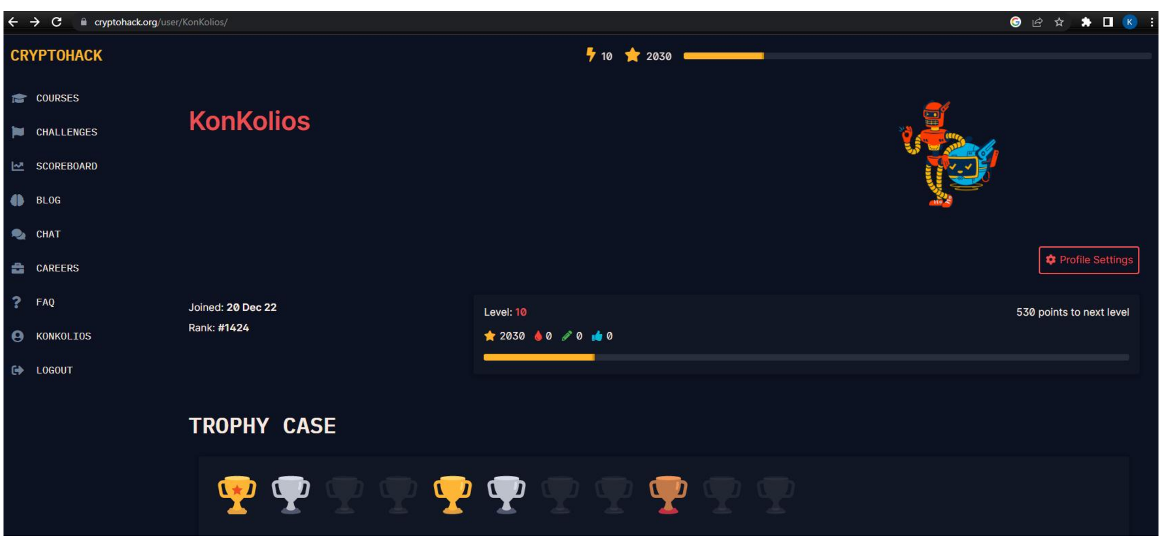
ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ - ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΜΣ «Πληροφορική»

Εργασία Μαθήματος
«Δίκτυα Υπολογιστών»

Όνομα φοιτητή – Αρ. Μητρώου	Κωνσταντίνος Κολιός ΜΠΠΛ - 21032
Ημερομηνία παράδοσης	06/02/2023
Συνολικοί Πόντοι	**1270**



General (235pt)

ASCII (5pt)

```
print(''.join(chr(i) for i in [99, 114, 121, 112, 116, 111, 123, 65, 83, 67,
73, 73, 95, 112, 114, 49, 110, 116, 52, 98, 108, 51, 125]))
```

Hex (5pt)

```
print(bytes.fromhex('63727970746f7b596f755f77696c6c5f62655f776f726b696e675
f776974685f6865785f737472696e67735f615f6c6f747d').decode())
```

Base64(10pt)

```
from base64 import b64encode
print(b64encode(bytes.fromhex('72bca9b68fc16ac7beeb8f849dca1d8a783e8acf967
9bf9269f7bf')).decode())
```

Bytes and Big Integers(10pt)

```
# Remove the "0x" prefix from the hexadecimal string
hex_representation =
hex(115151950638623188999316854888137473957755162872896826364999652827146
37259206269)[2:]

# Convert the hexadecimal string to bytes and decode the bytes to a string
decoded_string = bytes.fromhex(hex_representation).decode()
print(decoded_string)
```

XOR Starter (10pt)

```
#Create a string by performing XOR between each character in the byte string
'label' and the value 13
print('crypto{%s}' % ''.join(chr(i^13) for i in b'label'))
```

XOR Properties (15pt)

```
key1 = 0xa6c8b6733c9b22de7bc0253266a3867df55acde8635e19c73313
key2 = key1 ^ 0x37dcb292030faa90d07eec17e3b1c6d8daf94c35d4c9191a5e1e
key3 = key2 ^ 0xc1545756687e7573db23aa1c3452a098b71a7fbf0fddddde5fc1
flag = key1 ^ key2 ^ key3 ^
0x04ee9855208a2cd59091d04767ae47963170d1660df7f56f5faf
print(bytes.fromhex(hex(flag)[2:]).decode())
```

Favourite byte (20pt)

```
enc =
bytes.fromhex('73626960647f6b206821204f21254f7d694f7624662065622127234f726
927756d')
key = enc[0] ^ ord('c')
print(''.join(chr(c ^ key) for c in enc))
```

You either know, XOR you don't (30pt)

```
from pwn import xor

#Convert hex string to bytes object
enc =
bytes.fromhex('0e0b213f26041e480b26217f27342e175d0e070a3c5b103e2526217f273
42e175d0e077e263451150104')

#Perform XOR between the encoded data and the key and decode the result to string
print(xor(enc, b'myXORkey').decode())
```

Lemur XOR (40pt)

```
from PIL import Image
from pwn import *
lemur = Image.open("lemur.png")
flag = Image.open("flag.png")
leak_bytes = xor(lemur.tobytes(), flag.tobytes())
leak = Image.frombytes(flag.mode, flag.size, leak_bytes)
leak.save('leak.png')
```

Encoding Challenge (40pt)

```python
from pwn import remote
from json import loads, dumps
from base64 import b64decode
from codecs import encode

# Connect to the remote server at socket.cryptohack.org on port 13377
io = remote('socket.cryptohack.org', 13377)
# Start an infinite loop
while True:
    # Receive a line of encoded data from the server and decode it into a
dictionary
    enc = loads(io.recvline().decode())
    print(enc)
    # Check if the key "flag" exists in the received dictionary
    if 'flag' in enc:
        # If it exists, break out of the loop
        break

    # Send the decoded data back to the server
    io.sendline(dumps({"decoded": {
        'base64': lambda e: b64decode(e).decode(),
        'hex'   : lambda e: bytes.fromhex(e).decode(),
        'rot13' : lambda e: encode(e, 'rot_13'),
        'bigint': lambda e: bytes.fromhex(e[2:]).decode(),
        'utf-8' : lambda e: ''.join([chr(c) for c in e])
    }[enc['type']](enc['encoded'])}).encode())
```

Transparency (40pt)

Certificate Transparency Search Tool

cryptohack.org Search

Include expired certificates Include subdomains (partial match)

Export to Excel Export to CSV

Issuer Name ×

Issuer Name	Serial Numbe	Subject CN	Valid Fro	Valid To	Validatic
Let's Encrypt (24)					
Let's Encrypt	37dd8f626dedd2	cryptohack.org	2022-11-13	2023-02-11	non-EV
Let's Encrypt	37dd8f626dedd2	cryptohack.org	2022-11-13	2023-02-11	non-EV
Let's Encrypt	4f663076f0857de	blog.cryptohack.org	2022-12-01	2023-03-01	non-EV
Let's Encrypt	40fc4f12f3fff4dbc	tls2.cryptohack.org	2022-12-02	2023-03-02	non-EV
Let's Encrypt	40fc4f12f3fff4dbc	tls2.cryptohack.org	2022-12-02	2023-03-02	non-EV
Let's Encrypt	4d103c998b027fc	tls3.cryptohack.org	2022-12-02	2023-03-02	non-EV
Let's Encrypt	4d103c998b027fc	tls3.cryptohack.org	2022-12-02	2023-03-02	non-EV
Let's Encrypt	47395cc2f70675f	tls2.cryptohack.org	2022-12-02	2023-03-02	non-EV
Let's Encrypt	47395cc2f70675f	tls2.cryptohack.org	2022-12-02	2023-03-02	non-EV
Let's Encrypt	4deecf121e3e4421	matrix.cryptohack.org	2022-12-10	2023-03-10	non-EV
Let's Encrypt	43c100f8c60f835	tls1.cryptohack.org	2022-12-13	2023-03-13	non-EV
Let's Encrypt	43c100f8c60f835	tls1.cryptohack.org	2022-12-13	2023-03-13	non-EV
Let's Encrypt	4e8bf20dab8a73	thetransparencyflagishe	2022-12-31	2023-03-31	non-EV

Details

General

Issuer Name:	Let's Encrypt
Serial Number:	4e8bf20dab8a733ead409eaadc897b28a66
Issuer CN:	R3
Subject CN:	thetransparencyflagishere.cryptohack.org
Validation:	non-EV
Valid From:	Sat Dec 31 2022 00:00:00 GMT+0200 (Eastern European Standard Time)
Valid To:	Fri Mar 31 2023 00:00:00 GMT+0300 (Eastern European Summer Time)
Signing Algorithm:	SHA-256
Key:	RSA-2048
Issuer DN:	cn=R3,o=Let's Encrypt,c=US
Subject DN:	cn=thetransparencyflagishere.cryptohack.org
Subject Org:	undefined

Log Names (2)

argon2023
xenon2023

Subject Alt Names (1)

thetransparencyflagishere.cryptohack.org


thetransparencyflagishere.cryptohack.org
crypto{thx_redpwn_for_inspiration}

DIFFIE-HELLMAN (380pt)

Diffie-Hellman Starter 1 (10pt)

```
print(pow(209, -1, 991))
```

Diffie-Hellman Starter 2 (20pt)

```
from galois import GF
print(GF(28151).primitive_element)
```

Diffie-Hellman Starter 3 (20pt)

```
p =
2410312426921032588552076022197566074856950548502459942654116941958108831682612228890093858261341614673227141477904012196503648957050582631942730706805009223062734745341073406696246014589361659774041027169249453200378729434170325843778659198143763193776859869524088940195577346119843545301547043747207749969763750084308926339295559968882457872412993810129130294592999947926365264059284647209730384947211681434464714438488520940127459844288859336526896320919633919
a =
9721074438370337962458643162004582468469045984889816058567658904788530882468973454873284910377102192220389309433658486261941098303091793930182167633275721201247601400180386739998376433775904344138666111324039795471506590538973555933944925869784000443754656572960275929483495892164153637226683613286895889965413700975590903351376764115959493358573417971489261516942995759702928098053144314470434694474859576699499890902023202343378903232934018623049865998847 32815
g = 2
print(pow(g,a,p))
```

Diffie-Hellman Starter 4 (30pt)

```python
p =
2410312426921032588552076022197566074856950548502459942654116941958108831
6826122288900938582613416146732271414779040121965036489570505826319427307
0680500922306273474534107340669624601458936165977404102716924945320037872
9434170325843778659198143763193776859869524088940195577346119843545301547
0437472077499697637500843089263392955599688824578724129938101291302945929
9994792636526405928464720973038494721168143446471443848852094012745984428
8859336526896320919633919
A =
7024994321759546827855454126497548290928917435151613399449582140071062529
1840101960595720462672604202133493023241393916394629829526272643847352371
5348398620304103314850874873318092855331950243692872932170834144240968669
2584583864184092319348082133205673559248373092105553222505605661664236182
2852295042658817525804101947316338953458239639109017317157438357756197807
3897484484042557968338534449101595589210690464760204955947727934598253048
8299847663103078045601
b =
1201923325290399034459852253577496302039577040944529672403437843349797684
0167805970589960962221948290951873387728102115996831454482299243226839490
9997137634404121779658615087734205322664846191267105664149142275601037153
3669619321037985057504773038837834826618093494613910047983133983589658344
3691529372703954589071507717917136906770122077739814262298488662138085608
7361034186017508616984173402642138677538346793591914270981958871120645031
04510489610448294420720
print(pow(A, b, p))
```

Diffie-Hellman Starter 5 (40pt)

```python
from Crypto.Util.Padding import unpad
from Crypto.Cipher import AES
from hashlib import sha1

# Define function to decrypt the flag
def decrypt_flag(shared_secret, iv, ciphertext):
    # Use the shared secret to generate a 16-byte key using SHA-1 hash
    key = sha1(str(shared_secret).encode()).digest()[:16]

    # Convert ciphertext from hexadecimal to bytes
    ciphertext = bytes.fromhex(ciphertext)

    # Convert initial vector (IV) from hexadecimal to bytes
    iv = bytes.fromhex(iv)

    # Decrypt the ciphertext using AES in CBC mode
    plaintext = AES.new(key, AES.MODE_CBC, iv).decrypt(ciphertext)

    # Unpad the plaintext and decode it to a string
    return unpad(plaintext, 16).decode()

# Define the values of the Diffie-Hellman key exchange parameters
g = 2
p =
2410312426921032588552076022197566074856950548502459942654116941958108831682612228890093858261341614673227141477904012196503648957050582631942730706805009223062734745341073406696246014589361659774041027169249453200378729434170325843778659198143763193776859869524088940195577346119843545301547043747207749969763750084308926339295559968882457872412993810129130294592999947926365264059284647209730384947211681434464714438488520940127459844288859336526896320919633919
A =
1122187391395429088805643595343734240130162497729319626922379075719903344835288775138092726256105120611590617376085472885586628796850866842996244817428650169240650005552679778301447403644679772065559147812363972160338058822076402196860116434682751657181328884890246888461019436424596554236091119763633160806204719282368797379442175034622656157747743189863758784409788192383460779088641161568318746958174777724771212328208277284248908457691527260275207729014237844
b =
1973950838149070289917857727149208859082493419256509515552190494112984362171906051908249347873362792287858097835318145076613851112206393293580481963396260656768691197379791755317707688618085811103119035485674240392644856613309952219078033008241654699770994942847228318456539853927914802647120912935802749471324804023198121104626411438845777063358591906682406946802611602106095068918427938682976726196259240014030356768721894557679440775421980644994861644314519444

# Calculate the shared secret using the Diffie-Hellman key exchange
```

```python
shared_secret = pow(A, b, p)

# Define the initial vector (IV) in hexadecimal
iv = "737561146ff8194f45290f5766ed6aba"

# Define the ciphertext in hexadecimal
ciphertext =
"39c99bf2f0c14678d6a5416faef954b5893c316fc3c48622ba1fd6a9fe85f3dc72a29c394
cf4bc8aff6a7b21cae8e12c"
print(decrypt_flag(shared_secret, iv, ciphertext))
```

Parameter Injection (60pt)

```python
from Crypto.Util.Padding import unpad
from json import loads, dumps
from Crypto.Cipher import AES
from hashlib import sha1
from pwn import remote

# Function to decrypt the flag
def decrypt_flag(shared_secret, iv, ciphertext):
    # Calculate the key from the shared secret
    # The key is the first 16 bytes of the sha1 hash of the shared secret
    key = sha1(str(shared_secret).encode()).digest()[:16]
   # Convert the ciphertext from hex to bytes
    ciphertext = bytes.fromhex(ciphertext)
    # Convert the iv from hex to bytes
    iv = bytes.fromhex(iv)

    # Decrypt the ciphertext using AES in CBC mode with the key and iv
    plaintext = AES.new(key, AES.MODE_CBC, iv).decrypt(ciphertext)

    # Return the decrypted flag as a string
    return plaintext.decode()

# Connect to the server
io = remote("socket.cryptohack.org", 13371)

# Read the first line from the server and ignore it
io.readline()

# Send a random p, g, and A to the server
io.sendline(dumps({"p":"0x123", "g":"0x123", "A":"0x123"}).encode())

# Read the next line from the server and ignore it
io.readline()

# Send B=1 to the server
# This will make the shared secret always equal to 1
io.sendline(dumps({"B":"0x01"}).encode())

# Wait for the message "from Alice:"
io.readuntil(b"from Alice: ")

# Read the encrypted flag from the server
recv = loads(io.readline())
iv, ciphertext = recv["iv"], recv["encrypted_flag"]
# Calculate the shared secret
shared_secret = 1
print(decrypt_flag(shared_secret, iv, ciphertext))
```

Export-grade (100pt)

```python
from sympy.ntheory.residue_ntheory import discrete_log
from Crypto.Util.Padding import unpad
from json import loads, dumps
from Crypto.Cipher import AES
from hashlib import sha1
from pwn import remote

# Function to decrypt a given ciphertext using a shared secret
def decrypt_flag(shared_secret, iv, ciphertext):
    # Compute the AES key using SHA-1 hash
    key = sha1(str(shared_secret).encode()).digest()[:16]
    # Convert ciphertext from hex to bytes
    ciphertext = bytes.fromhex(ciphertext)
    # Convert initialization vector from hex to bytes
    iv = bytes.fromhex(iv)
    # Decrypt the ciphertext using AES in CBC mode
    plaintext = AES.new(key, AES.MODE_CBC, iv).decrypt(ciphertext)
    # Return the decrypted plaintext as a string
    return plaintext.decode()

# Connect to the remote socket
io = remote("socket.cryptohack.org", 13379)

# Read the welcome message and discard it
io.readline()

# Send the supported protocols to the server
io.sendline(dumps({"supported": ["DH64"]}).encode())

# Read the server's response
io.readline()

# Choose the DH64 protocol
io.sendline(dumps({"chosen": "DH64"}).encode())

# Read the public parameters from Alice
io.readuntil(b"from Alice: ")
recv = loads(io.readline())

# Extract the parameters
p = int(recv["p"], 16)
g = int(recv["g"], 16)
A = int(recv["A"], 16)

# Compute the secret key using the discrete log
a = discrete_log(p, A, g) #since p is small

# Read the public parameters from Bob
io.readuntil(b"from Bob: ")
```

```
recv = loads(io.readline())
B = int(recv["B"], 16)

# Read the encrypted flag and IV from Alice
io.readuntil(b"from Alice: ")
recv = loads(io.readline())
iv = recv["iv"]
ciphertext = recv["encrypted_flag"]

# Compute the shared secret using the Diffie-Hellman key exchange
shared_secret = pow(B, a, p)

# Print the decrypted flag
print(decrypt_flag(shared_secret, iv, ciphertext))
```

Static Client (100pt)

```python
from sympy.ntheory.residue_ntheory import discrete_log
from Crypto.Util.Padding import unpad
from json import loads, dumps
from Crypto.Cipher import AES
from hashlib import sha1
from pwn import remote

# Function to decrypt the flag
def decrypt_flag(shared_secret, iv, ciphertext):
    # Derive the AES key from the shared secret using SHA1
    key = sha1(str(shared_secret).encode()).digest()[:16]
    # Convert the ciphertext from hex to bytes
    ciphertext = bytes.fromhex(ciphertext)
    # Convert the IV from hex to bytes
    iv = bytes.fromhex(iv)
    # Decrypt the ciphertext using AES in CBC mode with the derived key and IV
    plaintext = AES.new(key, AES.MODE_CBC, iv).decrypt(ciphertext)
    # Remove padding from the decrypted plaintext and return the result as a
string
    return unpad(plaintext, 16).decode()

# Connect to the remote server
io = remote("socket.cryptohack.org", 13373)

# Read until the string "from Alice: " and then receive the JSON object from
the server
io.readuntil(b"from Alice: ")
recv = loads(io.readline())

# Extract the values of A, g, and p from the JSON object and store them in the
corresponding variables
A = int(recv["A"], 16)
g = int(recv["g"], 16)
p = int(recv["p"], 16)
io.readuntil(b"from Alice: ")
recv = loads(io.readline())
iv = recv["iv"]
ciphertext = recv["encrypted"]
# The smooth number to send to the server
smooth_p =
0x72b20ce22e5616f923901a946b02b2ad0417882d9172d88c1940fec763b0cdf02ca5862c
fa70e47fb8fd10615bf61187cd564a017355802212a526453e1fb9791014f070d77f8ff4dd
54a6d1d58969293734e0b6bc22f3ceea788aa33be35eed4bdc1c8ceb94084399d98e13e69a
2b9fa6c5583836a15798ba1a10edd81160a15662cdf587df6b816c570f9b11a466d1b4c328
180f614e964f3a5ec61c3f2b759b21687a122f9faefc86fe69a3efd14829639596eb7f2de6
eab6b444d06233d34d0651e6fed17db4d0025e58db7cad8824c3e93ed24df588a0a4530be2
676e995f870172b9e765ec2886bce140000000000000000000000000000000000000000000
000000000000000000000000000000000001
io.sendline(dumps({"g":hex(g),"A": hex(A),"p": hex(smooth_p)}).encode())
```

```python
io.readuntil(b"Bob says to you: ")
B = int(loads(io.readline())["B"], 16)

# Calculate b using the discrete log function
b = discrete_log(smooth_p, B, 2)
shared_secret = pow(A, b, p)

# Decrypt the flag using the shared secret, IV, and ciphertext
print(decrypt_flag(shared_secret, iv, ciphertext))
```

RSA (335PT)

RSA 1 (10pt)

```
print(pow(101, 17, 22663))
```

RSA 2 (15pt)

```
print(pow(12, 65537, 17*23))
```

RSA 3 (20pt)

```
p = 857504083339712752489993810777
q = 1029224947942998075080348647219
print((p-1)*(q-1))
```

RSA 4 (20pt)

```
print(pow(65537, -1,
882564595536224140639625987657529300394956519977044270821168))
```

RSA 5 (20pt)

```
N = 882564595536224140639625987659416029426239230804614613279163
c = 77578995801157823671636298847186723593814843845525223303932
d = 121832886702415731577073962957377780195510499965398469843281
print(pow(c,d,N))
```

Factoring (15pt)

```
from sympy import factorint

N = 510143758735509025530880200653196460532653147
factors = factorint(N)
print(factors)
```

Inferius Prime (30pt)

```
# Modulus in RSA encryption
n = 742449129124467073921545687640895127535705902454369756401331

# Public key exponent in RSA encryption
e = 3

# Ciphertext message
c = 39207274348578481322317340648475596807303160111338236677373

# First prime number used to generate n
p = 752708788837165590355094155871

# Second prime number used to generate n
q = 986369682585281993933185289261

# Decrypted message from ciphertext
print(bytes.fromhex(hex(pow(c, pow(e, -1, (p-1)*(q-1)), n))[2:]).decode())
```

Monoprime (30pt)

```
n =
171731371218065444125482536302245915415603318380280392385291836472299752747934607246477508507827284075763910264995326010251268493630501989810855418416643352631102434317900028697993224868629935657273062472544675693365930943308086634291936846505861203914449338007760990051788980485462592823446469606824421932591
e = 65537
c =
161367550346730604451454756189028938964941280347662098798775466019463375610700074840105776873791605070092554650190486030367121011578171525759600774739890458414593857709994072516290998135846956596662071379067305011746842247628316996977338024343628757374524136260758515864509435302781735938531030576289086798942
print(bytes.fromhex(hex(pow(c, pow(e, -1, n-1), n))[2:]).decode())
```

Square Eyes (35pt)

```
import math

# n is the modulus used in the encryption process
n =
535860808044009550029177135708168016201451343147313565371014459027743491739422885443084705720731409713775527993719682583669164873806842043288439828071789970694759080842162253955259590552283047728782812946845160334801782088068154453021936721710269050985805054692096738777321796153384024897615594493453068138341203673749514094546000253631902991617197847584519694152122765406982133526594928685232381934742152195861380221224370858128736975959176861651044370378539093990198336298572944512738570839396588590096813217791191895941380464803377602779240663133834952329316862399581950590588006371221334128215409197603236942597674756728212232134056562716399155080108881105952768189193728827484667349378091100068224404684701674782399200373192433062767622841264055426035349769018117299620554803902490432339600566432246795818167460916180647394169157647245603555692735630862148715428791242764799469896924753470539857080767170052783918273180304835318388177089674231640910337743789750979216202573226794240332797892868276309400253925932223895530714169648116569013581643192341931800785254715083294526325980247219218364118877864892068185905587410977152737936310734712276956663192182487672474651103240004173381041237906849437490609652395748868434296753449

# p is the square root of n
p = int(math.isqrt(n))

# e is a public key used in the encryption process
e = 65537

# c is the ciphertext (the encrypted message)
c =
2225028859741824295009483898405634152915347268913545739073295125564396328109219279052204867278074366680359293024427542259527866024922504480203412177336464729822862223388605660761619777860956759445522323914812787820193462839009596771670266368302520670487597202516718111058647569724495547940966885025629807079171218371644528053562232396674283745310132242492367274184667845174514466834132589971388067076980563188513333661165819462428837210575342101036356974189393390097403614434491507672459254969638032776897417674577487775755539964915035731988499983726435005007850876000232292458554577437739427313453671492956668188219600633325930981748162455965093222648173134777571527681591366164711307355510889316052064146089646772869610726671696699221157 98
```

```
58343256636614000348314424312091234787780782558468305222263909641198187849
03333020048870521276556916349557185145935552039892821420628508088395488188 8
66850926245549088928386256045359866291952222493514569443588539650078065153
08293770303716119211812073622173978053039621121001907837630619099458897178
78397740711340114311597934724670601992737526668932871436226135393872881664
51122278956525605913800265140387548492071131652253626060425526953216159482
4301047729082877262812899724246757871448545439896

# Decrypt the ciphertext using the private key
# pow(c, pow(e, -1, p * (p - 1)), n) calculates the modular exponentiation of
c^(e^-1) mod n
# The hex function is used to convert the result to a hexadecimal representation
# The [2:] is used to remove the 0x prefix from the hexadecimal representation
# bytes.fromhex converts the hexadecimal representation to bytes
# decode is used to convert the bytes to a string
decrypted_message = bytes.fromhex(hex(pow(c, pow(e, -1, p * (p - 1)),
n))[2:]).decode()
print(decrypted_message)
```

Salty (20pt)

```
c =
449812307182121836042747859257931454426554650252645540460282513111164494127
485
print(bytes.fromhex(hex(c)[2:]).decode())
```

Modulus Inutilis (50pt)

```
from sympy import cbrt
# The number to be extracted the cube root of
c =
243251053617903760309941844835411292373350655973075480264001352919865180151
222189820473358411037759381328642957324889519192337152355302808400638052620
580409813222660643570085177957

# Calculate the cube root of c using cbrt function
# Then convert the result to hexadecimal string representation
# Finally, decode the hexadecimal string to get the final result
print(bytes.fromhex(hex(cbrt(c))[2:]).decode())
```

Everything is Big (70pt)

```
n =
0x8da7d2ec7bf9b322a539afb9962d4d2ebeb3e3d449d709b80a51dc680a14c87ffa863edf
c7b5a2a542a0fa610febe2d967b58ae714c46a6eccb44cd5c90d1cf5e271224aa3367e5a13
305f2744e2e56059b17bf520c95d521d34fdad3b0c12e7821a3169aa900c711e6923ca1a26
c71fc5ac8a9ff8c878164e2434c724b68b508a030f86211c1307b6f90c0cd489a27fdc5e61
90f6193447e0441a49edde165cf6074994ea260a21ea1fc7e2dfb038df437f02b9ddb7b524
4a9620c8eca858865e83bab3413135e76a54ee718f4e431c29d3cb6e353a75d74f831bed2c
c7bdce553f25b617b3bdd9ef901e249e43545c91b0cd8798b27804d61926e317a2b745
e =
0x86d357db4e1b60a2e9f9f25e2db15204c820b6e8d8d04d29db168c890bc8a6c1e31b9316
c9680174e128515a00256b775a1a8ccca9c6936f1b4c2298c03032cda4dd8eca1145828d31
466bf56bfcf0c6a8b4a1b2fb27de7a57fae7430048d7590734b2f05b6443ad60d896068024
09d2fa4c6767ad42bffae01a8ef1364418362e133fa7b2770af64a68ad50ad8d2bd5cebb99
ceb13368fb31a6e7503e753f8638e21a96af1b6498c18578ba89b98d70fa482ad137d28fe7
01b4b77baa25d5e84c81b26ee9bddf8cbb51a071c60dd57714de379cd4bc14932809ba1852
4a0a18e4133665cfc46e2c4fcfbc28e0a0957e5513a7307c422b87a6182d0b6a074b4d
c =
0x6a2f2e401a54eeb5dab1e6d5d80e92a6ca189049e22844c825012b8f0578f95b269b1964
4c7c8af3d544840d380ed75fdf86844aa8976622fa0501eaec0e5a1a5ab09d3d1037e55501
c4e270060470c9f4019ced6c4e67673843daf2fd71c64f3dd8939ae322f2b79d283b338205
2d076ebe9bb50b0042f1f7dd7beadf0f5686926ade9fc8370283ead781a21896e7a878d99e
77c3bb1f470401062c0e0327fd85da1cf12901635f1df310e8f8c7d87aff5a01dbbecd739c
d8f36462060d0eb237af8d613e2d9cebb67d612bcfc353ef2cd44b7ac85e471287eb04ae9b
388b66ea8eb32429ae96dba5da8206894fa8c58a7440a127fceb5717a2eaa3c29f25f7
p =
1155072904368046818539725137858552290923340803568747178834342382355326644
1400698329642751264652299576298636563034566154936812894976200811116395627
6428241298812018796646817754026642839135083991257146569562480983392095388
3885778004271138804076391318427661198573263589390952751421487658123810872
4839614805837919
q =
1548158388307357562668398970021323145386759091352499548525421001795901920
5525710060175952340140938004946326626532378040655469224801222448546946835
5325698979825072726357567352976501667578692853621821116743497998613306224
8575206392716857722110467090148761044817669090365223492235063268529437099
40047836080845531

# convert c to the corresponding bytes and decode it
# using the pow() function to calculate c^(e^-1) % ((p-1)*(q-1)) modulo n
print(bytes.fromhex(hex(pow(c, pow(e, -1, (p-1)*(q-1)), n))[2:]).decode())
```

Crossed Wires (100pt)

```
# Import required modules from Crypto library and gmpy2 library
from Crypto.Util.number import long_to_bytes
from gmpy2 import *

# Modulus
N =
21711308225346315542706844618441565741046498277716979943478360598053144971
37995691657537034344898860190585457202963584662625948729795030523166110985
58549474942091352055892586435179615215949243684986720642932082308024410773
90193682958095111922082677813175804775628884377724377647428385841831277059
27417298228054523776555996922870750685756121526849102409706392033772178367
30605301816371615774015891265585561825468967833073705172750465227040473857
86111489447064794210010802761708615907245523492585896286374996088089317826
16279827852829620697790027443182982920610322717183927088747643689949442837
1323874689055690729986771
# Private key
d =
27344116772511480307231380057161097338388665453755276020182551593196310266
53190783670493107936401603981429171880504360560494771017246468702902647370
95422031245254134285874759057627377510787045085353371711668432697626300643
57333820458079718907620187477295740210574303317780339823591848381597473312
36538501849965329264774927607570410347019418407451937875684373454982306923
17840316121681723789096265121471883195421520063765110390720934790085782472
26532171795481481456871813772205448645218082301227309674529814353553349321
04265488075777638608041325256776275200067541533022527964743478554948792578
057708522350812154888097
# Public key
e = 0x10001

# Different public exponents
e1, e2, e3, e4, e5 = 106979, 108533, 69557, 97117, 103231
e_p = [e1, e2, e3, e4, e5]

# Ciphertext
c =
20304610279578186738172766224224793119885071262464464448863461184092225736
05474797698517967390544150268912621628289770450874540379905473412158396885
39997916042816151541007362591314534243853643246302296711853437781728072626
40709301838274824603101692485662726226902121105591137437331463201881264245
```

```
562214012160875177167442010952439360623396658974413900469093836794752270399520074596329058725874834082188697377597949405779039139194196065364426213208345461407030771089787529200057105746584493554722790592530472869581310117300343461207750821737840042745530876391793484035024644475535353227851321505537398888106855012746117

# Calculating phi value
tmp = e*d - 1
k = 2
phi = (tmp // k)

# Initializing an array to store the private exponents
d_p = [0, 0, 0, 0, 0]

# Calculating the private exponents for each of the public exponents
for j in range(5):
    d_p[j] = invert(e_p[j], phi)

# Decrypting the ciphertext
tmpC = c
for j in range(5):
    tmpC = pow(tmpC, d_p[j], N)

# Converting the decrypted message to bytes format
tmpC = (long_to_bytes(tmpC))

# Printing the decrypted message
print(tmpC)
```

Signing Server (60pt)

```python
# Import the necessary libraries
from pwn import *
from json import dumps, loads

# Connect to the remote server
io = remote("socket.cryptohack.org", 13374)

# Receive the initial message from the server
io.recvline()

# Send a request to get the secret
io.sendline(dumps({"option": "get_secret"}).encode())

# Sign the secret and send it back to the server
signed_secret = loads(io.recvline())["secret"]
io.sendline(dumps({"option": "sign", "msg": signed_secret}).encode())

# Decode and print the signature received from the server
signature = loads(io.recvline())["signature"]
print(bytes.fromhex(signature[2:]).decode())
```

Let’s Decrypt (80pt)

```
from pwn import *
from json import dumps, loads
from pkcs1 import emsa_pkcs1_v15

# Define the message and signature
msg = "I am Mallory.*own CryptoHack.org"
sig =
0x55c231eebc642cd1e44199e10937ee8b9e93c0c2d10a18b7b53a207fb1ddd4e6c2e08368
a1943187bb1efe0378567340a0851710c426f609aa79d3b5bb3f8efe7f531cfdb54a9fba9e
77e3ca2adcecdc299ebf601bd8926dd6ed4e7e71f96ef61cc041159eb0584ff4ce9f0d9e5c
b49a91ba15226740f378340e40805aff2e20e275b783aa43a0ac670ec1af2d4e834acceda1
89add6ed7daf64ed8f9f9718f030c8a7d64afee7cf33beef5f790611eaef40e7c978e2355f
3039a6df4f38113ce83ed669a733ce6a93e1fb04fdd6c28815beb6b62f886a47150fbdd346
68aa7ff55787874a7b6787a5942da4d73b3197eb792b39d0e338f48fc5f4c01a16a178

# Calculate the value of "m"
m = int(bytes.hex(emsa_pkcs1_v15.encode(msg.encode(), 256)), 16)

# Connect to the remote server
io = remote("socket.cryptohack.org", 13391)

# Receive the initial message from the server
io.recvline()

# Send a request to verify the signature
io.sendline(dumps({"option": "verify","msg": msg,"N": hex(sig-m),"e":
"0x01"}).encode())

# Print the result of the verification
print(loads(io.recvline())["msg"])
```

Blinding Light (120pt)

```python
from pwn import *
from json import loads, dumps

# Public key
n =
0x954e1412ba207b8a246ea515e81425aeb5471cf5062b6497b2c76312ccf150498779ca54
0464b09fe573df68b0cfdcac124ba799b8546b45b49eaae9fadd630d1b5562a9993c6a3da7
2d5222e24aa6e1f9c663bfd07f31f0cdef87a54f2fbf7151afc3fd329bd16692dcfa6794c3
d94d00fb2e11b49557a491be3e510f0c3e22163487df65e54d68f43a3ecea44e48dc929f2d
321c6bfdb2c6c233c704e0618041ace0be91f637f423e6161b36a1fe0f04445ee1f48dc596
0659706bbcb97c1667c5f17d0f2395dad348a88f3efb7fa06f99f7963749679eb697cd178f
ce6f65cfee5b6c9c36096c96f5b5532a6a3b44127afe27f10015dd71a644d455f800d5
e = 0x10001

#Encode the message to be signed
m = int(bytes.hex(b"admin=True"), 16)

#Connect to the remote server
io = remote("socket.cryptohack.org", 13376)

#Receive the initial message from the server and print it
io.recvline()

#Send a request to sign the message
io.sendline(dumps({"option":"sign","msg":str(hex((2**e*m) %
n))[2:]}).encode())

#Receive the signature from the server
s = int(loads(io.recvline())["signature"],16)*pow(2,-1,n) % n

#Send a request to verify the signature
io.sendline(dumps({"option":"verify","msg":str(hex(m)[2:]),"signature":str
(hex(s)[2:])}).encode())

#Receive the result of the verification and print it
print(loads(io.recvline())["response"])
```