ASIS CTF Finals - Bit Game

Steliyan Syarov

GENERAL INFO:

- Released in the last 1 / 3 of the CTF
- Reverse crypto challenge

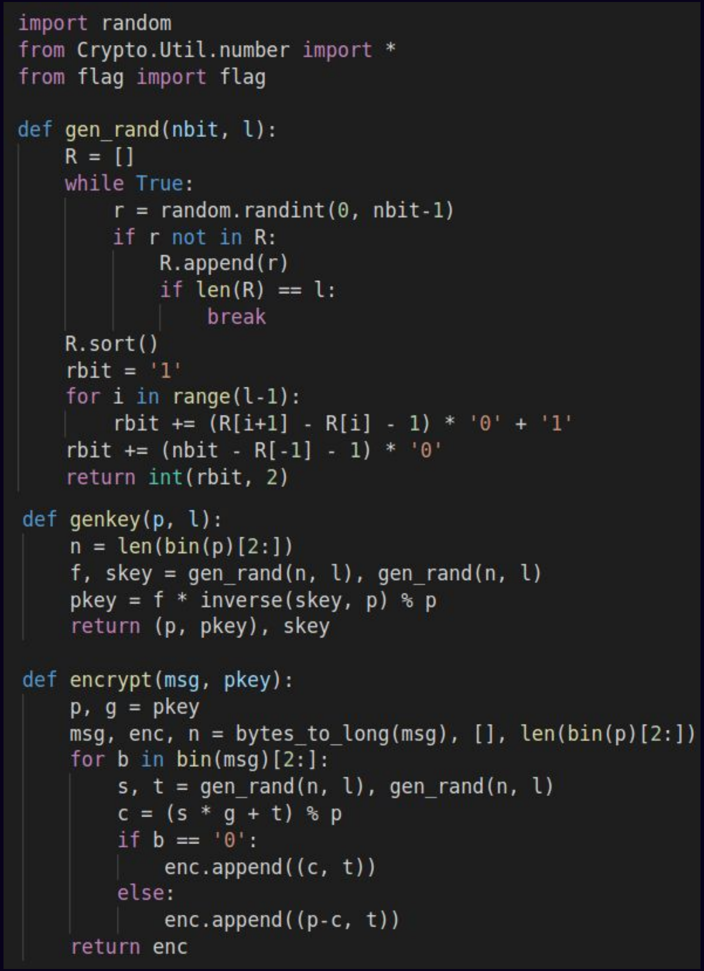
- Description:

  A game for grown ups, just play with bits to find out the sought after secret.

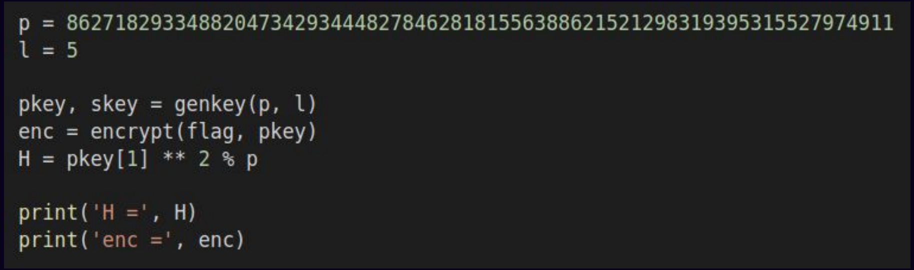
  2 files given: single_bits.py + output.txt

- Research + Math!

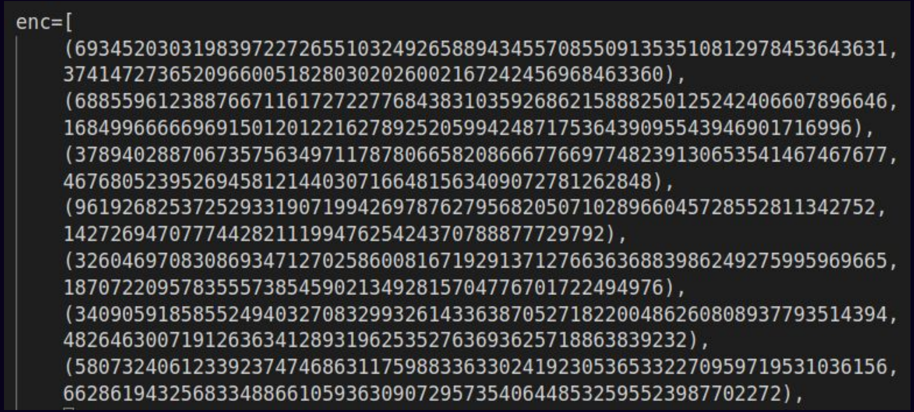
What was given...



```
import random
from Crypto.Util.number import *
from flag import flag

def gen_rand(nbit, l):
    R = []
    while True:
        r = random.randint(0, nbit-1)
        if r not in R:
            R.append(r)
            if len(R) == l:
                break
    R.sort()
    rbit = '1'
    for i in range(l-1):
        rbit += (R[i+1] - R[i] - 1) * '0' + '1'
    rbit += (nbit - R[-1] - 1) * '0'
    return int(rbit, 2)

def genkey(p, l):
    n = len(bin(p)[2:])
    f, skey = gen_rand(n, l), gen_rand(n, l)
    pkey = f * inverse(skey, p) % p
    return (p, pkey), skey

def encrypt(msg, pkey):
    p, g = pkey
    msg, enc, n = bytes_to_long(msg), [], len(bin(p)[2:])
    for b in bin(msg)[2:]:
        s, t = gen_rand(n, l), gen_rand(n, l)
        c = (s * g + t) % p
        if b == '0':
            enc.append((c, t))
        else:
            enc.append((p-c, t))
    return enc
```

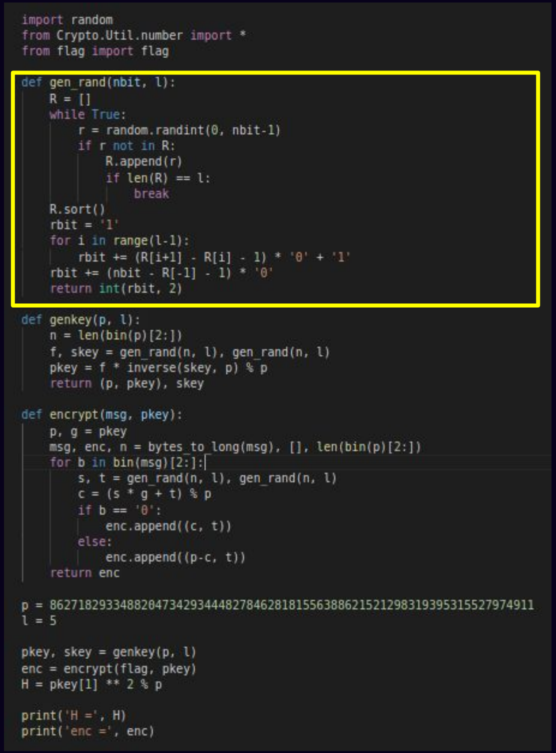


```
p = 862718293348820473429344482784628181556388621521298319395315527974911
l = 5

pkey, skey = genkey(p, l)
enc = encrypt(flag, pkey)
H = pkey[1] ** 2 % p

print('H =', H)
print('enc =', enc)
```

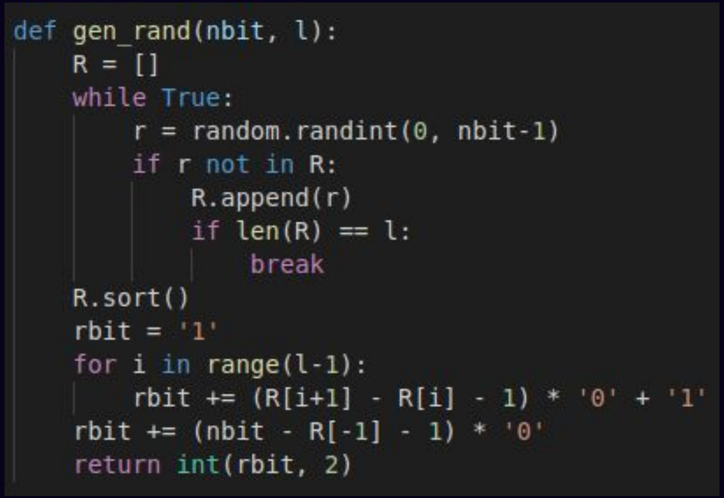


```
enc=[
    (693452030319839722726551032492658894345570855091353510812978453643631,
    374147273652096600518280302026002167242456968463360),
    (688559612388766711617272277684383103592686215888250125242406607896646,
    168499666669691501201221627892520599424871753643909554394690171 6996),
    (378940288706735756349711787806658208666776697748239130653541467467677,
    467680523952694581214403071664815634090727812628 48),
    (961926825372529331907199426978762795682050710289660457285528113427 52,
    142726947077744282111994762542437078887772 9792),
    (326046970830869347127025860081671929137127663636883986249275995969665,
    187072209578355573854590213492815704776701722494976),
    (340905918585524940327083299326143363870527182200486260808937793514394,
    482646300719126363412893196253527636936257188638 39232),
    (580732406123392374746863117598833633024192305365332270959719531036156,
    662861943256833488661059363090729573540644853259552398770 2272),
```

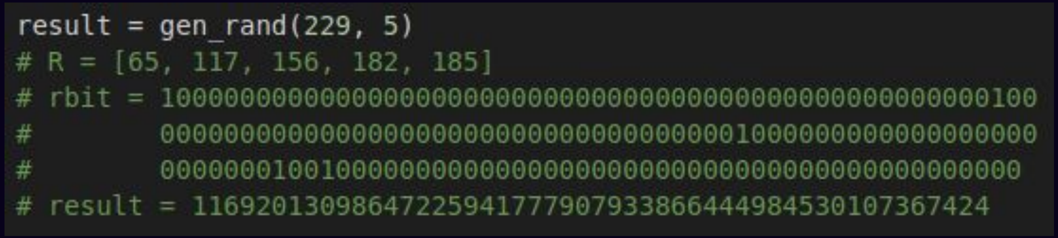
What does the code do ?



```python
import random
from Crypto.Util.number import *
from flag import flag

def gen_rand(nbit, l):
    R = []
    while True:
        r = random.randint(0, nbit-1)
        if r not in R:
            R.append(r)
            if len(R) == l:
                break
    R.sort()
    rbit = '1'
    for i in range(l-1):
        rbit += (R[i+1] - R[i] - 1) * '0' + '1'
    rbit += (nbit - R[-1] - 1) * '0'
    return int(rbit, 2)

def genkey(p, l):
    n = len(bin(p)[2:])
    f, skey = gen_rand(n, l), gen_rand(n, l)
    pkey = f * inverse(skey, p) % p
    return (p, pkey), skey

def encrypt(msg, pkey):
    p, g = pkey
    msg, enc, n = bytes_to_long(msg), [], len(bin(p)[2:])
    for b in bin(msg)[2:]:
        s, t = gen_rand(n, l), gen_rand(n, l)
        c = (s * g + t) % p
        if b == '0':
            enc.append((c, t))
        else:
            enc.append((p-c, t))
    return enc

p = 862718293348820473429344482784628181556388621521298319395315527974911
l = 5

pkey, skey = genkey(p, l)
enc = encrypt(flag, pkey)
H = pkey[1] ** 2 % p

print('H =', H)
print('enc =', enc)
```

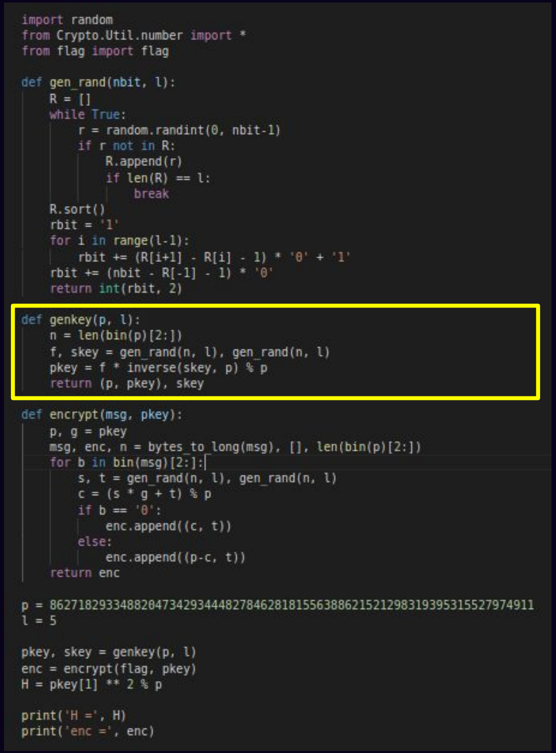


```python
def gen_rand(nbit, l):
    R = []
    while True:
        r = random.randint(0, nbit-1)
        if r not in R:
            R.append(r)
            if len(R) == l:
                break
    R.sort()
    rbit = '1'
    for i in range(l-1):
        rbit += (R[i+1] - R[i] - 1) * '0' + '1'
    rbit += (nbit - R[-1] - 1) * '0'
    return int(rbit, 2)
```

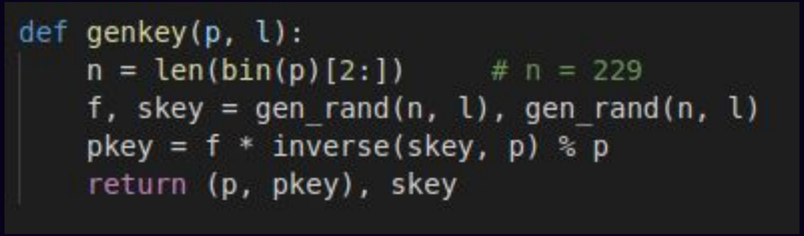


```python
result = gen_rand(229, 5)
# R = [65, 117, 156, 182, 185]
# rbit = 1000000000000000000000000000000000000000000000000000100
#        0000000000000000000000000000000000001000000000000000000
#        000000010010000000000000000000000000000000000000000000
# result = 11692013098647225941777907933866444984530107367424
```

What does the code do ?

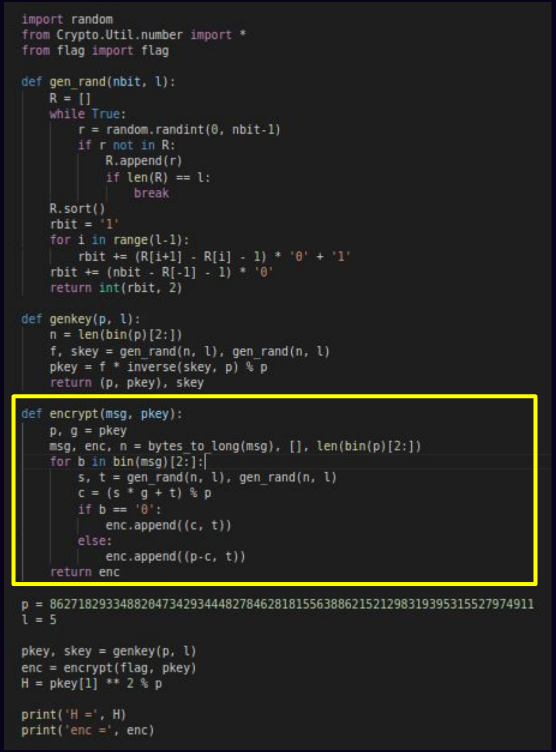


```
import random
from Crypto.Util.number import *
from flag import flag

def gen_rand(nbit, l):
    R = []
    while True:
        r = random.randint(0, nbit-1)
        if r not in R:
            R.append(r)
            if len(R) == l:
                break
    R.sort()
    rbit = '1'
    for i in range(l-1):
        rbit += (R[i+1] - R[i] - 1) * '0' + '1'
    rbit += (nbit - R[-1] - 1) * '0'
    return int(rbit, 2)

def genkey(p, l):
    n = len(bin(p)[2:])
    f, skey = gen_rand(n, l), gen_rand(n, l)
    pkey = f * inverse(skey, p) % p
    return (p, pkey), skey

def encrypt(msg, pkey):
    p, g = pkey
    msg, enc, n = bytes_to_long(msg), [], len(bin(p)[2:])
    for b in bin(msg)[2:]:
        s, t = gen_rand(n, l), gen_rand(n, l)
        c = (s * g + t) % p
        if b == '0':
            enc.append((c, t))
        else:
            enc.append((p-c, t))
    return enc

p = 862718293348820473429344482784628181556388621521298319395315527974911
l = 5

pkey, skey = genkey(p, l)
enc = encrypt(flag, pkey)
H = pkey[1] ** 2 % p

print('H =', H)
print('enc =', enc)
```

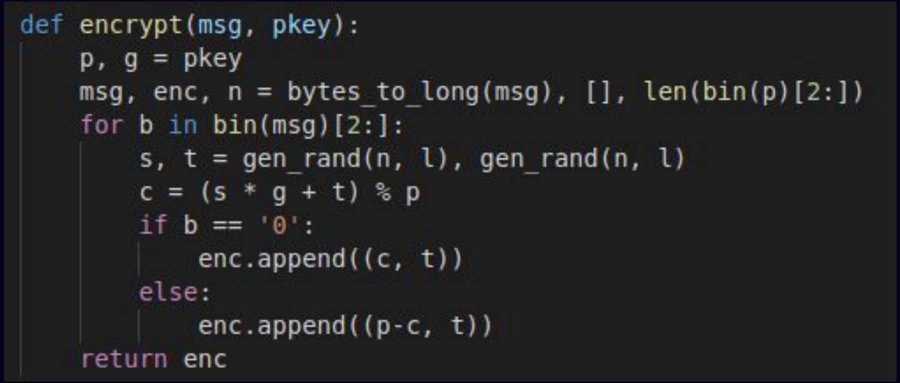


```
def genkey(p, l):
    n = len(bin(p)[2:])      # n = 229
    f, skey = gen_rand(n, l), gen_rand(n, l)
    pkey = f * inverse(skey, p) % p
    return (p, pkey), skey
```

What does the code do ?

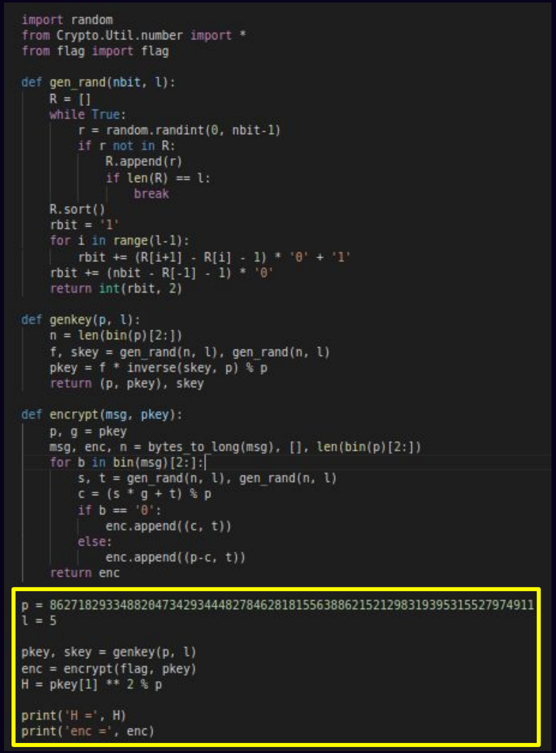


```python
import random
from Crypto.Util.number import *
from flag import flag

def gen_rand(nbit, l):
    R = []
    while True:
        r = random.randint(0, nbit-1)
        if r not in R:
            R.append(r)
            if len(R) == l:
                break
    R.sort()
    rbit = '1'
    for i in range(l-1):
        rbit += (R[i+1] - R[i] - 1) * '0' + '1'
    rbit += (nbit - R[-1] - 1) * '0'
    return int(rbit, 2)

def genkey(p, l):
    n = len(bin(p)[2:])
    f, skey = gen_rand(n, l), gen_rand(n, l)
    pkey = f * inverse(skey, p) % p
    return (p, pkey), skey

def encrypt(msg, pkey):
    p, g = pkey
    msg, enc, n = bytes_to_long(msg), [], len(bin(p)[2:])
    for b in bin(msg)[2:]:
        s, t = gen_rand(n, l), gen_rand(n, l)
        c = (s * g + t) % p
        if b == '0':
            enc.append((c, t))
        else:
            enc.append((p-c, t))
    return enc

p = 862718293348820473429344482784628181556388621521298319395315527974911
l = 5

pkey, skey = genkey(p, l)
enc = encrypt(flag, pkey)
H = pkey[1] ** 2 % p

print('H =', H)
print('enc =', enc)
```

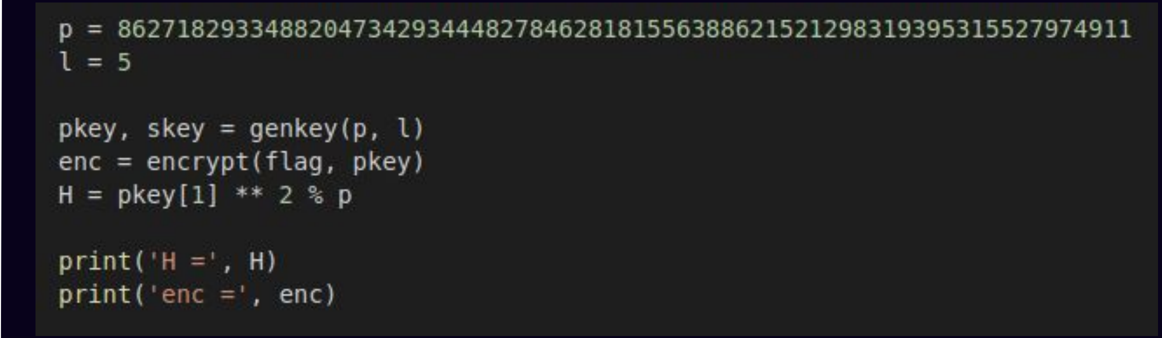


```python
def encrypt(msg, pkey):
    p, g = pkey
    msg, enc, n = bytes_to_long(msg), [], len(bin(p)[2:])
    for b in bin(msg)[2:]:
        s, t = gen_rand(n, l), gen_rand(n, l)
        c = (s * g + t) % p
        if b == '0':
            enc.append((c, t))
        else:
            enc.append((p-c, t))
    return enc
```

WHAT DOES THE CODE DO ?



```python
import random
from Crypto.Util.number import *
from flag import flag

def gen_rand(nbit, l):
    R = []
    while True:
        r = random.randint(0, nbit-1)
        if r not in R:
            R.append(r)
            if len(R) == l:
                break
    R.sort()
    rbit = '1'
    for i in range(l-1):
        rbit += (R[i+1] - R[i] - 1) * '0' + '1'
    rbit += (nbit - R[-1] - 1) * '0'
    return int(rbit, 2)

def genkey(p, l):
    n = len(bin(p)[2:])
    f, skey = gen_rand(n, l), gen_rand(n, l)
    pkey = f * inverse(skey, p) % p
    return (p, pkey), skey

def encrypt(msg, pkey):
    p, g = pkey
    msg, enc, n = bytes_to_long(msg), [], len(bin(p)[2:])
    for b in bin(msg)[2:]:
        s, t = gen_rand(n, l), gen_rand(n, l)
        c = (s * g + t) % p
        if b == '0':
            enc.append((c, t))
        else:
            enc.append((p-c, t))
    return enc

p = 862718293348820473429344482784628181556388621521298319395315527974911
l = 5

pkey, skey = genkey(p, l)
enc = encrypt(flag, pkey)
H = pkey[1] ** 2 % p

print('H =', H)
print('enc =', enc)
```



```python
p = 862718293348820473429344482784628181556388621521298319395315527974911
l = 5

pkey, skey = genkey(p, l)
enc = encrypt(flag, pkey)
H = pkey[1] ** 2 % p

print('H =', H)
print('enc =', enc)
```

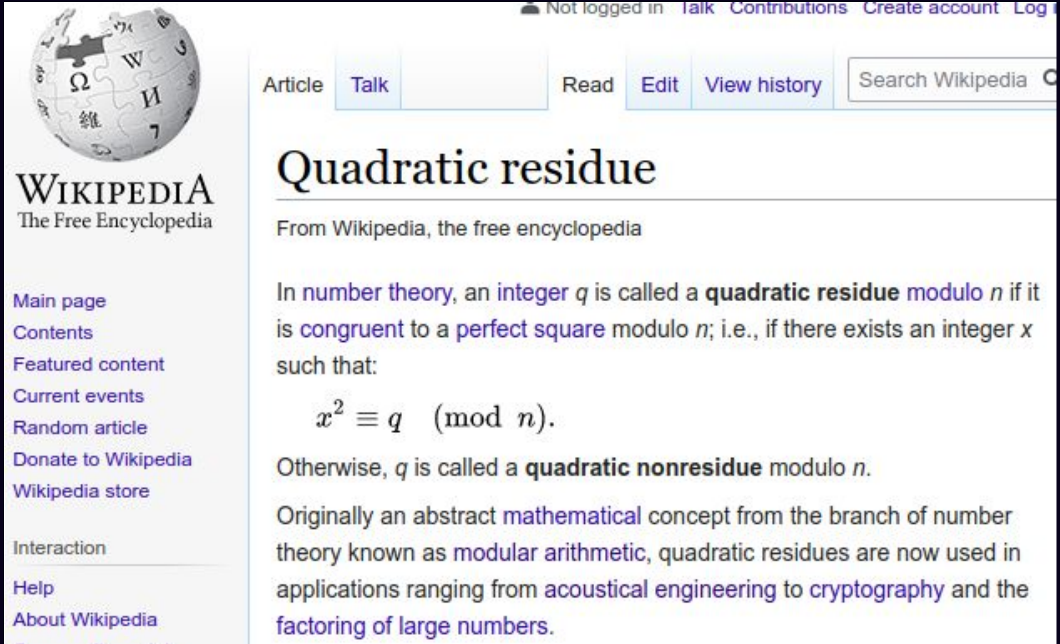
WHERE DO WE BEGIN ?

H = pkey[1] ** 2 % p
H = 381704527450191606347421195235742637659723827441243208291869156144963
p = 862718293348820473429344482784628181556388621521298319395315527974911

a = x^2 % p

What is a Quadratic Residue ?

`H = pkey[1] ** 2 % p` => $pkey^2 \equiv H \pmod{p}$



Quadratic residue

From Wikipedia, the free encyclopedia

In number theory, an integer *q* is called a **quadratic residue modulo** *n* if it is congruent to a perfect square modulo *n*; i.e., if there exists an integer *x* such that:

$$x^2 \equiv q \pmod{n}.$$

Otherwise, *q* is called a **quadratic nonresidue** modulo *n*.

Originally an abstract mathematical concept from the branch of number theory known as modular arithmetic, quadratic residues are now used in applications ranging from acoustical engineering to cryptography and the factoring of large numbers.

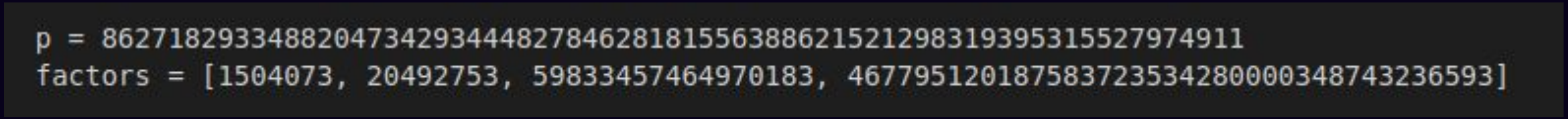
HOW TO SOLVE A QUADRATIC RESIDUE ?

Our **p** is not a prime number !

Factorization of **p** to factors **pi** +

Find solutions to each factor: $\mathbf{pkey^2 \equiv H \pmod{pi}}$ +

Chinese remainder theorem to combine the solutions



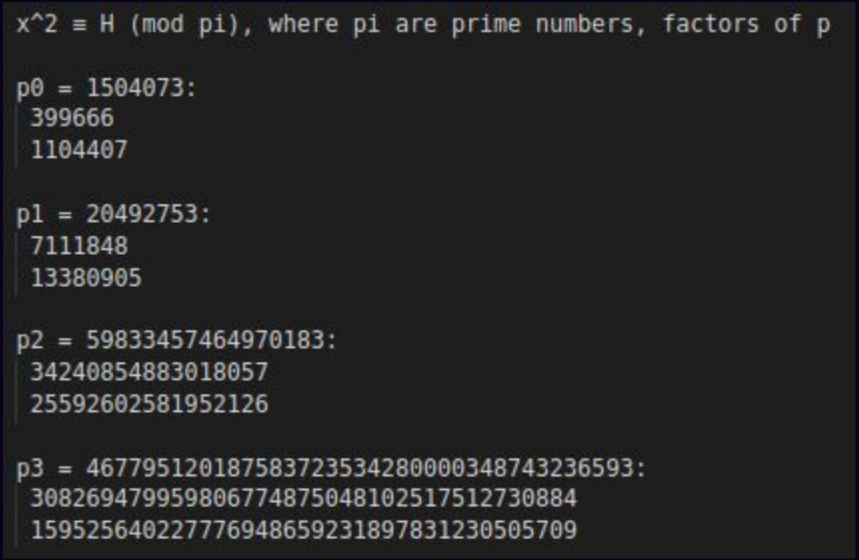
```
p = 862718293348820473429344482784628181556388621521298319395315527974911
factors = [1504073, 20492753, 59833457464970183, 467795120187583723534280000348743236593]
```

HOW TO FIND SOLUTIONS FOR THE FACTORS ?

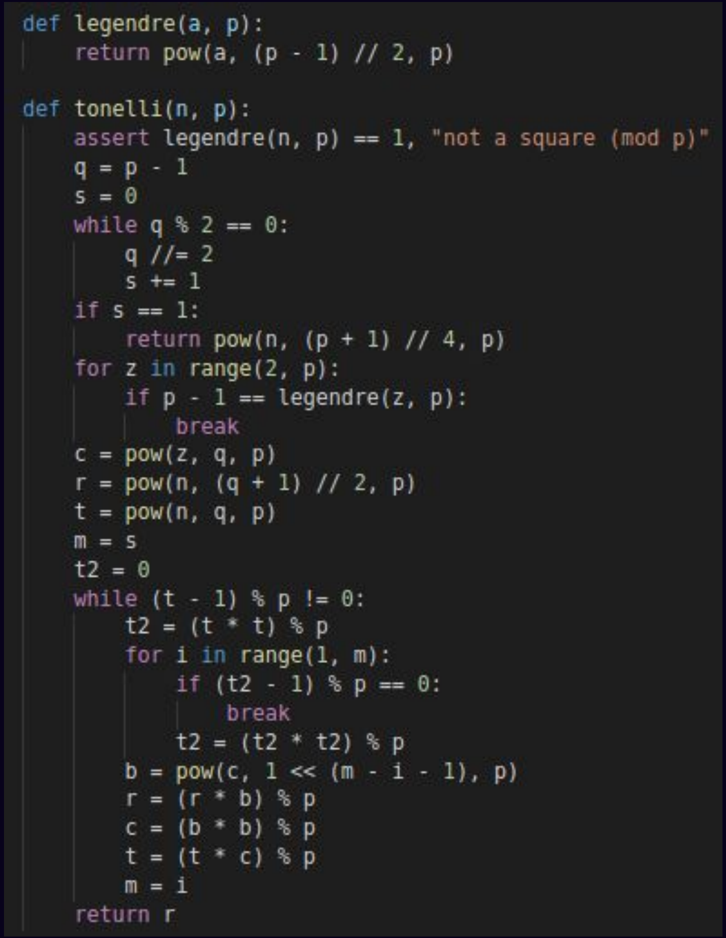
1) If **a** is a quadratic residue **(mod p)** and **$p \equiv 3 \pmod{4}$** then **$a^{(p+1)/4}$** is a solution to **$x^2 \equiv a \pmod{p}$**.

2) In all other cases there is no analogous formula and one may use the Tonelli-Shanks algorithm.

Tonelli-Shanks algorithm

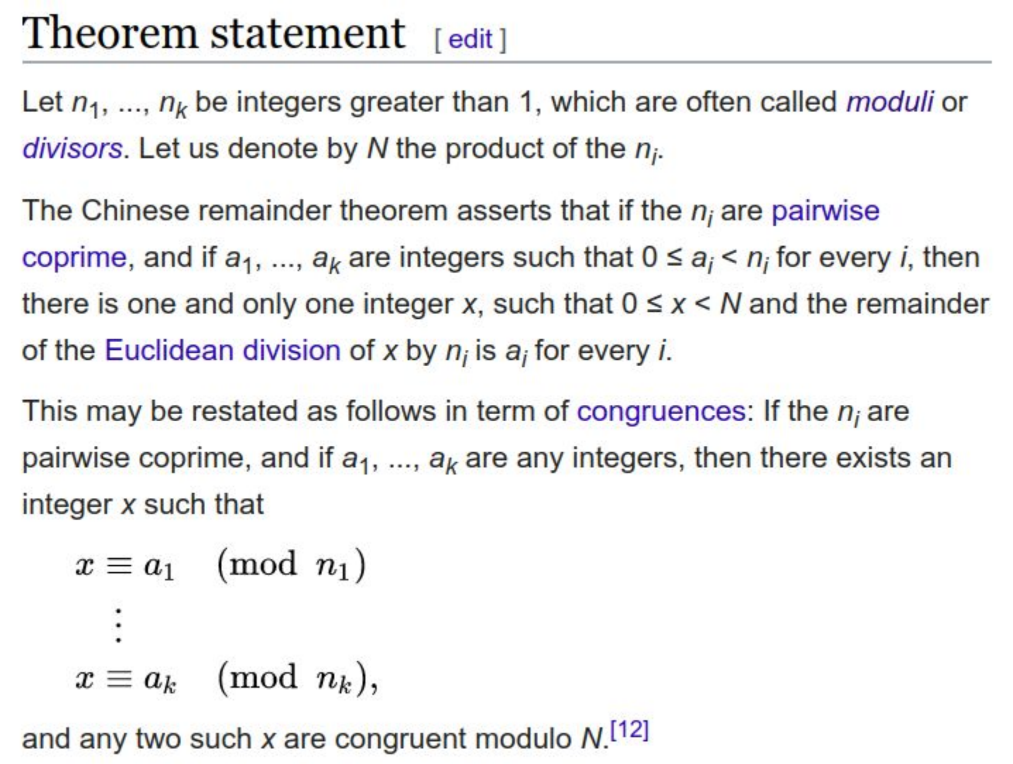


```
x^2 ≡ H (mod pi), where pi are prime numbers, factors of p

p0 = 1504073:
 399666
 1104407

p1 = 20492753:
 7111848
 13380905

p2 = 59833457464970183:
 34240854883018057
 25592602581952126

p3 = 467795120187583723534280000348743236593:
 308269479959806774875048102517512730884
 159525640227776948659231897831230505709
```



```python
def legendre(a, p):
    return pow(a, (p - 1) // 2, p)

def tonelli(n, p):
    assert legendre(n, p) == 1, "not a square (mod p)"
    q = p - 1
    s = 0
    while q % 2 == 0:
        q //= 2
        s += 1
    if s == 1:
        return pow(n, (p + 1) // 4, p)
    for z in range(2, p):
        if p - 1 == legendre(z, p):
            break
    c = pow(z, q, p)
    r = pow(n, (q + 1) // 2, p)
    t = pow(n, q, p)
    m = s
    t2 = 0
    while (t - 1) % p != 0:
        t2 = (t * t) % p
        for i in range(1, m):
            if (t2 - 1) % p == 0:
                break
            t2 = (t2 * t2) % p
        b = pow(c, 1 << (m - i - 1), p)
        r = (r * b) % p
        c = (b * b) % p
        t = (t * c) % p
        m = i
    return r
```

WHAT IS THE CHINESE REMAINDER THEOREM ?



Theorem statement [edit]

Let $n_1, \ldots, n_k$ be integers greater than 1, which are often called *moduli* or *divisors*. Let us denote by N the product of the n_i.

The Chinese remainder theorem asserts that if the n_i are pairwise coprime, and if $a_1, \ldots, a_k$ are integers such that $0 \leq a_i < n_i$ for every i, then there is one and only one integer x, such that $0 \leq x < N$ and the remainder of the Euclidean division of x by n_i is a_i for every i.

This may be restated as follows in term of congruences: If the n_i are pairwise coprime, and if $a_1, \ldots, a_k$ are any integers, then there exists an integer x such that

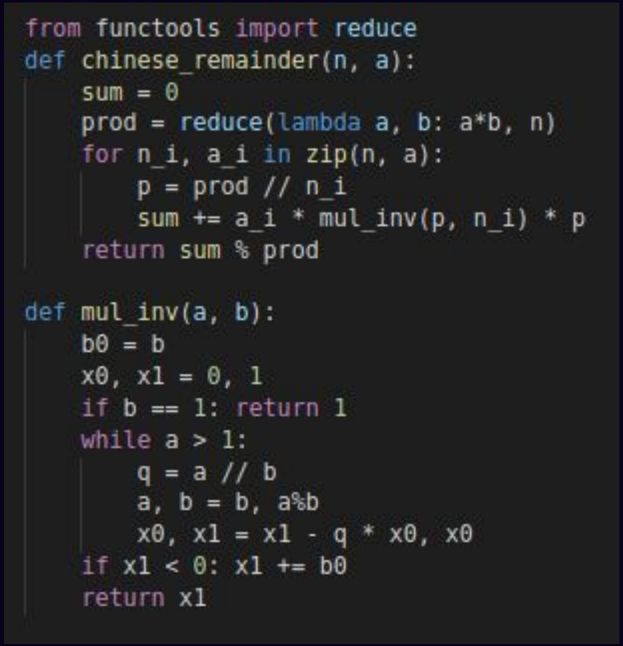
$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ &\vdots \\ x &\equiv a_k \pmod{n_k}, \end{aligned}$$

and any two such x are congruent modulo N.[12]

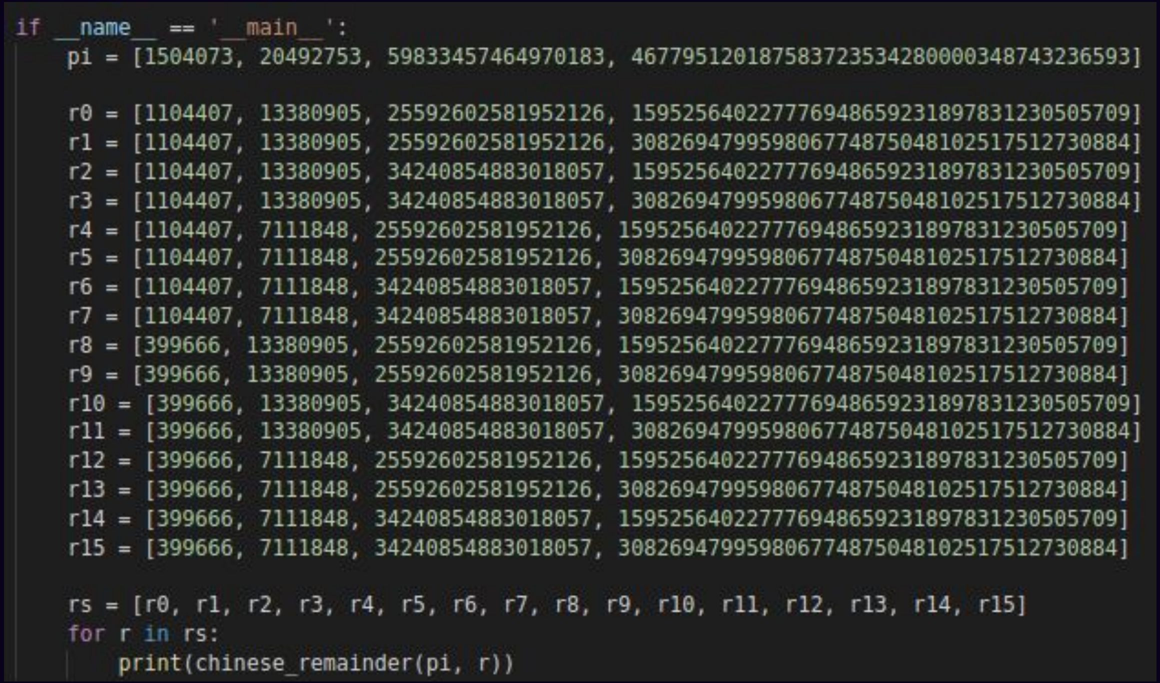
Looking for a solution of:
$H \equiv pkey^2 \pmod{p}$

Factors pi must be pairwise coprime! => gcd (pi, pj) = 1

HAVING TROUBLE APPLYING THE THEOREM...



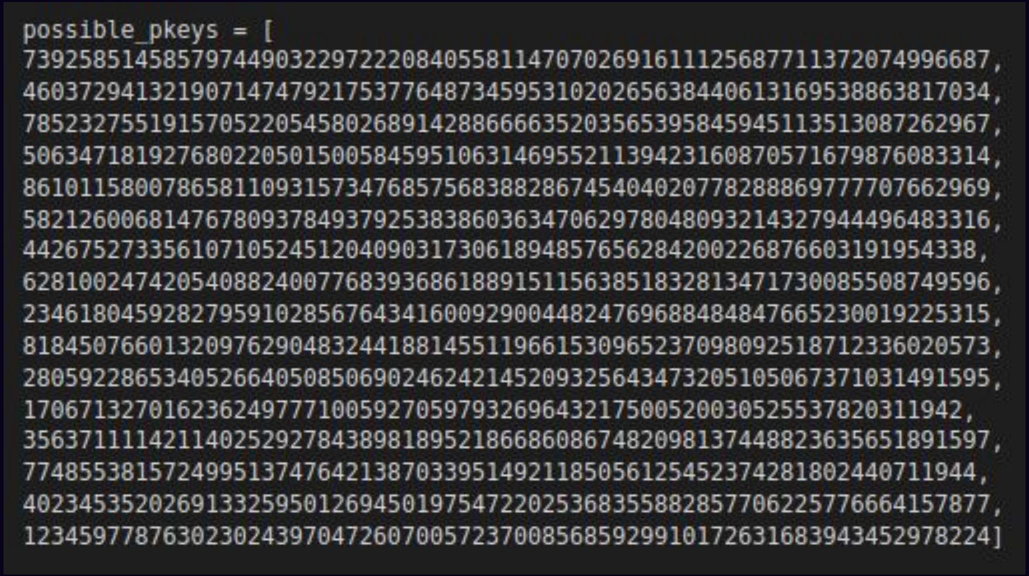
```python
from functools import reduce
def chinese_remainder(n, a):
    sum = 0
    prod = reduce(lambda a, b: a*b, n)
    for n_i, a_i in zip(n, a):
        p = prod // n_i
        sum += a_i * mul_inv(p, n_i) * p
    return sum % prod

def mul_inv(a, b):
    b0 = b
    x0, x1 = 0, 1
    if b == 1: return 1
    while a > 1:
        q = a // b
        a, b = b, a%b
        x0, x1 = x1 - q * x0, x0
    if x1 < 0: x1 += b0
    return x1
```



```python
if __name__ == '__main__':
    pi = [1504073, 20492753, 59833457464970183, 467795120187583723534280000348743236593]

    r0 = [1104407, 13380905, 25592602581952126, 159525640227776948659231897831230505709]
    r1 = [1104407, 13380905, 25592602581952126, 308269479959806774875048102517512730884]
    r2 = [1104407, 13380905, 34240854883018057, 159525640227776948659231897831230505709]
    r3 = [1104407, 13380905, 34240854883018057, 308269479959806774875048102517512730884]
    r4 = [1104407, 7111848, 25592602581952126, 159525640227776948659231897831230505709]
    r5 = [1104407, 7111848, 25592602581952126, 308269479959806774875048102517512730884]
    r6 = [1104407, 7111848, 34240854883018057, 159525640227776948659231897831230505709]
    r7 = [1104407, 7111848, 34240854883018057, 308269479959806774875048102517512730884]
    r8 = [399666, 13380905, 25592602581952126, 159525640227776948659231897831230505709]
    r9 = [399666, 13380905, 25592602581952126, 308269479959806774875048102517512730884]
    r10 = [399666, 13380905, 34240854883018057, 159525640227776948659231897831230505709]
    r11 = [399666, 13380905, 34240854883018057, 308269479959806774875048102517512730884]
    r12 = [399666, 7111848, 25592602581952126, 159525640227776948659231897831230505709]
    r13 = [399666, 7111848, 25592602581952126, 308269479959806774875048102517512730884]
    r14 = [399666, 7111848, 34240854883018057, 159525640227776948659231897831230505709]
    r15 = [399666, 7111848, 34240854883018057, 308269479959806774875048102517512730884]

    rs = [r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15]
    for r in rs:
        print(chinese_remainder(pi, r))
```

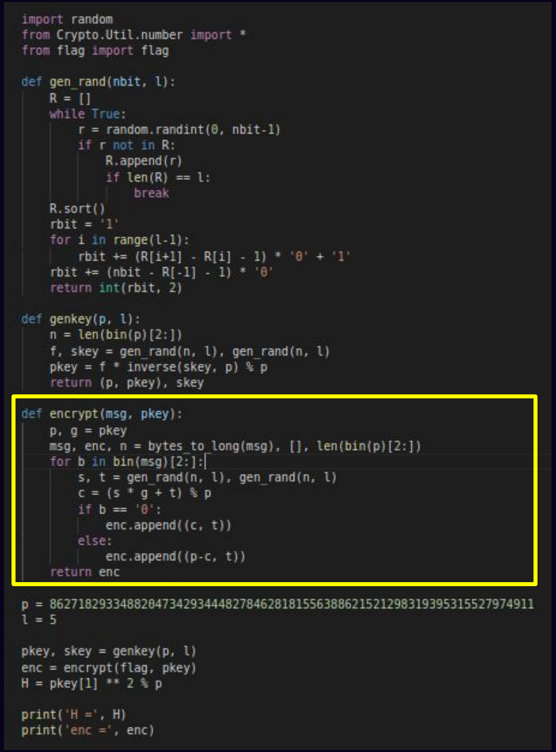
RESULTS OF THE CHINESE REMAINDER THEOREM:

YESSS!



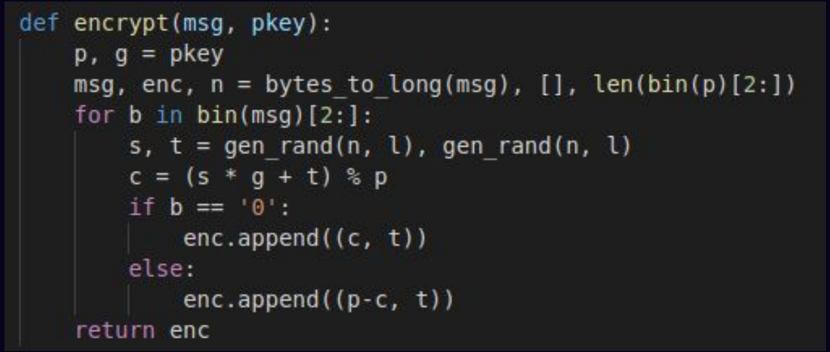
```
possible_pkeys = [
7392585145857974490322972220840558114707026916111256877113720749966687,
460372941321907147479217537764873459531020265638440613169538863817034,
7852327551915705220545802689142886666352035653958459451135130872629 67,
5063471819276802205015005845951063146955211394231608705716798760833 14,
8610115800786581109315734768575683882867454040207782888697777076629 69,
5821260068147678093784937925383860363470629780480932143279444964833 16,
442675273356107105245120409031730618948576562842002268766031919543 38,
]
```

NEXT STEP AFTER FINDING THE PUBLIC KEY …

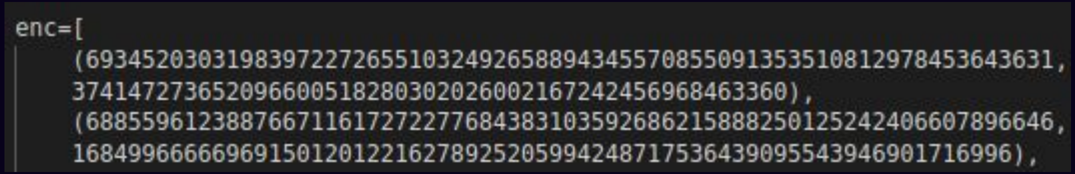


```
import random
from Crypto.Util.number import *
from flag import flag

def gen_rand(nbit, l):
    R = []
    while True:
        r = random.randint(0, nbit-1)
        if r not in R:
            R.append(r)
            if len(R) == l:
                break
    R.sort()
    rbit = '1'
    for i in range(l-1):
        rbit += (R[i+1] - R[i] - 1) * '0' + '1'
    rbit += (nbit - R[-1] - 1) * '0'
    return int(rbit, 2)

def genkey(p, l):
    n = len(bin(p)[2:])
    f, skey = gen_rand(n, l), gen_rand(n, l)
    pkey = f * inverse(skey, p) % p
    return (p, pkey), skey

def encrypt(msg, pkey):
    p, g = pkey
    msg, enc, n = bytes_to_long(msg), [], len(bin(p)[2:])
    for b in bin(msg)[2:]:
        s, t = gen_rand(n, l), gen_rand(n, l)
        c = (s * g + t) % p
        if b == '0':
            enc.append((c, t))
        else:
            enc.append((p-c, t))
    return enc

p = 862718293348820473429344482784628181556388621521298319395315527974911
l = 5

pkey, skey = genkey(p, l)
enc = encrypt(flag, pkey)
H = pkey[1] ** 2 % p

print('H =', H)
print('enc =', enc)
```

… reversing c = (s * g + t) % p

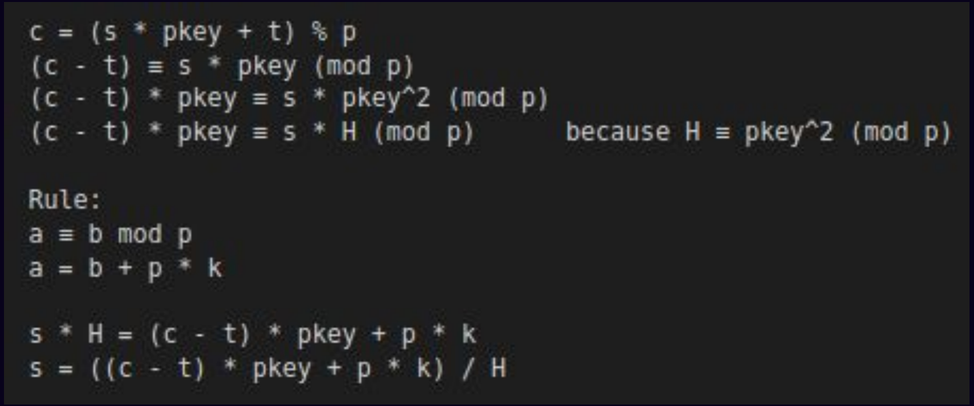


```
def encrypt(msg, pkey):
    p, g = pkey
    msg, enc, n = bytes_to_long(msg), [], len(bin(p)[2:])
    for b in bin(msg)[2:]:
        s, t = gen_rand(n, l), gen_rand(n, l)
        c = (s * g + t) % p
        if b == '0':
            enc.append((c, t))
        else:
            enc.append((p-c, t))
    return enc
```

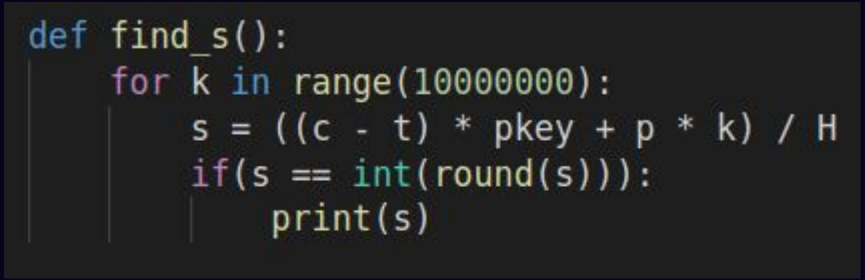


```
enc=[
    (693452030319839722726551032492658894345570855091353510812978453643631,
    374147273652096600518280302026002167242456968463360),
    (688559612388766711617272277684383103592686215888250125242406607896646,
    1684996666696915012012216278925205994248717536439095543946901716996),
```

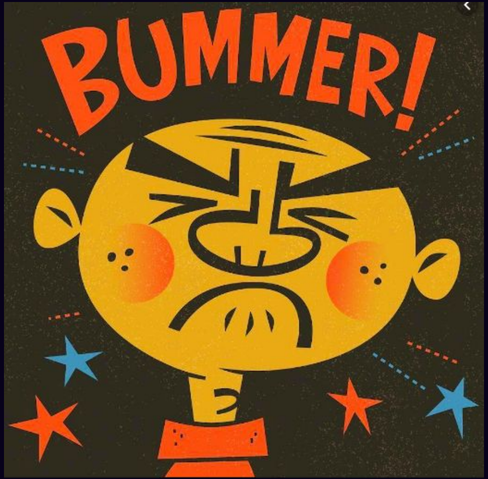
Brute forcing 's' ?



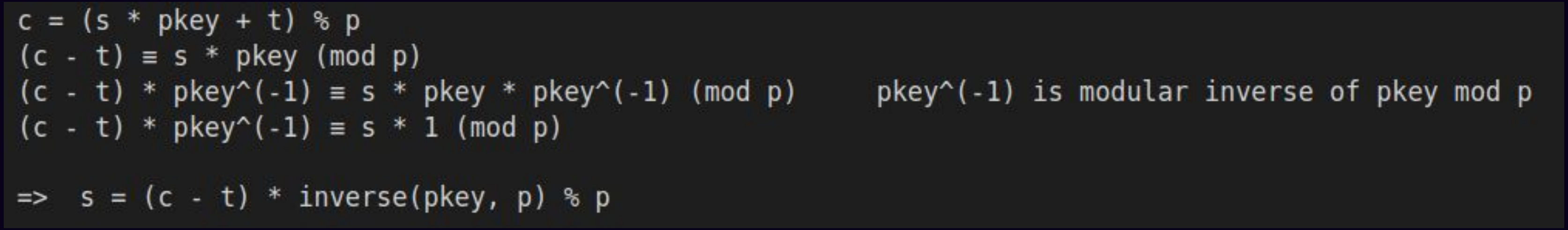
```
c = (s * pkey + t) % p
(c - t) ≡ s * pkey (mod p)
(c - t) * pkey ≡ s * pkey^2 (mod p)
(c - t) * pkey ≡ s * H (mod p)      because H ≡ pkey^2 (mod p)

Rule:
a ≡ b mod p
a = b + p * k

s * H = (c - t) * pkey + p * k
s = ((c - t) * pkey + p * k) / H
```



```python
def find_s():
    for k in range(10000000):
        s = ((c - t) * pkey + p * k) / H
        if(s == int(round(s))):
            print(s)
```



THE PROPER WAY TO REVERSE THE EXPRESSION !



```
c = (s * pkey + t) % p
(c - t) ≡ s * pkey (mod p)
(c - t) * pkey^(-1) ≡ s * pkey * pkey^(-1) (mod p)     pkey^(-1) is modular inverse of pkey mod p
(c - t) * pkey^(-1) ≡ s * 1 (mod p)

=>  s = (c - t) * inverse(pkey, p) % p
```

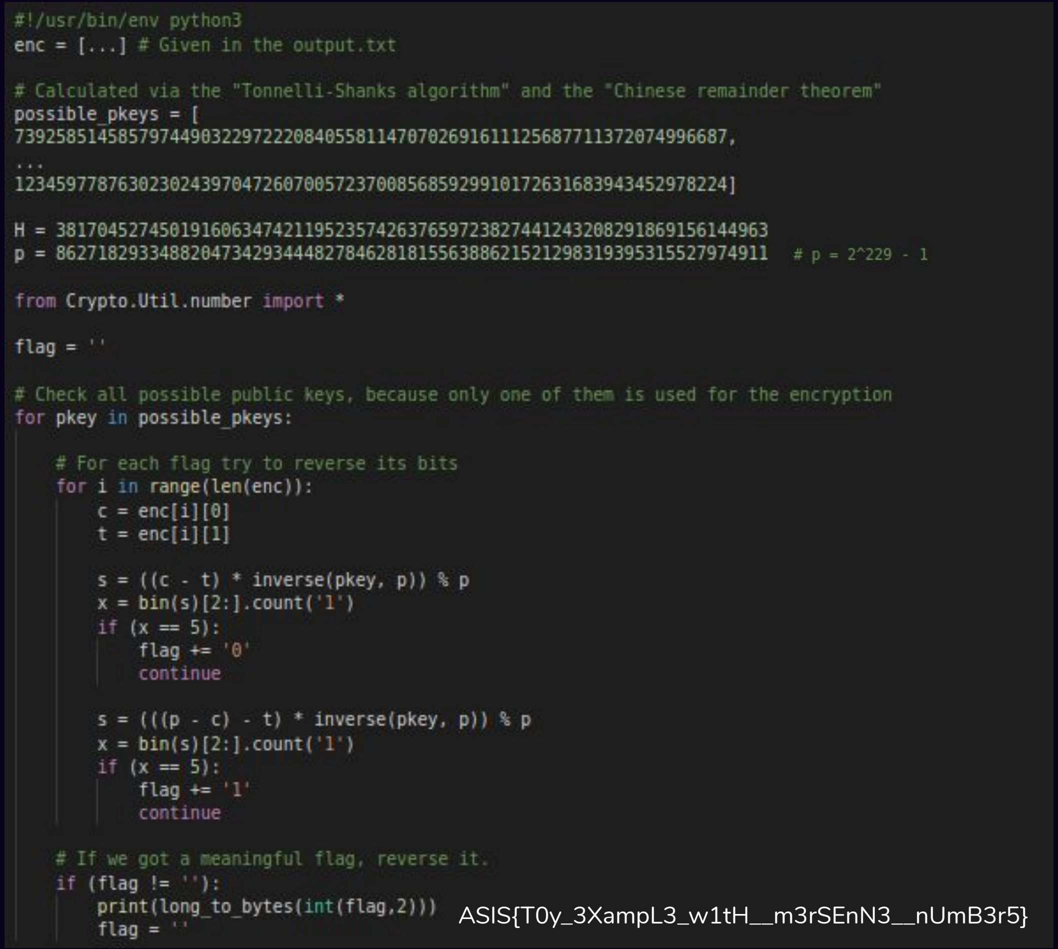
Modular inverse:

$$ax \equiv 1 \pmod{m}$$

$$aa^{-1} \equiv 1 \pmod{m}$$

Note: modular inverse of a number **n mod m** exists only if **gcd(n, m) = 1**
The library I used "*Crypto.Util.number*" doesn't check this and might produce wrong answers. Better alternatives are: "*gmpy2*" and "*sage*".

Final exploit:



```python
#!/usr/bin/env python3
enc = [...] # Given in the output.txt

# Calculated via the "Tonnelli-Shanks algorithm" and the "Chinese remainder theorem"
possible_pkeys = [
739258514585797449032297222084055811470702691611125687711372074996687,
...
1234597787630230243970472607005723700856859299101726316839434529782241

H = 381704527450191606347421195235742637659723827441243208291869156144963
p = 862718293348820473429344482784628181556388621521298319395315527974911   # p = 2^229 - 1

from Crypto.Util.number import *

flag = ''

# Check all possible public keys, because only one of them is used for the encryption
for pkey in possible_pkeys:

    # For each flag try to reverse its bits
    for i in range(len(enc)):
        c = enc[i][0]
        t = enc[i][1]

        s = ((c - t) * inverse(pkey, p)) % p
        x = bin(s)[2:].count('1')
        if (x == 5):
            flag += '0'
            continue

        s = (((p - c) - t) * inverse(pkey, p)) % p
        x = bin(s)[2:].count('1')
        if (x == 5):
            flag += '1'
            continue

    # If we got a meaningful flag, reverse it.
    if (flag != ''):
        print(long_to_bytes(int(flag,2)))
        flag = ''
```

ASIS{T0y_3XampL3_w1tH__m3rSEnN3__nUmB3r5}

COUNTERMEASURES.

1. Choosing **p** to be prime, so that it can’t be factored into smaller numbers.
2. Using a better random number generator.

Thank you

FOR YOUR ATTENTION !