

# *ΕΡΓΑΣΙΑ ΜΑΘΗΜΑΤΟΣ*

---

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

Τμήμα Πληροφορικής



Μάθημα: «ΚΡΥΠΤΟΓΡΑΦΙΑ (5ο εξ.)»

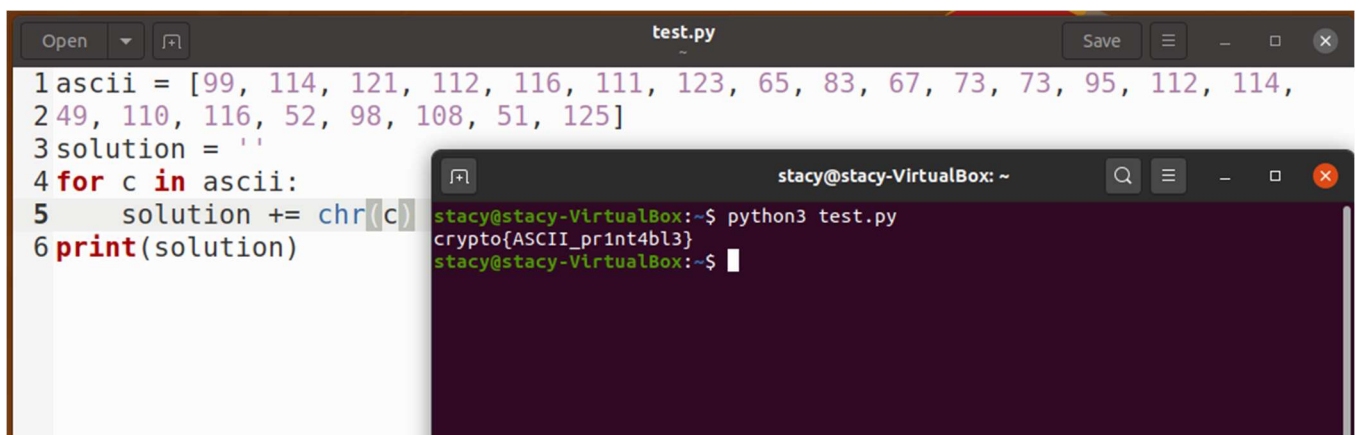
Π18101 – ΑΝΑΣΤΑΣΙΑ ΙΩΑΝΝΑ ΜΕΞΑ

Έχω λύσει συνολικά 22 challenges από την ιστοσελίδα του Cryptohack συγκεντρώνοντας 600 πόντους. Συνολικά στο προφίλ μου (<https://cryptohack.org/user/anastasiamexa/>) θα δείτε να αναγράφεται ο αριθμός 610 στο σκορ, ωστόσο δεν μετράω τους έξτρα 10 πόντους, διότι προέρχονται από την κατηγορία δοκιμασιών Introduction. Πιο συγκεκριμένα έχω λύσει 2 δοκιμασίες από την κατηγορία General, 1 από το Mathematics, 14 από το RSA και 5 από το Diffie-Hellman.

Παρακάτω αναλύονται όλες οι λύσεις και με screenshots που δείχνουν το αποτέλεσμα των scripts. Επίσης, μέσα στο zip της εργασίας υπάρχει ο φάκελος με το όνομα Scripts, που περιέχει μέσα όλα τα αρχεία python με τον κώδικα για κάθε challenge.

## GENERAL

- ENCODING - ASCII

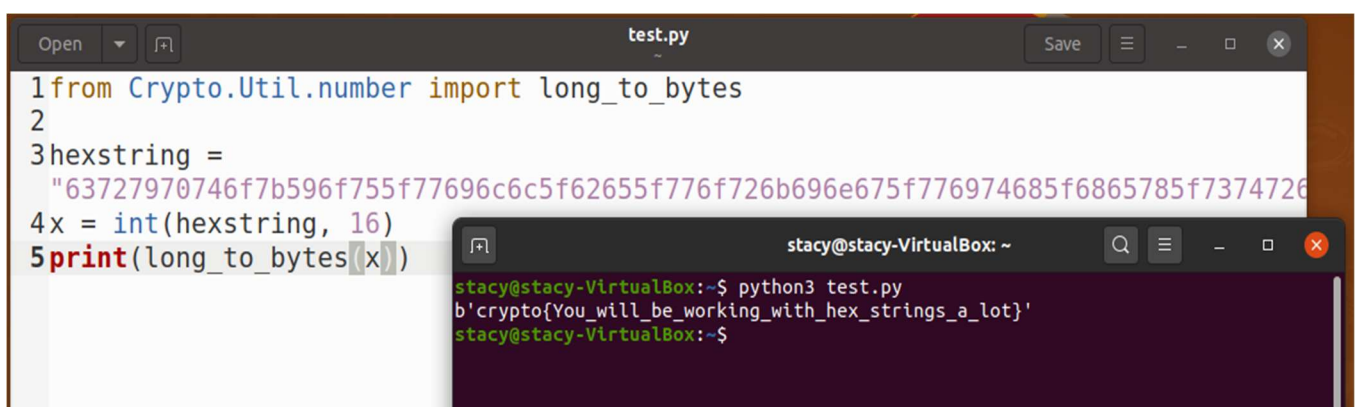


```
test.py
1 ascii = [99, 114, 121, 112, 116, 111, 123, 65, 83, 67, 73, 73, 95, 112, 114,
2 49, 110, 116, 52, 98, 108, 51, 125]
3 solution = ''
4 for c in ascii:
5     solution += chr(c)
6 print(solution)

stacy@stacy-VirtualBox: ~
stacy@stacy-VirtualBox:~$ python3 test.py
crypto{ASCII_print4bl3}
```

Η δοκιμασία ζητάει να μετατρέψουμε τους αριθμούς στον αντίστοιχο χαρακτήρα ASCII με σκοπό, να βρούμε το flag. Εύκολα λοιπόν, αποθηκεύοντας σε μια λίστα τους αριθμούς που μας δίνονται, διαπερνάω ένα-ένα τα στοιχεία της λίστας μετατρέποντάς το, στον αντίστοιχο ASCII χαρακτήρα με την συνάρτηση chr() και προσθέτω το αποτέλεσμα κάθε φορά στο string solution. Τέλος, εκτυπώνω το string.

- ENCODING - Hex



```
test.py
1 from Crypto.Util.number import long_to_bytes
2
3 hexstring =
4 "63727970746f7b596f755f77696c6c5f62655f776f726b696e675f776974685f6865785f7374726
5 print(long_to_bytes(x))

stacy@stacy-VirtualBox: ~
stacy@stacy-VirtualBox:~$ python3 test.py
b'crypto{You_will_be_working_with_hex_strings_a_lot}'
```

Η δοκιμασία ζητάει να αποκρυπτογραφήσουμε το hexstring σε bytes ώστε να βρούμε το flag. Χρησιμοποιώντας λοιπόν την συνάρτηση int(), κάνω casting το δεκαεξαδικό string σε bytes και τέλος χρησιμοποιώ την συνάρτηση long\_to\_bytes για να τυπώσω στην κατάλληλη μορφή το flag.

## MATHEMATICS

- MODULAR MATH - Chinese Remainder Theorem



```
1 from functools import reduce
2
3 def chinese_remainder(n, a):
4     sum=0
5     prod=reduce(lambda a, b: a*b, n)
6     for n_i, a_i in zip(n,a):
7         p=prod//n_i
8         sum += a_i* mul_inv(p, n_i)*p
9     return sum % prod
10
11 def mul_inv(a, b):
12     b0= b
13     x0, x1= 0,1
14     if b== 1: return 1
15     while a>1 :
16         q=a// b
17         a, b= b, a%b
18         x0, x1=x1 -q *x0, x0
19     if x1<0 : x1+= b0
20     return x1
21 """
22 x mod 5 ≡ 2
23 x mod 11 ≡ 3
24 x mod 17 ≡ 5
25 ...
26 x mod 935 ≡ a
27 """
28 n=[5,11,17]
29 a=[2,3,5]
30 print(chinese_remainder(n,a))
```

```
stacy@stacy-VirtualBox: ~
stacy@stacy-VirtualBox:~$ python3 test.py
872
stacy@stacy-VirtualBox:~$
```

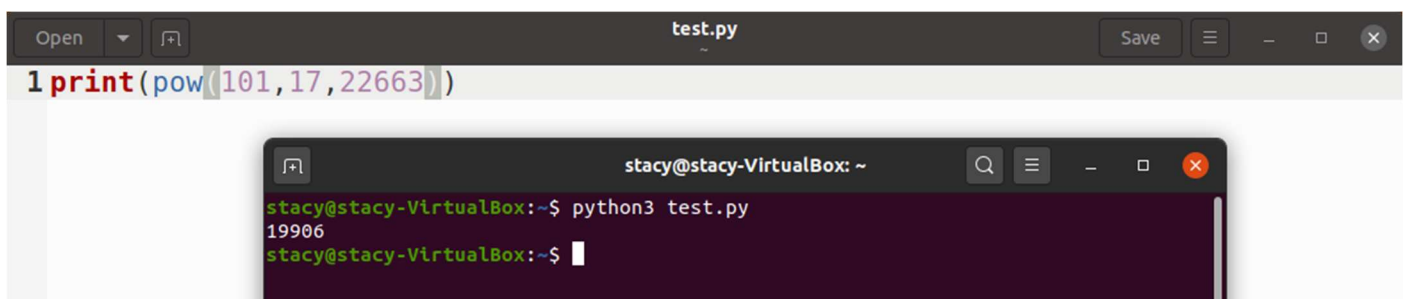
Η δοκιμασία ζητάει να λύσουμε το Κινεζικό Θεώρημα Υπολοίπων και να βρούμε τον αριθμό  $a$ , για τον οποίο ισχύει  $x \equiv a \pmod{935}$ . Αυτό κάνει και ο παραπάνω κώδικας, όπου ως αποτέλεσμα παίρνουμε ότι  $a = 872$ .

Σημείωση: Βρήκα τον κώδικα στην σελίδα

[https://rosettacode.org/wiki/Chinese\\_remainder\\_theorem#Python](https://rosettacode.org/wiki/Chinese_remainder_theorem#Python)

## RSA

- STARTER - RSA Starter 1

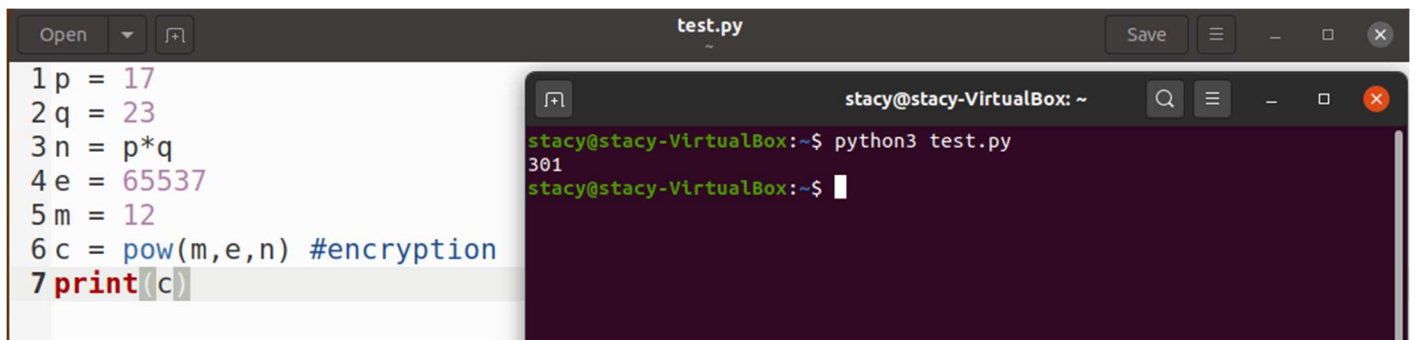


```
1 print(pow(101,17,22663))
```

```
stacy@stacy-VirtualBox: ~
stacy@stacy-VirtualBox:~$ python3 test.py
19906
stacy@stacy-VirtualBox:~$
```

Η δοκιμασία ζητάει να λύσουμε το  $101^{17} \pmod{22663}$ . Αυτό επιτυγχάνεται εύκολα, χρησιμοποιώντας την συνάρτηση `pow()` όπως φαίνεται στο παραπάνω screenshot. Το αποτέλεσμα είναι 19906.

- STARTER - RSA Starter 2



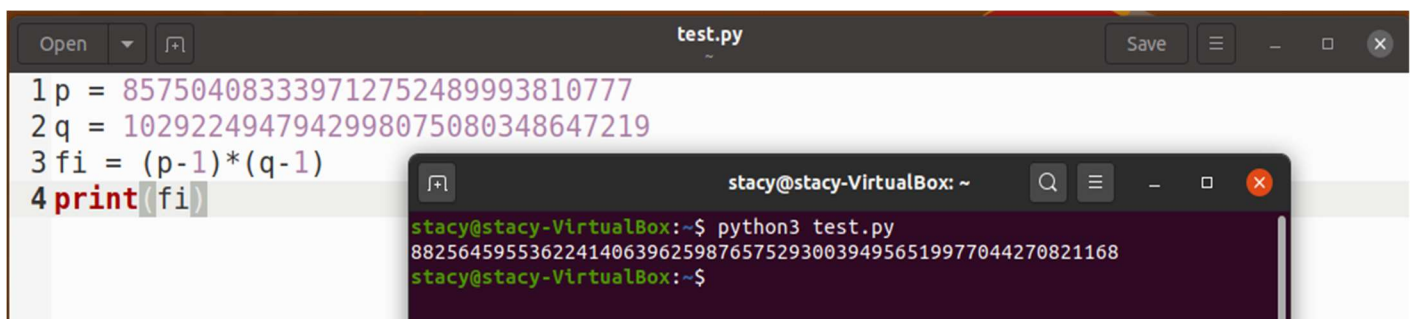
The screenshot shows a code editor window titled 'test.py' with the following Python code:

```
1 p = 17
2 q = 23
3 n = p*q
4 e = 65537
5 m = 12
6 c = pow(m,e,n) #encryption
7 print(c)
```

Below the code editor is a terminal window titled 'stacy@stacy-VirtualBox: ~'. It shows the command 'python3 test.py' being executed, resulting in the output '301'.

Η δοκιμασία ζητάει να κρυπτογραφήσουμε τον αριθμό 12 (m) και μας δίνει τους παράγοντες του n, αλλά και το e. Απλά εφαρμόζουμε την διαδικασία κρυπτογράφησης του αλγορίθμου RSA, όπως φαίνεται παραπάνω και εκτυπώνουμε το αποτέλεσμα (c = 301).

- STARTER - RSA Starter 3



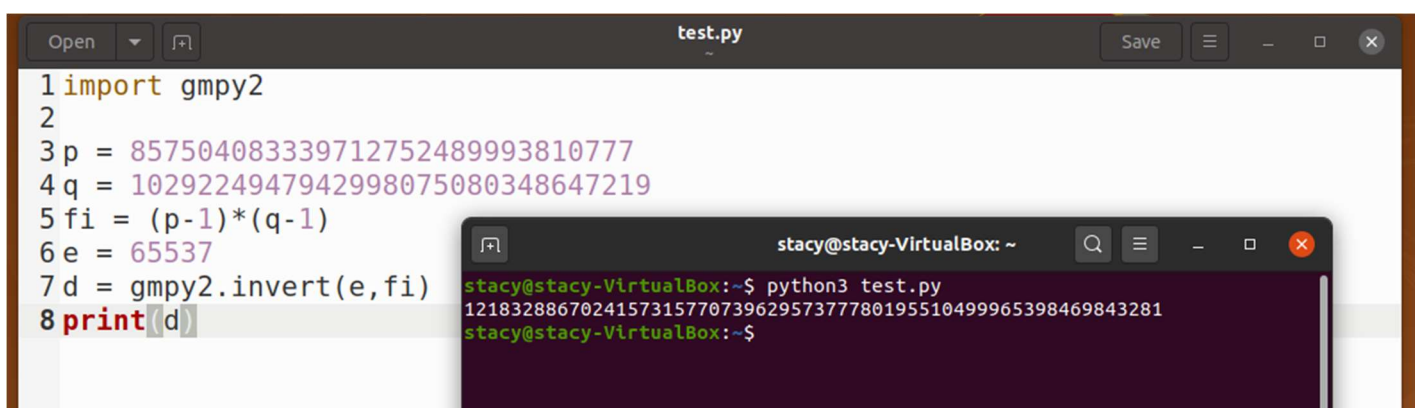
The screenshot shows a code editor window titled 'test.py' with the following Python code:

```
1 p = 857504083339712752489993810777
2 q = 1029224947942998075080348647219
3 fi = (p-1)*(q-1)
4 print(fi)
```

Below the code editor is a terminal window titled 'stacy@stacy-VirtualBox: ~'. It shows the command 'python3 test.py' being executed, resulting in the output '882564595536224140639625987657529300394956519977044270821168'.

Η δοκιμασία ζητάει να βρούμε το  $\phi(n)$  (Συνάρτηση του Euler). Ξέρουμε ότι  $n = p*q$ , άρα ο τύπος είναι  $\phi(n) = (p-1)*(q-1)$ . Τα p, q μας δίνονται οπότε απλά εφαρμόζω τον τύπο και εκτυπώνω το αποτέλεσμα.

- STARTER - RSA Starter 4



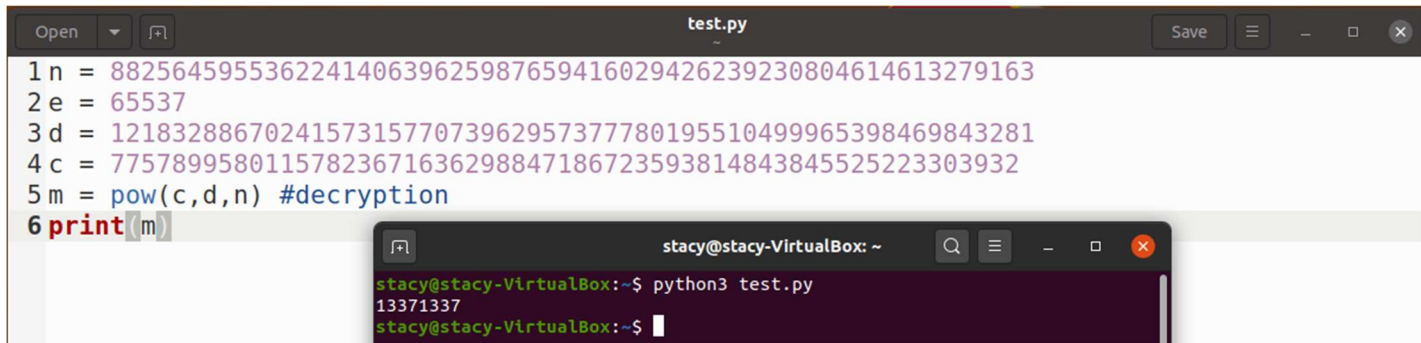
The screenshot shows a code editor window titled 'test.py' with the following Python code:

```
1 import gmpy2
2
3 p = 857504083339712752489993810777
4 q = 1029224947942998075080348647219
5 fi = (p-1)*(q-1)
6 e = 65537
7 d = gmpy2.invert(e,fi)
8 print(d)
```

Below the code editor is a terminal window titled 'stacy@stacy-VirtualBox: ~'. It shows the command 'python3 test.py' being executed, resulting in the output '121832886702415731577073962957377780195510499965398469843281'.

Η δοκιμασία ζητάει να βρούμε το ιδιωτικό κλειδί d. Μας δίνει τα p, q και e, άρα έχουμε όλα τα δεδομένα που χρειαζόμαστε. Από τα p, q μπορούμε να υπολογίσουμε το  $\phi(n)$  όπως είδαμε και στο προηγούμενο challenge και χρησιμοποιώ την συνάρτηση invert() της βιβλιοθήκης gmpy2 για να βρω το αποτέλεσμα.

- STARTER - RSA Starter 5



The image shows a code editor window titled 'test.py' with the following Python code:

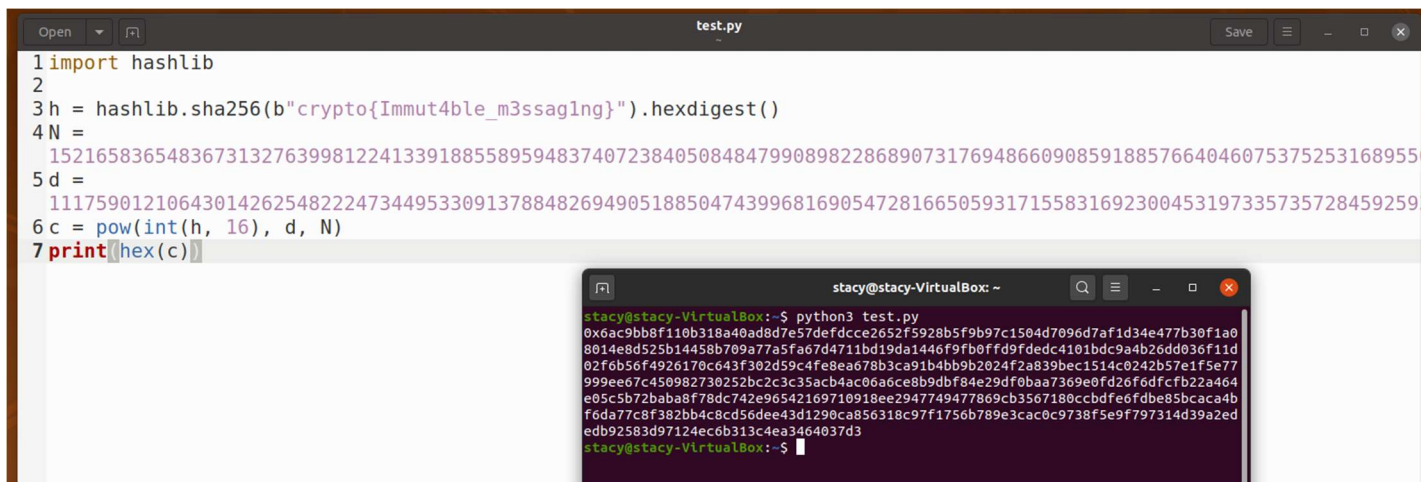
```
1 n = 882564595536224140639625987659416029426239230804614613279163
2 e = 65537
3 d = 121832886702415731577073962957377780195510499965398469843281
4 c = 77578995801157823671636298847186723593814843845525223303932
5 m = pow(c,d,n) #decryption
6 print(m)
```

Below the code editor is a terminal window titled 'stacy@stacy-VirtualBox: ~' showing the execution of the script:

```
stacy@stacy-VirtualBox:~$ python3 test.py
13371337
stacy@stacy-VirtualBox:~$
```

Η δοκιμασία ζητάει να αποκρυπτογραφήσουμε το  $c$ , χρησιμοποιώντας το ιδιωτικό κλειδί  $d$  που βρήκαμε στην προηγούμενη δοκιμασία. Μας δίνονται τα  $n$  και  $e$ , επομένως έχουμε όλα τα δεδομένα και άρα χρησιμοποιούμε την διαδικασία αποκρυπτογράφησης του αλγορίθμου RSA, όπως φαίνεται παραπάνω και εκτυπώνουμε το αποτέλεσμα.

- STARTER - RSA Starter 6



The image shows a code editor window titled 'test.py' with the following Python code:

```
1 import hashlib
2
3 h = hashlib.sha256(b"crypto{Immut4ble_m3ssag1ng}").hexdigest()
4 N =
15216583654836731327639981224133918855895948374072384050848479908982286890731769486609085918857664046075375253168955
5 d =
11175901210643014262548222473449533091378848269490518850474399681690547281665059317155831692300453197335735728459259
6 c = pow(int(h, 16), d, N)
7 print(hex(c))
```

Below the code editor is a terminal window titled 'stacy@stacy-VirtualBox: ~' showing the execution of the script:

```
stacy@stacy-VirtualBox:~$ python3 test.py
0x6ac9bb8f110b318a40ad8d7e57defdcce2652f5928b5f9b97c1504d7096d7af1d34e477b30f1a0
8014e8d525b14458b709a77a5fa67d4711bd19da1446f9fb0ffdd9fdedc4101bdc9a4b26dd036f11d
02f6b56f4926170c643f302d59c4fe8ea678b3ca91b4bb9b2024f2a839bec1514c0242b57e1f5e77
999ee67c450982730252bc2c3c35acb4ac06a6ce8b9dbf84e29df0baa7369e0fd26fdcfcb22a464
e05c5b72baba8f78dc742e96542169710918ee2947749477869cb3567180ccbdfe6fdb85bcaca4b
f6da77c8f382bb4c8cd56dee43d1290ca856318c97f1756b789e3cac0c9738f5e9f797314d39a2ed
edb92583d97124ec6b313c4ea3464037d3
stacy@stacy-VirtualBox:~$
```

Η δοκιμασία ζητάει να υπογράψουμε το flag. Για να γίνει αυτό, πρέπει πρώτα να το hashάρουμε, αυτό εκτελείται στην γραμμή 3 του κώδικα, και έπειτα να εκτελέσουμε την γνωστή διαδικασία κρυπτογράφησης του RSA. Τέλος, εκτυπώνουμε το αποτέλεσμα (Προσοχή! Το αποτέλεσμα είναι το string που φαίνεται στο terminal, εκτός από τους δύο πρώτους χαρακτήρες 0x).



- PRIMES PART 1 - Factoring

factorize 510143758735509025530880200653196460532653147

Extended Keyboard Upload Examples Random

Input interpretation:

factor 510143758735509025530880200653196460532653147

Decimal approximation: 5.1014375873550902553088020065319646053265314700000000000000...  $\times 10^{44}$  More digits

Divisors: 1 | 19704762736204164635843 | 25889363174021185185929 | 510143758735509025530880200653196460532653147 (4 divisors) Step-by-step solution

Download Page POWERED BY THE WOLFRAM LANGUAGE

Η δοκιμασία ζητάει να βρούμε τον μικρότερο παράγοντα του αριθμού 510143758735509025530880200653196460532653147. Για να το βρω, χρησιμοποίησα την σελίδα <https://www.wolframalpha.com/>. Το αποτέλεσμα φαίνεται στο παραπάνω screenshot και είναι τονισμένο με κίτρινο χρώμα.

- PRIMES PART 1 - Inferius Prime

```
1 import gmpy2
2 from Crypto.Util.number import long_to_bytes
3
4 p = 752708788837165590355094155871
5 q = 986369682585281993933185289261
6 n = 742449129124467073921545687640895127535705902454369756401331
7 fi = (p-1)*(q-1)
8 e = 3
9 d = gmpy2.invert(e, fi)
10 c = 39207274348578481322317340648475596807303160111338236677373
11 m = pow(c,d,n) #decryption
12 decrypted = long_to_bytes(m)
13 print(decrypted)
```

```
stacy@stacy-VirtualBox: ~$ pip install pycrypto
Command 'pip' not found, but there are 18 similar ones.
stacy@stacy-VirtualBox: ~$ pip3 install pycrypto
Collecting pycrypto
  Downloading pycrypto-2.6.1.tar.gz (446 kB)
    | 446 kB 966 kB/s
Building wheels for collected packages: pycrypto
  Building wheel for pycrypto (setup.py) ... done
  Created wheel for pycrypto: filename=pycrypto-2.6.1-cp38-cp38-linux_x86_64.whl
    size=498472 sha256=2e2b1e5dea8b12b329cf69b150fdf3381653759ee1f95f1a0c028f4c8d44
    9dc6
    Stored in directory: /home/stacy/.cache/pip/wheels/d0/99/d0/0298ea019d63f1d63a
    0965b9944b719e875f9bd0ffcdcf293
Successfully built pycrypto
Installing collected packages: pycrypto
Successfully installed pycrypto-2.6.1
stacy@stacy-VirtualBox: ~$ pip3 install pycrypto
Requirement already satisfied: pycrypto in ./local/lib/python3.8/site-packages
(2.6.1)
stacy@stacy-VirtualBox: ~$ python3 test.py
b'crypto(N33d big pRim35)'
```

Η δοκιμασία ζητάει να αποκρυπτογραφήσουμε το μήνυμα c. Μας δίνονται επίσης, το e και το n. Άρα, πρέπει να βρούμε τους παράγοντες του n, ώστε να μπορούμε να υπολογίσουμε το φ(n). Για να τους βρω, χρησιμοποίησα την σελίδα <http://www.factordb.com/index.php> (όχι το wolframalpha.com γιατί δεν μπορούσε να τους υπολογίσει για πολύ μεγάλους αριθμούς...). Το αποτέλεσμα φαίνεται στο παρακάτω screenshot:

Search Sequences Report results Factor tables Status D

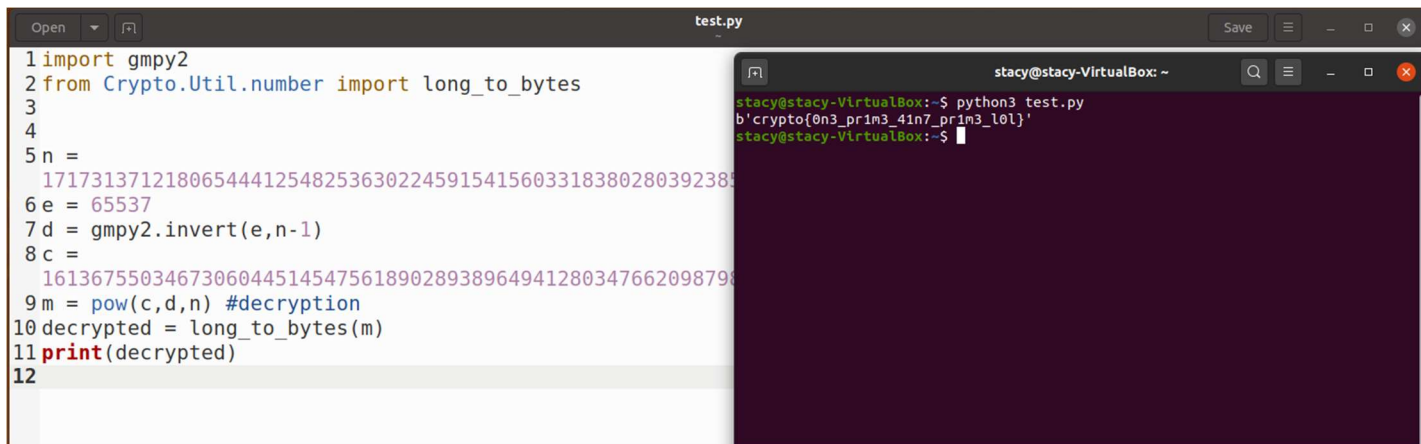
742449129124467073921545687640895127535705902454369756401331 Factorize!

Result:

status (2)	digits	number
FF	60 (show)	7424491291...31<60> = 752708788837165590355094155871<30> · 986369682585281993933185289261<30>

Έπειτα, εφάρμοσα πάλι την διαδικασία κρυπτογράφησης του RSA και εκτύπωσα το flag.

- PRIMES PART 1 - Monoprime



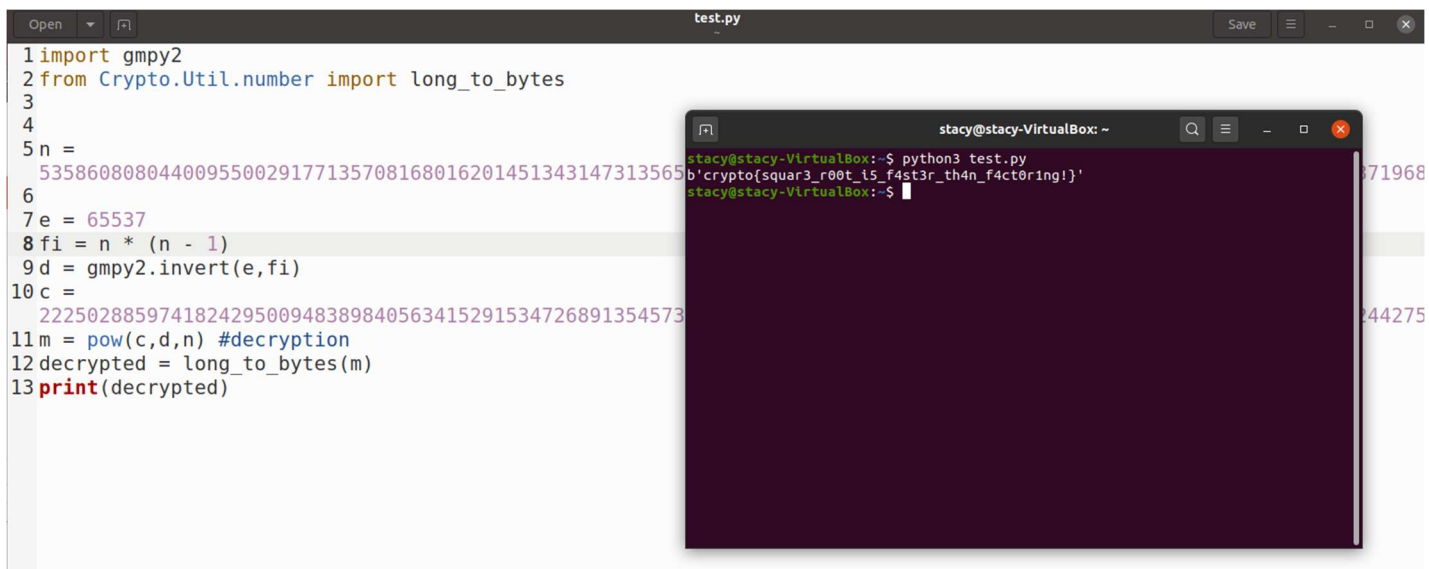
The screenshot shows a Python script named `test.py` and its execution in a terminal. The script imports `gmpy2` and `Crypto.Util.number`, then defines `n`, `e`, and `d`. It calculates `m = pow(c, d, n)` and prints the decrypted message. The terminal output shows the decrypted message: `b'crypto{0n3_pr1m3_41n7_pr1m3_l0l}'`.

```
1 import gmpy2
2 from Crypto.Util.number import long_to_bytes
3
4
5 n =
6 e = 65537
7 d = gmpy2.invert(e, n-1)
8 c =
9 m = pow(c, d, n) #decryption
10 decrypted = long_to_bytes(m)
11 print(decrypted)
12
```

```
stacy@stacy-VirtualBox: ~$ python3 test.py
b'crypto{0n3_pr1m3_41n7_pr1m3_l0l}'
stacy@stacy-VirtualBox: ~$
```

Η δοκιμασία ζητάει πάλι να αποκρυπτογραφήσουμε το μήνυμα `c`. Μας δίνονται επίσης, το `e` και το `n`. Η διαφορά εδώ είναι ότι δεν χρησιμοποιήθηκαν 2 πρώτοι αριθμοί ( $p, q$ ) για να δημιουργηθεί το `n`, αλλά ένας... ο `n`. Άρα, διαφέρει το  $\phi(n)$  όπου εδώ είναι ίσο με  $n-1$ . Εκτελείται πάλι η ίδια διαδικασία και εκτυπώνεται το αποτέλεσμα.

- PRIMES PART 1 - Square Eyes



The screenshot shows a Python script named `test.py` and its execution in a terminal. The script imports `gmpy2` and `Crypto.Util.number`, then defines `n`, `e`, and `fi`. It calculates `m = pow(c, d, n)` and prints the decrypted message. The terminal output shows the decrypted message: `b'crypto{squar3_r00t_15_f4st3r_th4n_f4ct0r1ng!'}`.

```
1 import gmpy2
2 from Crypto.Util.number import long_to_bytes
3
4
5 n =
6 e = 65537
7 fi = n * (n - 1)
8 d = gmpy2.invert(e, fi)
9 c =
10 m = pow(c, d, n) #decryption
11 decrypted = long_to_bytes(m)
12 print(decrypted)
13
```

```
stacy@stacy-VirtualBox: ~$ python3 test.py
b'crypto{squar3_r00t_15_f4st3r_th4n_f4ct0r1ng!}'
stacy@stacy-VirtualBox: ~$
```

Η δοκιμασία ζητάει πάλι να αποκρυπτογραφήσουμε το μήνυμα `c`. Μας δίνονται επίσης, το `e` και το `n`. Η διαφορά εδώ είναι ότι δεν χρησιμοποιήθηκαν 2 πρώτοι αριθμοί ( $p, q$ ) για να δημιουργηθεί το `n`, αλλά ένας, ο οποίος χρησιμοποιήθηκε 2 φορές. Άρα, διαφέρει το  $\phi(n)$  όπου εδώ είναι ίσο με  $(n-1)*n$ , αφού χρησιμοποιούμε τον ίδιο πρώτο αριθμό 2 φορές, πολλαπλασιάζουμε με `n`. Εκτελείται πάλι η ίδια διαδικασία και εκτυπώνεται το αποτέλεσμα.

- PRIMES PART 1 - Manyprime

```

1 import gmpy2
2 from Crypto.Util.number import long_to_bytes
3
4
5 n =
58064239189884319292956385687089779965088315271876176293229248225215259127987142156916203719041903643504179773988038
6 e = 65537
7 c =
32072149053462443414999372352732297796055651075062835485626073209810969258133840999998337613135491837004762515045472
8
9 factors = [9282105380008121879, 9303850685953812323, 9389357739583927789, 10336650220878499841,
10638241655447339831, 11282698189561966721, 11328768673634243077, 11403460639036243901, 11473665579512371723,
11492065299277279799, 11530534813954192171, 11665347949879312361, 12132158321859677597, 12834461276877415051,
12955403765595949597, 1297397233677979701, 13099895578757581201, 13572286589428162097, 14100640260554622013,
14178869592193599187, 14278240802299816541, 14523070016044624039, 14963354250199553339, 15364597561881860737,
15669758663523555763, 15824122791679574573, 15998365463074268941, 16656402470578844539, 16898740504023346457,
17138336856793050757, 17174065872156629921, 17281246625998849649]
10
11 fi = 1
12 for f in factors:
13     fi *= (f-1)
14
15 d = gmpy2.invert(e,fi)
16 m = pow(c,d,n) #decryption
17 decrypted = long_to_bytes(m)
18 print(decrypted)

```

```

stacy@stacy-VirtualBox: ~
stacy@stacy-VirtualBox:~$ python3 test.py
b'crypto{squar3_r00t_i5_f4st3r_th4n_f4ct0ring!}'
stacy@stacy-VirtualBox:~$ python3 test.py
b'crypto{700_m4ny_5m4ll_f4c70r5}'
stacy@stacy-VirtualBox:~$

```

Η δοκιμασία ζητάει πάλι να αποκρυπτογραφήσουμε το μήνυμα c. Μας δίνονται επίσης, το e και το n. Η διαφορά εδώ είναι ότι δεν χρησιμοποιήθηκαν 2 πρώτοι αριθμοί (p,q) για να δημιουργηθεί το n, αλλά 30. Για να τους υπολογίσω, χρησιμοποίησα πάλι την σελίδα <http://www.factordb.com/index.php> και το αποτέλεσμα φαίνεται στο παρακάτω screenshot:

580642391898843192929563856870897799650883152718761762932292482252152591279871421569162037190419031

Factorize!

status	digits	number
FF	612	<div> <div>5806423918_37&lt;612&gt; = 9282105380008121879_&lt;19&gt; · 9303850685953812323_&lt;19&gt; · 9389357739583927789_&lt;19&gt; · 10336650220878499841_&lt;20&gt; · 10638241655447339831_&lt;20&gt; · 11282698189561966721_&lt;20&gt; · 11328768673634243077_&lt;20&gt; · 11403460639036243901_&lt;20&gt; · 11473665579512371723_&lt;20&gt; · 11492065299277279799_&lt;20&gt; · 11530534813954192171_&lt;20&gt; · 11665347949879312361_&lt;20&gt; · 12132158321859677597_&lt;20&gt; · 12834461276877415051_&lt;20&gt; · 12955403765595949597_&lt;20&gt; · 1297397233677979701_&lt;20&gt; · 13099895578757581201_&lt;20&gt; · 13572286589428162097_&lt;20&gt; · 14100640260554622013_&lt;20&gt; · 14178869592193599187_&lt;20&gt; · 14278240802299816541_&lt;20&gt; · 14523070016044624039_&lt;20&gt; · 14963354250199553339_&lt;20&gt; · 15364597561881860737_&lt;20&gt; · 15669758663523555763_&lt;20&gt; · 15824122791679574573_&lt;20&gt; · 15998365463074268941_&lt;20&gt; · 16656402470578844539_&lt;20&gt; · 16898740504023346457_&lt;20&gt; · 17138336856793050757_&lt;20&gt; · 17174065872156629921_&lt;20&gt; · 17281246625998849649_&lt;20&gt;</div> </div>

More information

ECM

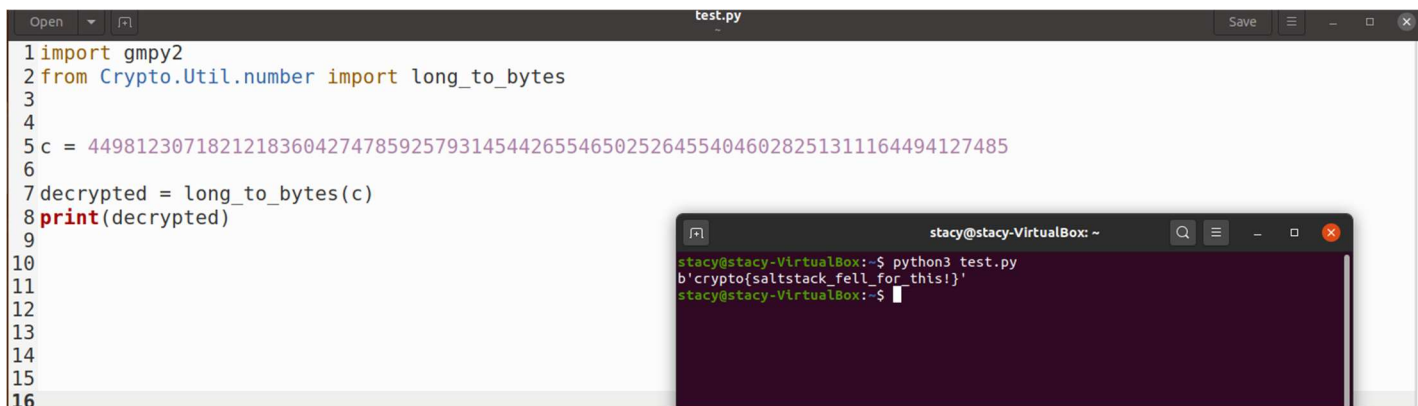
factordb.com - 284 queries to generate this page (0.05 seconds) ([limits](#)) ([imprint](#)) ([Privacy Policy](#))

Πήρα έναν-έναν τους παράγοντες και τους έβαλα στην λίστα factors. Για να υπολογίσω το  $\phi(n)$ , διαπερνάω την λίστα factors και υπολογίζω το γινόμενο κάθε φορά του ήδη υπάρχοντος  $\phi$  με το  $f - 1$ , όπου  $f$  είναι ο τρέχον παράγοντας. Εκτελείται πάλι η ίδια διαδικασία και εκτυπώνεται το αποτέλεσμα.

Σημείωση: Είχα ξεχάσει να κάνω clear το terminal μετά που έλυσα την προηγούμενη δοκιμασία και για αυτό φαίνονται και τα 2 αποτελέσματα... για την τρέχουσα δοκιμασία το σωστό αποτέλεσμα είναι το δεύτερο (crypto{700\_m4ny\_5m4ll\_f4c70r5}).



- PUBLIC EXPONENT - Salty

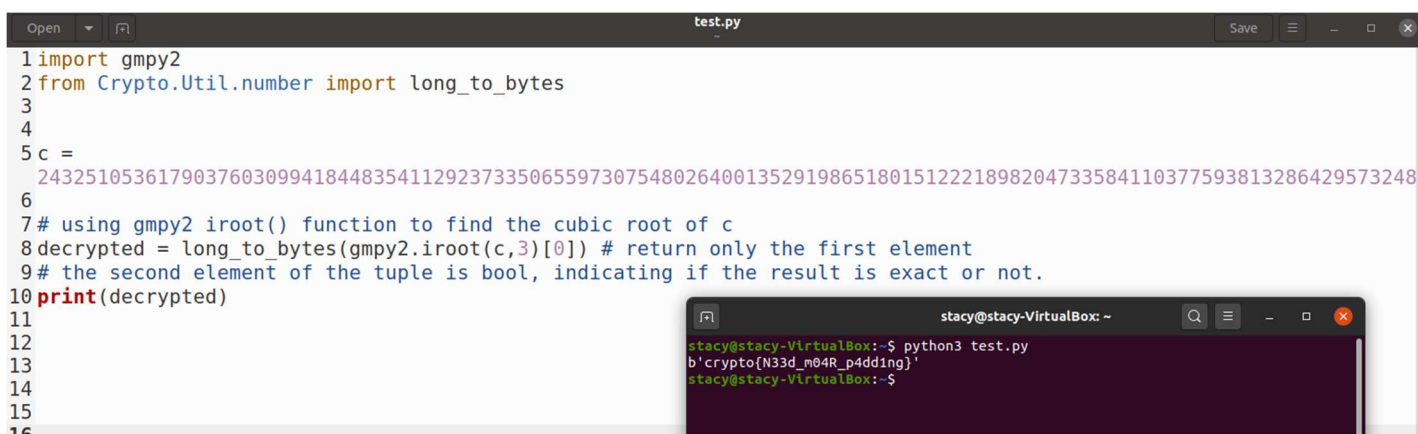


```
test.py
1 import gmpy2
2 from Crypto.Util.number import long_to_bytes
3
4
5 c = 44981230718212183604274785925793145442655465025264554046028251311164494127485
6
7 decrypted = long_to_bytes(c)
8 print(decrypted)
9
10
11
12
13
14
15
16
```

```
stacy@stacy-VirtualBox: ~
stacy@stacy-VirtualBox:~$ python3 test.py
b'crypto{saltstack_fell_for_this!}'
stacy@stacy-VirtualBox:~$
```

Η δοκιμασία ζητάει πάλι να αποκρυπτογραφήσουμε το μήνυμα  $c$ . Μας δίνεται επίσης, ότι  $e = 1$  και  $d = -1$ . Άρα, αρκεί απλά να χρησιμοποιώ την συνάρτηση `long_to_bytes` για να τυπώσω στην κατάλληλη μορφή το flag. Ουσιαστικά έγινε τζάμπα η διαδικασία της κρυπτογράφησης...

- PUBLIC EXPONENT - Modulus Inutilis

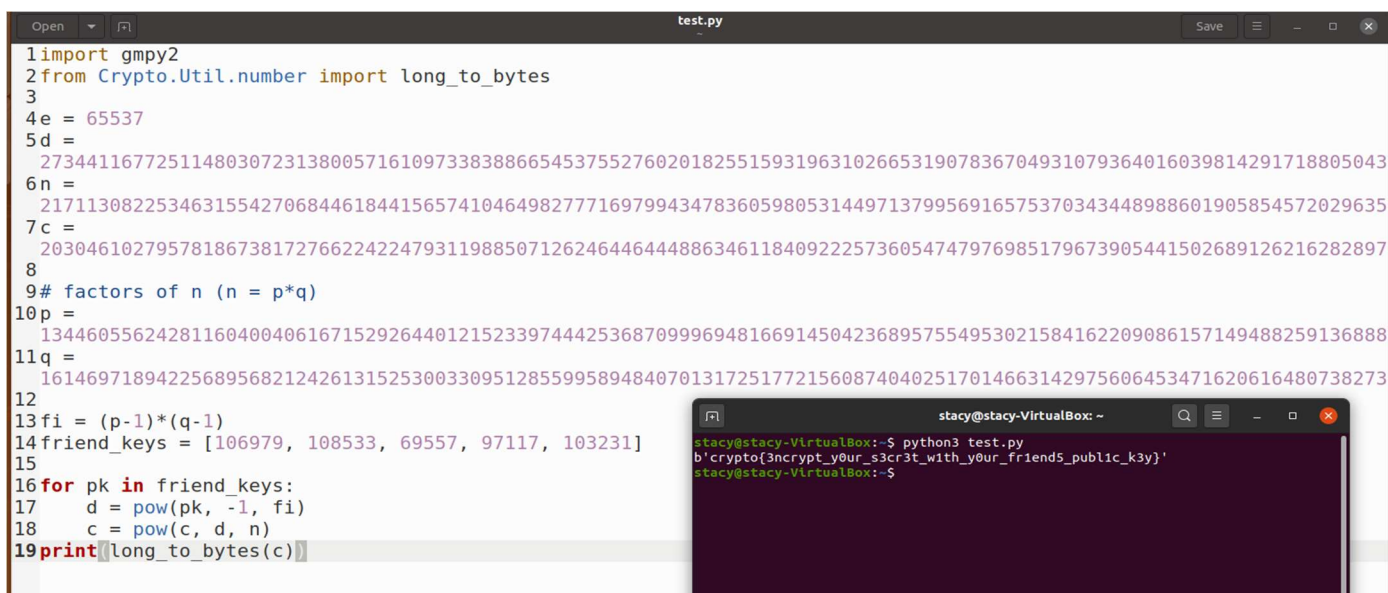


```
test.py
1 import gmpy2
2 from Crypto.Util.number import long_to_bytes
3
4
5 c =
  2432510536179037603099418448354112923733506559730754802640013529198651801512221898204733584110377593813286429573248
6
7 # using gmpy2 iroot() function to find the cubic root of c
8 decrypted = long_to_bytes(gmpy2.iroot(c,3)[0]) # return only the first element
9 # the second element of the tuple is bool, indicating if the result is exact or not.
10 print(decrypted)
11
12
13
14
15
16
```

```
stacy@stacy-VirtualBox: ~
stacy@stacy-VirtualBox:~$ python3 test.py
b'crypto{N33d_m04R_p4dd1ng}'
stacy@stacy-VirtualBox:~$
```

Η δοκιμασία ζητάει πάλι να αποκρυπτογραφήσουμε το μήνυμα  $c$ . Μας δίνεται επίσης, ότι  $e = 3$  και  $d = -1$ . Άρα, αρκεί απλά να βρω την κυβική ρίζα του  $c$  (αφού  $e = 3$ ) και να χρησιμοποιήσω την συνάρτηση `long_to_bytes` για να τυπώσω στην κατάλληλη μορφή το flag. Ουσιαστικά πάλι έγινε τζάμπα η διαδικασία της κρυπτογράφησης...

- PUBLIC EXPONENT - Crossed Wires



```
test.py
1 import gmpy2
2 from Crypto.Util.number import long_to_bytes
3
4 e = 65537
5 d =
  27344116772511480307231380057161097338388665453755276020182551593196310266531907836704931079364016039814291718805043
6 n =
  21711308225346315542706844618441565741046498277716979943478360598053144971379956916575370343448988601905854572029635
7 c =
  20304610279578186738172766224224793119885071262464464448863461184092225736054747976985179673905441502689126216282897
8
9 # factors of n (n = p*q)
10 p =
  13446055624281160400406167152926440121523397444253687099969481669145042368957554953021584162209086157149488259136888
11 q =
  16146971894225689568212426131525300330951285599589484070131725177215608740402517014663142975606453471620616480738273
12
13 fi = (p-1)*(q-1)
14 friend_keys = [106979, 108533, 69557, 97117, 103231]
15
16 for pk in friend_keys:
17     d = pow(pk, -1, fi)
18     c = pow(c, d, n)
19 print(long_to_bytes(c))
```

```
stacy@stacy-VirtualBox: ~
stacy@stacy-VirtualBox:~$ python3 test.py
b'crypto{3ncrypt_y0ur_s3cr3t_w1th_y0ur_fr1end5_puBl1c_k3y}'
stacy@stacy-VirtualBox:~$
```

Η δοκιμασία ζητάει πάλι να αποκρυπτογραφήσουμε το μήνυμα  $c$ , το οποίο όμως έχει περάσει από 5 ακόμα φίλους... Μας δίνονται τα  $c$ ,  $n$ ,  $d$ ,  $d$  των φίλων και  $e$ . Πρέπει, αρχικά να υπολογίσουμε τους 2 παράγοντες  $(p, q)$  του  $n$ . Πάλι, χρησιμοποιώ την σελίδα <http://www.factordb.com/index.php> τα  $p$  και  $q$  φαίνονται στα παρακάτω screenshots:

Additional information (Internal ID 1100000001531000381)	
Digits (Base 10)	309
Number	161469718942256895682124261315253003309512855995894840701317251772156087404025170146631429756064534716206164807382734456 43809273274367793224010769460318383691408352089793973150914149255603969984103815563896440419666191368964699279209687091 969164697704779792586727943470780308857107052647197945528236341228473

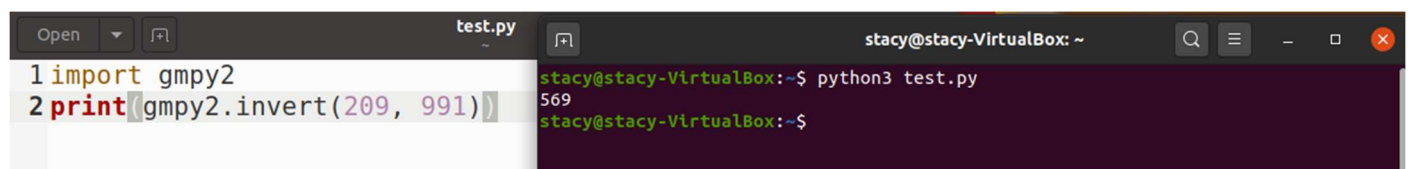
  

Additional information (Internal ID 1100000001489276241)	
Digits (Base 10)	309
Number	134460556242811604004061671529264401215233974442536870999694816691450423689575549530215841622090861571494882591368883283 016107051686642467260643894947947473532769025695530343815260424314855023688439603651834585971233941772580950216838838690 315383700689885536546289584980534945897919914730948196240662991266027

Τώρα έχω ότι χρειάζομαι για να εφαρμόσω την γνωστή διαδικασία. Πρέπει ωστόσο, να αποκρυπτογραφώ κάθε φορά το μήνυμα χρησιμοποιώντας κάθε φορά το αντίστοιχο κλειδί του κάθε φίλου. Αυτό γίνεται στις γραμμές 16-18 του κώδικα όπως φαίνεται και στο παραπάνω screenshot. Τέλος, χρησιμοποιήσω την συνάρτηση `long_to_bytes` για να τυπώσω στην κατάλληλη μορφή το flag.

## DIFFIE-HELLMAN

- STARTER - Diffie-Hellman Starter 1

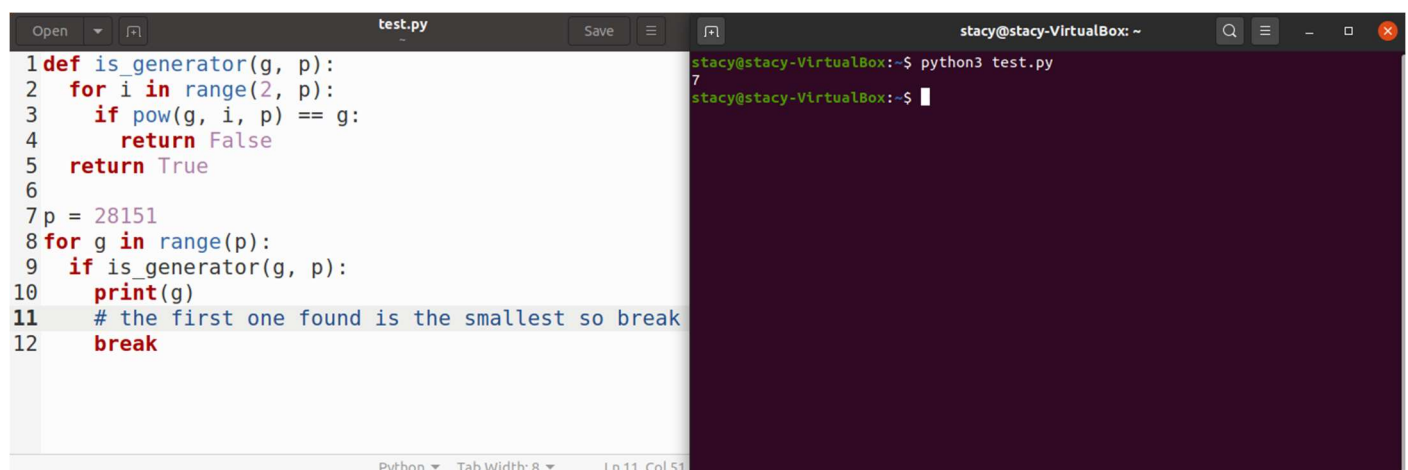


```
test.py
1 import gmpy2
2 print(gmpy2.invert(209, 991))

stacy@stacy-VirtualBox: ~
stacy@stacy-VirtualBox:~$ python3 test.py
569
stacy@stacy-VirtualBox:~$
```

Η δοκιμασία ζητάει απλά να υπολογίσουμε το  $d$ , για το οποίο ισχύει ότι  $g * d \bmod 991 = 1$  και  $g = 209$ . Χρησιμοποιώ απλά την συνάρτηση `invert()` της βιβλιοθήκης `gmpy2` και εκτυπώνω το αποτέλεσμα ( $d = 569$ ).

- STARTER - Diffie-Hellman Starter 2

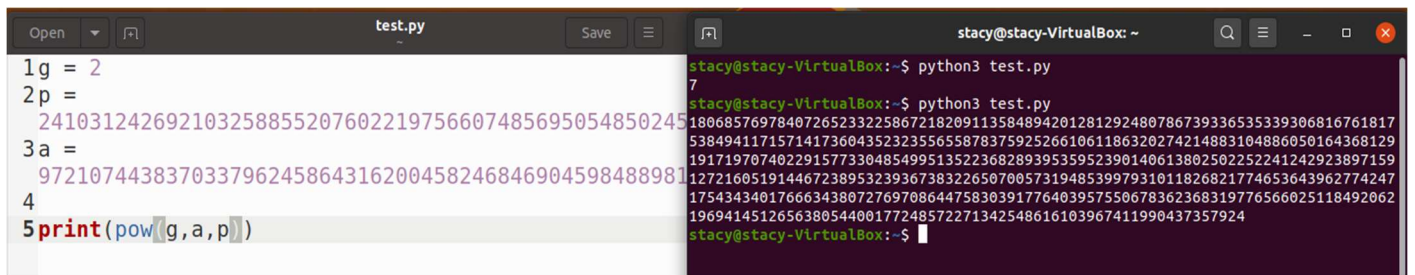


```
test.py
1 def is_generator(g, p):
2     for i in range(2, p):
3         if pow(g, i, p) == g:
4             return False
5     return True
6
7 p = 28151
8 for g in range(p):
9     if is_generator(g, p):
10        print(g)
11    # the first one found is the smallest so break
12    break

stacy@stacy-VirtualBox: ~
stacy@stacy-VirtualBox:~$ python3 test.py
7
stacy@stacy-VirtualBox:~$
```

Η δοκιμασία ζητάει να υπολογίσουμε το μικρότερο  $g$  (generator). Ο παραπάνω κώδικας κάνει ακριβώς αυτό, δηλαδή ελέγχει εάν η σειρά κάθε στοιχείου είναι ίση με τον πρώτο και αν ναι έχουμε βρει τον generator. Το αποτέλεσμα είναι  $g = 7$ .

- STARTER - Diffie-Hellman Starter 3



The screenshot shows a code editor on the left with a file named 'test.py' containing the following Python code:

```
1 g = 2
2 p =
241031242692103258855207602219756607485695054850245
3 a =
972107443837033796245864316200458246846904598488981
4
5 print(pow(g, a, p))
```

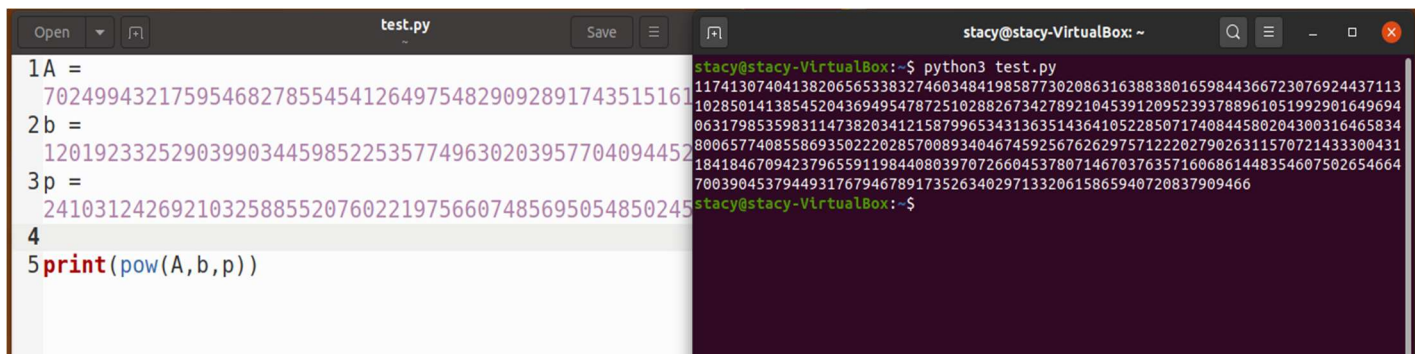
On the right, a terminal window shows the execution of the script:

```
stacy@stacy-VirtualBox: ~$ python3 test.py
7
stacy@stacy-VirtualBox: ~$ python3 test.py
18068576978407265233225867218209113584894201281292480786739336535339306816761817
53849411715714173604352323556558783759252661061186320274214883104886050164368129
19171970740229157733048549951352236828939535952390140613802502252241242923897159
12721605191446723895323936738322650700573194853997931011826821774653643962774247
17543434017666343807276970864475830391776403957550678362368319776566025118492062
196941451265638054400177248572271342548616103967411990437357924
stacy@stacy-VirtualBox: ~$
```

Η δοκιμασία ζητάει απλά να υπολογίσουμε το αποτέλεσμα του  $g^a \bmod p$ . Μας δίνονται τα  $g$ ,  $p$  και  $a$ , άρα είναι όλα γνωστά. Επομένως, απλά χρησιμοποιώ την συνάρτηση `pow()` και εκτυπώνω το αποτέλεσμα.

Σημείωση: Είχα ξεχάσει πάλι να κάνω `clear` το terminal μετά που έλυσα την προηγούμενη δοκιμασία και για αυτό φαίνονται και τα 2 αποτελέσματα... για την τρέχουσα δοκιμασία το σωστό αποτέλεσμα είναι το δεύτερο.

- STARTER - Diffie-Hellman Starter 4



The screenshot shows a code editor on the left with a file named 'test.py' containing the following Python code:

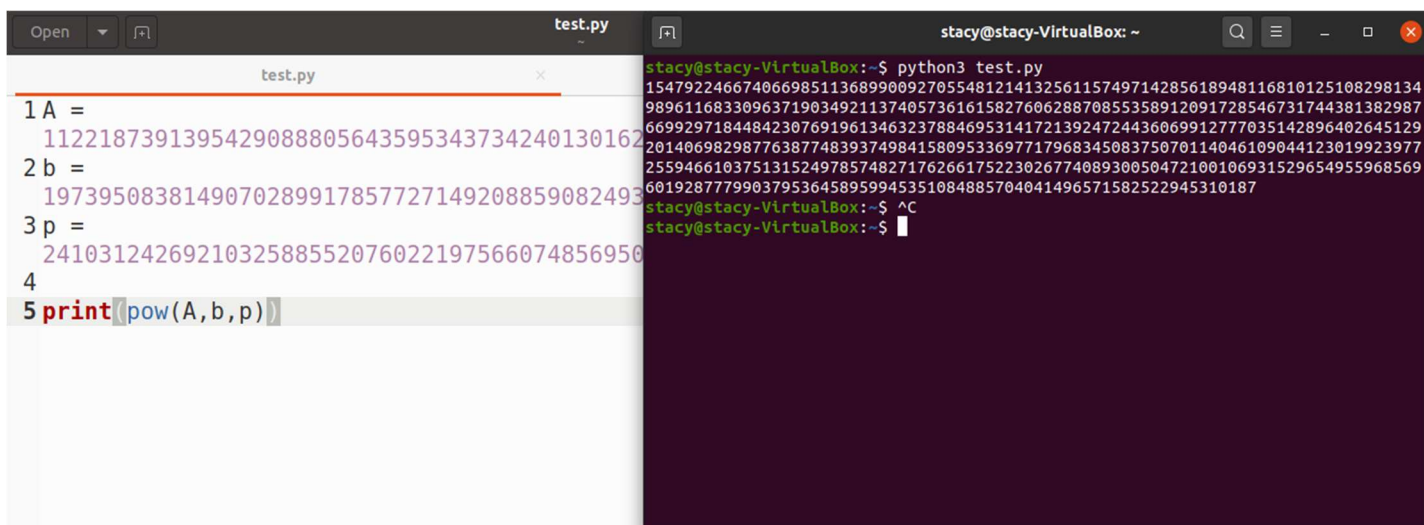
```
1 A =
702499432175954682785545412649754829092891743515161
2 b =
120192332529039903445985225357749630203957704094452
3 p =
241031242692103258855207602219756607485695054850245
4
5 print(pow(A, b, p))
```

On the right, a terminal window shows the execution of the script:

```
stacy@stacy-VirtualBox: ~$ python3 test.py
11741307404138206565338327460348419858773020863163883801659844366723076924437113
10285014138545204369495478725102882673427892104539120952393788961051992901649694
06317985359831147382034121587996534313635143641052285071740844580204300316465834
80065774085586935022202857008934046745925676262975712220279026311570721433300431
18418467094237965591198440803970726604537807146703763571606861448354607502654664
700390453794493176794678917352634029713320615865940720837909466
stacy@stacy-VirtualBox: ~$
```

Η δοκιμασία ζητάει απλά να υπολογίσουμε το αποτέλεσμα του  $A^b \bmod p$ . Μας δίνονται τα  $A$ ,  $p$  και  $b$ , άρα είναι όλα γνωστά. Επομένως, απλά χρησιμοποιώ την συνάρτηση `pow()` και εκτυπώνω το αποτέλεσμα.

- STARTER - Diffie-Hellman Starter 5



The screenshot shows a code editor on the left with a file named 'test.py' containing the following Python code:

```
1 A =
1122187391395429088805643595343734240130162
2 b =
1973950838149070289917857727149208859082493
3 p =
2410312426921032588552076022197566074856950
4
5 print(pow(A, b, p))
```

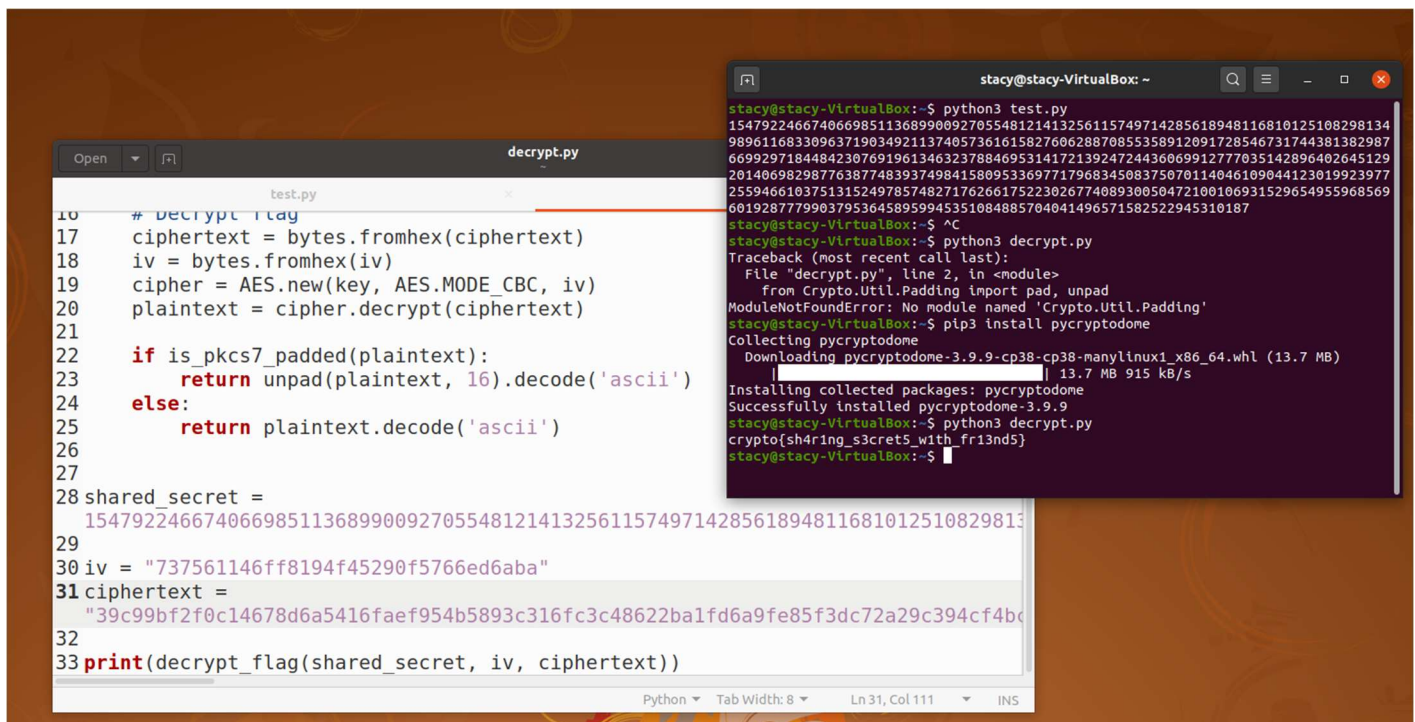
On the right, a terminal window shows the execution of the script:

```
stacy@stacy-VirtualBox: ~$ python3 test.py
15479224667406698511368990092705548121413256115749714285618948116810125108298134
98961168330963719034921137405736161582760628870855358912091728546731744381382987
66992971844842307691961346323788469531417213924724436069912777035142896402645129
20140698298776387748393749841580953369771796834508375070114046109044123019923977
25594661037513152497857482717626617522302677408930050472100106931529654955968569
601928777990379536458959945351084885704041496571582522945310187
stacy@stacy-VirtualBox: ~$ ^C
stacy@stacy-VirtualBox: ~$
```

Η δοκιμασία ζητάει ξανά, απλά να υπολογίσουμε το αποτέλεσμα του  $A^b \bmod p$ . Μας δίνονται τα  $A$ ,  $p$  και  $b$ , άρα είναι όλα γνωστά. Επομένως, απλά χρησιμοποιώ την συνάρτηση `pow()` και εκτυπώνω το αποτέλεσμα. Ωστόσο, για να βρω το flag πρέπει να δώσω τιμή σε 3 μεταβλητές στο αρχείο `decrypt.py`



που μας δίνεται. Το iv και ciphertext μας δίνονται από την εκφώνηση και το shared\_secret είναι αυτό που βρήκαμε στο προηγούμενο screenshot. Επομένως, αν τα συμπληρώσω και τρέξω τον κώδικα παίρνω το flag, όπως φαίνεται στο παρακάτω screenshot:



The image shows a code editor with a file named `decrypt.py` and a terminal window. The code in `decrypt.py` defines a function `decrypt_flag` that takes a shared secret, an IV, and a ciphertext as input. It uses the `Crypto` module to create an AES cipher in CBC mode and decrypt the ciphertext. The main part of the script sets the `shared_secret`, `iv`, and `ciphertext` variables and then prints the result of `decrypt_flag`.

```
10 # decrypt flag
17 ciphertext = bytes.fromhex(ciphertext)
18 iv = bytes.fromhex(iv)
19 cipher = AES.new(key, AES.MODE_CBC, iv)
20 plaintext = cipher.decrypt(ciphertext)
21
22 if is_pkcs7_padded(plaintext):
23     return unpad(plaintext, 16).decode('ascii')
24 else:
25     return plaintext.decode('ascii')
26
27
28 shared_secret =
29 15479224667406698511368990092705548121413256115749714285618948116810125108298134
30 iv = "737561146ff8194f45290f5766ed6aba"
31 ciphertext =
32 "39c99bf2f0c14678d6a5416faef954b5893c316fc3c48622ba1fd6a9fe85f3dc72a29c394cf4b"
33 print(decrypt_flag(shared_secret, iv, ciphertext))
```

The terminal window shows the execution of the script. It first runs `python3 test.py`, which outputs a long string of hexadecimal characters. Then, it runs `python3 decrypt.py`, which results in a `ModuleNotFoundError: No module named 'Crypto.Util.Padding'`. The user then runs `pip3 install pycryptodome`, which successfully installs the package. Finally, the user runs `python3 decrypt.py` again, which outputs the flag: `crypto{sh4r1ng_s3cret5_w1th_fr13nd5}`.