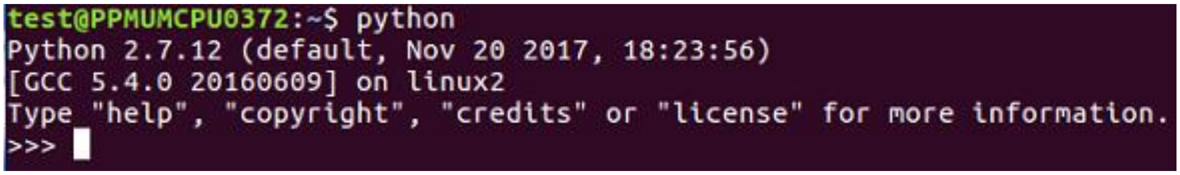
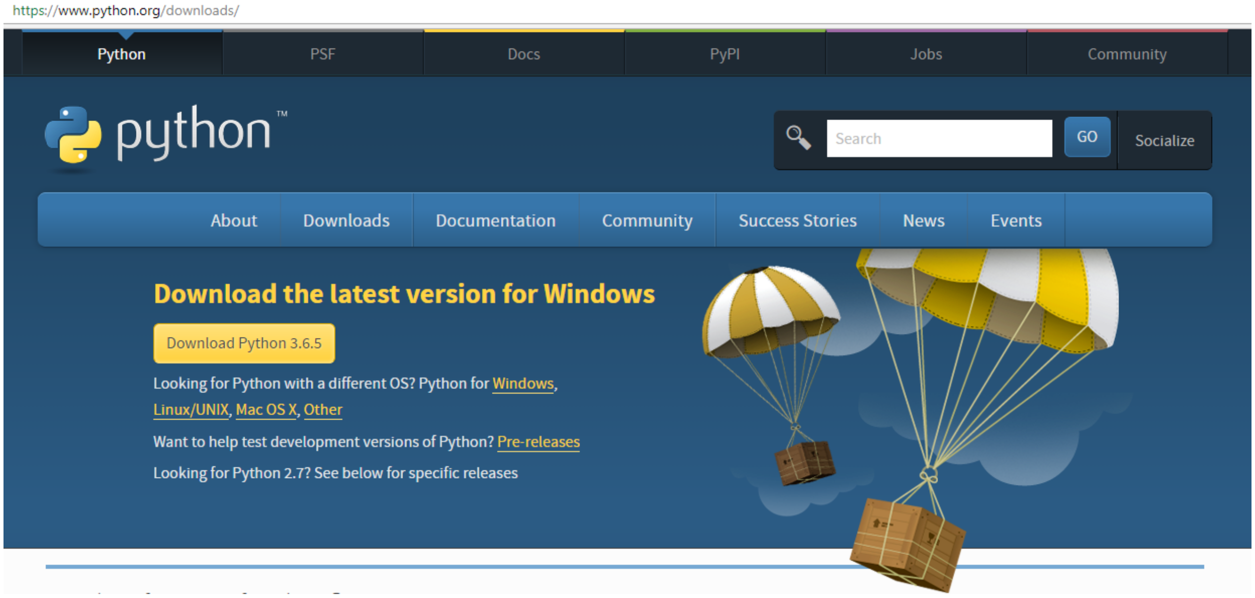
Chapter 1: Obfuscation



```
test@PPMUMCPU0372:~$ python
Python 2.7.12 (default, Nov 20 2017, 18:23:56)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```



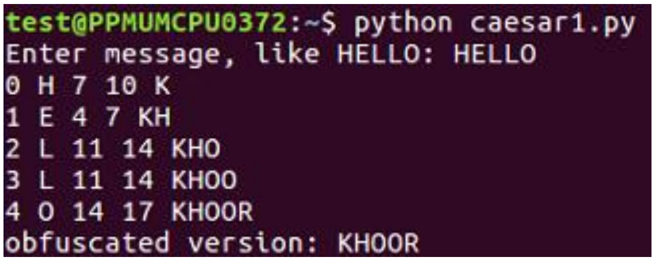
https://www.python.org/downloads/

Download the latest version for Windows

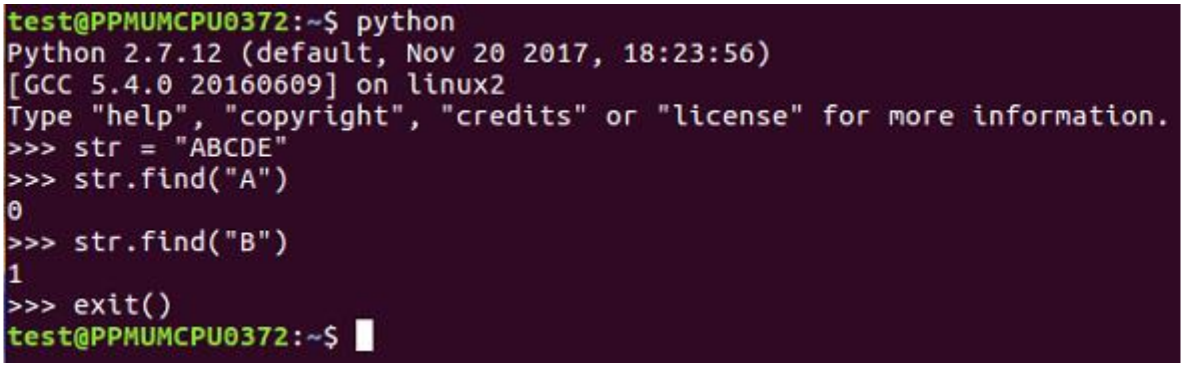
Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Pre-releases

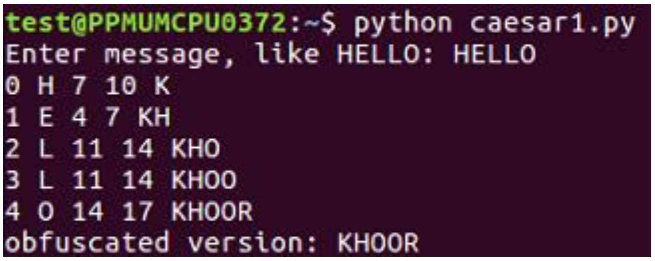
Looking for Python 2.7? See below for specific releases



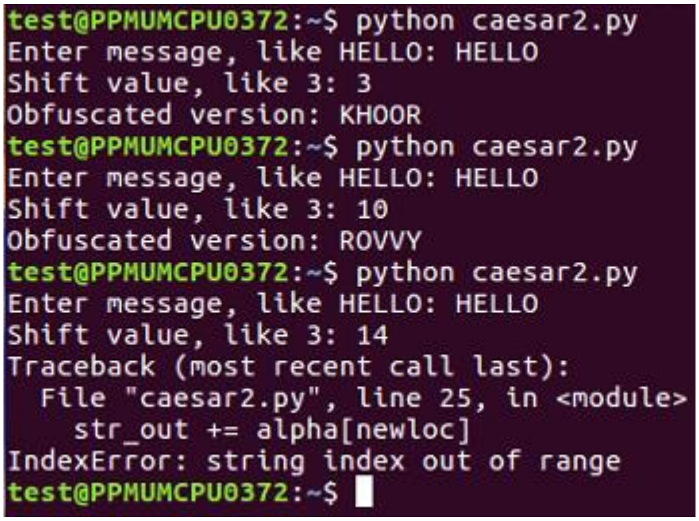
```
test@PPMUMCPU0372:~$ python caesar1.py
Enter message, like HELLO: HELLO
0 H 7 10 K
1 E 4 7 KH
2 L 11 14 KHO
3 L 11 14 KHOO
4 O 14 17 KHOOR
obfuscated version: KHOOR
```



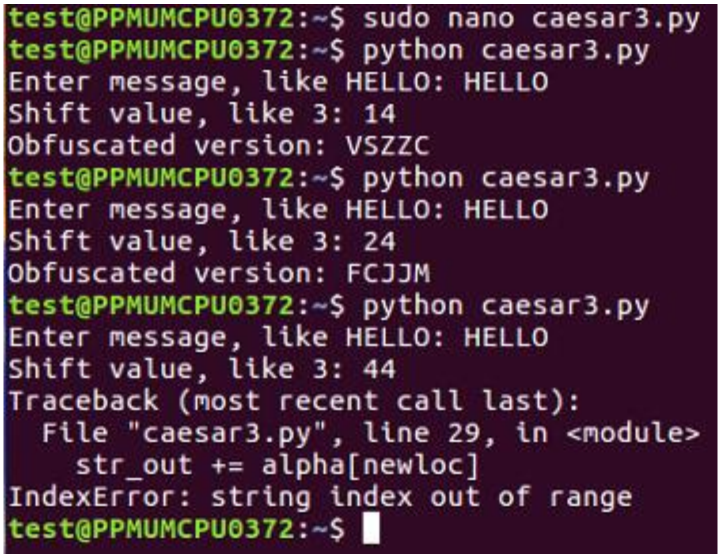
```
test@PPMUMCPU0372:~$ python
Python 2.7.12 (default, Nov 20 2017, 18:23:56)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> str = "ABCDE"
>>> str.find("A")
0
>>> str.find("B")
1
>>> exit()
test@PPMUMCPU0372:~$ 
```



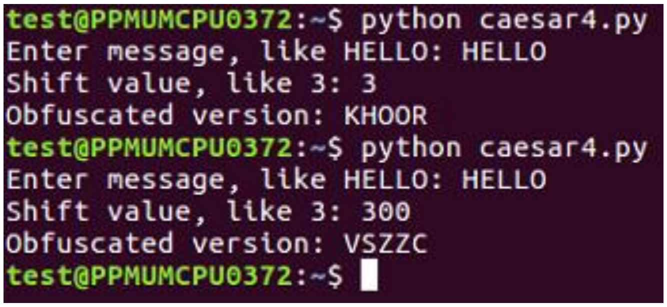
```
test@PPMUMCPU0372:~$ python caesar1.py
Enter message, like HELLO: HELLO
0 H 7 10 K
1 E 4 7 KH
2 L 11 14 KHO
3 L 11 14 KHOO
4 O 14 17 KHOOR
obfuscated version: KHOOR
```



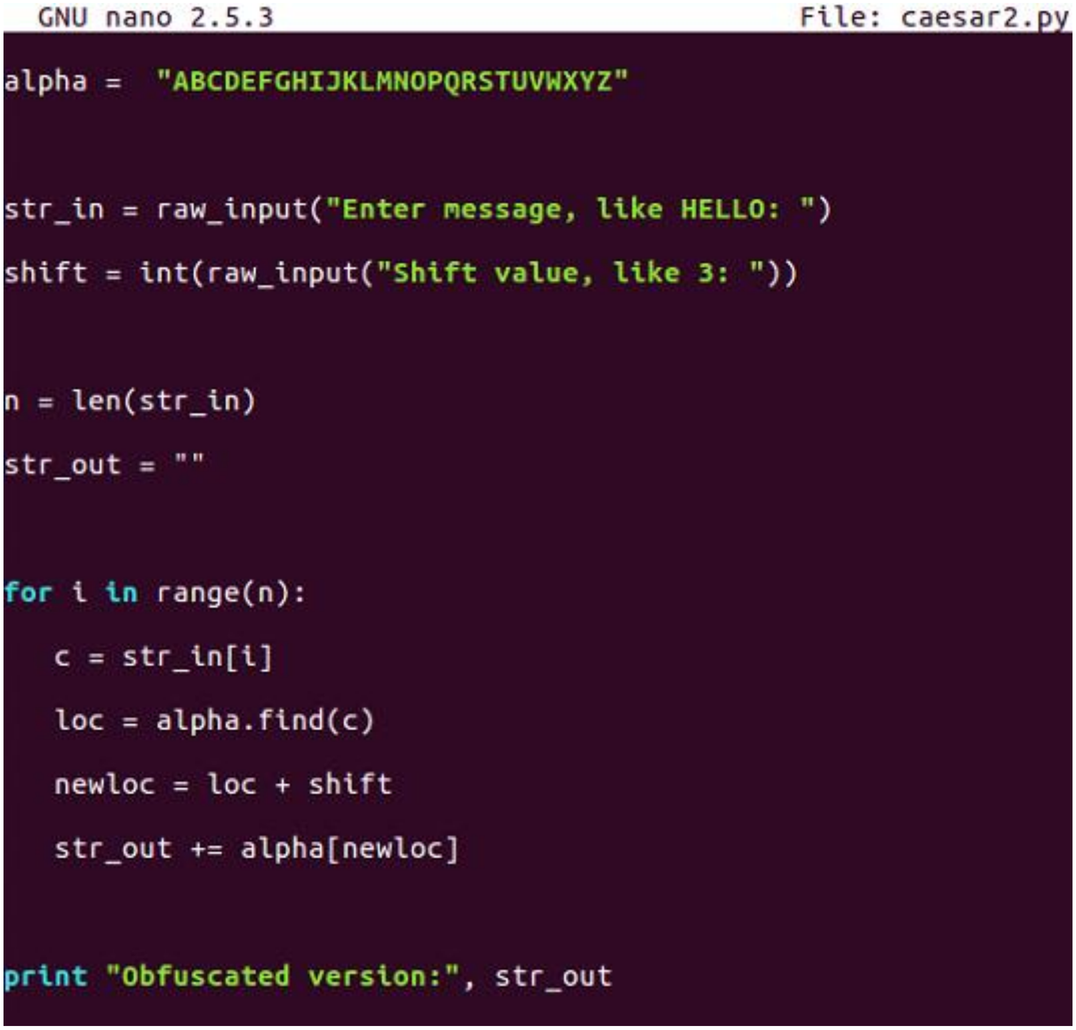
```
test@PPMUMCPU0372:~$ python caesar2.py
Enter message, like HELLO: HELLO
Shift value, like 3: 3
Obfuscated version: KHOOR
test@PPMUMCPU0372:~$ python caesar2.py
Enter message, like HELLO: HELLO
Shift value, like 3: 10
Obfuscated version: ROVVY
test@PPMUMCPU0372:~$ python caesar2.py
Enter message, like HELLO: HELLO
Shift value, like 3: 14
Traceback (most recent call last):
  File "caesar2.py", line 25, in <module>
    str_out += alpha[newloc]
IndexError: string index out of range
test@PPMUMCPU0372:~$ 
```



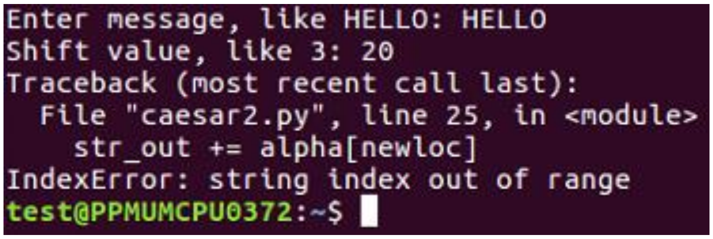
```
test@PPMUMCPU0372:~$ sudo nano caesar3.py
test@PPMUMCPU0372:~$ python caesar3.py
Enter message, like HELLO: HELLO
Shift value, like 3: 14
Obfuscated version: VSZZC
test@PPMUMCPU0372:~$ python caesar3.py
Enter message, like HELLO: HELLO
Shift value, like 3: 24
Obfuscated version: FCJJM
test@PPMUMCPU0372:~$ python caesar3.py
Enter message, like HELLO: HELLO
Shift value, like 3: 44
Traceback (most recent call last):
  File "caesar3.py", line 29, in <module>
    str_out += alpha[newloc]
IndexError: string index out of range
test@PPMUMCPU0372:~$
```



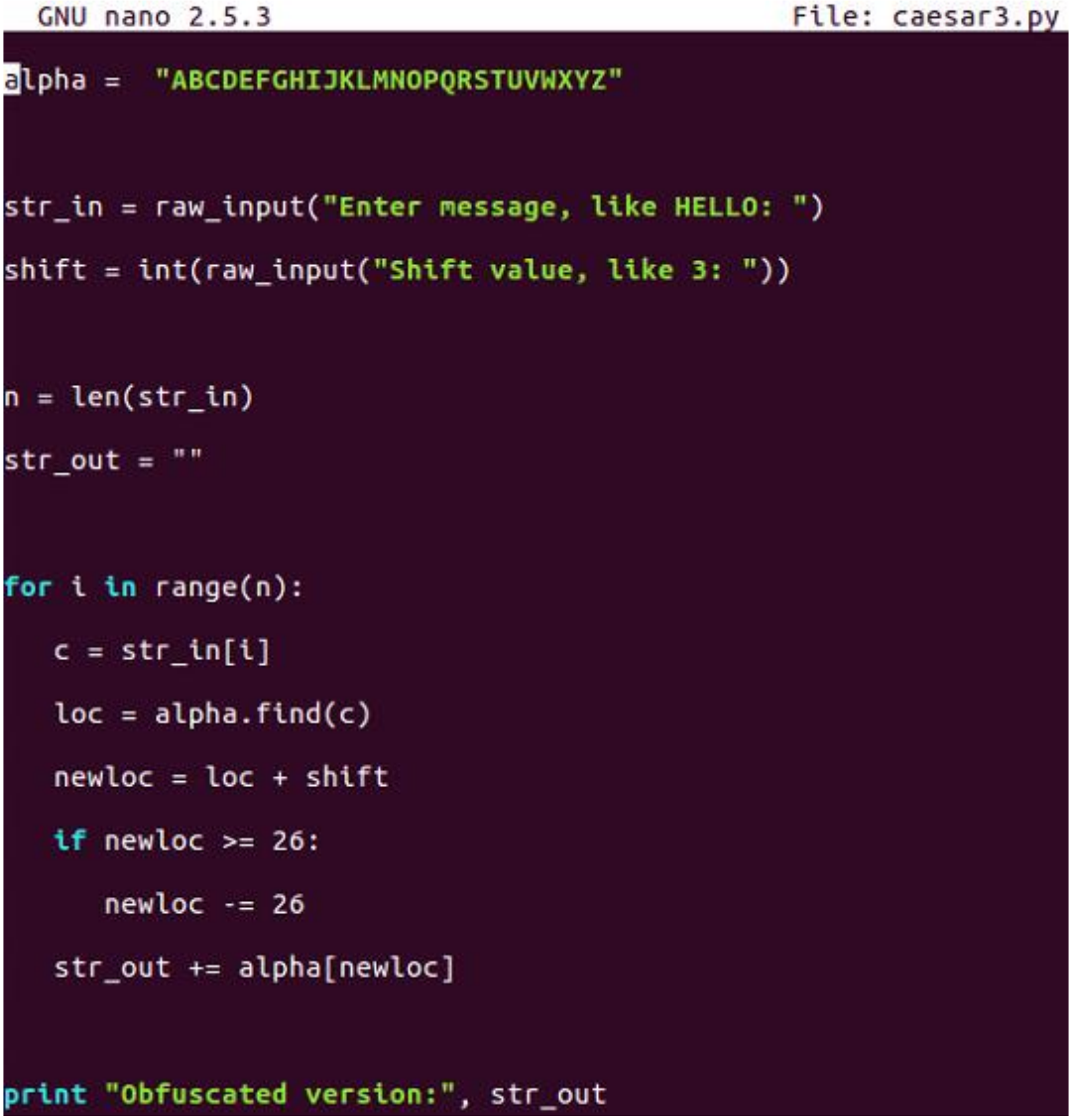
```
test@PPMUMCPU0372:~$ python caesar4.py
Enter message, like HELLO: HELLO
Shift value, like 3: 3
Obfuscated version: KHOOR
test@PPMUMCPU0372:~$ python caesar4.py
Enter message, like HELLO: HELLO
Shift value, like 3: 300
Obfuscated version: VSZZC
test@PPMUMCPU0372:~$
```



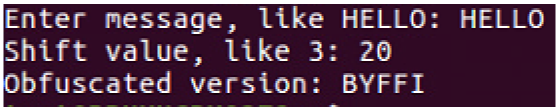
```
alpha =  "ABCDEFGHIJKLMNOPQRSTUVWXYZ"


str_in = raw_input("Enter message, like HELLO: ")
shift = int(raw_input("Shift value, like 3: "))


n = len(str_in)
str_out = ""


for i in range(n):
   c = str_in[i]
   loc = alpha.find(c)
   newloc = loc + shift
   str_out += alpha[newloc]


print "Obfuscated version:", str_out
```



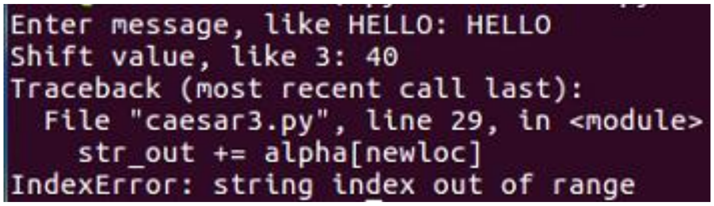
```
Enter message, like HELLO: HELLO
Shift value, like 3: 20
Traceback (most recent call last):
  File "caesar2.py", line 25, in <module>
    str_out += alpha[newloc]
IndexError: string index out of range
test@PPMUMCPU0372:~$ 
```



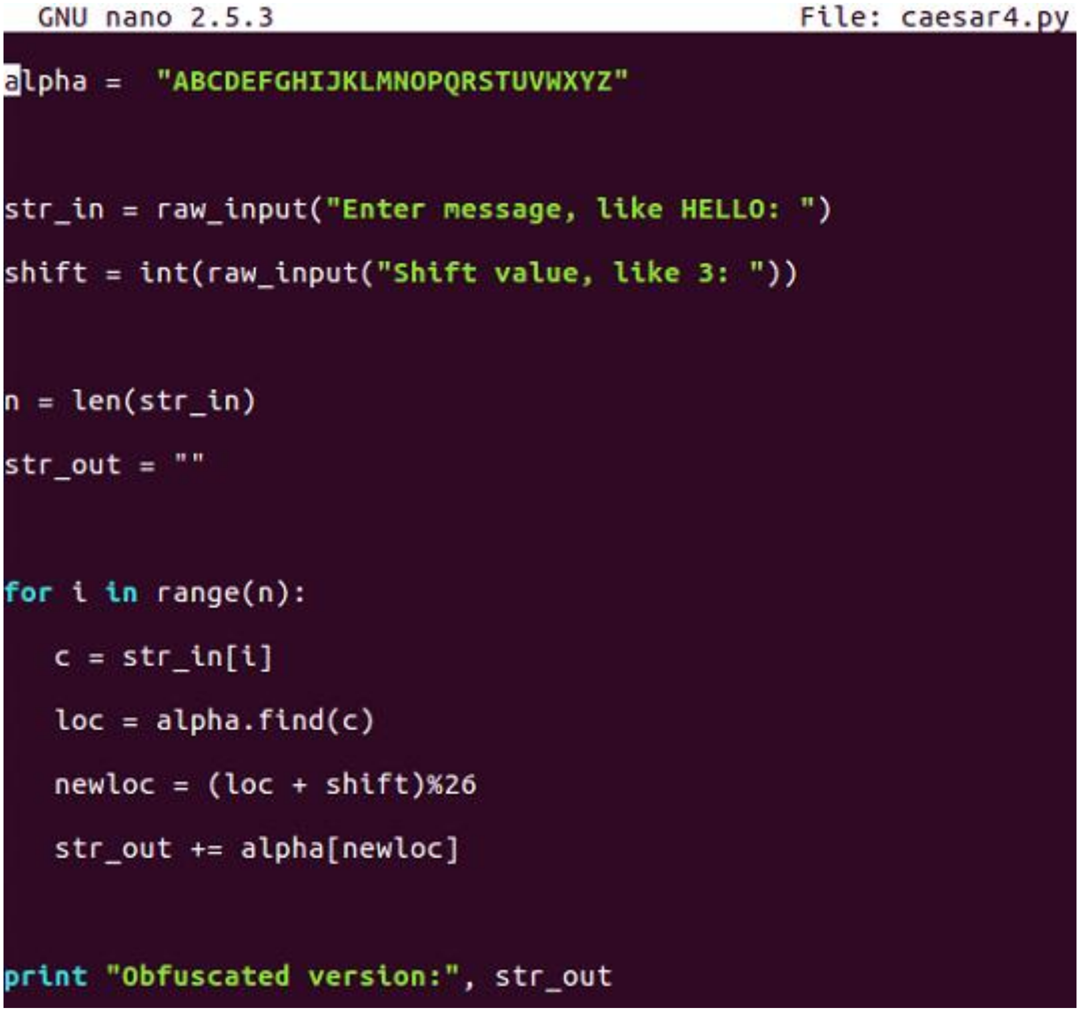
```
  GNU nano 2.5.3                                    File: caesar3.py

alpha =  "ABCDEFGHIJKLMNOPQRSTUVWXYZ"



str_in = raw_input("Enter message, like HELLO: ")

shift = int(raw_input("Shift value, like 3: "))



n = len(str_in)

str_out = ""



for i in range(n):

   c = str_in[i]

   loc = alpha.find(c)

   newloc = loc + shift

   if newloc >= 26:

      newloc -= 26

   str_out += alpha[newloc]



print "Obfuscated version:", str_out
```



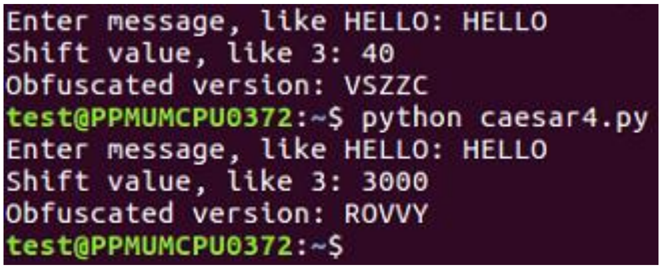
```
Enter message, like HELLO: HELLO
Shift value, like 3: 20
Obfuscated version: BYFFI
```



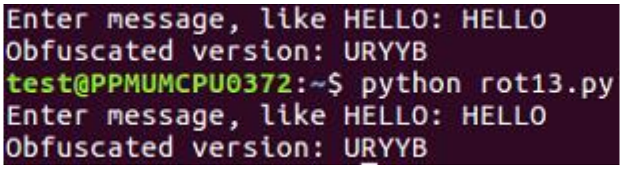
```
Enter message, like HELLO: HELLO
Shift value, like 3: 40
Traceback (most recent call last):
  File "caesar3.py", line 29, in <module>
    str_out += alpha[newloc]
IndexError: string index out of range
```



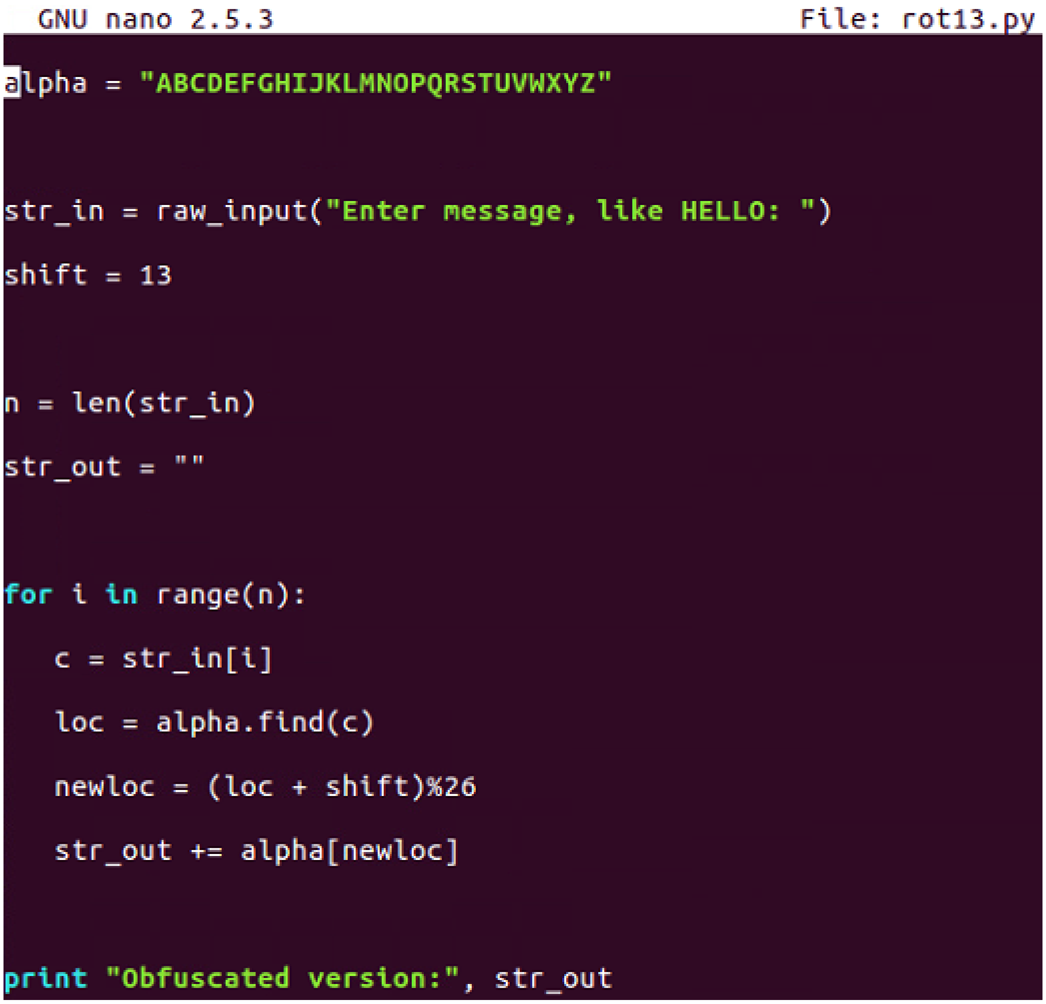
```
  GNU nano 2.5.3                                     File: caesar4.py

alpha =  "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

str_in = raw_input("Enter message, like HELLO: ")
shift = int(raw_input("Shift value, like 3: "))

n = len(str_in)
str_out = ""

for i in range(n):
   c = str_in[i]
   loc = alpha.find(c)
   newloc = (loc + shift)%26
   str_out += alpha[newloc]

print "Obfuscated version:", str_out
```



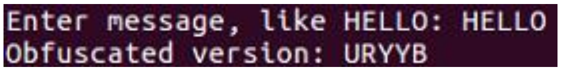
```
Enter message, like HELLO: HELLO
Shift value, like 3: 40
Obfuscated version: VSZZC
test@PPMUMCPU0372:~$ python caesar4.py
Enter message, like HELLO: HELLO
Shift value, like 3: 3000
Obfuscated version: ROVVY
test@PPMUMCPU0372:~$
```



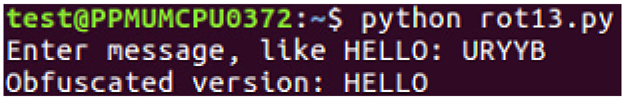
```
Enter message, like HELLO: HELLO
Obfuscated version: URYYB
test@PPMUMCPU0372:~$ python rot13.py
Enter message, like HELLO: HELLO
Obfuscated version: URYYB
```



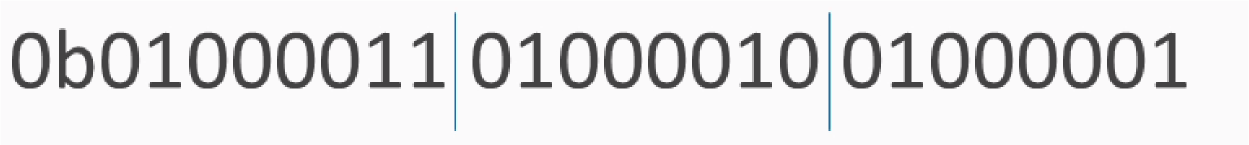
```
  GNU nano 2.5.3                                    File: rot13.py

alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

str_in = raw_input("Enter message, like HELLO: ")
shift = 13

n = len(str_in)
str_out = ""

for i in range(n):
   c = str_in[i]
   loc = alpha.find(c)
   newloc = (loc + shift)%26
   str_out += alpha[newloc]

print "Obfuscated version:", str_out
```



```
Enter message, like HELLO: HELLO
Obfuscated version: URYYB
```



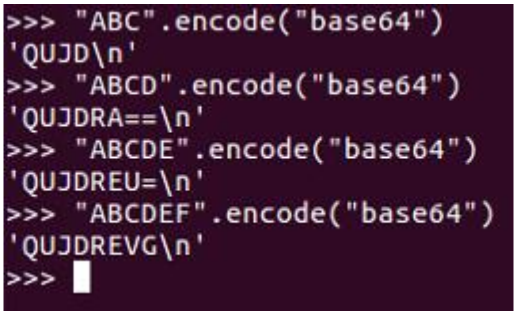
```
test@PPMUMCPU0372:~$ python rot13.py
Enter message, like HELLO: URYYB
Obfuscated version: HELLO
```



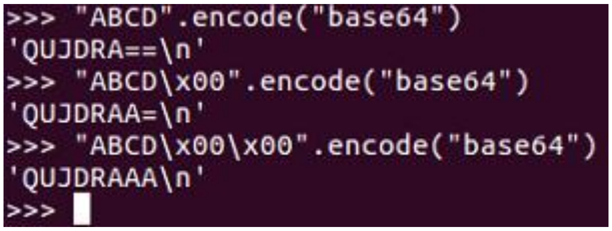
0b01000011 01000010 01000001

- A, B, C, ... Z for 0 - 25
- a, b, c, ... z for 26 - 51
- 0, 1, 2, ... 9 for 52 - 61
- +, / for 62 and 63

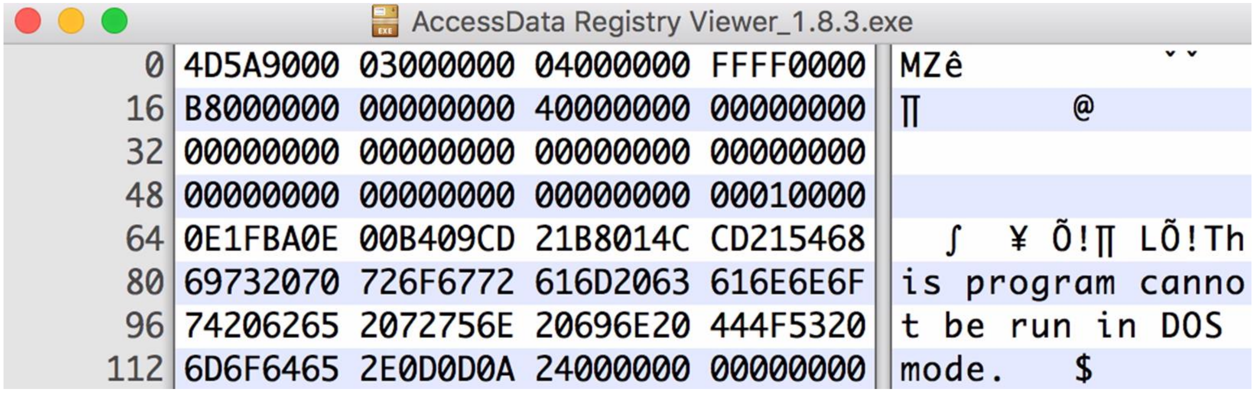
- 'ABC' is 0b01000001 01000010 01000011
- 6-bit groups 0b010000 010100 001001 000011
- Decimal 16 20 9 3
- BASE64 Q U J D



```
>>> "ABC".encode("base64")
'QUJD\n'
>>> "ABCD".encode("base64")
'QUJDRA==\n'
>>> "ABCDE".encode("base64")
'QUJDREU=\n'
>>> "ABCDEF".encode("base64")
'QUJDREVG\n'
>>> 
```



```
>>> "ABCD".encode("base64")
'QUJDRA==\n'
>>> "ABCD\x00".encode("base64")
'QUJDRAA=\n'
>>> "ABCD\x00\x00".encode("base64")
'QUJDRAAA\n'
>>> 
```

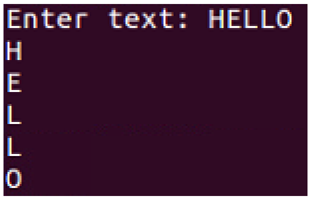


```
  0 4D5A9000 03000000 04000000 FFFF0000  MZê              ˇˇ
 16 B8000000 00000000 40000000 00000000  ∏        @
 32 00000000 00000000 00000000 00000000
 48 00000000 00000000 00000000 00010000
 64 0E1FBA0E 00B409CD 21B8014C CD215468    ∫   ¥ Õ!∏ LÕ!Th
 80 69732070 726F6772 616D2063 616E6E6F  is program canno
 96 74206265 2072756E 20696E20 444F5320  t be run in DOS
112 6D6F6465 2E0D0D0A 24000000 00000000  mode.    $
```

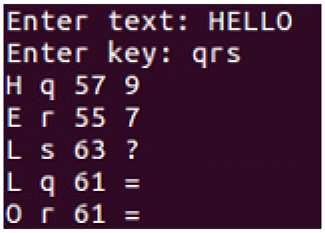
Single-byte and Multi-byte XOR

- Single-Byte XOR: Use the same key for every byte
- 'ABC' 0b01000001 01000010 01000011
- 'B' repeated 0b01000010 01000010 01000010
- XOR 0b00000011 00000000 00000001

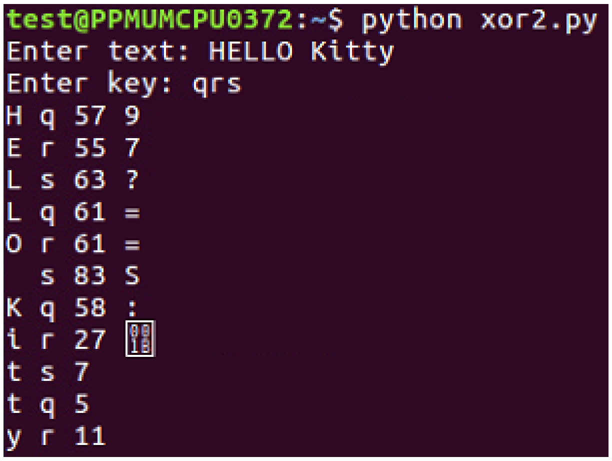
- Multi-Byte XOR: Repeat a pattern
- 'ABC' 0b01000001 01000010 01000011
- 'BC' repeated 0b01000010 01000011 01000010
- XOR 0b00000011 00000001 00000001



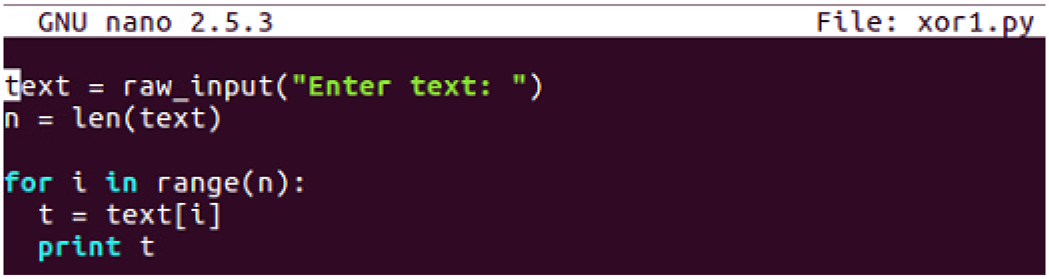
```
Enter text: HELLO
H
E
L
L
O
```



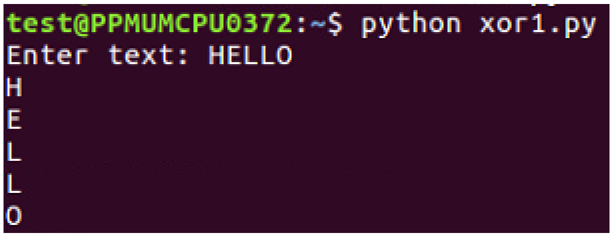
```
Enter text: HELLO
Enter key: qrs
H q 57 9
E r 55 7
L s 63 ?
L q 61 =
O r 61 =
```



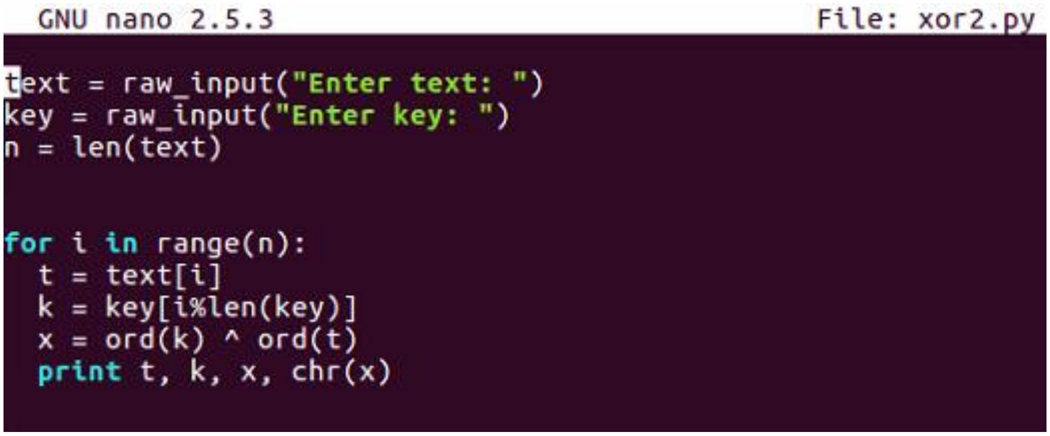
```
test@PPMUMCPU0372:~$ python xor2.py
Enter text: HELLO Kitty
Enter key: qrs
H q 57 9
E r 55 7
L s 63 ?
L q 61 =
O r 61 =
  s 83 S
K q 58 :
i r 27 ␛
t s 7
t q 5
y r 11
```



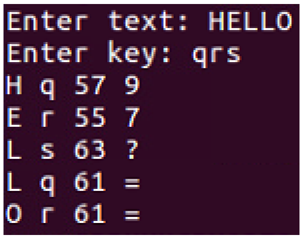
```
  GNU nano 2.5.3                                File: xor1.py

text = raw_input("Enter text: ")
n = len(text)

for i in range(n):
  t = text[i]
  print t
```



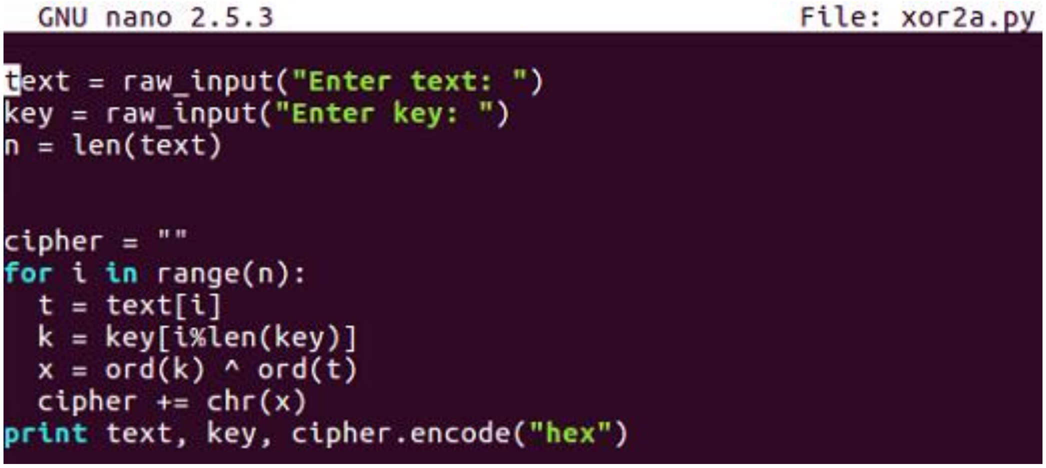
```
test@PPMUMCPU0372:~$ python xor1.py
Enter text: HELLO
H
E
L
L
O
```



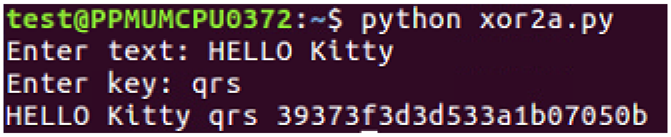
```
  GNU nano 2.5.3                                          File: xor2.py

text = raw_input("Enter text: ")
key = raw_input("Enter key: ")
n = len(text)


for i in range(n):
  t = text[i]
  k = key[i%len(key)]
  x = ord(k) ^ ord(t)
  print t, k, x, chr(x)
```



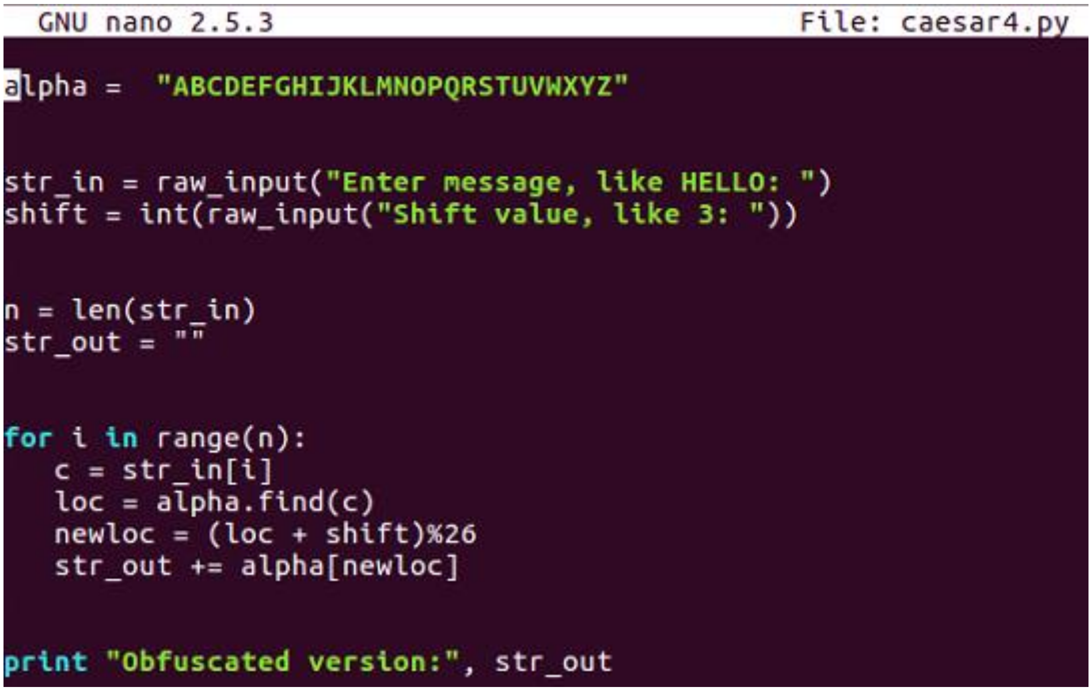
```
Enter text: HELLO
Enter key: qrs
H q 57 9
E r 55 7
L s 63 ?
L q 61 =
O r 61 =
```



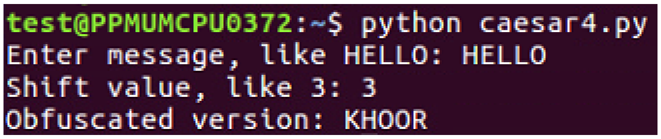
```
  GNU nano 2.5.3                                         File: xor2a.py

text = raw_input("Enter text: ")
key = raw_input("Enter key: ")
n = len(text)


cipher = ""
for i in range(n):
  t = text[i]
  k = key[i%len(key)]
  x = ord(k) ^ ord(t)
  cipher += chr(x)
print text, key, cipher.encode("hex")
```



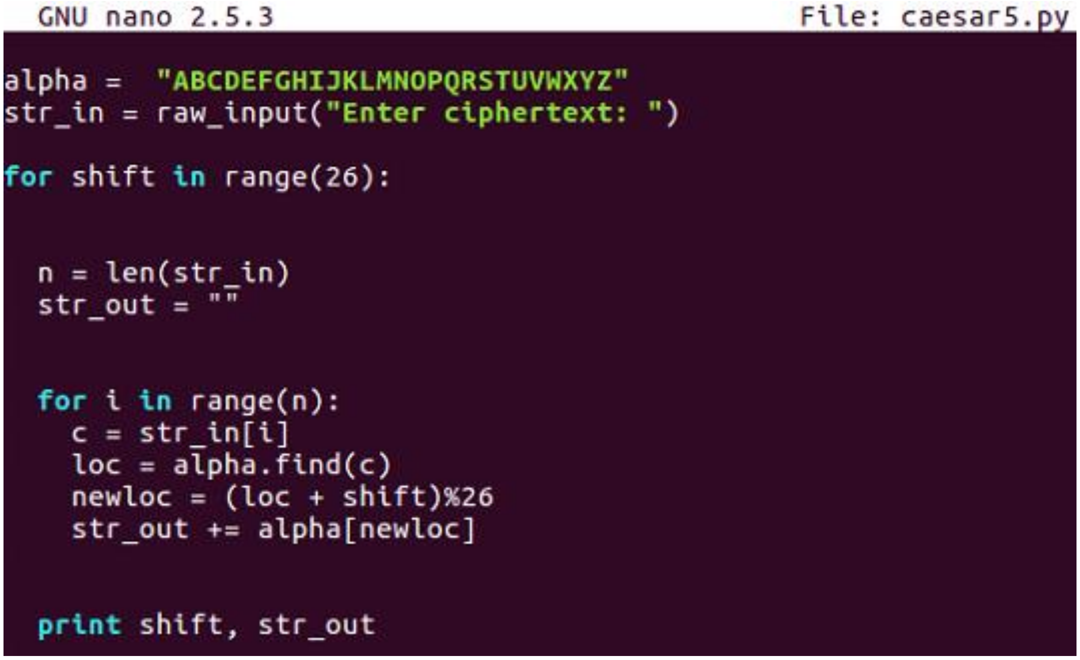
```
test@PPMUMCPU0372:~$ python xor2a.py
Enter text: HELLO Kitty
Enter key: qrs
HELLO Kitty qrs 39373f3d3d533a1b07050b
```



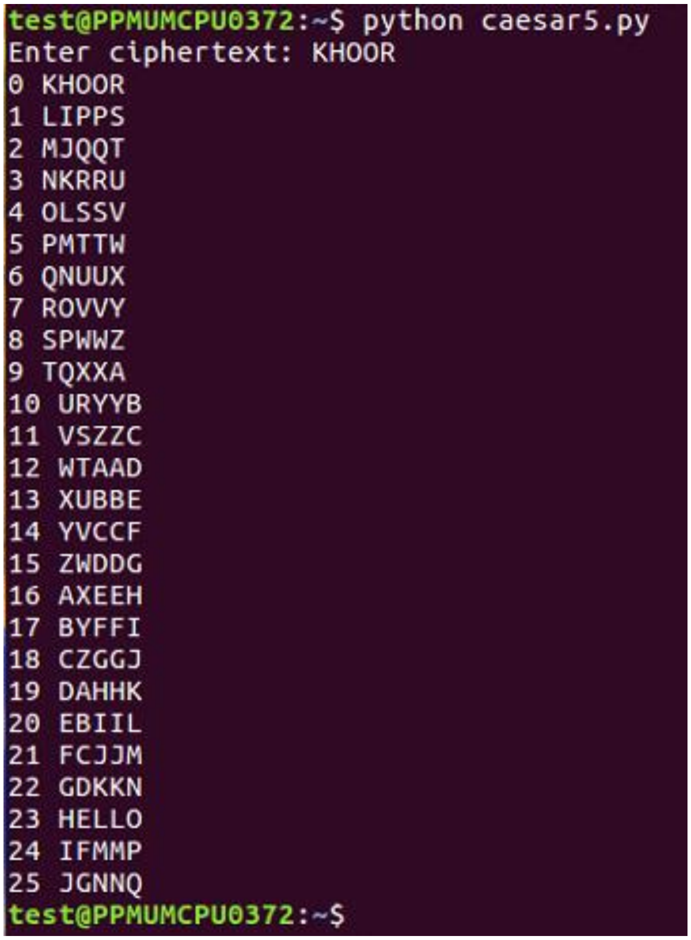
```
  GNU nano 2.5.3                                  File: caesar4.py

alpha =  "ABCDEFGHIJKLMNOPQRSTUVWXYZ"


str_in = raw_input("Enter message, like HELLO: ")
shift = int(raw_input("Shift value, like 3: "))


n = len(str_in)
str_out = ""


for i in range(n):
   c = str_in[i]
   loc = alpha.find(c)
   newloc = (loc + shift)%26
   str_out += alpha[newloc]


print "Obfuscated version:", str_out
```



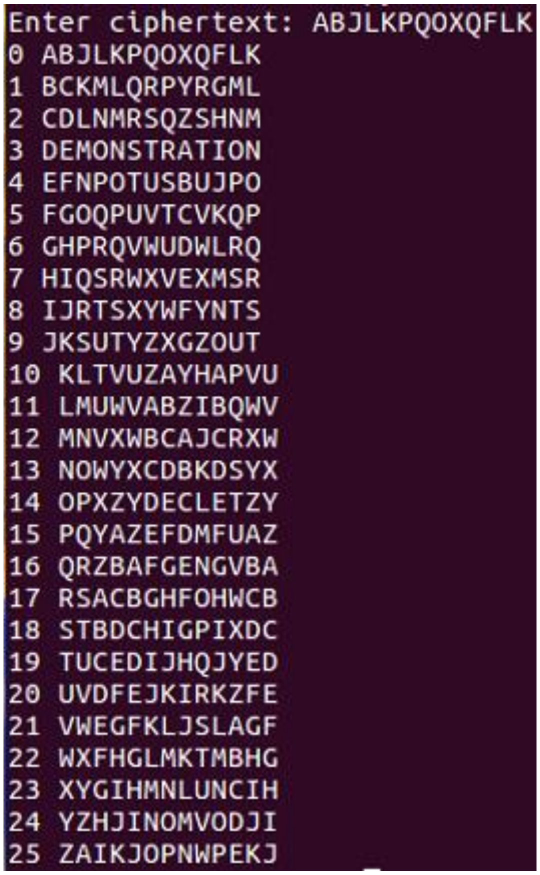
```
test@PPMUMCPU0372:~$ python caesar4.py
Enter message, like HELLO: HELLO
Shift value, like 3: 3
Obfuscated version: KHOOR
```



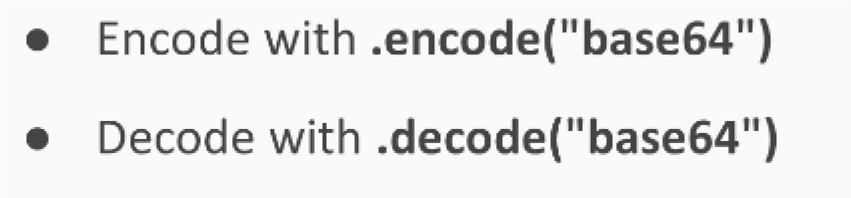
```
  GNU nano 2.5.3                                  File: caesar5.py

alpha =  "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
str_in = raw_input("Enter ciphertext: ")

for shift in range(26):


  n = len(str_in)
  str_out = ""


  for i in range(n):
    c = str_in[i]
    loc = alpha.find(c)
    newloc = (loc + shift)%26
    str_out += alpha[newloc]


  print shift, str_out
```



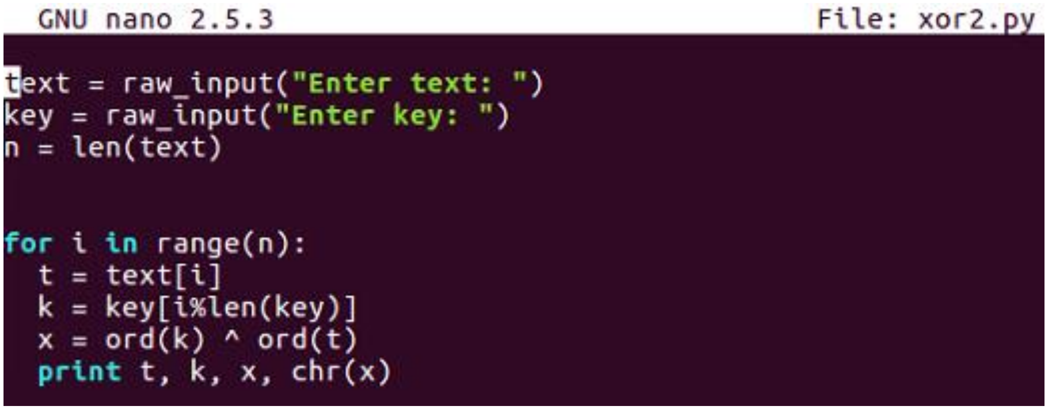
```
test@PPMUMCPU0372:~$ python caesar5.py
Enter ciphertext: KHOOR
0 KHOOR
1 LIPPS
2 MJQQT
3 NKRRU
4 OLSSV
5 PMTTW
6 QNUUX
7 ROVVY
8 SPWWZ
9 TQXXA
10 URYYB
11 VSZZC
12 WTAAD
13 XUBBE
14 YVCCF
15 ZWDDG
16 AXEEH
17 BYFFI
18 CZGGJ
19 DAHHK
20 EBIIL
21 FCJJM
22 GDKKN
23 HELLO
24 IFMMP
25 JGNNQ
test@PPMUMCPU0372:~$
```



```
Enter ciphertext: ABJLKPQOXQFLK
0 ABJLKPQOXQFLK
1 BCKMLQRPYRGML
2 CDLNMRSQZSHNM
3 DEMONSTRATION
4 EFNPOTUSBUJPO
5 FGOQPUVTCVKQP
6 GHPRQVWUDWLRQ
7 HIQSRWXVEXMSR
8 IJRTSXYWFYNTS
9 JKSUTYZXGZOUT
10 KLTVUZAYHAPVU
11 LMUWVABZIBQWV
12 MNVXWBCAJCRXW
13 NOWYXCDBKDSYX
14 OPXZYDECLETZY
15 PQYAZEFDMFUAZ
16 QRZBAFGENGVBA
17 RSACBGHFOHWCB
18 STBDCHIGPIXDC
19 TUCEDIJHQJYED
20 UVDFEJKIRKZFE
21 VWEGFKLJSLAGF
22 WXFHGLMKTMBHG
23 XYGIHMNLUNCIH
24 YZHJINOMVODJI
25 ZAIKJOPNWPEKJ
```

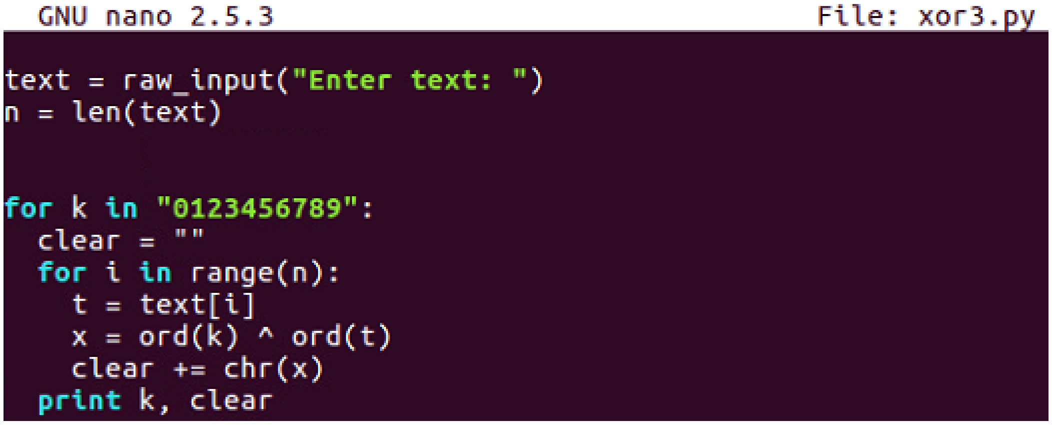


- Encode with **.encode("base64")**
- Decode with **.decode("base64")**

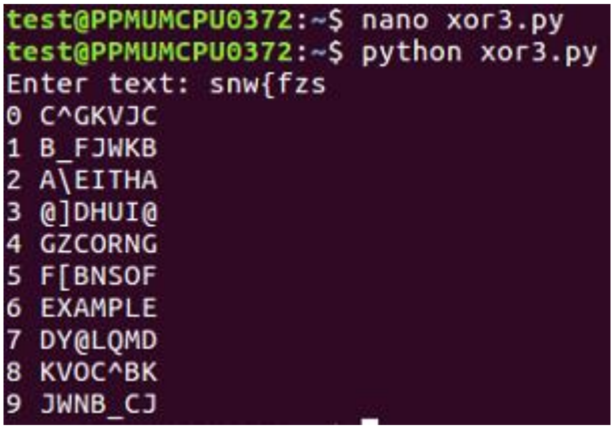


```
  GNU nano 2.5.3                                   File: xor2.py

text = raw_input("Enter text: ")
key = raw_input("Enter key: ")
n = len(text)


for i in range(n):
  t = text[i]
  k = key[i%len(key)]
  x = ord(k) ^ ord(t)
  print t, k, x, chr(x)
```

- Ciphertext: **snw{fzs**
- Key is **6**



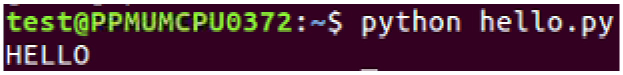
```
  GNU nano 2.5.3                                          File: xor3.py

text = raw_input("Enter text: ")
n = len(text)


for k in "0123456789":
  clear = ""
  for i in range(n):
    t = text[i]
    x = ord(k) ^ ord(t)
    clear += chr(x)
  print k, clear
```



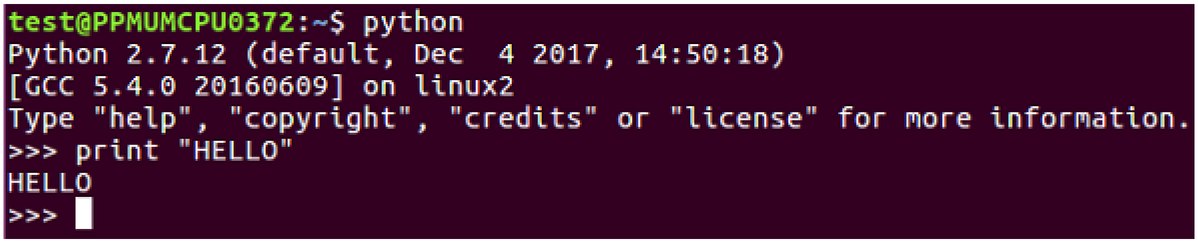
```
test@PPMUMCPU0372:~$ nano xor3.py
test@PPMUMCPU0372:~$ python xor3.py
Enter text: snw{fzs
0 C^GKVJC
1 B_FJWKB
2 A\EITHA
3 @]DHUI@
4 GZCORNG
5 F[BNSOF
6 EXAMPLE
7 DY@LQMD
8 KVOC^BK
9 JWNB_CJ
```

- Decipher this: **kquht}**
- Key is a single digit

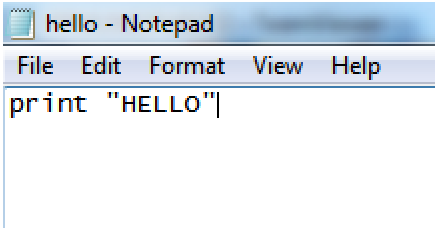
- Decipher this: **70155d5c45415d5011585446424c**
- Key is two digits of ASCII



```
test@PPMUMCPU0372:~$ python hello.py
HELLO
```

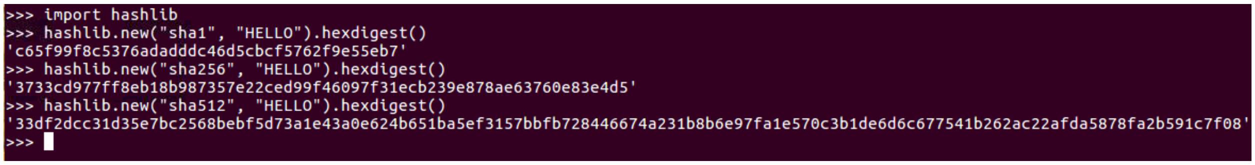


```
test@PPMUMCPU0372:~$ python
Python 2.7.12 (default, Dec  4 2017, 14:50:18)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print "HELLO"
HELLO
>>> 
```

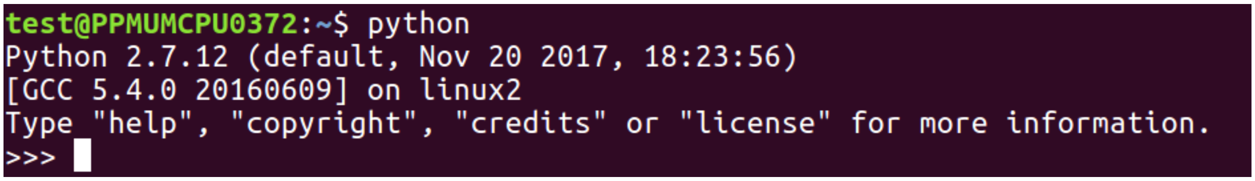


```
print "HELLO"
```

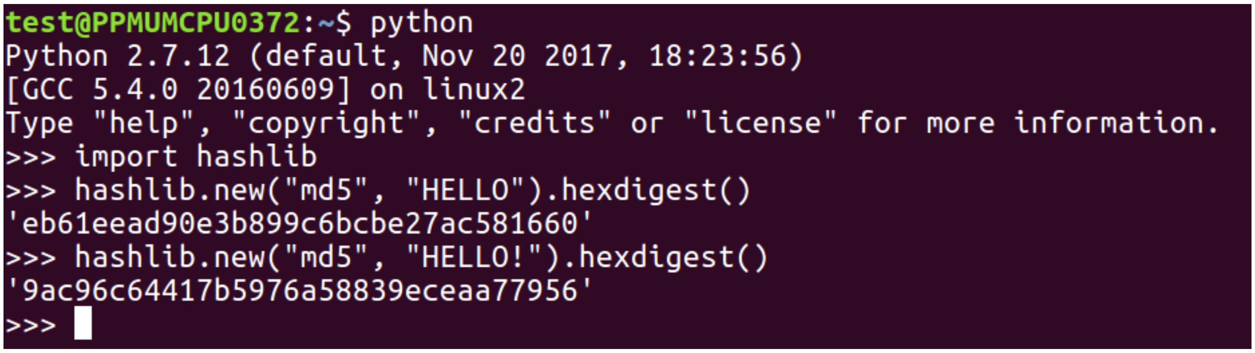
Chapter 2: Hashing



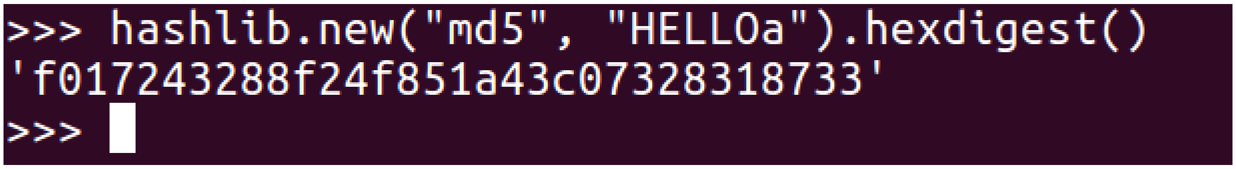
```
>>> import hashlib
>>> hashlib.new("sha1", "HELLO").hexdigest()
'c65f99f8c5376adadddc46d5cbcf5762f9e55eb7'
>>> hashlib.new("sha256", "HELLO").hexdigest()
'3733cd977ff8eb18b987357e22ced99f46097f31ecb239e878ae63760e83e4d5'
>>> hashlib.new("sha512", "HELLO").hexdigest()
'33df2dcc31d35e7bc2568bebf5d73a1e43a0e624b651ba5ef3157bbfb728446674a231b8b6e97fa1e570c3b1de6d6c677541b262ac22afda5878fa2b591c7f08'
>>> 
```



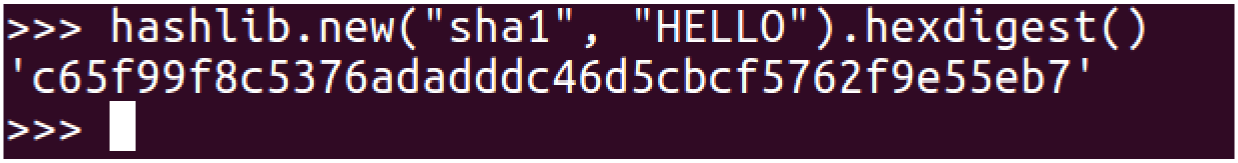
```
test@PPMUMCPU0372:~$ python
Python 2.7.12 (default, Nov 20 2017, 18:23:56)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```



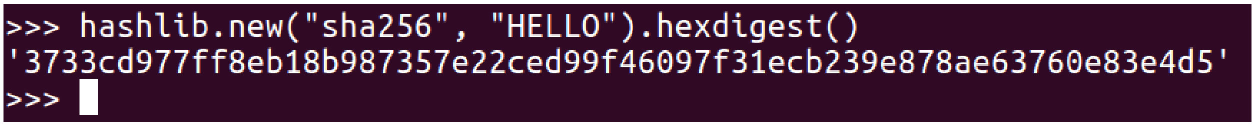
```
test@PPMUMCPU0372:~$ python
Python 2.7.12 (default, Nov 20 2017, 18:23:56)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import hashlib
>>> hashlib.new("md5", "HELLO").hexdigest()
'eb61eead90e3b899c6bcbe27ac581660'
>>> hashlib.new("md5", "HELLO!").hexdigest()
'9ac96c64417b5976a58839eceaa77956'
>>> 
```



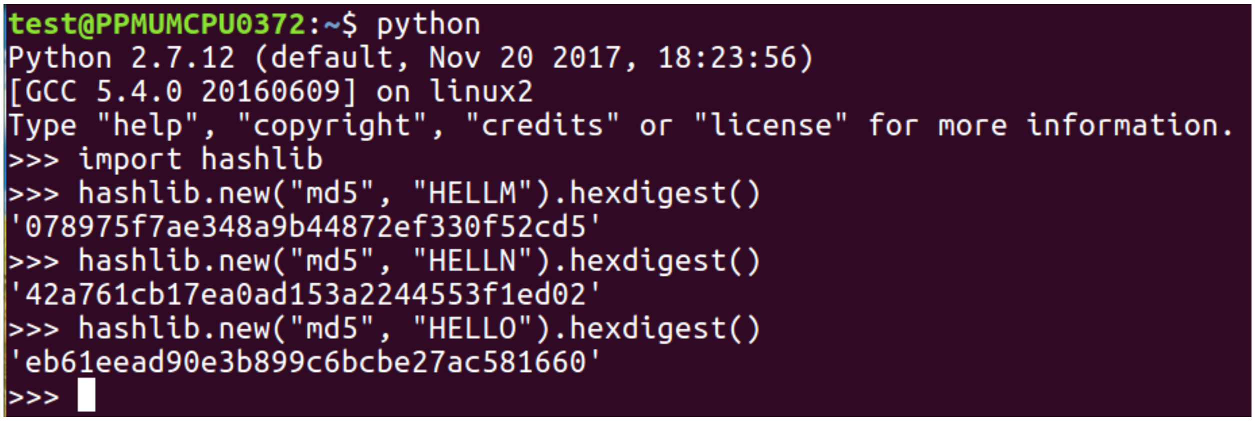
```
>>> hashlib.new("md5", "HELLOa").hexdigest()
'f017243288f24f851a43c07328318733'
>>> 
```



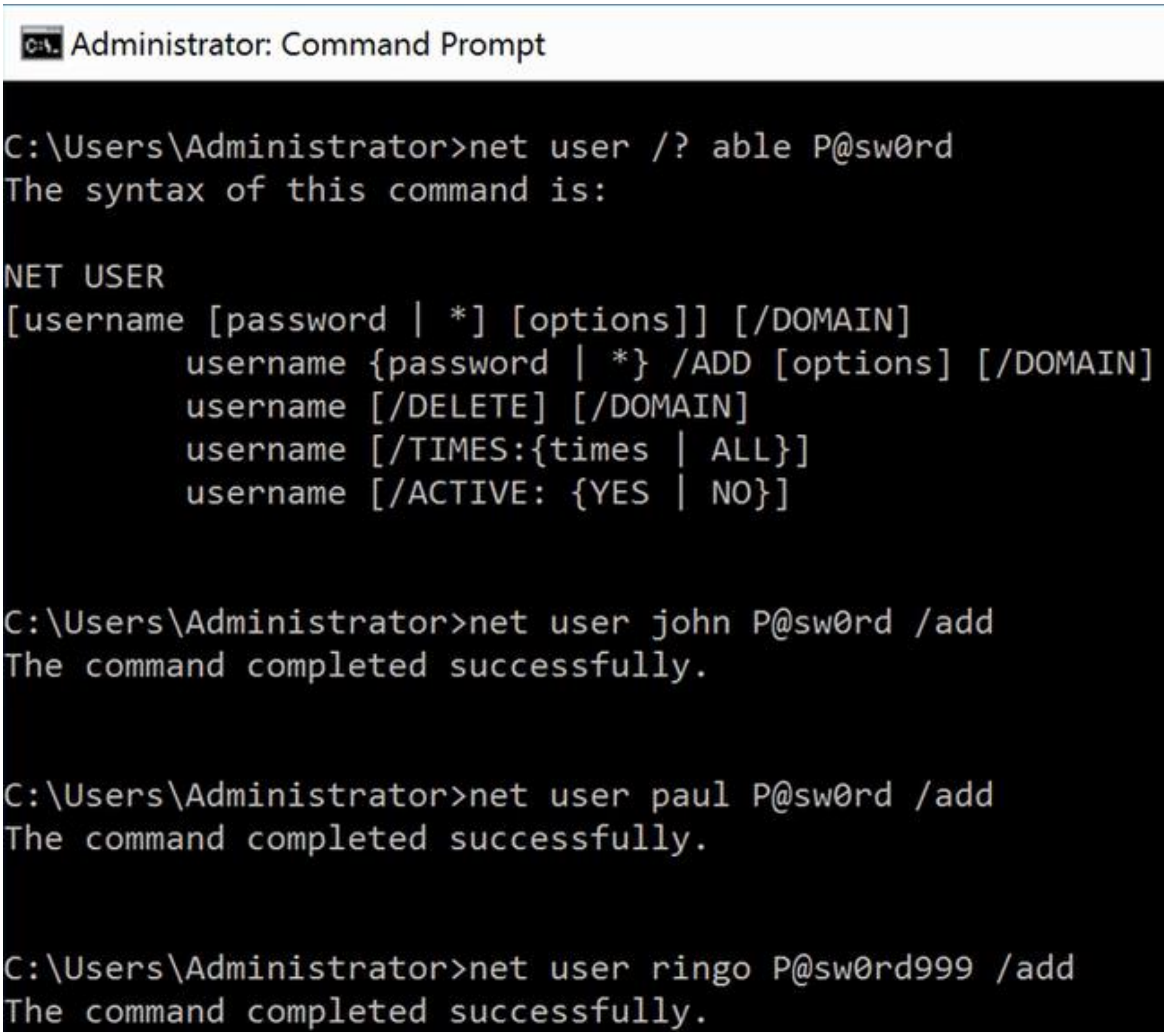
```
>>> hashlib.new("sha1", "HELLO").hexdigest()
'c65f99f8c5376adadddc46d5cbcf5762f9e55eb7'
>>> 
```



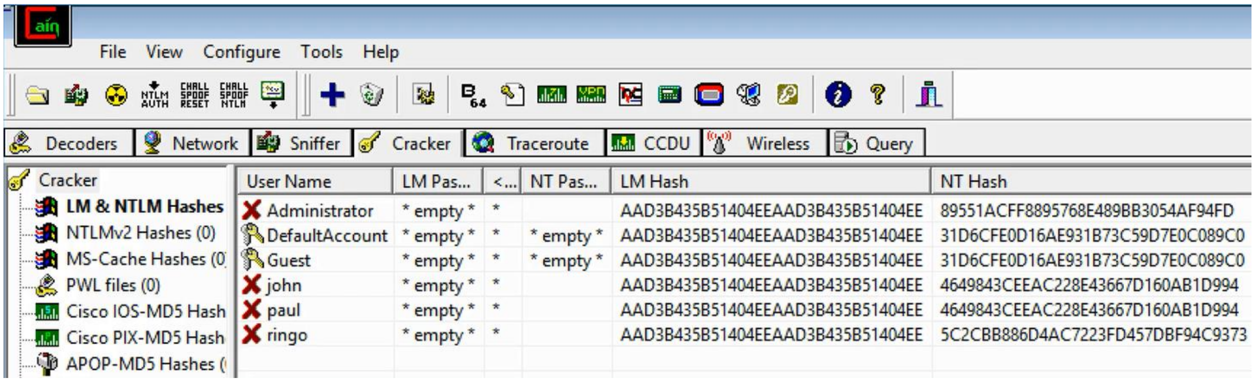
```
>>> hashlib.new("sha256", "HELLO").hexdigest()
'3733cd977ff8eb18b987357e22ced99f46097f31ecb239e878ae63760e83e4d5'
>>> 
```



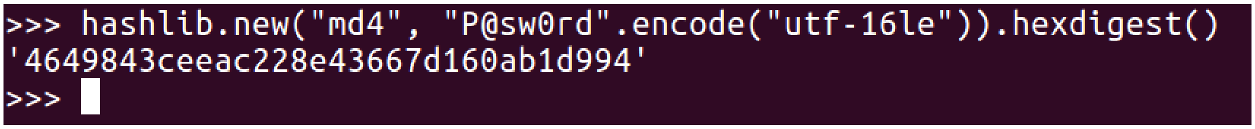
```
test@PPMUMCPU0372:~$ python
Python 2.7.12 (default, Nov 20 2017, 18:23:56)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import hashlib
>>> hashlib.new("md5", "HELLM").hexdigest()
'078975f7ae348a9b44872ef330f52cd5'
>>> hashlib.new("md5", "HELLN").hexdigest()
'42a761cb17ea0ad153a2244553f1ed02'
>>> hashlib.new("md5", "HELLO").hexdigest()
'eb61eead90e3b899c6bcbe27ac581660'
>>>
```



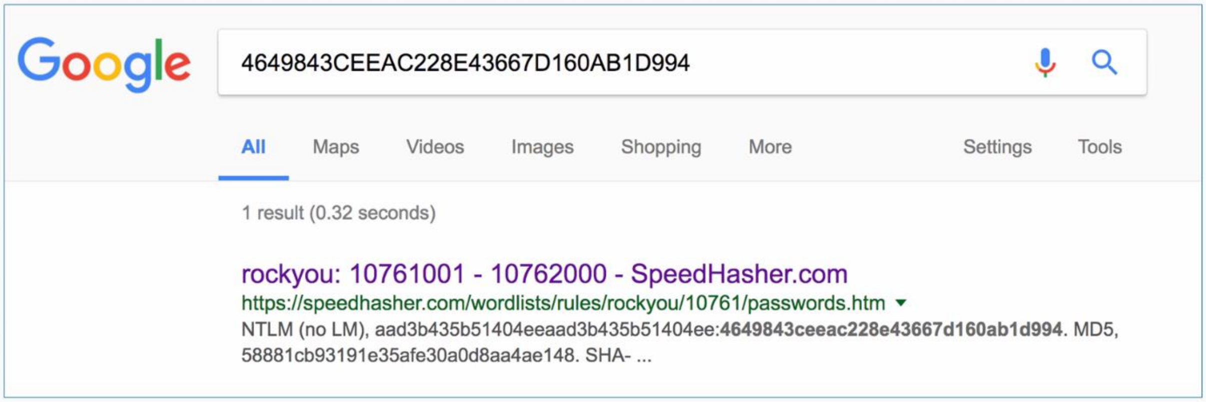
```
C:\Users\Administrator>net user /? able P@sw0rd
The syntax of this command is:

NET USER
[username [password | *] [options]] [/DOMAIN]
         username {password | *} /ADD [options] [/DOMAIN]
         username [/DELETE] [/DOMAIN]
         username [/TIMES:{times | ALL}]
         username [/ACTIVE: {YES | NO}]


C:\Users\Administrator>net user john P@sw0rd /add
The command completed successfully.


C:\Users\Administrator>net user paul P@sw0rd /add
The command completed successfully.


C:\Users\Administrator>net user ringo P@sw0rd999 /add
The command completed successfully.
```



User Name	LM Pas...	<...	NT Pas...	LM Hash	NT Hash
Administrator	* empty *	*		AAD3B435B51404EEAAD3B435B51404EE	89551ACFF8895768E489BB3054AF94FD
DefaultAccount	* empty *	*	* empty *	AAD3B435B51404EEAAD3B435B51404EE	31D6CFE0D16AE931B73C59D7E0C089C0
Guest	* empty *	*	* empty *	AAD3B435B51404EEAAD3B435B51404EE	31D6CFE0D16AE931B73C59D7E0C089C0
john	* empty *	*		AAD3B435B51404EEAAD3B435B51404EE	4649843CEEAC228E43667D160AB1D994
paul	* empty *	*		AAD3B435B51404EEAAD3B435B51404EE	4649843CEEAC228E43667D160AB1D994
ringo	* empty *	*		AAD3B435B51404EEAAD3B435B51404EE	5C2CBB886D4AC7223FD457DBF94C9373



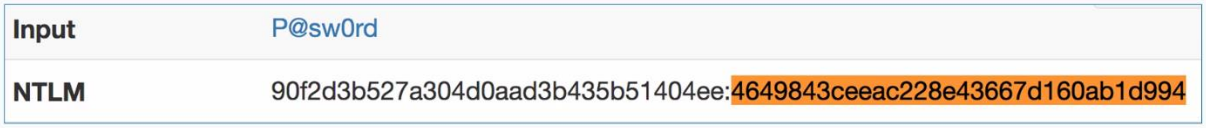
```
>>> hashlib.new("md4", "P@sw0rd".encode("utf-16le")).hexdigest()
'4649843ceeac228e43667d160ab1d994'
>>> 
```



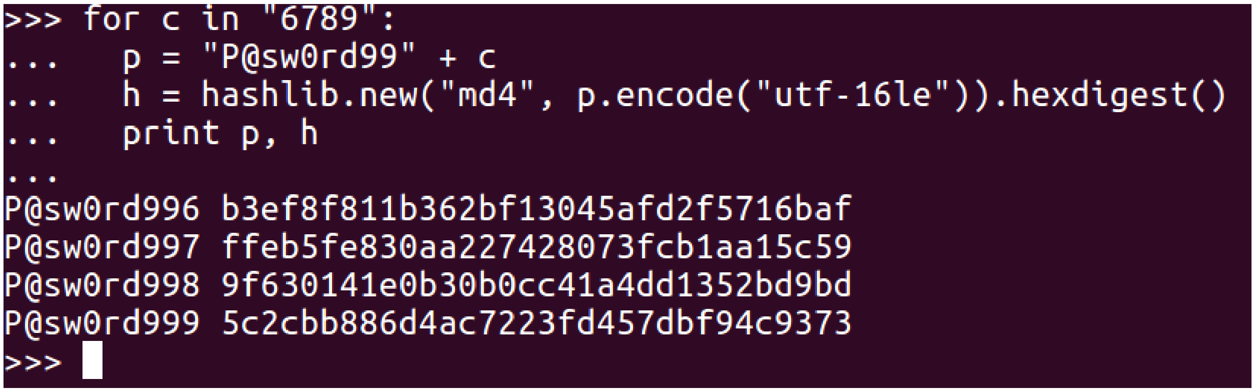
4649843CEEAC228E43667D160AB1D994

1 result (0.32 seconds)

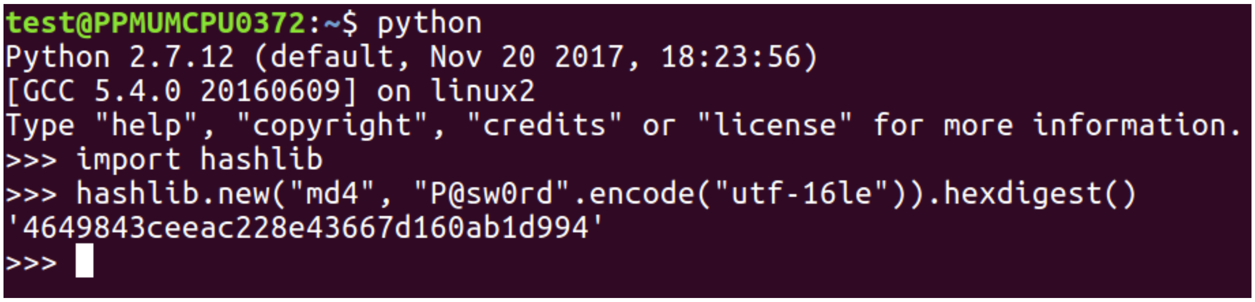
rockyou: 10761001 - 10762000 - SpeedHasher.com
https://speedhasher.com/wordlists/rules/rockyou/10761/passwords.htm
NTLM (no LM), aad3b435b51404eeaad3b435b51404ee:**4649843ceeac228e43667d160ab1d994**. MD5, 58881cb93191e35afe30a0d8aa4ae148. SHA- ...



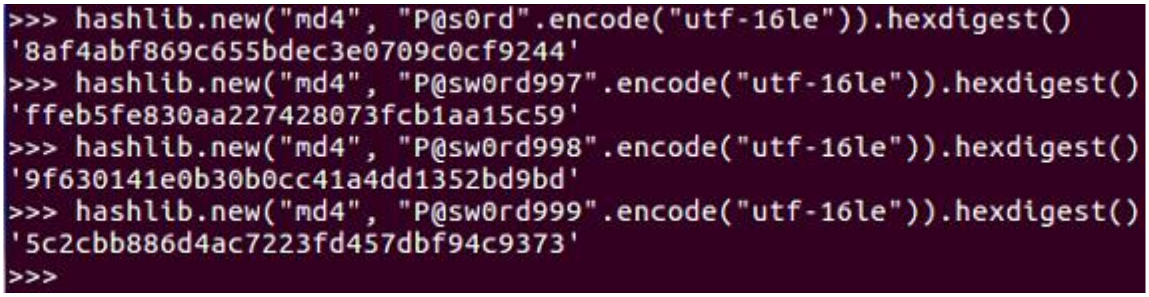
Input	P@sw0rd
NTLM	90f2d3b527a304d0aad3b435b51404ee:4649843ceeac228e43667d160ab1d994



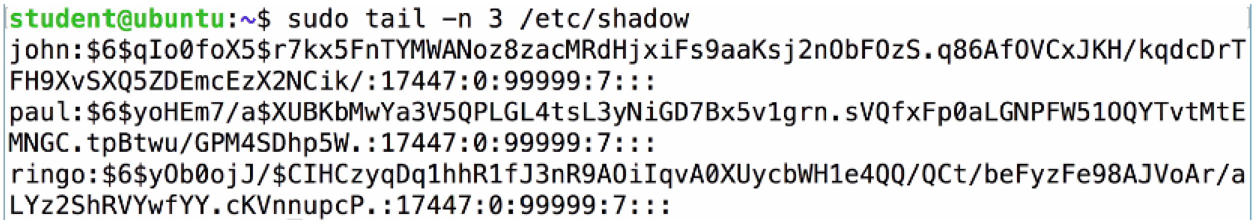
```
>>> for c in "6789":
...     p = "P@sw0rd99" + c
...     h = hashlib.new("md4", p.encode("utf-16le")).hexdigest()
...     print p, h
...
P@sw0rd996 b3ef8f811b362bf13045afd2f5716baf
P@sw0rd997 ffeb5fe830aa227428073fcb1aa15c59
P@sw0rd998 9f630141e0b30b0cc41a4dd1352bd9bd
P@sw0rd999 5c2cbb886d4ac7223fd457dbf94c9373
>>> 
```



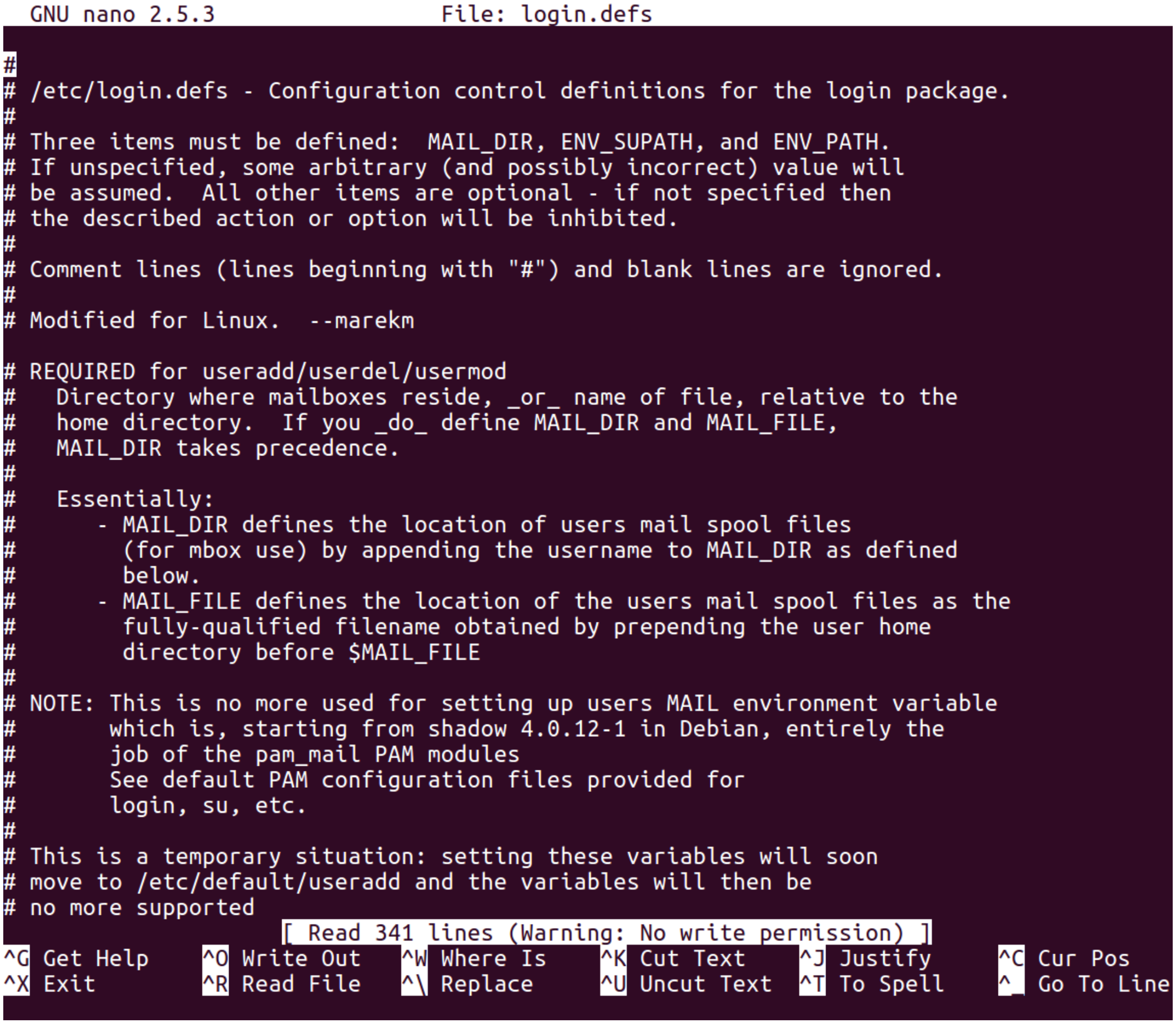
```
test@PPMUMCPU0372:~$ python
Python 2.7.12 (default, Nov 20 2017, 18:23:56)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import hashlib
>>> hashlib.new("md4", "P@sw0rd".encode("utf-16le")).hexdigest()
'4649843ceeac228e43667d160ab1d994'
>>> 
```



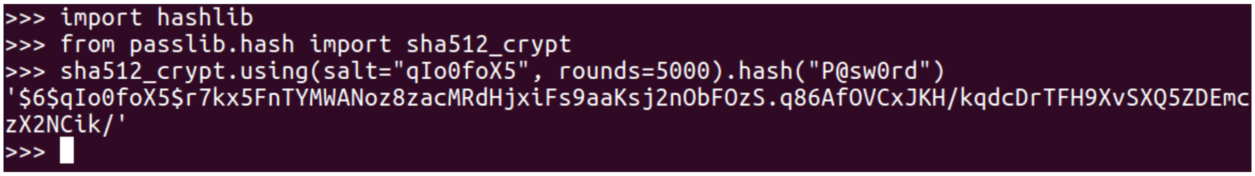
```
>>> hashlib.new("md4", "P@s0rd".encode("utf-16le")).hexdigest()
'8af4abf869c655bdec3e0709c0cf9244'
>>> hashlib.new("md4", "P@sw0rd997".encode("utf-16le")).hexdigest()
'ffeb5fe830aa227428073fcb1aa15c59'
>>> hashlib.new("md4", "P@sw0rd998".encode("utf-16le")).hexdigest()
'9f630141e0b30b0cc41a4dd1352bd9bd'
>>> hashlib.new("md4", "P@sw0rd999".encode("utf-16le")).hexdigest()
'5c2cbb886d4ac7223fd457dbf94c9373'
>>>
```



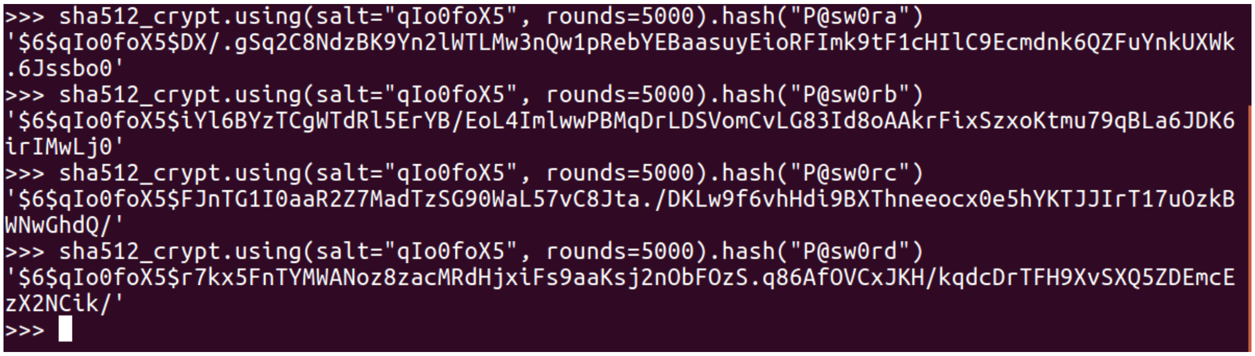
```
student@ubuntu:~$ sudo tail -n 3 /etc/shadow
john:$6$qIo0foX5$r7kx5FnTYMWANoz8zacMRdHjxiFs9aaKsj2nObFOzS.q86AfOVCxJKH/kqdcDrT
FH9XvSXQ5ZDEmcEzX2NCik/:17447:0:99999:7:::
paul:$6$yoHEm7/a$XUBKbMwYa3V5QPLGL4tsL3yNiGD7Bx5v1grn.sVQfxFp0aLGNPFW51OQYTvtMtE
MNGC.tpBtwu/GPM4SDhp5W.:17447:0:99999:7:::
ringo:$6$yOb0ojJ/$CIHCzyqDq1hhR1fJ3nR9AOiIqvA0XUycbWH1e4QQ/QCt/beFyzFe98AJVoAr/a
LYz2ShRVYwfYY.cKVnnupcP.:17447:0:99999:7:::
```



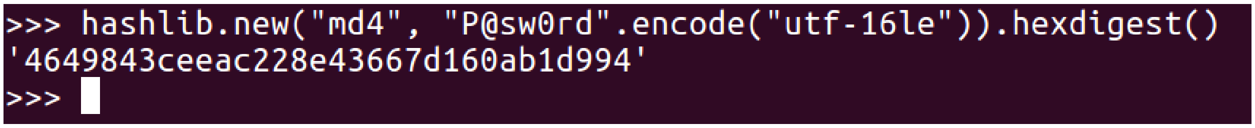
```
  GNU nano 2.5.3                 File: login.defs

#
# /etc/login.defs - Configuration control definitions for the login package.
#
# Three items must be defined:  MAIL_DIR, ENV_SUPATH, and ENV_PATH.
# If unspecified, some arbitrary (and possibly incorrect) value will
# be assumed.  All other items are optional - if not specified then
# the described action or option will be inhibited.
#
# Comment lines (lines beginning with "#") and blank lines are ignored.
#
# Modified for Linux.  --marekm

# REQUIRED for useradd/userdel/usermod
#   Directory where mailboxes reside, _or_ name of file, relative to the
#   home directory.  If you _do_ define MAIL_DIR and MAIL_FILE,
#   MAIL_DIR takes precedence.
#
#   Essentially:
#      - MAIL_DIR defines the location of users mail spool files
#        (for mbox use) by appending the username to MAIL_DIR as defined
#        below.
#      - MAIL_FILE defines the location of the users mail spool files as the
#        fully-qualified filename obtained by prepending the user home
#        directory before $MAIL_FILE
#
# NOTE: This is no more used for setting up users MAIL environment variable
#       which is, starting from shadow 4.0.12-1 in Debian, entirely the
#       job of the pam_mail PAM modules
#       See default PAM configuration files provided for
#       login, su, etc.
#
# This is a temporary situation: setting these variables will soon
# move to /etc/default/useradd and the variables will then be
# no more supported
                     [ Read 341 lines (Warning: No write permission) ]
^G Get Help     ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify      ^C Cur Pos
^X Exit         ^R Read File    ^\ Replace      ^U Uncut Text   ^T To Spell     ^_ Go To Line
```



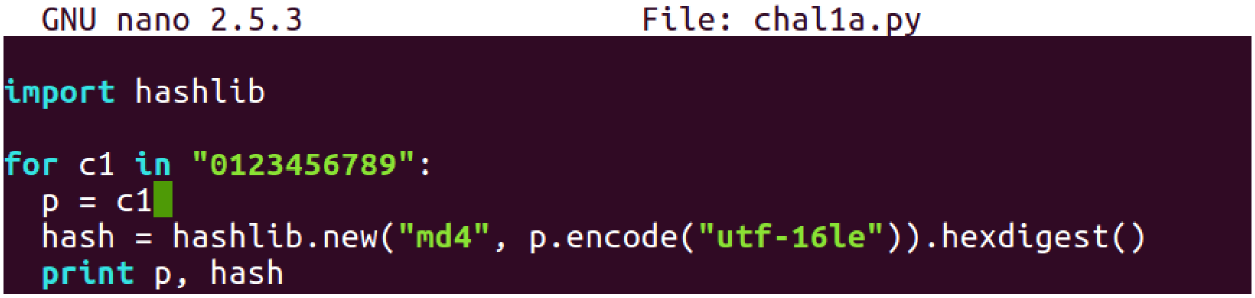
```
>>> import hashlib
>>> from passlib.hash import sha512_crypt
>>> sha512_crypt.using(salt="qIo0foX5", rounds=5000).hash("P@sw0rd")
'$6$qIo0foX5$r7kx5FnTYMWANoz8zacMRdHjxiFs9aaKsj2nObFOzS.q86AfOVCxJKH/kqdcDrTFH9XvSXQ5ZDEmc
zX2NCik/'
>>> 
```



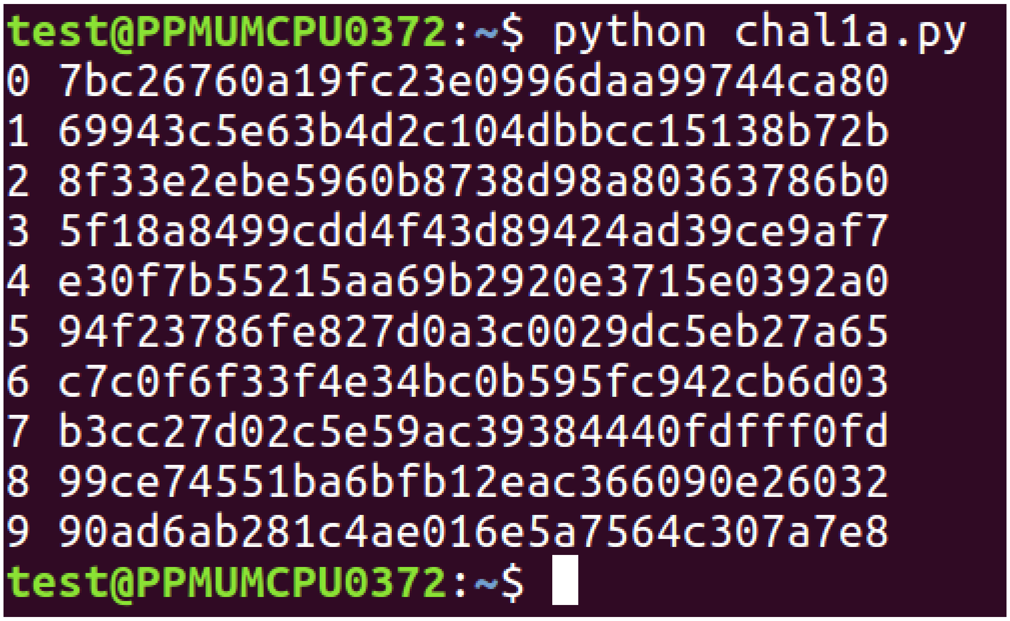
```
>>> sha512_crypt.using(salt="qIo0foX5", rounds=5000).hash("P@sw0ra")
'$6$qIo0foX5$DX/.gSq2C8NdzBK9Yn2lWTLMw3nQw1pRebYEBaasuyEioRFImk9tF1cHIlC9Ecmdnk6QZFuYnkUXWk
.6Jssbo0'
>>> sha512_crypt.using(salt="qIo0foX5", rounds=5000).hash("P@sw0rb")
'$6$qIo0foX5$iYl6BYzTCgWTdRl5ErYB/EoL4ImlwwPBMqDrLDSVomCvLG83Id8oAAkrFixSzxoKtmu79qBLa6JDK6
irIMwLj0'
>>> sha512_crypt.using(salt="qIo0foX5", rounds=5000).hash("P@sw0rc")
'$6$qIo0foX5$FJnTG1I0aaR2Z7MadTzSG90WaL57vC8Jta./DKLw9f6vhHdi9BXThneeocx0e5hYKTJJIrT17uOzkB
WNwGhdQ/'
>>> sha512_crypt.using(salt="qIo0foX5", rounds=5000).hash("P@sw0rd")
'$6$qIo0foX5$r7kx5FnTYMWANoz8zacMRdHjxiFs9aaKsj2nObFOzS.q86AfOVCxJKH/kqdcDrTFH9XvSXQ5ZDEmcE
zX2NCik/'
>>> 
```



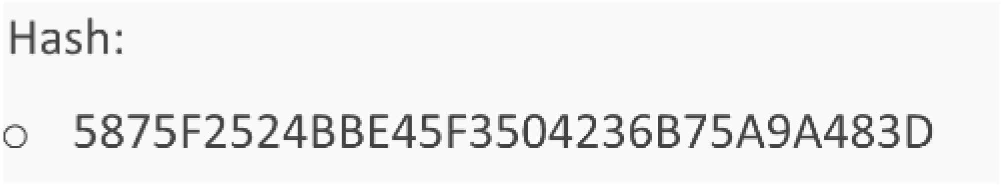
```
>>> hashlib.new("md4", "P@sw0rd".encode("utf-16le")).hexdigest()
'4649843ceeac228e43667d160ab1d994'
>>> 
```



```
  GNU nano 2.5.3                 File: chal1a.py

import hashlib

for c1 in "0123456789":
  p = c1
  hash = hashlib.new("md4", p.encode("utf-16le")).hexdigest()
  print p, hash
```

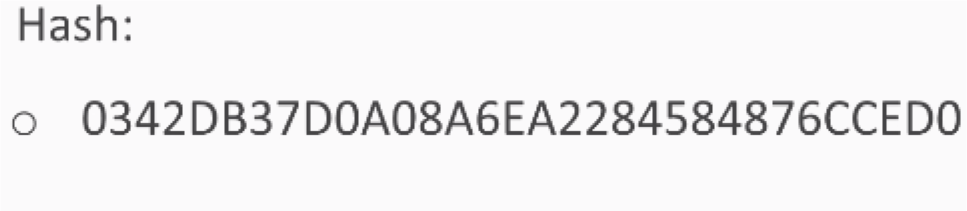


```
test@PPMUMCPU0372:~$ python chal1a.py
0 7bc26760a19fc23e0996daa99744ca80
1 69943c5e63b4d2c104dbbcc15138b72b
2 8f33e2ebe5960b8738d98a80363786b0
3 5f18a8499cdd4f43d89424ad39ce9af7
4 e30f7b55215aa69b2920e3715e0392a0
5 94f23786fe827d0a3c0029dc5eb27a65
6 c7c0f6f33f4e34bc0b595fc942cb6d03
7 b3cc27d02c5e59ac39384440fdfff0fd
8 99ce74551ba6bfb12eac366090e26032
9 90ad6ab281c4ae016e5a7564c307a7e8
test@PPMUMCPU0372:~$ 
```



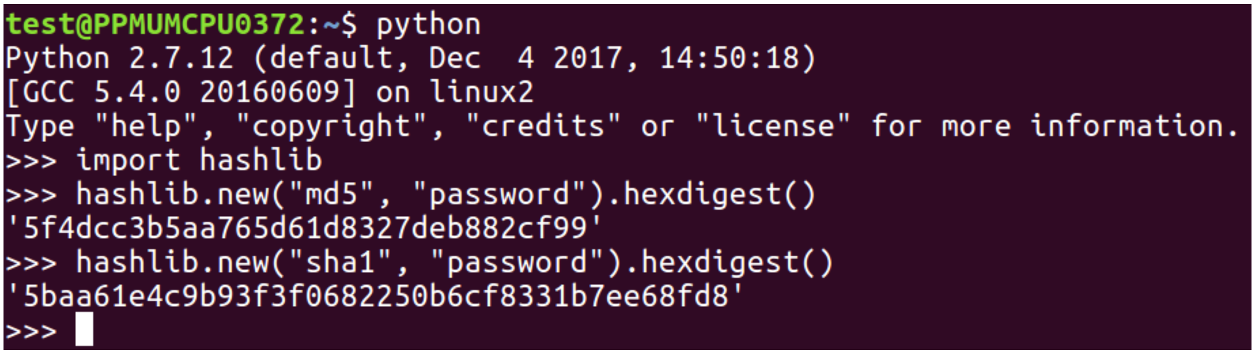
Hash:

○ 5875F2524BBE45F3504236B75A9A483D

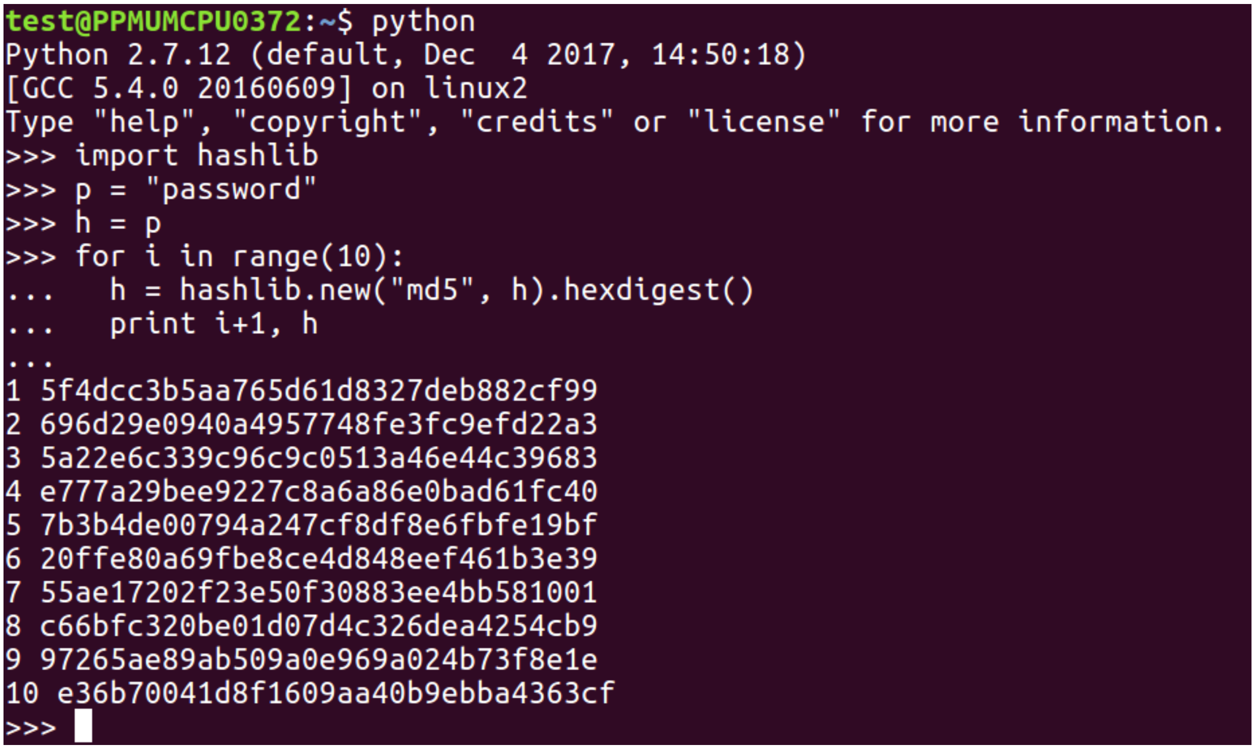


Hash:

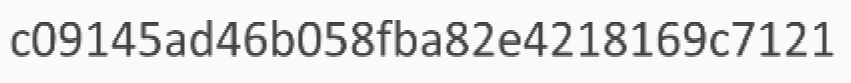
○ 0342DB37D0A08A6EA2284584876CCED0



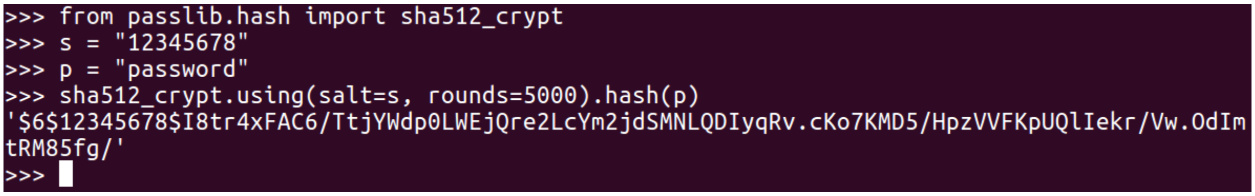
```
test@PPMUMCPU0372:~$ python
Python 2.7.12 (default, Dec  4 2017, 14:50:18)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import hashlib
>>> hashlib.new("md5", "password").hexdigest()
'5f4dcc3b5aa765d61d8327deb882cf99'
>>> hashlib.new("sha1", "password").hexdigest()
'5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8'
>>> 
```



```
test@PPMUMCPU0372:~$ python
Python 2.7.12 (default, Dec  4 2017, 14:50:18)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import hashlib
>>> p = "password"
>>> h = p
>>> for i in range(10):
...   h = hashlib.new("md5", h).hexdigest()
...   print i+1, h
...
1 5f4dcc3b5aa765d61d8327deb882cf99
2 696d29e0940a4957748fe3fc9efd22a3
3 5a22e6c339c96c9c0513a46e44c39683
4 e777a29bee9227c8a6a86e0bad61fc40
5 7b3b4de00794a247cf8df8e6fbfe19bf
6 20ffe80a69fbe8ce4d848eef461b3e39
7 55ae17202f23e50f30883ee4bb581001
8 c66bfc320be01d07d4c326dea4254cb9
9 97265ae89ab509a0e969a024b73f8e1e
10 e36b70041d8f1609aa40b9ebba4363cf
>>> 
```

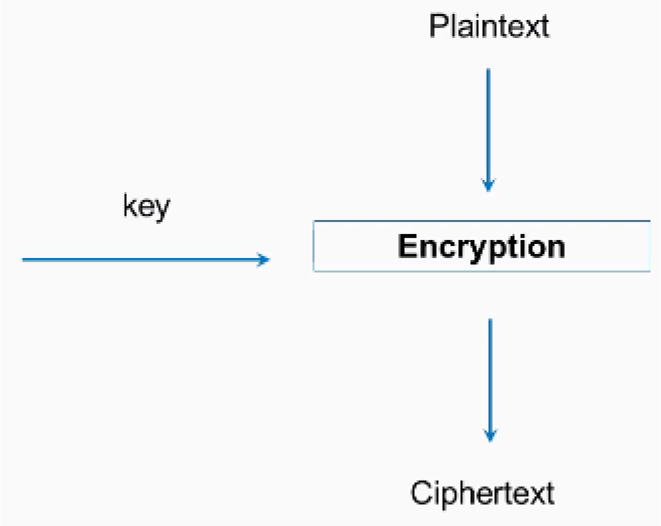


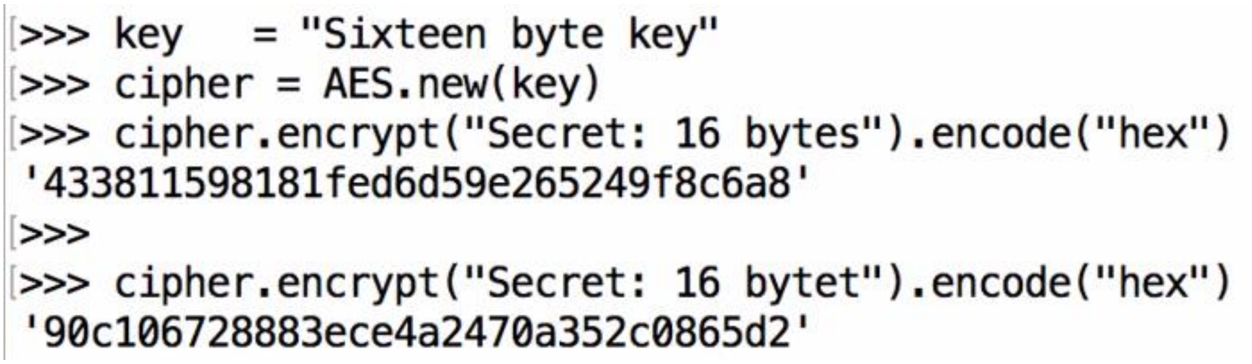
```
c09145ad46b058fba82e4218169c7121
```



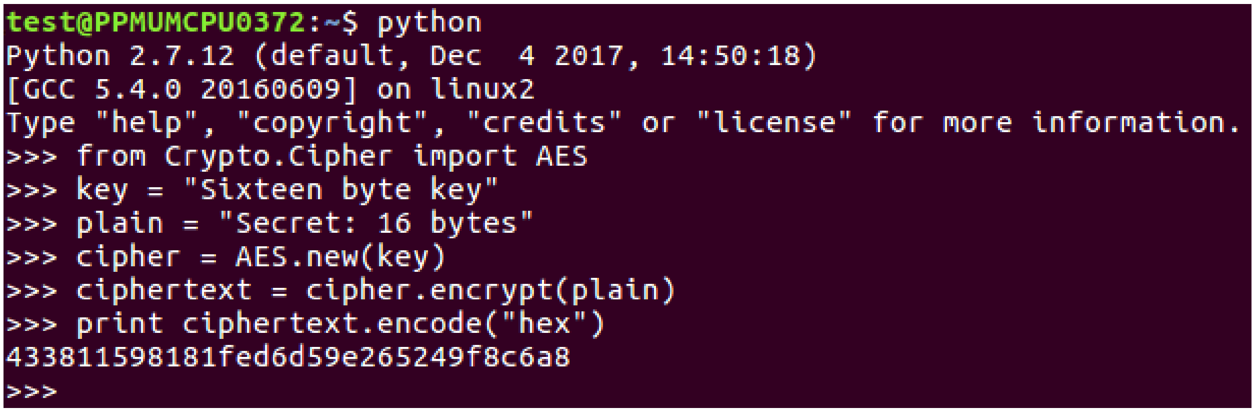
```
>>> from passlib.hash import sha512_crypt
>>> s = "12345678"
>>> p = "password"
>>> sha512_crypt.using(salt=s, rounds=5000).hash(p)
'$6$12345678$I8tr4xFAC6/TtjYWdp0LWEjQre2LcYm2jdSMNLQDIyqRv.cKo7KMD5/HpzVVFKpUQlIekr/Vw.OdIm
tRM85fg/'
>>> 
```

Chapter 3: Strong Encryption

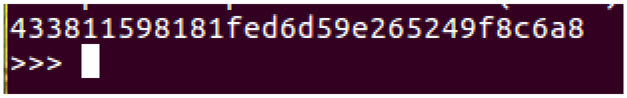




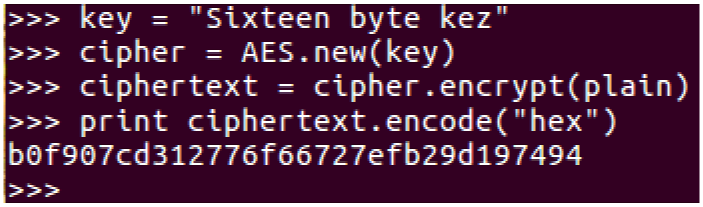
```
[>>> key    = "Sixteen byte key"
[>>> cipher = AES.new(key)
[>>> cipher.encrypt("Secret: 16 bytes").encode("hex")
'433811598181fed6d59e265249f8c6a8'
[>>>
[>>> cipher.encrypt("Secret: 16 bytet").encode("hex")
'90c106728883ece4a2470a352c0865d2'
```



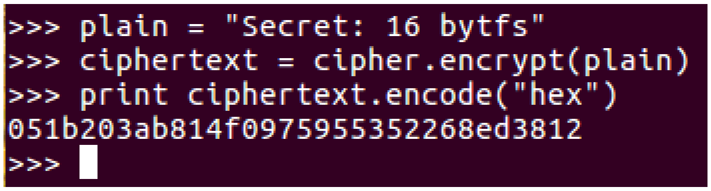
```
test@PPMUMCPU0372:~$ python
Python 2.7.12 (default, Dec  4 2017, 14:50:18)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from Crypto.Cipher import AES
>>> key = "Sixteen byte key"
>>> plain = "Secret: 16 bytes"
>>> cipher = AES.new(key)
>>> ciphertext = cipher.encrypt(plain)
>>> print ciphertext.encode("hex")
433811598181fed6d59e265249f8c6a8
>>>
```



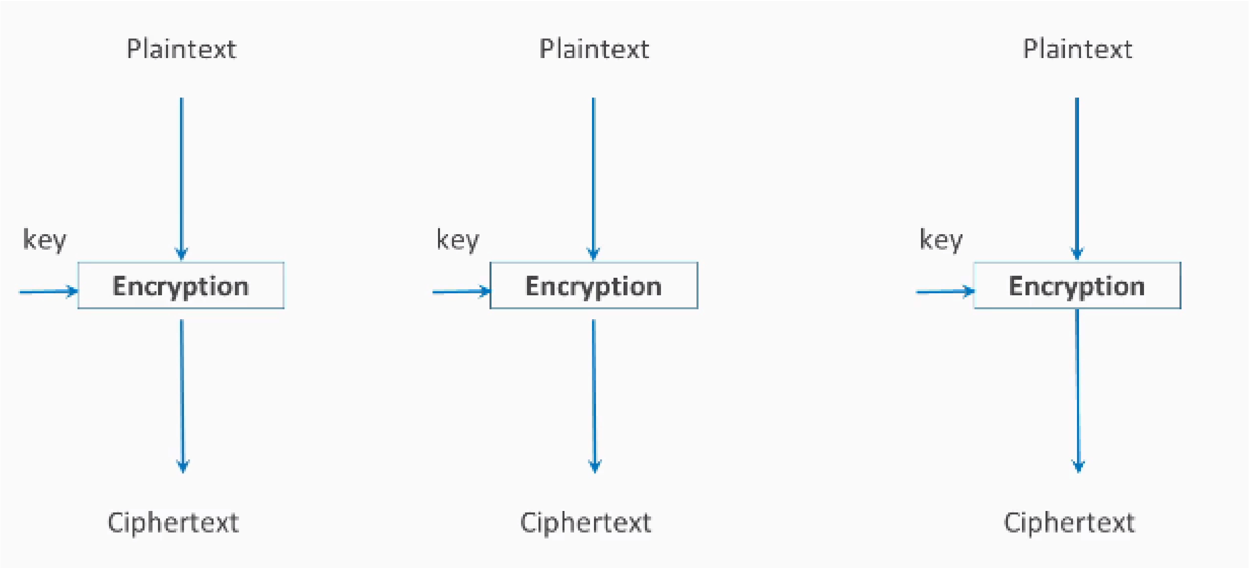
```
433811598181fed6d59e265249f8c6a8
>>> 
```

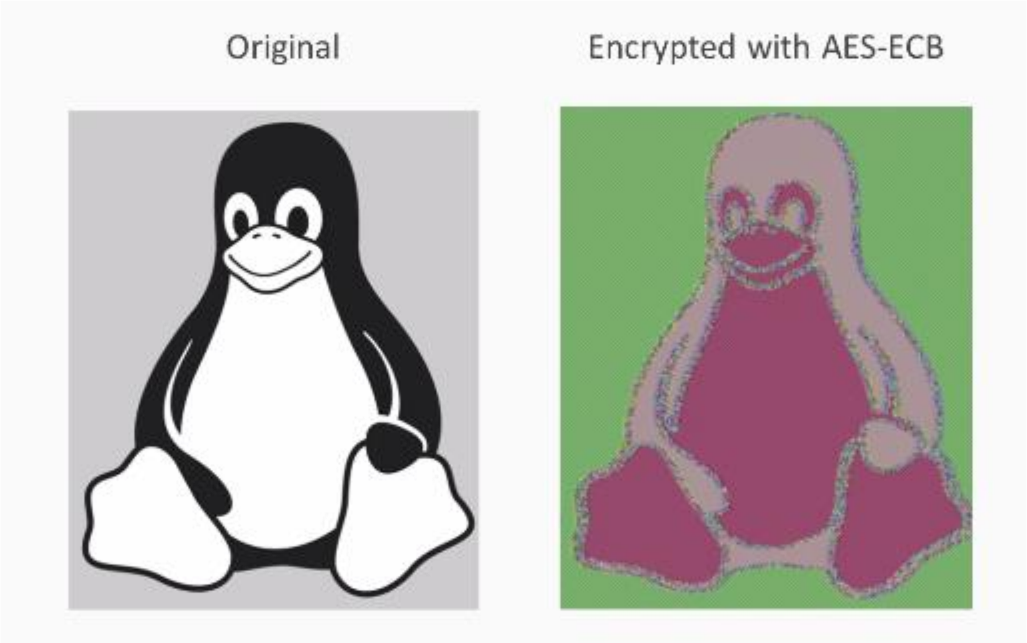


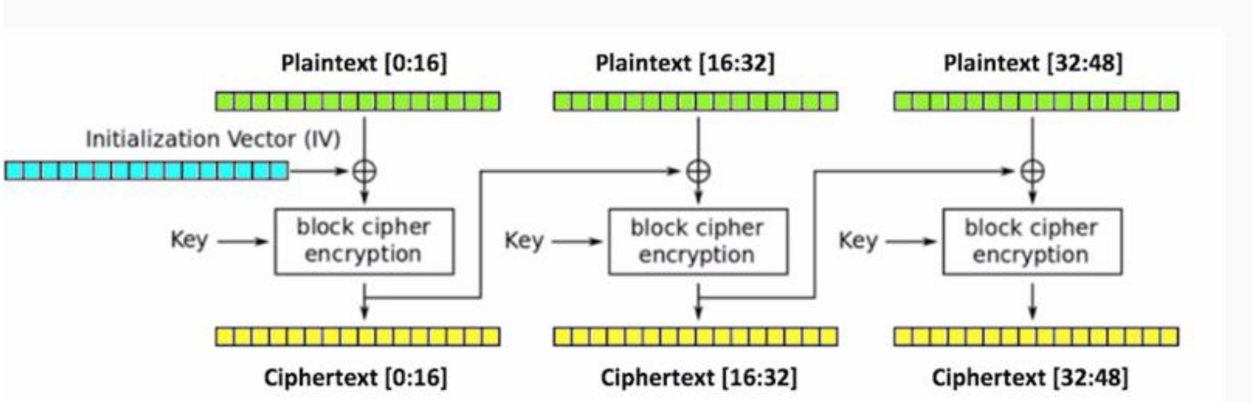
```
>>> key = "Sixteen byte kez"
>>> cipher = AES.new(key)
>>> ciphertext = cipher.encrypt(plain)
>>> print ciphertext.encode("hex")
b0f907cd312776f66727efb29d197494
>>>
```



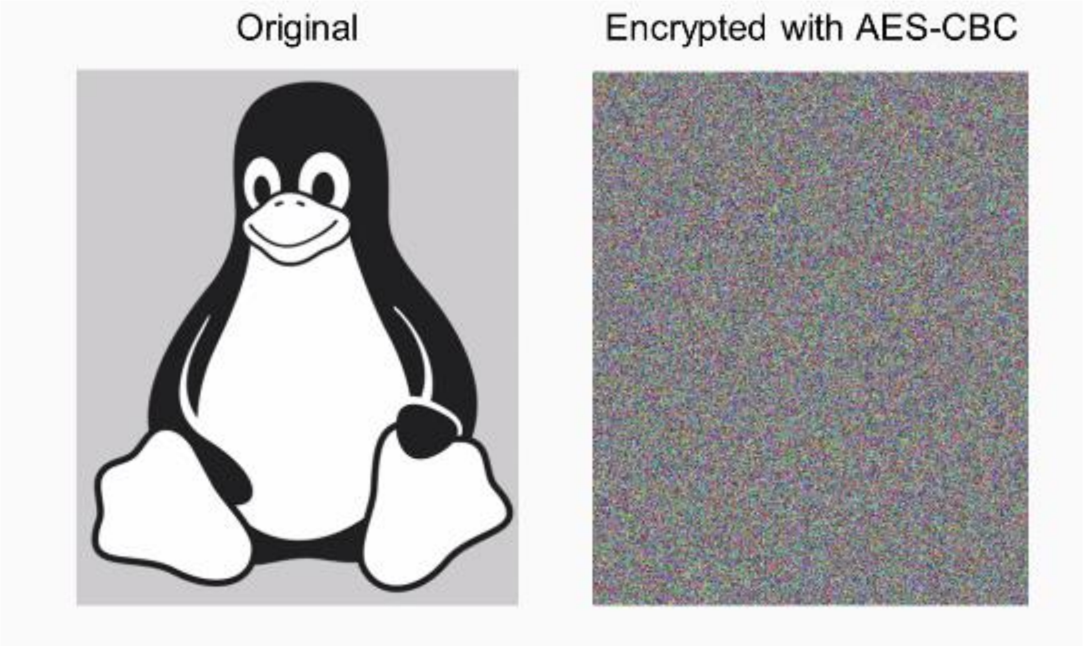
```
>>> plain = "Secret: 16 bytfs"
>>> ciphertext = cipher.encrypt(plain)
>>> print ciphertext.encode("hex")
051b203ab814f0975955352268ed3812
>>> 
```

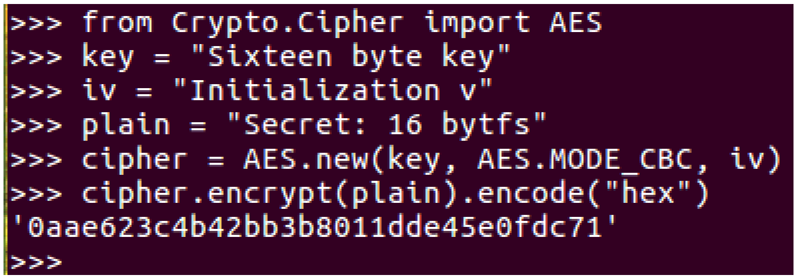




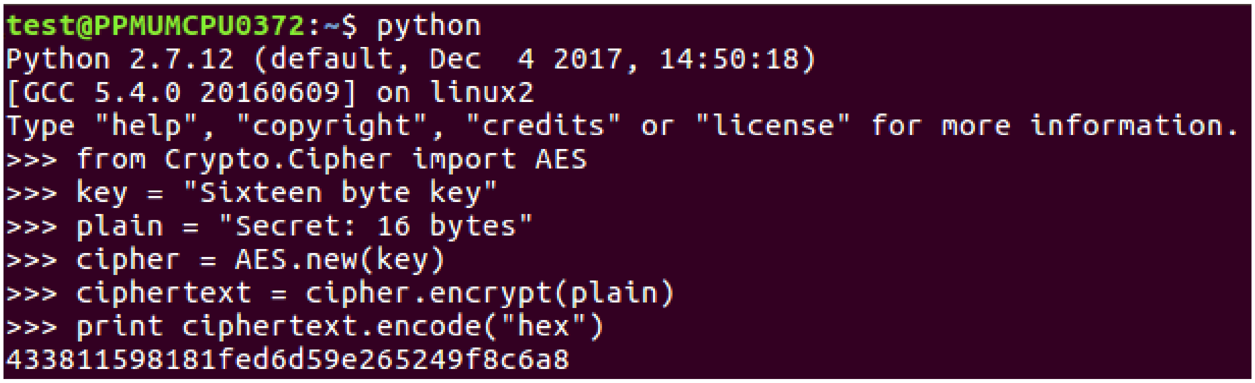


Cipher Block Chaining (CBC) mode encryption

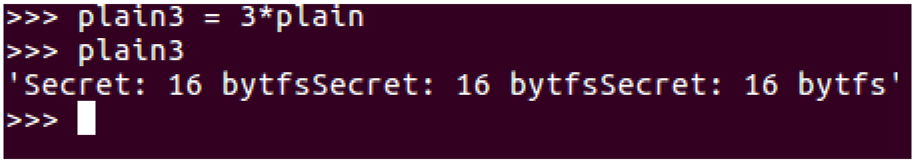




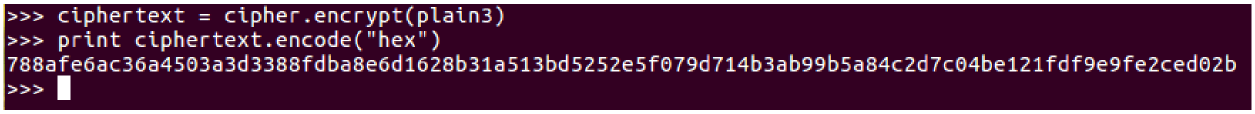
```
>>> from Crypto.Cipher import AES
>>> key = "Sixteen byte key"
>>> iv = "Initialization v"
>>> plain = "Secret: 16 bytfs"
>>> cipher = AES.new(key, AES.MODE_CBC, iv)
>>> cipher.encrypt(plain).encode("hex")
'0aae623c4b42bb3b8011dde45e0fdc71'
>>>
```



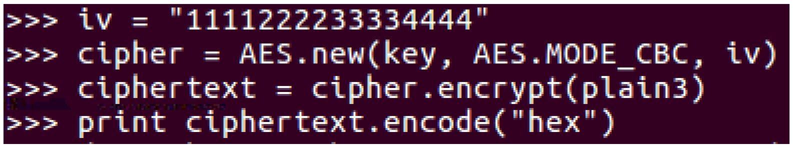
```
test@PPMUMCPU0372:~$ python
Python 2.7.12 (default, Dec  4 2017, 14:50:18)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from Crypto.Cipher import AES
>>> key = "Sixteen byte key"
>>> plain = "Secret: 16 bytes"
>>> cipher = AES.new(key)
>>> ciphertext = cipher.encrypt(plain)
>>> print ciphertext.encode("hex")
433811598181fed6d59e265249f8c6a8
```



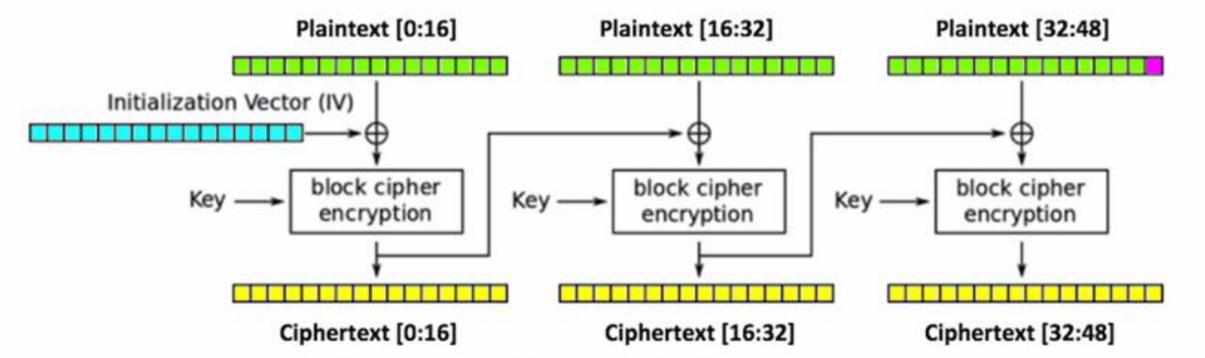
```
>>> plain3 = 3*plain
>>> plain3
'Secret: 16 bytfsSecret: 16 bytfsSecret: 16 bytfs'
>>> 
```



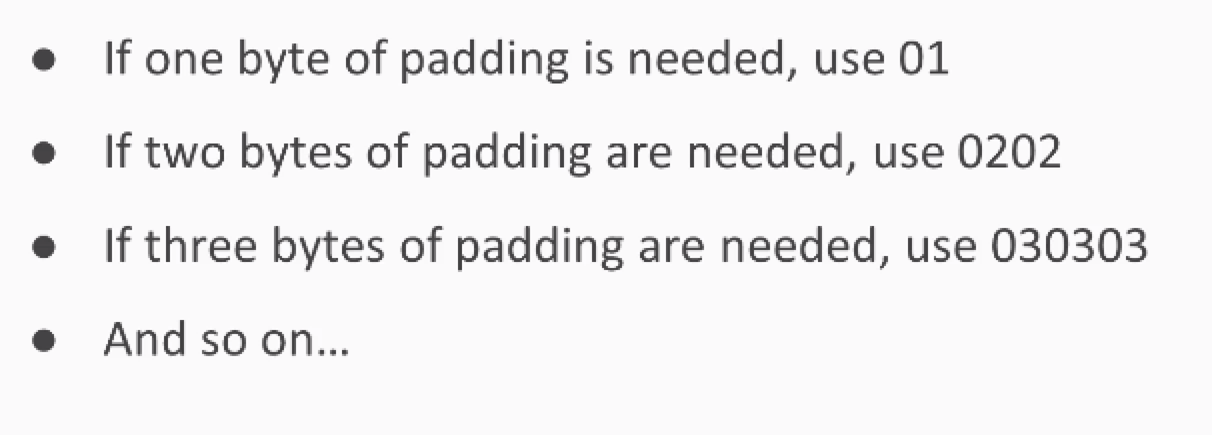
```
>>> ciphertext = cipher.encrypt(plain3)
>>> print ciphertext.encode("hex")
788afe6ac36a4503a3d3388fdba8e6d1628b31a513bd5252e5f079d714b3ab99b5a84c2d7c04be121fdf9e9fe2ced02b
>>> 
```



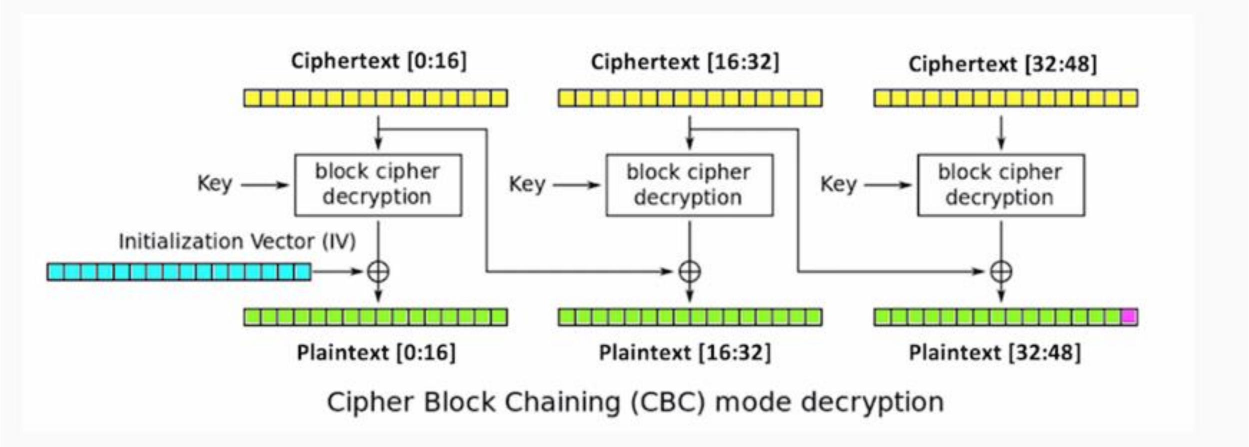
```
>>> iv = "1111222233334444"
>>> cipher = AES.new(key, AES.MODE_CBC, iv)
>>> ciphertext = cipher.encrypt(plain3)
>>> print ciphertext.encode("hex")
```



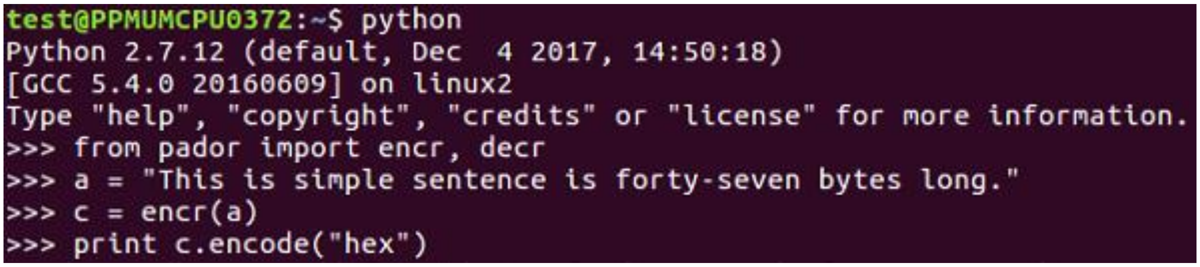
Cipher Block Chaining (CBC) mode encryption



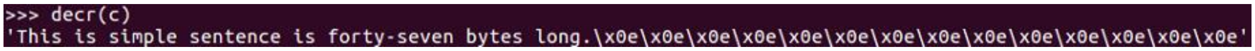
- If one byte of padding is needed, use 01
- If two bytes of padding are needed, use 0202
- If three bytes of padding are needed, use 030303
- And so on...



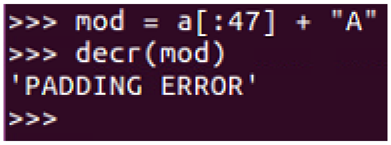
Cipher Block Chaining (CBC) mode decryption



```
test@PPMUMCPU0372:~$ python
Python 2.7.12 (default, Dec  4 2017, 14:50:18)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from pador import encr, decr
>>> a = "This is simple sentence is forty-seven bytes long."
>>> c = encr(a)
>>> print c.encode("hex")
```

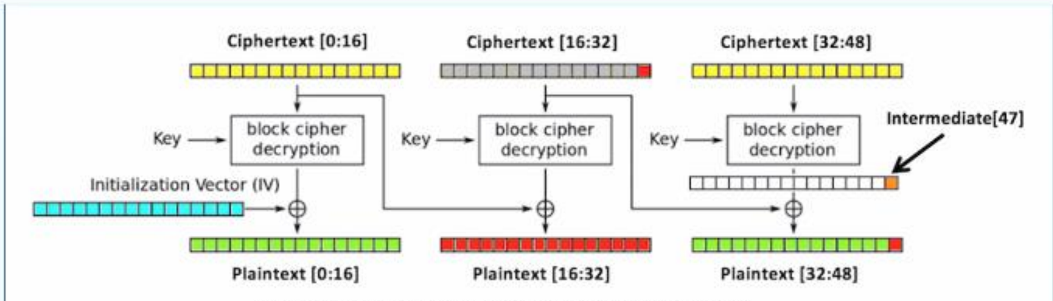


```
>>> decr(c)
'This is simple sentence is forty-seven bytes long.\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e'
```

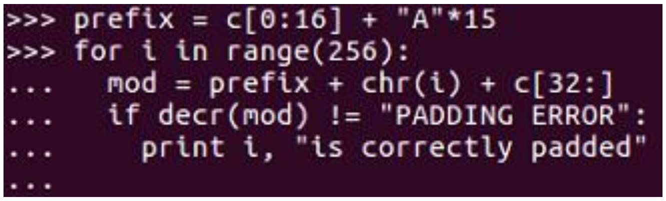


```
>>> mod = a[:47] + "A"
>>> decr(mod)
'PADDING ERROR'
>>>
```

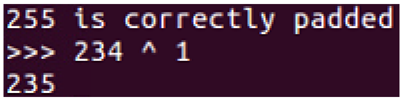
- Change ciphertext[16:31] to any bytes
- Change ciphertext[31] until padding is valid
- Intermediate[47] must be 1



Cipher Block Chaining (CBC) mode decryption

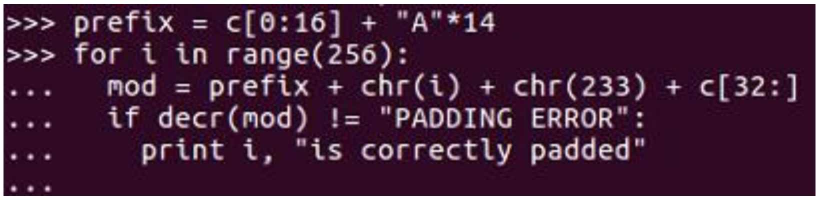


```
>>> prefix = c[0:16] + "A"*15
>>> for i in range(256):
...   mod = prefix + chr(i) + c[32:]
...   if decr(mod) != "PADDING ERROR":
...     print i, "is correctly padded"
...
```

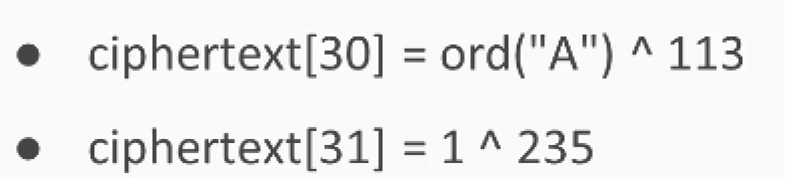


```
255 is correctly padded
>>> 234 ^ 1
235
```

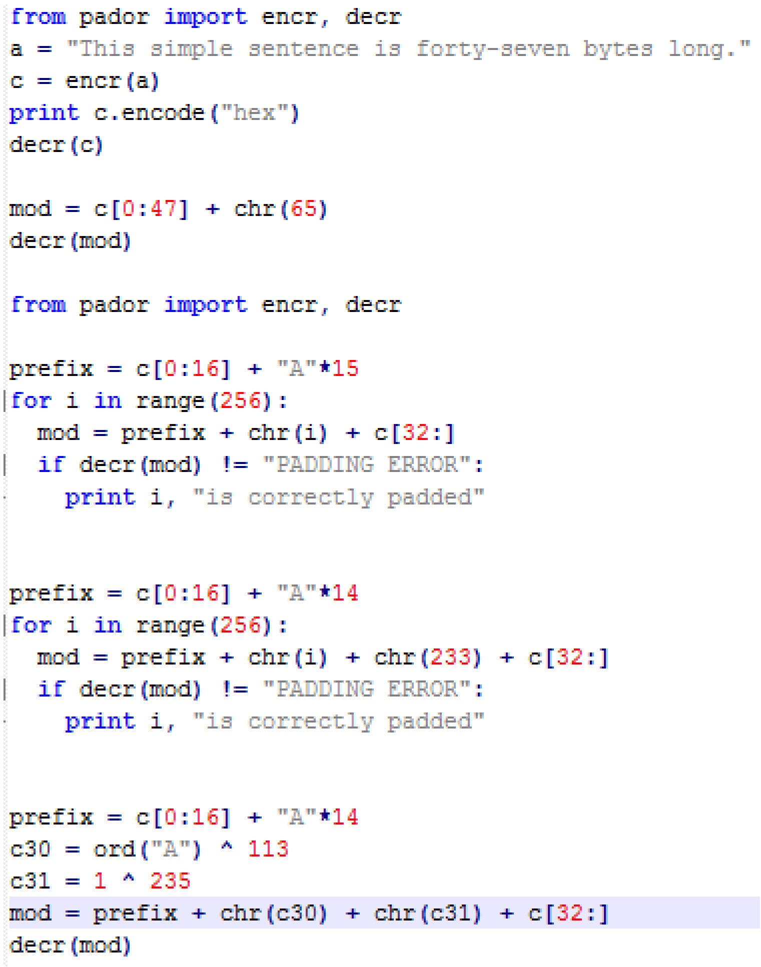
- Two bytes of padding:
  - ciphertext[46] =ciphertext[47] = 2
  - Set ciphertext[31] = 235 ^ 2 = 233



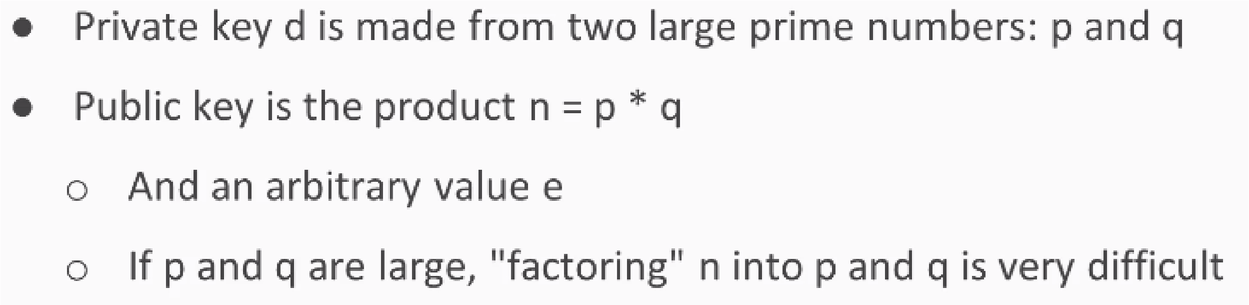
```
>>> prefix = c[0:16] + "A"*14
>>> for i in range(256):
...     mod = prefix + chr(i) + chr(233) + c[32:]
...     if decr(mod) != "PADDING ERROR":
...       print i, "is correctly padded"
...
```



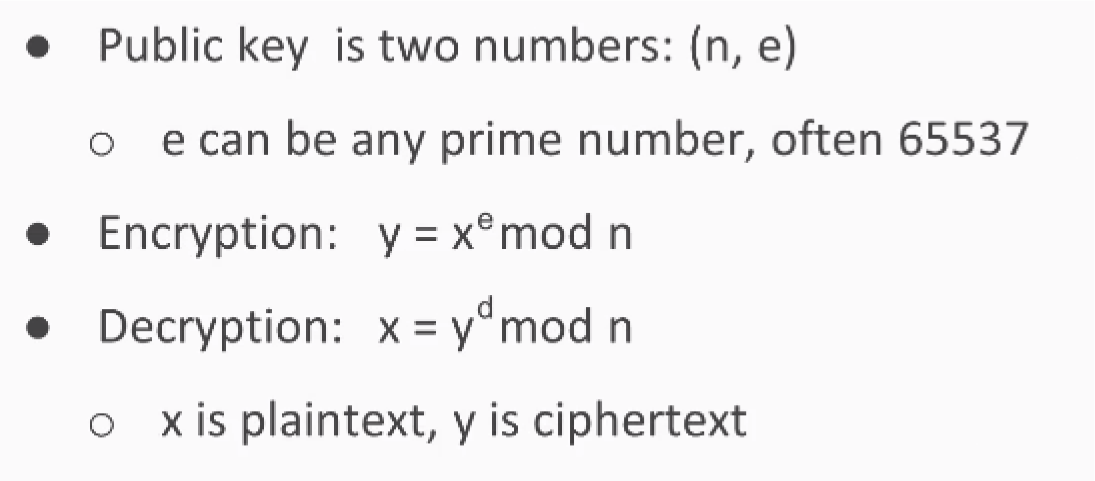
- ciphertext[30] = ord("A") ^ 113
- ciphertext[31] = 1 ^ 235



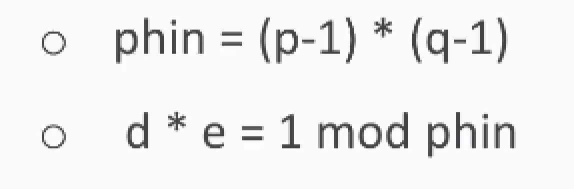
```
from pador import encr, decr
a = "This simple sentence is forty-seven bytes long."
c = encr(a)
print c.encode("hex")
decr(c)

mod = c[0:47] + chr(65)
decr(mod)

from pador import encr, decr

prefix = c[0:16] + "A"*15
for i in range(256):
  mod = prefix + chr(i) + c[32:]
  if decr(mod) != "PADDING ERROR":
    print i, "is correctly padded"


prefix = c[0:16] + "A"*14
for i in range(256):
  mod = prefix + chr(i) + chr(233) + c[32:]
  if decr(mod) != "PADDING ERROR":
    print i, "is correctly padded"


prefix = c[0:16] + "A"*14
c30 = ord("A") ^ 113
c31 = 1 ^ 235
mod = prefix + chr(c30) + chr(c31) + c[32:]
decr(mod)
```



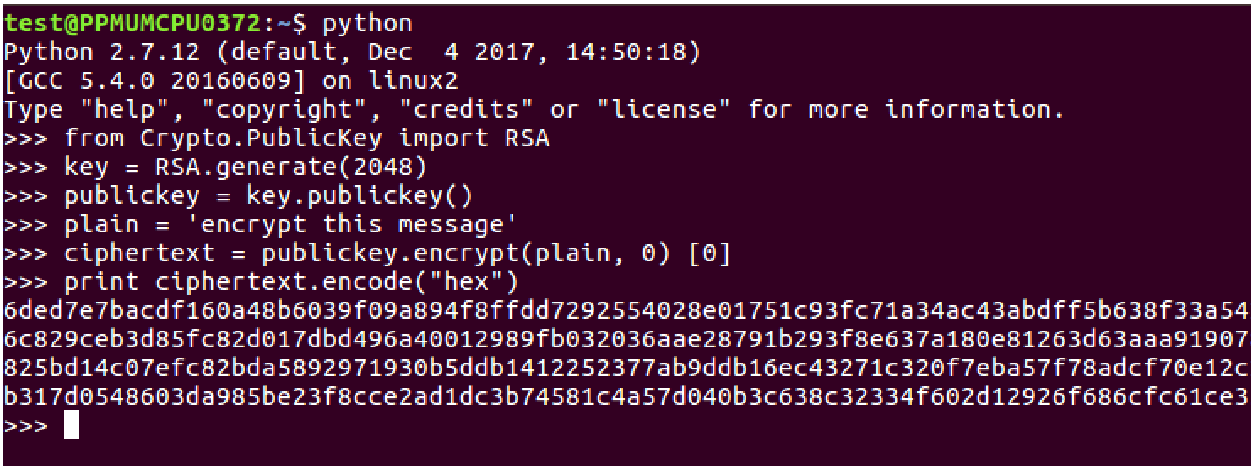
- Private key d is made from two large prime numbers: p and q
- Public key is the product n = p * q
  - And an arbitrary value e
  - If p and q are large, "factoring" n into p and q is very difficult



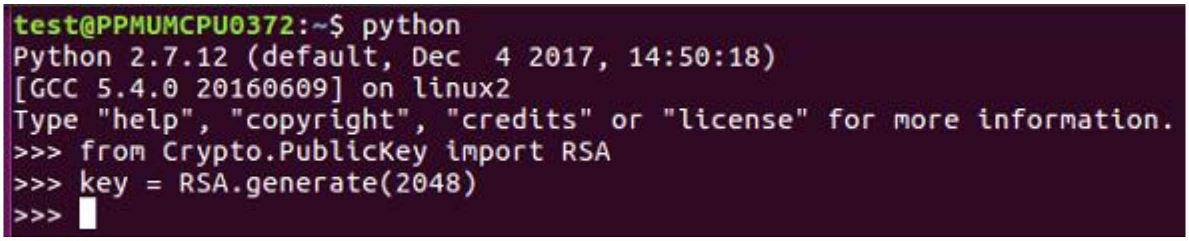
- Public key is two numbers: (n, e)
  - e can be any prime number, often 65537
- Encryption: $y = x^e \bmod n$
- Decryption: $x = y^d \bmod n$
  - x is plaintext, y is ciphertext



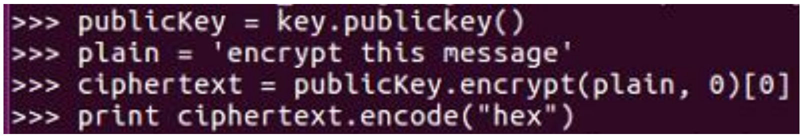
- phin = (p-1) * (q-1)
- d * e = 1 mod phin



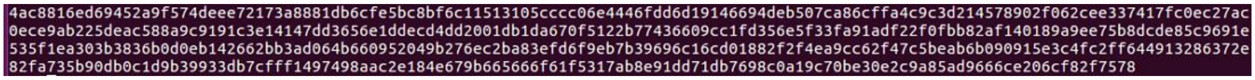
```
test@PPMUMCPU0372:~$ python
Python 2.7.12 (default, Dec  4 2017, 14:50:18)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from Crypto.PublicKey import RSA
>>> key = RSA.generate(2048)
>>> publickey = key.publickey()
>>> plain = 'encrypt this message'
>>> ciphertext = publickey.encrypt(plain, 0) [0]
>>> print ciphertext.encode("hex")
6ded7e7bacdf160a48b6039f09a894f8ffdd7292554028e01751c93fc71a34ac43abdff5b638f33a54
6c829ceb3d85fc82d017dbd496a40012989fb032036aae28791b293f8e637a180e81263d63aaa91907
825bd14c07efc82bda5892971930b5ddb1412252377ab9ddb16ec43271c320f7eba57f78adcf70e12c
b317d0548603da985be23f8cce2ad1dc3b74581c4a57d040b3c638c32334f602d12926f686cfc61ce3
>>> 
```



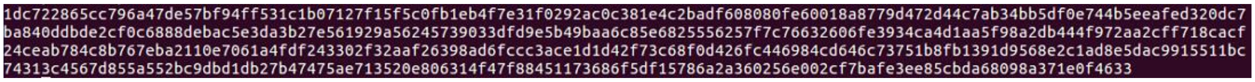
```
test@PPMUMCPU0372:~$ python
Python 2.7.12 (default, Dec  4 2017, 14:50:18)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from Crypto.PublicKey import RSA
>>> key = RSA.generate(2048)
>>> 
```



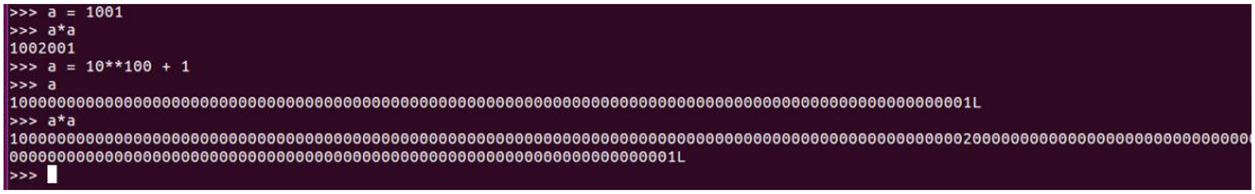
```
>>> publicKey = key.publickey()
>>> plain = 'encrypt this message'
>>> ciphertext = publicKey.encrypt(plain, 0)[0]
>>> print ciphertext.encode("hex")
```



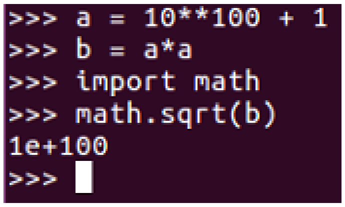
```
4ac8816ed69452a9f574deee72173a8881db6cfe5bc8bf6c11513105cccc06e4446fdd6d19146694deb507ca86cffa4c9c3d214578902f062cee337417fc0ec27ac
0ece9ab225deac588a9c9191c3e14147dd3656e1ddecd4dd2001db1da670f5122b77436609cc1fd356e5f33fa91adf22f0fbb82af140189a9ee75b8dcde85c9691e
535f1ea303b3836b0d0eb142662bb3ad064b660952049b276ec2ba83efd6f9eb7b39696c16cd01882f2f4ea9cc62f47c5beab6b090915e3c4fc2ff644913286372e
82fa735b90db0c1d9b39933db7cfff1497498aac2e184e679b665666f61f5317ab8e91dd71db7698c0a19c70be30e2c9a85ad9666ce206cf82f7578
```



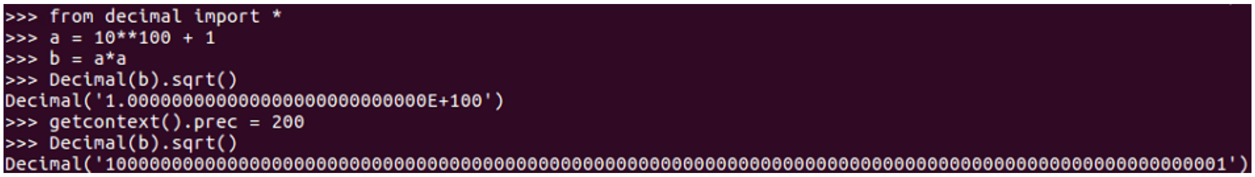
```
1dc722865cc796a47de57bf94ff531c1b07127f15f5c0fb1eb4f7e31f0292ac0c381e4c2badf608080fe60018a8779d472d44c7ab34bb5df0e744b5eeafed320dc7
ba840ddbde2cf0c6888debac5e3da3b27e561929a56245739033dfd9e5b49baa6c85e6825556257f7c76632606fe3934ca4d1aa5f98a2db444f972aa2cff718cacf
24ceab784c8b767eba2110e7061a4fdf243302f32aaf26398ad6fccc3ace1d1d42f73c68f0d426fc446984cd646c73751b8fb1391d9568e2c1ad8e5dac9915511bc
74313c4567d855a552bc9dbd1db27b47475ae713520e806314f47f88451173686f5df15786a2a360256e002cf7bafe3ee85cbda68098a371e0f4633
```



```
>>> a = 1001
>>> a*a
1002001
>>> a = 10**100 + 1
>>> a
10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001L
>>> a*a
10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000002000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000001L
>>> 
```



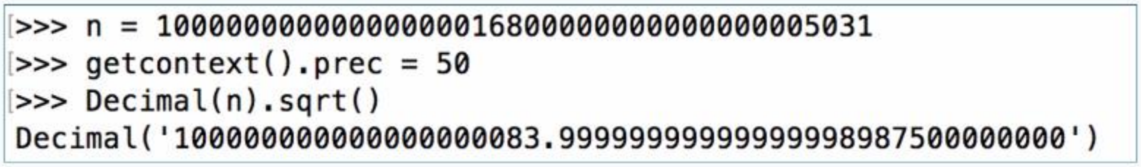
```
>>> a = 10**100 + 1
>>> b = a*a
>>> import math
>>> math.sqrt(b)
1e+100
>>> 
```



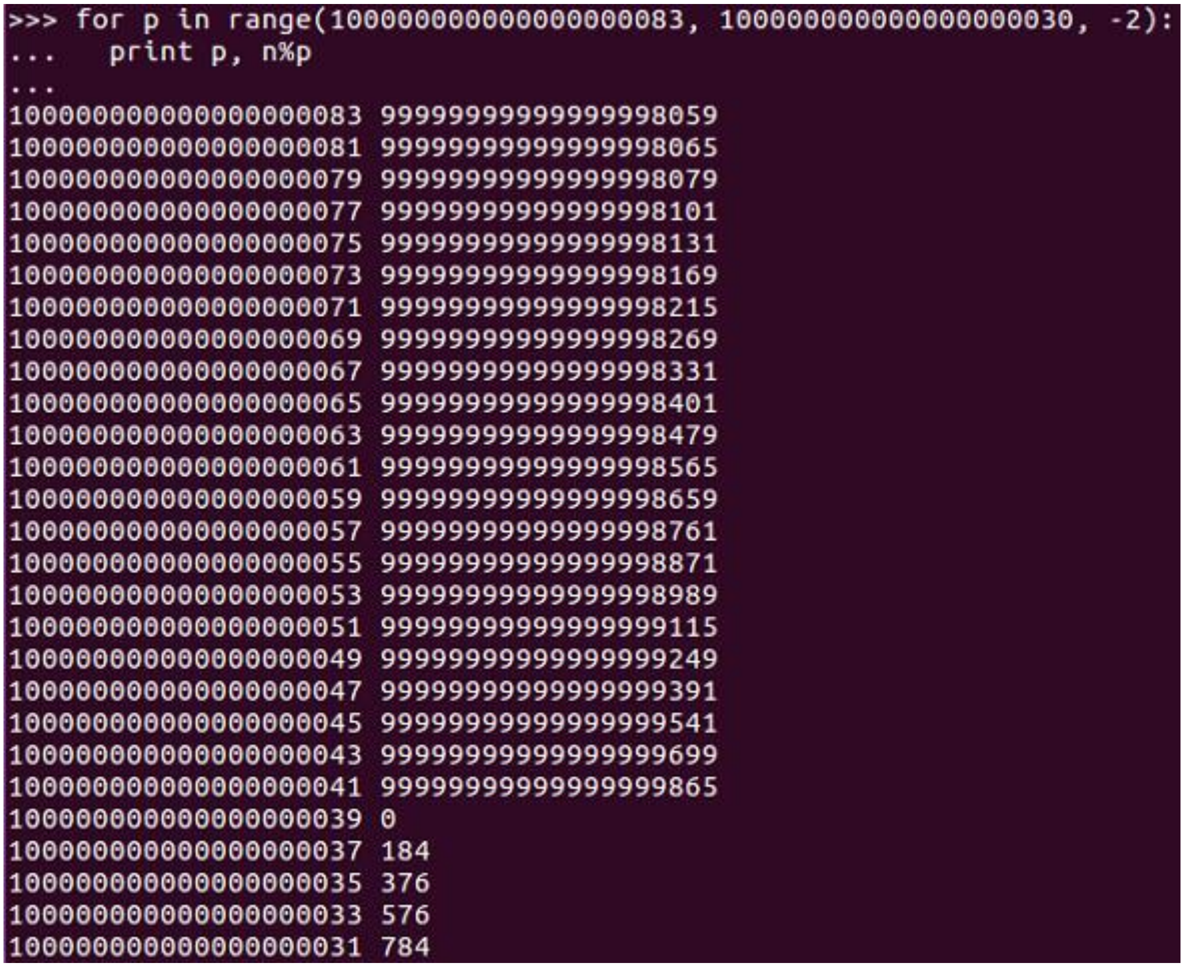
```
>>> from decimal import *
>>> a = 10**100 + 1
>>> b = a*a
>>> Decimal(b).sqrt()
Decimal('1.000000000000000000000000000E+100')
>>> getcontext().prec = 200
>>> Decimal(b).sqrt()
Decimal('10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001')
```

Example:

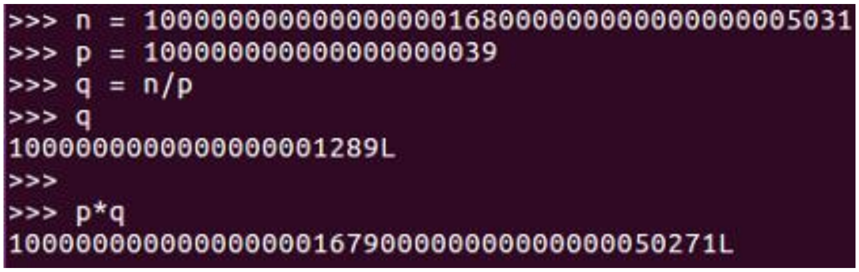
- 10000000000000000016800000000000000005031



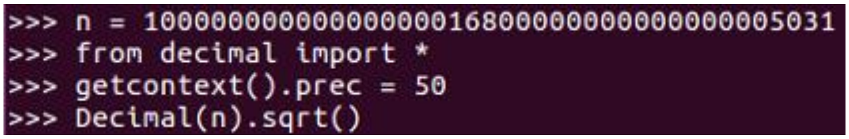
```
>>> n = 10000000000000000016800000000000000005031
>>> getcontext().prec = 50
>>> Decimal(n).sqrt()
Decimal('100000000000000000083.99999999999999998987500000000')
```



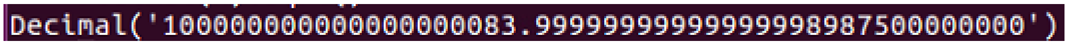
```
>>> for p in range(100000000000000000083, 100000000000000000030, -2):
...     print p, n%p
...
100000000000000000083 99999999999999998059
100000000000000000081 99999999999999998065
100000000000000000079 99999999999999998079
100000000000000000077 99999999999999998101
100000000000000000075 99999999999999998131
100000000000000000073 99999999999999998169
100000000000000000071 99999999999999998215
100000000000000000069 99999999999999998269
100000000000000000067 99999999999999998331
100000000000000000065 99999999999999998401
100000000000000000063 99999999999999998479
100000000000000000061 99999999999999998565
100000000000000000059 99999999999999998659
100000000000000000057 99999999999999998761
100000000000000000055 99999999999999998871
100000000000000000053 99999999999999998989
100000000000000000051 99999999999999999115
100000000000000000049 99999999999999999249
100000000000000000047 99999999999999999391
100000000000000000045 99999999999999999541
100000000000000000043 99999999999999999699
100000000000000000041 99999999999999999865
100000000000000000039 0
100000000000000000037 184
100000000000000000035 376
100000000000000000033 576
100000000000000000031 784
```



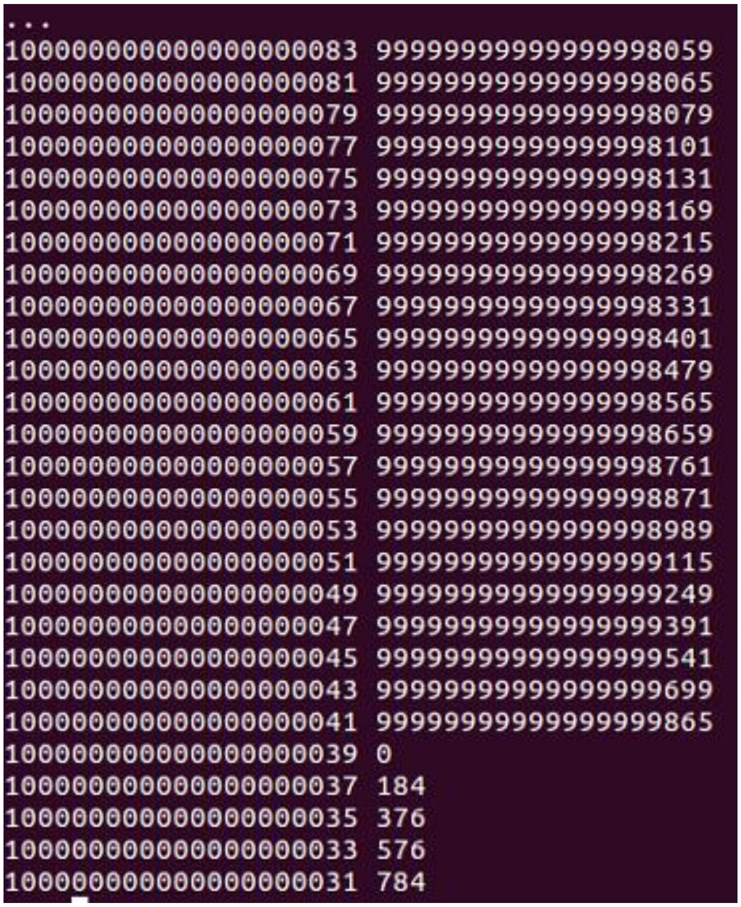
```
>>> n = 100000000000000000168000000000000000005031
>>> p = 100000000000000000039
>>> q = n/p
>>> q
1000000000000000001289L
>>>
>>> p*q
1000000000000000001679000000000000000050271L
```



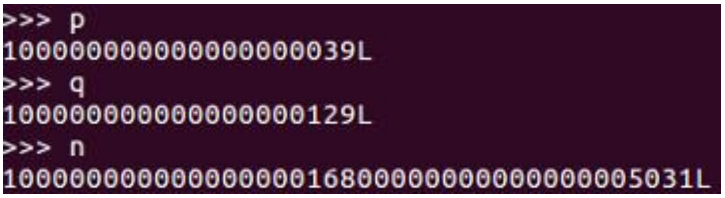
```
>>> n = 10000000000000000016800000000000000005031
>>> from decimal import *
>>> getcontext().prec = 50
>>> Decimal(n).sqrt()
```



```
Decimal('100000000000000000083.99999999999999998987500000000')
```



```
...
100000000000000000083 99999999999999998059
100000000000000000081 99999999999999998065
100000000000000000079 99999999999999998079
100000000000000000077 99999999999999998101
100000000000000000075 99999999999999998131
100000000000000000073 99999999999999998169
100000000000000000071 99999999999999998215
100000000000000000069 99999999999999998269
100000000000000000067 99999999999999998331
100000000000000000065 99999999999999998401
100000000000000000063 99999999999999998479
100000000000000000061 99999999999999998565
100000000000000000059 99999999999999998659
100000000000000000057 99999999999999998761
100000000000000000055 99999999999999998871
100000000000000000053 99999999999999998989
100000000000000000051 99999999999999999115
100000000000000000049 99999999999999999249
100000000000000000047 99999999999999999391
100000000000000000045 99999999999999999541
100000000000000000043 99999999999999999699
100000000000000000041 99999999999999999865
100000000000000000039 0
100000000000000000037 184
100000000000000000035 376
100000000000000000033 576
100000000000000000031 784
```



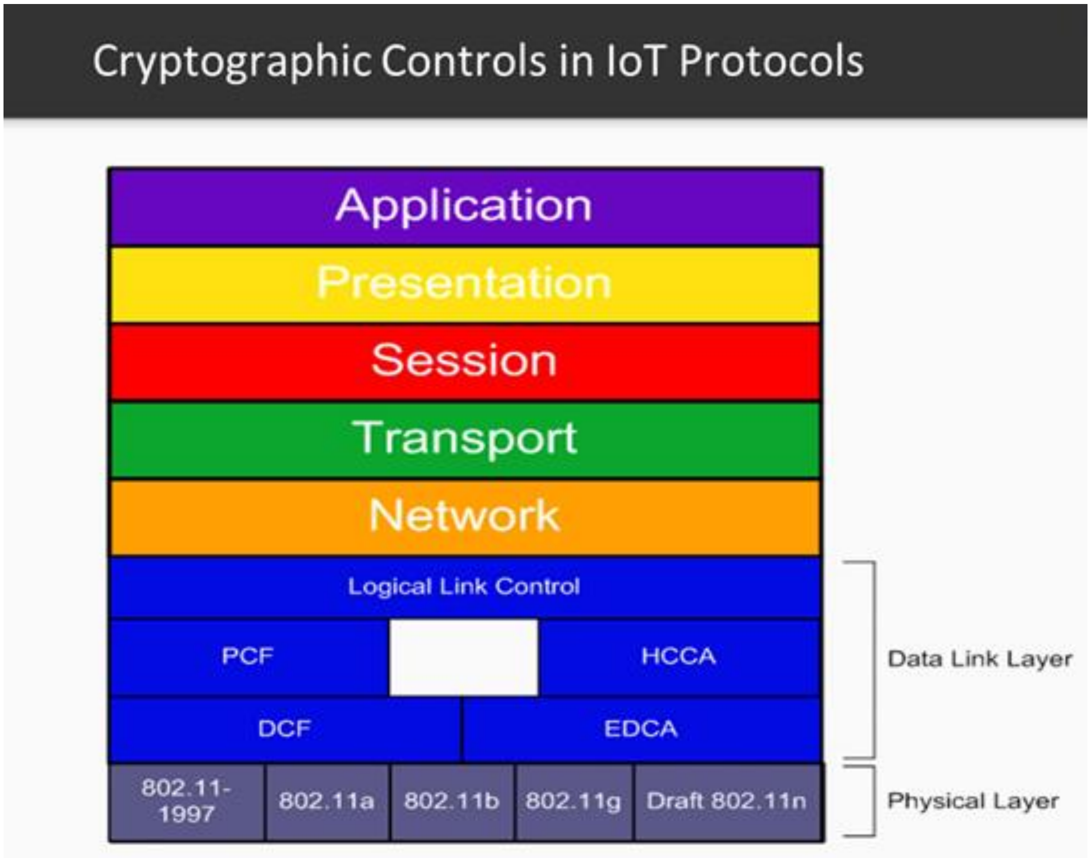
```
>>> p
100000000000000000039L
>>> q
100000000000000000129L
>>> n
10000000000000000016800000000000000005031L
```

First challenge: Factor this number

- 1234592592962967901296297037045679111122222098932964637053765599260929646321154446111289984805767

Second challenge: Factor this number

- 245731949077587003410793632769772440172121093648772379511569661065308222834597845272487909241946260280128792103441259245182932059730438317062685471060402660920755731093250407425954390905112202199219



Cryptographic Controls in IoT Protocols

Application

Presentation

Session

Transport

Network

Logical Link Control

PCF

HCCA

DCF

EDCA

802.11-1997

802.11a

802.11b

802.11g

Draft 802.11n

Data Link Layer

Physical Layer