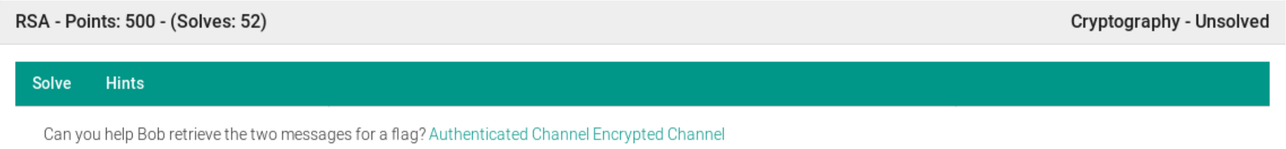
Cryptography – RSA – 500 Points :

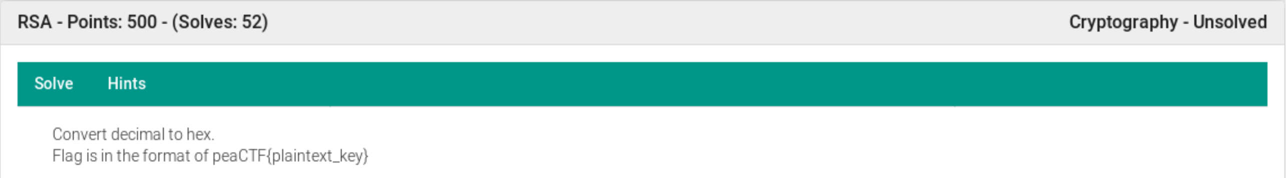


RSA - Points: 500 - (Solves: 52)

Cryptography - Unsolved

Can you help Bob retrieve the two messages for a flag? Authenticated Channel Encrypted Channel

Hint :

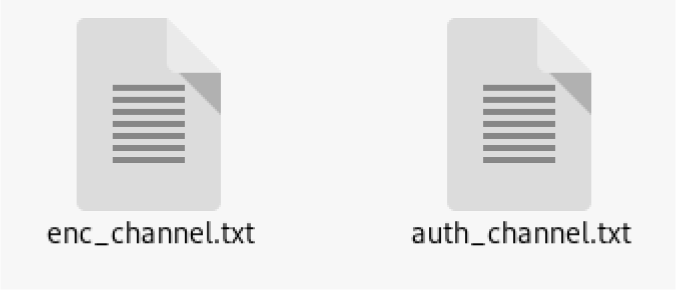


RSA - Points: 500 - (Solves: 52)

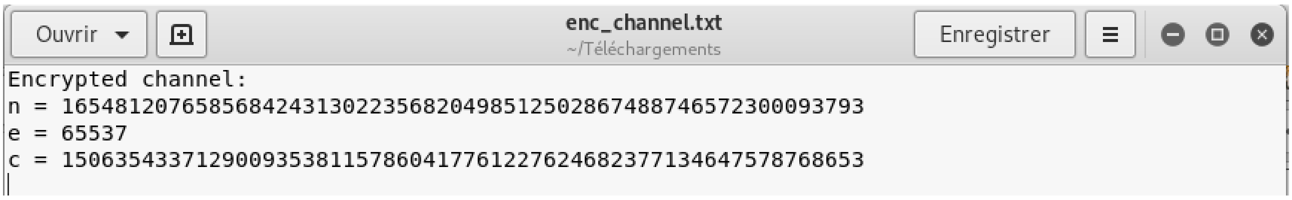
Cryptography - Unsolved

Convert decimal to hex.
Flag is in the format of peaCTF{plaintext_key}

First, download the two files attachement.

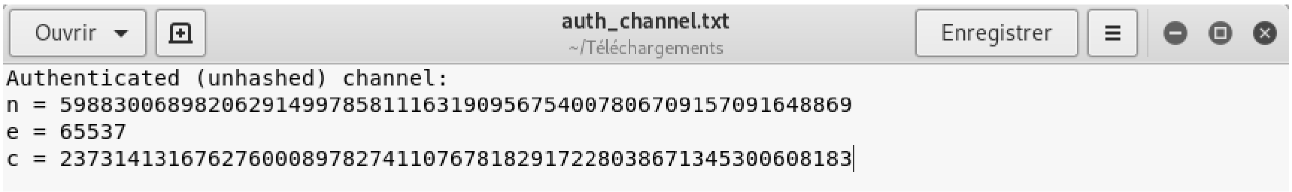


enc_channel.txt content :



```
Encrypted channel:
n = 165481207658568424313022356820498512502867488746572300093793
e = 65537
c = 150635433712900935381157860417761227624682377134647578768653
```

auth_channel.txt content :



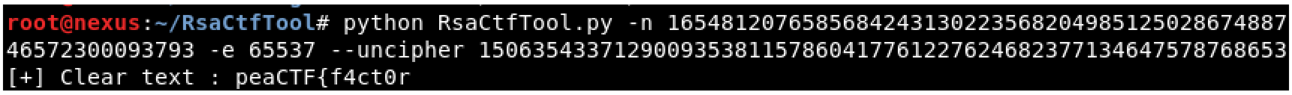
```
Authenticated (unhashed) channel:
n = 598830068982062914997858111631909567540078067091570916488869
e = 65537
c = 237314131676276000897827411076781829172280386713453006081832
```

Part 1 – enc_channel.txt : Solved with RsaCtfTool

First, we will need RsaCtfTool python script. Follow those step for download it and install all dependecies.

1. git clone https://github.com/Ganapati/RsaCtfTool.git
2. cd RsaCtfTool
3. apt-get install libmpc-dev
4. pip2 install gmpy2
5. pip2 install -r optional-requirements.txt
6. git clone https://github.com/hellman/libnum.git
7. cd libnum
8. python setup.py build
9. python setup.py install
10. apt-get install python3-crypto
11. apt-get install python3-gmpy2

Once the tool is ready, use it for decode enc_channel.txt, with the -n parameter (for the modulus), the -e parameter (for the exponent) and the --uncipher parameter (for the private exponent).



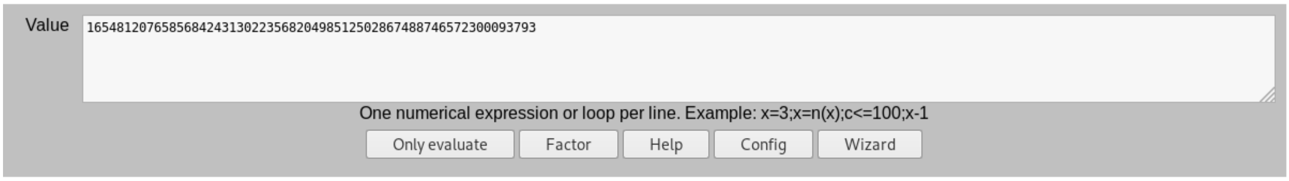
```
root@nexus:~/RsaCtfTool# python RsaCtfTool.py -n 1654812076585684243130223568204985125028674887
46572300093793 -e 65537 --uncipher 150635433712900935381157860417761227624682377134647578768653
[+] Clear text : peaCTF{f4ct0r
```

First part flag : peaCTF{f4ct0r

Part 1 – enc_channel.txt : Solved Manually

First we need to factorise « n » for have the « p » and « q » value, an online tool allow us to do that.

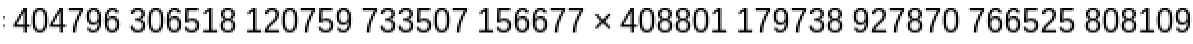
Source : https://www.alpertron.com.ar/ECM.HTM



Value 165481207658568424313022356820498512502867488746572300093793

One numerical expression or loop per line. Example: x=3;x=n(x);c<=100;x-1

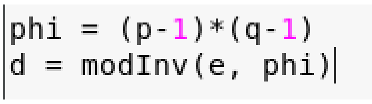
Once factored you will have « p » and « q » values separed by a « x ».



404796 306518 120759 733507 156677 × 408801 179738 927870 766525 808109

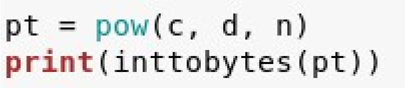
Now it’s time for some scripting with python.

We need to found the opposite of « e » modulo « n » by multiply « p-1*q-1 ».



```
phi = (p-1)*(q-1)
d = modInv(e, phi)
```

And finnaly we need to decrypt « c ».



```
pt = pow(c, d, n)
print(inttobytes(pt))
```

I used libctf for this script.

Source : https://github.com/arisada/libctf

Installation :
1. git clone https://github.com/arisada/libctf.git
2. export PYTHONPATH=/full/path/of/libctf

Code :

```
#!/usr/bin/env python

from libctf import *

def extendedEuclid(a, b):
    x,y, u,v = 0,1, 1,0
    while a != 0:
        q, r = b//a, b%a
        m, n = x-u*q, y-v*q
        b,a, x,y, u,v = a,r, u,v, m,n
    return b, x, y

def modInv(a, m):
    """returns the multiplicative inverse of a in modulo m as a
       positive value between zero and m-1"""
    # notice that a and m need to co-prime to each other.
    linearCombination = extendedEuclid(a, m)
    return linearCombination[1] % m

n = 165481207658568424313022356820498512502867488746572300093793
e = 65537
c = 150635433712900935381157860417761227624682377134647578768653
p,q = 404796306518120759733507156677, 408801179738927870766525808109
p*q == n

phi = (p-1)*(q-1)
d = modInv(e, phi)

pt = pow(c, d, n)
print(inttobytes(pt))
```

Part 2 – auth_channel.txt : Solved Manually

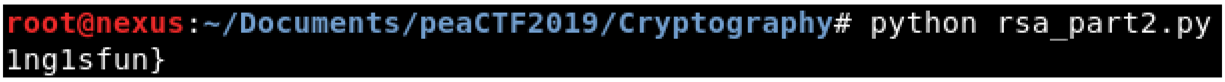
For this part, we only need value of « n, e, c » for decrypt with « inttobytes ».

We must do the opposite operation. In a real case we would not have a flag but a data structure with the signed message hash.

Code :

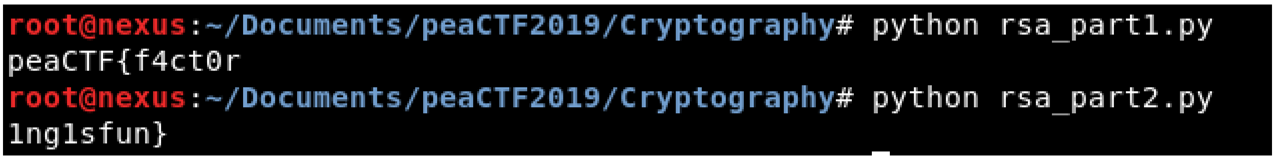
```
#!/usr/bin/env python

from libctf import *

## Quelque Valeur
n = 59883006898206291499785811163190956754007806709157091648869
e = 65537
c = 23731413167627600089782741107678182917228038671345300608183

## Dechiffrer ciphertext
print(inttobytes(RSA(n=n, e=e).encrypt(c)))
```

Execute the script and you will got the second part of the flag.



```
root@nexus:~/Documents/peaCTF2019/Cryptography# python rsa_part2.py
1ng1sfun}
```

Second Part Flag = 1ng1sfun}



```
root@nexus:~/Documents/peaCTF2019/Cryptography# python rsa_part1.py
peaCTF{f4ct0r
root@nexus:~/Documents/peaCTF2019/Cryptography# python rsa_part2.py
1ng1sfun}
```

Flag = peaCTF{f4ct0r1ng1sfun}