaaN'\xa2\xbl\t\xd2"
secret key (sk): b'\xfak\x10\x05v\xf4\xb2\t\xcb$\x1a\x0b5'\xe1\xb8Y\xcbg\xd9\xcfkE\x0c\x5b\xebP\xf0\x02T\xdd'
Decrypted Message: b'Harshil Panchal 110096129'
Client: Harshil Panchal 110096129

[10/26/23]seed@VM:~$ python3 TCP_client.py
Secret Key: 166609872700337901425387095382522941629536899571165818718264802919785837856273516313483980129930438917192858027095297984
You: Harshil Panchal 110096129
Cipher Text (C): b'q{x\xd81G\xe4[P\xb5S\xd0\xa90\x9e\x04u\xca\x9e\xe6\xd3\xd4|\r\xb3'
Tag: b"\x08\xdc\x9b0\xa2e3\xf2U^\x82\xe9(\xaa\x1a(\xaf%n-2'\xd8d\x1d\xaaN'\xa2\xbl\t\xd2"
Message: b"\x00\x00\x00\x19\x00\x00\x00 q{x\xd81G\xe4[P\xb5S\xd0\xa90\x9e\x04u\xca\x9e\xe6\xd3\xd4|\r\xb3\x08\xdc\x9b0\xa2e3\xf2U^\x82\xe9(\xaa\x1a(\xaf%n-2'\xd8d\x1d\xaaN'\xa2\xbl\t\xd2'w\xcl\xf5,3 \xec\xf8E:\xec\xde\x8bE"

```