

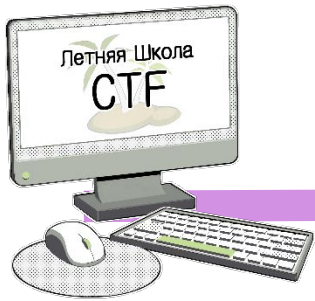
# Разбор задач CryptoCTF 2020

<http://10.10.10.19:8080/s/eQK4y9924FINXgR>

Задача	Категории	Баллы	Задача		Баллы
Trailing Bits	trivial	29	Classic	classic	226
Amsterdam	encoding	55	Complex to Hell	matrices	271
Gambler	algebraic, polynomial	86	Strip	googling, algebraic	285
Three Ravens	algebraic, pow	90	Fatima	pow, algebraic, encoding	316
Model	algebraic, pow	112	Namura	knapsack	354
One Line Crypto	trivial	142	Decent RSA	algebraic	398
Abbot	encoding	187	GenGol	algebraic, pow	423
Mad Hat	matrices	209	Open Band	?	477
Butterfly Effect	algebraic	209	Chilli (bug in task)	collisions	477
Heaven	stream cipher	226			

ASIS

[cryp.toc.tf/](http://cryp.toc.tf/)



# Модульная арифметика

**Простое число** - натуральное число, имеющее ровно два различных натуральных делителя — единицу и самого себя.

**Взятие  $x$  по модулю ( $x \% m$ )** – вычисление остатка от деления  $x$  на  $m$

$a$  и  $b$  **сравнимы** по модулю  $m$ , если их остатки при делении на  $m$  равны ( $a \equiv b \% m$ ). **Класс вычетов** – множество всех чисел, сравнимых с  $a$  по модулю  $m$ .

Отношение сравнимости является отношением эквивалентности (симметричным, транзитивным, рефлексивное)

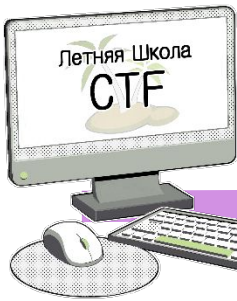
Допустимые операции со сравнениями:  $a + m * x \equiv b \% m$ ;  $a - b \equiv 0 \% m$ ;  $a + x \equiv b + x \% m$ ;  
 $a * x \equiv b * x \% m$ ;  $a^x \equiv b^x \% m$ ;  $a / k \equiv b / k \% m$  если  $\text{НОД}(k, m)=1$

**Обратное число по модулю** – такое  $i$ , что  $a * i \equiv 1 \% m$ . Существует для всех  $a$ , взаимнопростых с  $m$

**Теорема Эйлера:**  $a^{\varphi(m)} \equiv 1 \% m$ ,  $\varphi(m)$  - функция Эйлера (количество натуральных чисел, не превышающих  $n$  и взаимно простых с ним)

# Encoding





# Abbot

```
import string
import random
from fractions import Fraction as frac
from secret import flag

def me(msg):
    if len(msg) == 1:
        return ord(msg)
    msg = msg[::-1]
    reducer = len(msg) - 1
    resultNum, resultDen = frac(ord(msg[0]), reducer).denominator, frac(ord(msg[0]), r
    reducer -= 1
    for i in range(1, len(msg)-1):
        result = ord(msg[i]) + frac(resultNum, resultDen)
        resultDen, resultNum = result.denominator, result.numerator
        resultDen, resultNum = resultNum, reducer * resultDen
        reducer -= 1
    result = ord(msg[-1]) + frac(resultNum, resultDen)
    resultDen, resultNum = result.denominator, result.numerator
    return (resultNum, resultDen)

def you(msg):
    if len(msg) == 1:
        return ord(msg)
    msg = msg[::-1]
    reducer = (-1) ** len(msg)
    result = frac(ord(msg[0]), reducer)
    resultNum, resultDen = result.denominator, result.numerator
    reducer *= -1
    for i in range(1, len(msg)-1):
        result = ord(msg[i]) + frac(resultNum, resultDen)
        resultDen, resultNum = result.denominator, result.numerator
        resultDen, resultNum = resultNum, reducer * resultDen
        reducer *= -1
    result = ord(msg[-1]) + frac(resultNum, resultDen)
    resultDen, resultNum = result.denominator, result.numerator
    return (resultNum, resultDen)
```

```
def us(msg):
    if len(msg) == 1:
        return ord(msg)
    msg = msg[::-1]
    reducer = (-1) ** int(frac(len(msg), len(msg)**2))
    result = frac(ord(msg[0]), reducer)
    resultNum, resultDen = result.denominator, result.numerator
    reducer **= -1
    reducer = int(reducer)
    for i in range(1, len(msg)-1):
        result = ord(msg[i]) + frac(resultNum, resultDen)
        resultDen, resultNum = result.denominator, result.numerator
        resultDen, resultNum = resultNum, reducer * resultDen
        reducer **= -1
        reducer = int(reducer)
    result = ord(msg[-1]) + frac(resultNum, resultDen)
    resultDen, resultNum = result.denominator, result.numerator
    return (resultNum, resultDen)

dict_encrypt = {
    1: me,
    2: you,
    3: us,
    4: you,
    5: me
}

cipher = [[] for _ in range(5)]
S = list(range(1,6))
random.shuffle(S)
print("enc = [")
for i in range(4):
    cipher[i] = dict_encrypt[S[i]](flag[int(i * len(flag) // 5) : int(i * len(flag) //
    print(cipher[i])
    print(", ")
i += 1
cipher[i] = dict_encrypt[S[i]](flag[int(i * len(flag) // 5) : int(i * len(flag) // 5 + len
    print(cipher[i])
    print(" ]")

enc = [(4874974328610108385835995981839358584964018454799387862L, 727446086721304042164046
```



# Abbot

```
def us(msg):  
    if len(msg) == 1:  
        return ord(msg)  
    msg = msg[::-1]  
    reducer = (-1) ** int(frac(len(msg), len(msg)**2))  
    result = frac(ord(msg[0]), reducer)  
    resultNum, resultDen = result.denominator, result.numerator  
    reducer **= -1  
    reducer = int(reducer)  
    for i in range(1, len(msg)-1):  
        result = ord(msg[i]) + frac(resultNum, resultDen)  
        resultDen, resultNum = result.denominator, result.numerator  
        resultDen, resultNum = resultNum, reducer * resultDen  
        reducer **= -1  
        reducer = int(reducer)  
    result = ord(msg[-1]) + frac(resultNum, resultDen)  
  
    resultDen, resultNum = result.denominator, result.numerator  
    return (resultNum, resultDen)
```



```
def us_(num, den):  
  
    text = ""  
    resultDen, resultNum = den, num  
    r = int(resultNum / resultDen)  
    text += chr(r)  
    result = frac(resultNum, resultDen) - r  
    resultDen, resultNum = result.denominator, result.numerator  
    while resultNum != 0 and resultDen != 0:  
        r = int(resultNum / resultDen)  
  
    result = (frac(resultNum, resultDen) - r)  
    resultDen, resultNum = result.numerator, result.denominator  
  
    if result == 1:  
        text += chr(r - 1)  
    else:  
        text += chr(r)  
  
    return text
```

```
for ct in enc:  
    print(us(ct[0], ct[1]))  
    print(me(ct[0], ct[1]))  
    print(you(ct[0], ct[1]))  
  
#CCTF{This_13_n0t_Arthur_Who_  
#_ASIS_Crypto_CTF_with_  
#_very_m0d3rn_arthur_Enc0d1ng!  
#_l0ves_Short_st0ries_This_IS_  
#_!_D0_you_Enj0y_IT_as_w311??}  
  
# CCTF{This_13_n0t_Arthur_Who_l0ves_Short_st0ries_This_IS_ ASIS_ Crypto_CTF_ with_very_m0d3rn_arthur_Enc0d1ng!!_D0_you_Enj0y_IT_as_w311??}
```

# Amsterdam



```
54
55
56 def comb(n, k):
57     if k > n:
58         return 0
59     k = min(k, n - k)
60     u = reduce(operator.mul, range(n, n - k, -1), 1)
61     d = reduce(operator.mul, range(1, k + 1), 1)
62     return u // d
63
64 def encrypt(msg, n, k):
65     msg = bytes_to_long(msg.encode('utf-8'))
66     print("msg", msg)
67     if msg >= comb(n, k):
68         return -1
69     m = ['1'] + ['0' for i in range(n - 1)]
70     for i in range(1, n + 1):
71         if msg >= comb(n - i, k):
72             m[i-1] = '1'
73             msg -= comb(n - i, k)
74             print("sub", comb(n - i, k), n, i, k)
75             k -= 1
76     m = int(''.join(m), 2)
77
78     print("encryption_res_1", bin(m))
79     i, z = 0, [0 for i in range(n - 1)]
80     c = 0
81     while (m > 0):
82         if m % 4 == 1:
83             c += 3 ** i
84             m -= 1
85         elif m % 4 == 3:
86             c += 2 * 3 ** i
87             m += 1
88         m //= 2
89         i += 1
90     print("enc_res_2", c)
91     return c
92
93 #enc = encrypt(flag, n, k)
94 #print('enc =', enc)
95
96 enc = 5550332817876280162274999855997378479609235817133438293571677699650888686
97
```

# Amsterdam



```
56 def comb(n, k):
57     if k > n:
58         return 0
59     k = min(k, n - k)
60     u = reduce(operator.mul, range(n, n - k, -1), 1)
61     d = reduce(operator.mul, range(1, k + 1), 1)
62     return u // d
63
64 def encrypt(msg, n, k):
65     msg = bytes_to_long(msg.encode('utf-8'))
66     print("msg", msg)
67     if msg >= comb(n, k):
68         return -1
69     m = ['1'] + ['0' for i in range(n - 1)]
70     for i in range(1, n + 1):
71         if msg >= comb(n - i, k):
72             m[i-1] = '1'
73             msg -= comb(n - i, k)
74             print("sub", comb(n - i, k), n, i, k)
75             k -= 1
76     m = int(''.join(m), 2)
77
78     print("encryption_res_1", bin(m))
79     i, z = 0, [0 for i in range(n - 1)]
80     c = 0
81     while (m > 0):
82         if m % 4 == 1:
83             c += 3 ** i
84             m -= 1
85         elif m % 4 == 3:
86             c += 2 * 3 ** i
87             m += 1
88         m //= 2
89         i += 1
90     print("enc_res_2", c)
91     return c
92
93 #enc = encrypt(flag, n, k)
94 #print('enc =', enc)
95
96 enc = 555033281787628016227499985599737847960923581713343829357167769965088686
```

```
128
129 def dec (enc):
130     arr = []
131     i = 0
132     while enc > 0:
133         m = enc % 3
134         arr.append(m)
135         enc //= 3
136         i += 1
137     c = 0
138
139     for m in arr[::-1]:
140         c *= 2
141         if m == 1:
142             c += 1
143         elif m == 2:
144             c -= 1
145     return c
146
147 def decrypt(msg, n, k):
148     pt = 0
149
150     for i in range(2, n + 1):
151         if msg[i-1] == '1':
152             pt += comb(n - i, k)
153             k -= 1
154
155     return long_to_bytes(pt)
156
157
158 msg = dec (enc)
159 print(msg_)
160 msg_ = bin(msg_)[2:]
161 n = len(msg_)
162
163 for k in range(n):
164     pt = decrypt(msg_, n, k)
165     if b'CTF{' in pt:
166         print(k, pt)
167         break
168
169 # b'...: CTF{with_Re3p3ct_for_Sch4lkwijk_dec3nt_Encoding!} :...'
```

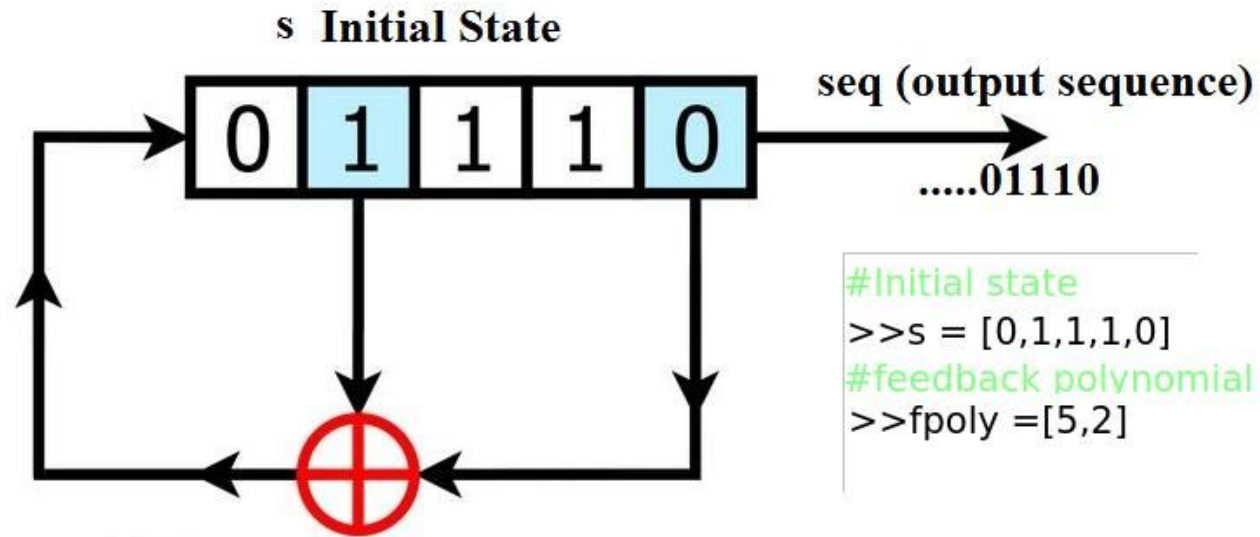
```
def decode(c): ## аналогичная функция из официальных writeups
c = c.str(base=3)
m = 0
for t in c:
    m *= 2
    if t == '1':
        m += 1
    elif t == '2':
        m -= 1
return m
```

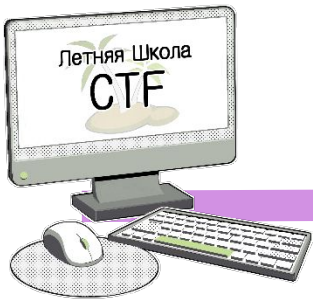


# Shift registers



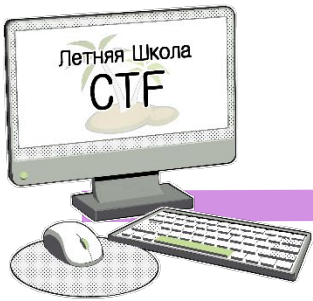
## LFSR





# Heaven

```
1 from bitstring import BitArray
2 from heaven import seventh_seal, oh, no, new_testament
3
4 def matthew_effect(shire, rohan):
5     gandalf = ''
6     for every, hobbit in enumerate(shire):
7         gandalf += oh if ord(hobbit) ^ ord(rohan[every]) == 0 else no
8     return gandalf
9
10 def born_to_die(isengard):
11     luke = 0
12     for book in new_testament:
13         luke ^= ord(isengard[book])
14     lizzy_grant = oh + isengard[:-1] if luke == 0 else no + isengard[:-1]
15     return lizzy_grant
16
17
18 david = len(seventh_seal)
19 elf = seventh_seal
20 lord = BitArray(bytes=bytes(open('flag.jpg', 'rb').read())).bin
21 bilbo = len(lord)
22 matthew = 0
23 princess_leia = ''
24 destiny = bilbo // david
25 apocalypse = bilbo % david
26 for i in range(32):
27     elf = born_to_die(elf)
28 while matthew < destiny:
29     princess_leia += matthew_effect(elf, lord[matthew * david_: (matthew + 1) * david])
30     elf = born_to_die(elf)
31     matthew += 1
32 princess_leia += matthew_effect(elf[:apocalypse], lord[matthew * david_:])
33 res = open('flag.enc', 'wb')
34 res.write(bytes(int(princess_leia[i_: i + 8], 2) for i in range(0, bilbo, 8)))
35
```



# Heaven

```
1 from bitstring import BitArray
2 from heaven import seventh_seal, oh, no, new_testament
3
4 def matthew_effect(shire, rohan):
5     gandalf = ''
6     for every, hobbit in enumerate(shire):
7         gandalf += oh if ord(hobbit) ^ ord(rohan[every]) == 0 else no
8     return gandalf
9
10 def born_to_die(isengard):
11     luke = 0
12     for book in new_testament:
13         luke ^= ord(isengard[book])
14     lizzy_grant = oh + isengard[:-1] if luke == 0 else no + isengard[:-1]
15     return lizzy_grant
16
17 |
18 david = len(seventh_seal)
19 elf = seventh_seal
20 lord = BitArray(bytes=bytes(open('flag.jpg', 'rb').read())).bin
21 bilbo = len(lord)
22 matthew = 0
23 princess_leia = ''
24 destiny = bilbo // david
25 apocalypse = bilbo % david
26 for i in range(32):
27     elf = born_to_die(elf)
28 while matthew < destiny:
29     princess_leia += matthew_effect(elf, lord[matthew * david_: (matthew + 1) * david])
30     elf = born_to_die(elf)
31     matthew += 1
32 princess_leia += matthew_effect(elf[:apocalypse], lord[matthew * david_:])
33 res = open('flag.enc', 'wb')
34 res.write(bytes(int(princess_leia[i_: i + 8], 2) for i in range(0, bilbo, 8)))
35
```

```
1 from bitstring import BitArray
2 #from heaven import seventh_seal, oh, no, new_testament
3
4 oh = '0'
5 no = '1'
6
7 def xor(shire, rohan):
8     gandalf = ''
9     for every, hobbit in enumerate(shire):
10         gandalf += oh if ord(hobbit) ^ ord(rohan[every]) == 0 else no
11     return gandalf
12
13 def shift(isengard):
14     luke = 0
15     for book in new_testament:
16         luke ^= ord(isengard[book])
17     lizzy_grant = oh + isengard[:-1] if luke == 0 else no + isengard[:-1]
18     return lizzy_grant
19
20 |
21 key = ""
22 key_len = len(key)
23
24 msg = BitArray(bytes=bytes(open('flag.jpg', 'rb').read())).bin
25 msg_len = len(msg)
26 ctr = 0
27 ct = ''
28 for i in range(32):
29     key = shift(key)
30 while ctr < msg_len // key_len:
31     ct += xor(key, msg[ctr * key_len_: (ctr + 1) * key_len])
32     key = shift(key)
33     ctr += 1
34
35 |
36 ct += xor(key[:msg_len % key_len], msg[ctr * key_len_:])
37 print(bytes(int(ct[i_: i + 8], 2) for i in range(0, msg_len, 8)))
```



# Heaven

1)

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ASCII
00000000	FF	D8	FE	E0	00	10	4A	46	49	46	00	01	01	01	00	48	яШяа..JFIF.....H
00000010	00	48	00	00	FF	DB	00	43	00	01	01	01	01	01	01	01	.Н..ЯЫ.С.....
00000020	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	.....
00000030	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	.....
00000040	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	.....
00000050	01	01	01	01	01	01	01	01	01	FF	DB	00	43	01	01	01	.....ЯЫ.С...
00000060	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	.....
00000070	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	.....
00000080	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	.....

```
f = open("flag.enc", "rb")
ct = f.read()
f.close()

k = "FFD8FFE0"
print(xor_(tobit(ct[:4]).hex(), tobit(k)))

stream = "1100011100101010110000110001110010"

keys = [
    "110001110010101011000"
    "0110001110010"
]
```

```
from bitstring import BitArray
#from heaven import seventh_seal, oh, no, new_testament

oh = '0'
no = '1'

def xor(shire, rohan):
    gandalf = ''
    for every, hobbit in enumerate(shire):
        gandalf += oh if ord(hobbit) ^ ord(rohan[every]) == 0 else no
    return gandalf

def shift(isengard):
    luke = 0
    for book in new_testament:
        luke ^= ord(isengard[book])
    lizzy_grant = oh + isengard[:-1] if luke == 0 else no + isengard[:-1]
    return lizzy_grant

key = ""
key_len = len(key)

msg = BitArray(bytes=bytes(open('flag.jpg', 'rb').read()).bin)
msg_len = len(msg)
ctr = 0
ct = ''
for i in range(32):
    key = shift(key)
while ctr < msg_len // key_len:
    ct += xor(key, msg[ctr * key_len_: (ctr + 1) * key_len])
    key = shift(key)
    ctr += 1

ct += xor(key[:msg_len % key_len], msg[ctr * key_len_:])
print(bytes(int(ct[i: i + 8], 2) for i in range(0, msg_len, 8)))
```



# Heaven

2)

JFIF APP0 marker segment		
Field	Size (bytes)	Description
APP0 marker	2	FF E0
Length	2	Length of segment excluding APP0 marker
Identifier	5	4A 46 49 46 00 = "JFIF" in ASCII, terminated by a null byte
JFIF version	2	First byte for major version, second byte for minor version ( 01 02 for 1.02)
Density units	1	Units for the following pixel density fields <ul style="list-style-type: none"> <li>• 00 : No units; width:height pixel aspect ratio = Ydensity:Xdensity</li> <li>• 01 : Pixels per inch (2.54 cm)</li> <li>• 02 : Pixels per centimeter</li> </ul>

Browser address bar showing: https://asecuritysite.com/information/jpeg?file=new\_logo4.jpg

What is Huffman Coding?

### Additional scanning signature analysis

Possible: [JPEG files, sig: FFD8FF] Count: 1  
Possible: [JPEG files, sig: 4A464946] Count: 1

### File hex characters

```
[00000000] FF D8 FF E0 00 10 4A 46 49 46 00 01 01 01 00 48 .....JFIF....H
[00000016] 00 48 00 00 FF DB 00 43 00 01 01 01 01 01 01 .H....C.....
[00000032] 01 01 01 01 01 01 01 01 01 01 01 01 01 01 .....
[00000048] 01 01 01 01 01 02 01 01 01 01 01 01 02 02 .....
[00000064] 02 02 02 02 02 02 02 02 02 02 02 02 02 .....
[00000080] 02 02 02 02 02 02 02 02 FF DB 00 43 01 01 .....C...
[00000096] 01 01 01 01 02 02 02 02 02 02 02 02 02 .....
[00000112] 02 02 02 02 02 02 02 02 02 02 02 02 02 .....
[00000128] 02 02 02 02 02 02 02 02 02 02 02 02 02 .....
```

```
key = "1100011100101011000"
key_len = len(key)

ct = BitArray(bytes=bytes(open('flag.enc', 'rb').read()).bin)
msg_len = len(ct)
ctr = 0
pt = ''
add = "000101" + " " * 100000

print(ct)
stream = ""

while ctr < msg_len // key_len:
    pt += xor(key, ct[ctr * key_len: (ctr + 1) * key_len])
    stream += key
    key = shift(key, add[0])
    add = add[1:]

    ctr += 1

pt += xor(key[msg_len % key_len], ct[ctr * key_len:])

print(stream)
print(pt)
print(tobyte(pt))

k = "FFD8FFE00104a4649460001"

#111100000000111111001000100001111101001101001101011100100001100111
#1100011100101011000 110001110010101100 11000111001010110 11000111
#11111111011000111 1111110000000000 00010000010010100 01100100
#f f d 8 _ _ f e 0 0 _ 1 0 4 a _ 6 4

#111100000000111111001000100001111101001101001101011100100001100111
#1100011100101011000011000111001010110000110001110010101100001100111
#1111111101100011111111110000000000000001000001001010010001100100
#f f d 8 f f e 0 0 0 1 0 4 a 4 6 4
```



# Heaven

2)

```
100
101 key = "1100011100101011000"
102 key_len = len(key)
103
104 ct = BitArray(bytes=bytes(open('flag.enc', 'rb').read())).bin
105 msg_len = len(ct)
106 ctr = 0
107 pt = ''
108 add = "0001011001011001011101010001" + "_" * 100000
109 #0,0,0,1,1,0,1,0,1,0,1,0,0,1,1,1,0,0,0,1,1,0,0,0,1,0,0,1,1,0,0,1,0,0,1,0,1,1,1,0,1,0,1,0,0,0,1"
110
111 print(ct[400:])
112 stream = ""
113
114 while ctr < msg_len // key_len:
115     pt += xor(key, ct[ctr * key_len: (ctr + 1) * key_len])
116     stream += key
117     key = shift(key, add[0])
118     add = add[1:]
119
120     ctr += 1
121
122 pt += xor(key[:msg_len % key_len], ct[ctr * key_len:])
123
124 print(stream[400:])
125 print(pt[400:])
126 print(tobyte(pt)[400:])
127
128 k = "FFD8FFE000104a4649460001"
```

CCTF{0Ne\_k3y\_t0\_rU1e\_7hem\_A11\_4Nd\_7o\_d3crYp7\_th3\_fL4g!}

UNIVERSITY OF Waterloo

Search uwaterloo.ca Search

### An Online Calculator of Berlekamp-Massey Algorithm

[Berlekamp-Massey algorithm](#) is an algorithm that will find the shortest linear feedback shift register (LFSR) for a given binary output sequence. Here we present a web-based implementation to compute the shortest LFSR and linear span of a given binary sequence. If you have any questions or suggestions, please do not hesitate to contact [Bo Zhu](#).

Please enter the binary sequence (separated by commas):

0,0,0,1,1,0,1,0,1,0,0,1,1,1,0,0,0,1,1,0,0,0,1,0,1,1,0,0,1,0,1,1,0,0,1,0,1,1,1,0,1,0,1,0,0,0,1

Press to Compute

LFSR:  $x^{19} + x^5 + x^2 + x^1 + 1$  Linear Span: 19 Time Used: 0.00 sec

Please note the output polynomial is using the form that its degree is always equal to the linear span. For example,  $x^3 + x + 1$

If the output is not what you expect, please check the input and run it on local computers.

University of Waterloo  
200 University Avenue West  
Waterloo, Ontario, Canada N2L 3G1  
519 888 4567

powered by Google App Engine

MAKING THE FUTURE SUPPORT WATERLOG

# Knapsack



открытый текст	1 1 1 1 1 0	0 0 1 1 0 0	0 0 0 0 0 0	0 0 0 0 0 1
вещи в рюкзаке	3 4 6 7 10 11	3 4 6 7 10 11	3 4 6 7 10 11	3 4 6 7 10 11
шифротекст	$3 + 4 + 6 + 7 + 10 = 30$	$6 + 7 = 13$	0	11

# Namura



```
def encrypt(pubkey, msg):  
    C = 0  
    for i in range(n):  
        C += pubkey[i] * int(msg[i])  
    return C  
  
flag = flag.lstrip('CCTF{').rstrip('}')  
bflag = bin(bytes_to_long(flag.encode('utf-8')))[2:]  
n = len(bflag)  
u = n - 30  
  
pubkey = keygen((n+1) // 3, n, u)  
  
print('pubkey =', pubkey)  
enc = encrypt(pubkey, bflag)  
print('enc =', enc)  
  
pubkey = [63673042463428268415078750502463684687819253083430137304541  
enc = 154657917005376465967753276253676484467260782425419406781078357]
```



# Namura



```
def encrypt(pubkey, msg):  
    C = 0  
    for i in range(n):  
        C += pubkey[i] * int(msg[i])  
    return C  
  
flag = flag.lstrip('CCTF{').rstrip('}')  
bflag = bin(bytes_to_long(flag.encode('utf-8')))[2:]  
n = len(bflag)  
u = n - 30  
  
pubkey = keygen((n+1) // 3, n, u)  
  
print('pubkey =', pubkey)  
enc = encrypt(pubkey, bflag)  
print('enc =', enc)  
  
pubkey = [63673042463428268415078750502463684687819253083430137304541  
enc = 154657917005376465967753276253676484467260782425419406781078357
```

## Knapsack Public-Key Cryptosystem Using Chinese Remainder Theorem

Yasuyuki MURAKAMI\*  
yasuyuki@isc.osakac.ac.jp

Takeshi NASAKO †  
nasako@m.ieice.org

$$B = \begin{pmatrix} 1 & 0 & \dots & 0 & -\lambda a_1 \\ 0 & 1 & \dots & 0 & -\lambda a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & -\lambda a_n \\ -1/2 & -1/2 & \dots & -1/2 & \lambda C \end{pmatrix},$$

# Namura



```
def encrypt(pubkey, msg):  
    C = 0  
    for i in range(n):  
        C += pubkey[i] * int(msg[i])  
    return C  
  
flag = flag.lstrip('CCTF{').rstrip('}')  
bflag = bin(bytes_to_long(flag.encode('utf-8')))[2:]  
n = len(bflag)  
u = n - 30  
  
pubkey = keygen((n+1) // 3, n, u)  
  
print('pubkey =', pubkey)  
enc = encrypt(pubkey, bflag)  
print('enc =', enc)  
  
pubkey = [63673042463428268415078750502463684687819253083430137304541  
enc = 1546579170053764659677532762536764844672607824254194067810783575
```

```
def LLL_for_knapsack_problem(vector, w):  
    K = 10000000  
    def check_01(line):  
        for l in line:  
            if l not in [0, 1]:  
                return False  
        return True  
  
    def check_0_1(line):  
        for l in line:  
            if l not in [0, -1]:  
                return False  
        return True  
  
    for i in range(len(vector)):  
        vector[i] *= K  
  
    l = len(vector)  
    A = matrix.identity(l)  
    B = zero_matrix(ZZ, l, 1)  
    C = matrix([vector])  
    D = [[-w * K]]  
    M = block_matrix(SR, [[A, B], [C, D]], subdivide=False)  
    M = Matrix(ZZ, l + 1, l + 1, M)  
    MR = M.T.LLL()  
    R = []  
    for line in MR:  
        if line[-1] == 0 and (check_01(line) or check_0_1(line)):  
            R.append(line[:-1])  
    return R  
  
pubkey = [636730424634282684150787505024636846878192530834301373045417941, 4434437  
enc = 154657917005376465967753276253676484467260782425419406781078357515  
  
print(LLL_for_knapsack_problem(pubkey, enc))  
for i in range(10000):  
    #print(shuffle(pubkey))  
    l = LLL_for_knapsack_problem(pubkey, enc)  
    if len(l) > 0:  
        print(l)  
        print(pubkey)  
        break
```

# Matrices



$$b_A = \begin{bmatrix} 4 & 1 & 11 & 10 \\ 5 & 5 & 9 & 5 \\ 3 & 9 & 0 & 10 \\ 1 & 3 & 3 & 2 \\ 12 & 7 & 3 & 4 \\ 6 & 5 & 11 & 4 \\ 3 & 3 & 5 & 0 \end{bmatrix} \times \begin{bmatrix} 6 \\ 9 \\ 11 \\ 11 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \\ 1 \\ 1 \\ 1 \\ 0 \\ -1 \end{bmatrix} \pmod{13} = \begin{bmatrix} 4 \\ 7 \\ 2 \\ 11 \\ 5 \\ 12 \\ 8 \end{bmatrix}$$

# Mad hat



```
import random
#from secret import p, flag

def transpose(x):...

def multiply(A, B):...

def sum_matrix(A, B):...

def keygen(p):
    d = random.randint(1, 2**64)
    if p % 4 == 1:
        Q = []
        for i in range(p):
            q = []
            for j in range(p):
                if i == j:
                    q.append(0)
                elif pow((i-j), int_((p-1) // 2), p) == 1:
                    q.append(1)
                else:
                    q.append(-1)
            Q.append(q)
        H = []
        r = []
        r.append(0)
        r.extend([1 for i in range(p)])
        H.append(r)
        for i in range(1, p + 1):
            r = []
            for j in range(p + 1):
                r.append(1 if j == 0 else r.append(Q[i-1][j-1]))
            H.append(r)

    H2 = [[0 for j in range(2*(p+1))] for i in range(2*(p+1))]
    for i in range(0, p+1):
        for j in range(0, p+1):
            if H[i][j] == 0:
                H2[i*2][j*2] = 1
                H2[i*2][j*2+1] = -1
                H2[i*2+1][j*2] = -1
                H2[i*2+1][j*2+1] = -1
            elif H[i][j] == 1:
                H2[i*2][j*2] = 1
                H2[i*2][j*2+1] = 1
                H2[i*2+1][j*2] = 1
                H2[i*2+1][j*2+1] = -1
            else:
                H2[i*2][j*2] = -1
                H2[i*2][j*2+1] = -1
                H2[i*2+1][j*2] = -1
                H2[i*2+1][j*2+1] = +1
    ID = [[(-1)**d if i == j else 0 for i in range(len(H2))] for j in range(len(H2))]
    H2 = multiply(ID, H2)
    return(H2, d)
else:
```

$m$  – ИСКОМЫЙ  
вектор

$K, d$  –  
закрытый ключ  
 $K$  – матрица,  $d$   
– натуральное  
число

$$c = m * K + [d, d \dots d]$$

```
return(H2, d)
else:
    Q = []
    for i in range(p):
        q = []
        for j in range(p):
            if i == j:
                q.append(0)
            elif pow((i-j), int_((p-1) // 2), p) == 1:
                q.append(1)
            else:
                q.append(-1)
        Q.append(q)
    H1 = []
    H1.append([1 for i in range(p+1)])
    for i in range(1, p + 1):
        r = []
        for j in range(p + 1):
            if j == 0:
                r.append(-1)
            elif i == j:
                r.append(1 + Q[i-1][j-1])
            else:
                r.append(Q[i-1][j-1])
        H1.append(r)
    ID = [[(-1)**d if i == j else 0 for i in range(len(H1))] for j in range(len(H1))]
    H1 = multiply(ID, H1)
    return(H1, d)

def encrypt(msg, key):
    matrix = key[0]
    d = key[1]
    m = [[ord(char) for char in msg]]
    de = [[-d for i in range(len(msg))]
    C = multiply(m, matrix)
    cipher = sum_matrix(C, de)
    return cipher

key = keygen(p)
flag = flag + (len(key[0][0]) - len(flag)) * flag[-1]
cipher = encrypt(flag, key)
print('cipher = ', cipher)

## 1 x 7d
cipher = [-3459749918754130611, -3459749918754138177, -3459749918754137803, -3459749918754138385,
-3459749918754138025, -3459749918754138097, -3459749918754138073, -3459749918754138245,
-3459749918754138183, -3459749918754138445, -3459749918754137991, -3459749918754138597,
```

# Mad hat



1) Какие передаваемые или генерируемые в самой функции параметры определяют ключевую матрицу?

```
import random
#from secret import p, flag

def transpose(x):...

def multiply(A, B):...

def sum_matrix(A, B):...

def keygen(p):
    d = random.randint(1, 2**64)
    if p % 4 == 1:
        Q = []
        for i in range(p):
            q = []
            for j in range(p):
                if i == j:
                    q.append(0)
                elif pow((i-j), int_((p-1) // 2), p) == 1:
                    q.append(1)
                else:
                    q.append(-1)
            Q.append(q)
        H = []
        r = []
        r.append(0)
        r.extend([1 for i in range(p)])
        H.append(r)
        for i in range(1, p + 1):
            r = []
            for j in range(p + 1):
                r.append(1 if j == 0 else r.append(Q[i-1][j-1])
            H.append(r)

        H2 = [[0 for j in range(2*(p+1))] for i in range(2*(p+1))]
        for i in range(0, p+1):
            for j in range(0, p+1):
                if H[i][j] == 0:
                    H2[i*2][j*2] = 1
                    H2[i*2][j*2+1] = -1
                    H2[i*2+1][j*2] = -1
                    H2[i*2+1][j*2+1] = -1
                elif H[i][j] == 1:
                    H2[i*2][j*2] = 1
                    H2[i*2][j*2+1] = 1
                    H2[i*2+1][j*2] = 1
                    H2[i*2+1][j*2+1] = -1
                else:
                    H2[i*2][j*2] = -1
                    H2[i*2][j*2+1] = -1
                    H2[i*2+1][j*2] = -1
                    H2[i*2+1][j*2+1] = +1
            ID = [[(-1)**d if i == j else 0 for i in range(len(H2))] for j in range(len(H2))]
            H2 = multiply(ID, H2)
            return(H2, d)
    else:
```

```
        return(H2, d)
    else:
        Q = []
        for i in range(p):
            q = []
            for j in range(p):
                if i == j:
                    q.append(0)
                elif pow((i-j), int_((p-1) // 2), p) == 1:
                    q.append(1)
                else:
                    q.append(-1)
            Q.append(q)
        H1 = []
        H1.append([1 for i in range(p+1)])
        for i in range(1, p + 1):
            r = []
            for j in range(p + 1):
                if j == 0:
                    r.append(-1)
                elif i == j:
                    r.append(1 + Q[i-1][j-1])
                else:
                    r.append(Q[i-1][j-1])
            H1.append(r)
        ID = [[(-1)**d if i == j else 0 for i in range(len(H1))] for j in range(len(H1))]
        H1 = multiply(ID, H1)
        return(H1, d)

def encrypt(msg, key):
    matrix = key[0]
    d = key[1]
    m = [[ord(char) for char in msg]]
    de = [[-d for i in range(len(msg))]
    C = multiply(m, matrix)
    cipher = sum_matrix(C, de)
    return cipher

key = keygen(p)
flag = flag + (len(key[0][0]) - len(flag)) * flag[-1]
cipher = encrypt(flag, key)
print('cipher = ', cipher)

## 1 x 7d
cipher = [-3459749918754130611, -3459749918754138177, -3459749918754137803, -3459749918754138385,
-3459749918754138025, -3459749918754138097, -3459749918754138073, -3459749918754138245,
-3459749918754138183, -3459749918754138445, -3459749918754137991, -3459749918754138597,
```

# Mad hat



2) как именно они влияют и что нужно для их нахождения

```
import random
#from secret import p, flag

def transpose(x):...

def multiply(A, B):...

def sum_matrix(A, B):...

def keygen(p):
    d = random.randint(1, 2**64)
    if p % 4 == 1:
        Q = []
        for i in range(p):
            q = []
            for j in range(p):
                if i == j:
                    q.append(0)
                elif pow((i-j), int_((p-1) // 2), p) == 1:
                    q.append(1)
                else:
                    q.append(-1)
            Q.append(q)
        H = []
        r = []
        r.append(0)
        r.extend([1 for i in range(p)])
        H.append(r)
        for i in range(1, p + 1):
            r = []
            for j in range(p + 1):
                r.append(1 if j == 0 else r.append(Q[i-1][j-1])
            H.append(r)

        H2 = [[0 for j in range(2*(p+1))] for i in range(2*(p+1))]
        for i in range(0, p+1):
            for j in range(0, p+1):
                if H[i][j] == 0:
                    H2[i*2][j*2] = 1
                    H2[i*2][j*2+1] = -1
                    H2[i*2+1][j*2] = -1
                    H2[i*2+1][j*2+1] = -1
                elif H[i][j] == 1:
                    H2[i*2][j*2] = 1
                    H2[i*2][j*2+1] = 1
                    H2[i*2+1][j*2] = 1
                    H2[i*2+1][j*2+1] = -1
                else:
                    H2[i*2][j*2] = -1
                    H2[i*2][j*2+1] = -1
                    H2[i*2+1][j*2] = -1
                    H2[i*2+1][j*2+1] = +1
            ID = [((-1)**d if i == j else 0 for i in range(len(H2))] for j in range(len(H2))]
            H2 = multiply(ID, H2)
            return(H2, d)
    else:
```

Для d имеет значение ТОЛЬКО ЧЕТНОСТЬ

p можно определить по размерности шифртекста:  
 $p_1 = 37$   
 $p_2 = 75$

```
        return(H2, d)
    else:
        Q = []
        for i in range(p):
            q = []
            for j in range(p):
                if i == j:
                    q.append(0)
                elif pow_((i-j), int_((p-1) // 2), p) == 1:
                    q.append(1)
                else:
                    q.append(-1)
            Q.append(q)
        H1 = []
        H1.append([1 for i in range(p+1)])
        for i in range(1, p + 1):
            r = []
            for j in range(p + 1):
                if j == 0:
                    r.append(-1)
                elif i == j:
                    r.append(1 + Q[i-1][j-1])
                else:
                    r.append(Q[i-1][j-1])
            H1.append(r)
        ID = [((-1)**d if i == j else 0 for i in range(len(H1))] for j in range(len(H1))]
        H1 = multiply(ID, H1)
        return(H1, d)

def encrypt(msg, key):
    matrix = key[0]
    d = key[1]
    m = [[ord(char) for char in msg]]
    de = [[-d for i in range(len(msg))]
    C = multiply(m, matrix)
    cipher = sum_matrix(C, de)
    return cipher

key = keygen(p)
flag = flag + (len(key[0][0]) - len(flag)) * flag[-1]
cipher = encrypt(flag, key)
print('cipher = ', cipher)

## 1 x 7d
cipher = [-3459749918754130611, -3459749918754138177, -3459749918754137803, -3459749918754138385,
-3459749918754138025, -3459749918754138097, -3459749918754138073, -3459749918754138245,
-3459749918754138183, -3459749918754138445, -3459749918754137991, -3459749918754138597,
```

# Mad hat



```
import numpy as np

p1 = 37
p2 = 75

matrices = [np.matrix(keygen(p1, 1)[0]), np.matrix(keygen(p2, 1)[0]),
            np.matrix(keygen(p1, 2)[0]), np.matrix(keygen(p2, 2)[0])]
inv_matrices = [np.linalg.inv(a) for a in matrices]
d0 = -3459749918754130000

for m in inv_matrices:
    for i in range(10000):
        cipher_ = sum_matrix([cipher], [[-d0 + i] * 76])
        cipher_ = np.matrix(cipher_).dot(m)
        flag = ""
        b = True
        for c in range(76):
            c = round(cipher_[0, c])
            if c < 0 or c >= 256:
                b = False
                break
            flag += chr(c)
        if b and "CCTF{" in flag:
            print(flag)

# CCTF{TH13_i3_Hadamard_rip_y0ung_&_brilliant_Paley!}}}}}}}}}}}}}}}}}}}}}}}}
```

```
136 import numpy as np
137 R = IntegerModRing(257)
138
139 p1 = 37
140 p2 = 75
141
142 matrices = [Matrix(R, keygen(p1, 1)[0]), Matrix(R, keygen(p2, 1)[0]),
143            Matrix(R, keygen(p1, 2)[0]), Matrix(R, keygen(p2, 2)[0])]
144 inv_matrices = [a.inverse() for a in matrices]
145 d0 = -3459749918754130000
146
147
148 for m in inv_matrices:
149     for i in range(257):
150         cipher_ = Matrix(R, [cipher]) + Matrix(R, [[-d0 + i] * 76])
151         cipher_ = cipher_ * m
152         flag = ""
153         b = True
154         for c in range(76):
155             c = cipher_[0, c]
156             if c < 0 or c >= 256:
157                 b = False
158                 break
159             flag += chr(c)
160         if b and "CCTF{" in flag:
161             print(flag)
162             #break
163
```

# Algebraic



```
sage: R.<x> = ZZ['x']; A.<Dx> = OreAlgebra(R)
```

```
sage: L = (5*x^2+3*x-7)*Dx^2 + (3*x^2+8*x-1)*Dx + (9*x^2-3*x+8)
```

```
sage: L.parent()
```

$\mathbf{Z}[x]\langle Dx \rangle$

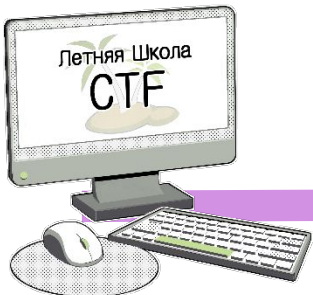
```
sage: B = OreAlgebra(QQ['x'], 'Dx')
```

```
sage: L = B(L)
```

```
sage: L.parent()
```

$\mathbf{Q}[x]\langle Dx \rangle$





# Gambling

```
nc 05.cr.yr.toc.tf 33371
```

```
+++++
+ Hi, there is a strong relation between philosophy and the gambling! +
+ Gamble as an ancient philosopher and find the flag :)           +
+++++
| Options:
|   [C]ipher flag!
|   [E]ncryption function!
|   [T]ry the encryption
|   [Q]uit
```

```
def encrypt(m, p, a, b):
    assert m < p and isPrime(p)
    return (m ** 3 + a * m + b) % p
```

$p, a, b$  – не известны

$f(x) \equiv x^3 + ax + b \pmod{p}$

$f(\text{flag}) = c$

Мы можем получить  $f(x)$  для любого  $x$



# Gambling

nc 05.cr.yp.toc.tf 33371

```
+++++
+ Hi, there is a strong relation between philosophy and the gambling! +
+ Gamble as an ancient philosopher and find the flag :) +
+++++
| Options:
| [C]ipher flag!
| [E]ncryption function!
| [T]ry the encryption
| [Q]uit
```

```
def encrypt(m, p, a, b):
    assert m < p and isPrime(p)
    return (m ** 3 + a * m + b) % p
```

$p, a, b$  – не известны

$f(x) \equiv x^3 + ax + b \pmod{p}$

$f(\text{flag}) = c$

Мы можем получить  $f(x)$  для любого  $x$

$f(0) \equiv b \pmod{p}$  – получаем  $b$

$f(1) \equiv 1 + a + b \pmod{p}$  – получаем  $a$

$$c_1 = x_1^3 + ax_1 + b = f(x_1) + k_1p$$

$$c_2 = x_2^3 + ax_2 + b = f(x_2) + k_2p$$

НОД ( $c_1 - f(x_1), c_2 - f(x_2)$ ) делится на  $p$

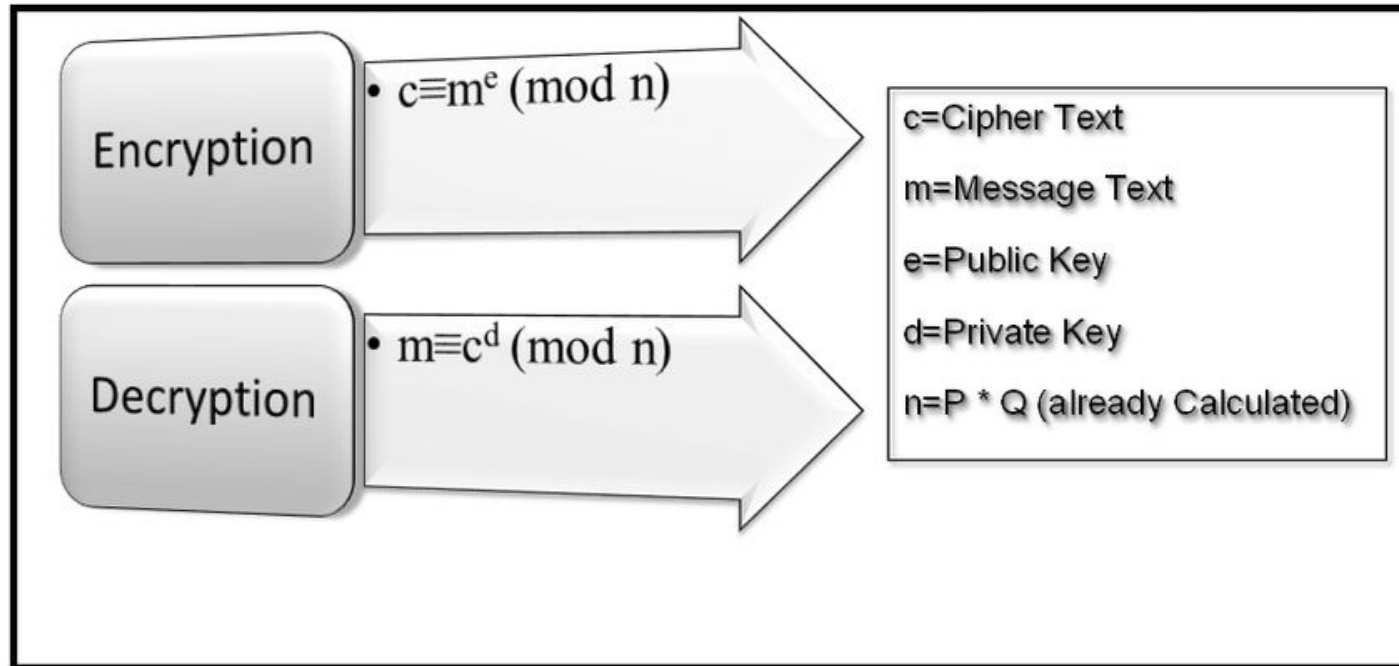
Решаем сравнение:

$$x^3 + ax + b - c \equiv 0 \pmod{p}$$

```
PR.<x> = PolynomialRing(GF(p))
f = x^3 + a * x + b - enc
rts = f.roots()
print(rts)
```

```
for root in rts:
    flag = root[0]
    print(long_to_bytes(flag))
```

# Algebraic, pow



# One line crypto



```
from Crypto.Util.number import *
from secret import m, n, x, y, flag

p, q = x**(m+1) - (x+1)**m, y**(n+1) - (y+1)**n
assert isPrime(p) and isPrime(q) and p < q < p << 3 and len(bin(p*q)[2:]) == 2048
enc = bytes_to_long(flag)
print(pow(enc, 0x10001, p*q))
```

$$p = x^{m+1} - (x+1)^m$$
$$q = y^{m+1} - (y+1)^m$$

# One line crypto



```
from Crypto.Util.number import *
from secret import m, n, x, y, flag

p, q = x**(m+1) - (x+1)**m, y**(n+1) - (y+1)**n
assert isPrime(p) and isPrime(q) and p < q < p << 3 and len(bin(p*q)[2:]) == 2048
enc = bytes_to_long(flag)
print(pow(enc, 0x10001, p*q))
```

$$p = x^{m+1} - (x+1)^m$$
$$q = y^{n+1} - (y+1)^n$$

```
from Crypto.Util.number import *
from gmpy2 import invert

res = 1460847413295235232889708071732546430843832262331984742844793394320

def f(p, i):
    return pow(p, i + 1) - pow(p + 1, i)

d13 = pow(2, 1028)
d7 = pow(2, 1021)
primes = []
for i in range(1000):
    for j in range(-1000, 1000):
        fn = f(j, i)
        if fn < d13 and fn > d7 and isPrime(fn):
            print(fn, i, j)
            primes.append(fn)

for i in range(len(primes)):
    for j in range(i, len(primes)):
        n = primes[i]*primes[j]
        if len(bin(n)[2:]) == 2048:
            phi = (primes[i]-1)*(primes[j]-1)
            d = invert(0x10001, phi)
            m = long to bytes(pow(res, d, n))
            if b'CCTF{" in m:
                print(m)

# b'CCTF{0N3_1!nE_CrYp70_iN_2020}'
```

# Three raven



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

def keygen(nbit):
    while True:
        p, q, r = [getPrime(nbit) for _ in range(3)]
        if isPrime(p + q + r):
            pubkey = (p * q * r, p + q + r)
            privkey = (p, q, r)
            return pubkey, privkey

def encrypt(msg, pubkey):
    enc = pow(bytes_to_long(msg.encode('utf-8')), 0x10001, pubkey[0] * pubkey[1])
    return enc

nbit = 512
pubkey, _ = keygen(nbit)
print('pubkey =', pubkey)

enc = encrypt(flag, pubkey)
print('enc =', enc)
```



$p, q, r$  – простые  
(секретный ключ, не  
известны)

$s = p + q + r$  - простое

$n = p * q * r * s$

$s, n / s$  – открытый ключ,  
известны

$c = m^e \% n$

$m = ?$

# Three ravens



Короткое сообщение (< длины любого из множителей  $n$ )  
Известен один из множителей  $n$

$$c \equiv m^e \pmod{p \cdot q \cdot r \cdot s}$$

$$c_{\_} \equiv m^e = k * (p \cdot q \cdot r \cdot s) + c$$

$$m^e \equiv c \pmod{s}$$

$$d \equiv e^{-1} \pmod{s}$$

$$c^d \equiv m^{ed} \equiv m \pmod{s}$$

```
def keygen(nbit):
    while True:
        p, q, r = [getPrime(nbit) for _ in range(3)]
        if isPrime(p + q + r):
            pubkey = (p * q * r, p + q + r)
            privkey = (p, q, r)
            return pubkey, privkey

def encrypt(msg, pubkey):
    enc = pow(bytes_to_long(msg.encode('utf-8')), 0x10001, pubkey[0] * pubkey[1])
    return enc

nbit = 512
pubkey, _ = keygen(nbit)
print('pubkey =', pubkey)

pubkey = (11180735511505417603835067658683342890958492172073834287759921283748260379
enc = 821805228222601189722970390776352121405425478527551188647686132806711749218379

def decrypt(mess, pubkey):
    d = invert(0x10001, pubkey[1] - 1)
    m = pow(mess, d, pubkey[1])
    return long_to_bytes(m)

print(decrypt(enc, pubkey))
# b'CTF{tH3_thr3E_r4V3n5_ThRe3_cR0w5}'
```



# Model

nc 04.cr.yp.toc.tf 8001

```
def genkey(nbit):  
    while True:  
        p, q = getPrime(nbit), getPrime(nbit)  
        if gcd((p-1) // 2, (q-1) // 2) == 1:  
            P, Q = (q-1) // 2, (p-1) // 2  
            r = inverse(Q, P)  
            e = 2 * r * Q - 1  
            return(p, q, e)  
  
def encrypt(msg, pubkey):  
    e, n = pubkey  
    return pow(bytes_to_long(msg), e, n)
```

$p, q$  – простые

$P = (q-1) // 2$

$Q = (p-1) // 2$

$r \equiv Q^{-1} \% P$

$e = 2 * r * Q - 1$

$c \equiv m^e \% p*q$





# Model

nc 04.cr.yr.toc.tf 8001

```
def genkey(nbit):  
    while True:  
        p, q = getPrime(nbit), getPrime(nbit)  
        if gcd((p-1) // 2, (q-1) // 2) == 1:  
            P, Q = (q-1) // 2, (p-1) // 2  
            r = inverse(Q, P)  
            e = 2 * r * Q - 1  
            return(p, q, e)  
  
def encrypt(msg, pubkey):  
    e, n = pubkey  
    return pow(bytes_to_long(msg), e, n)
```

$p, q$  – простые  
 $P = (q-1) // 2$   
 $Q = (p-1) // 2$   
 $r \equiv Q^{-1} \% P$   
 $e = 2 * r * Q - 1$

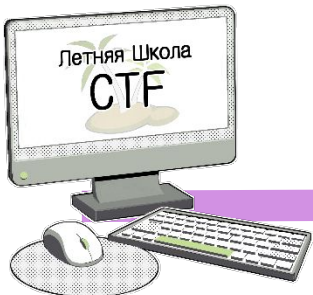
$c \equiv m^e \% p*q$

$$e = 2 * Q^{-1} * Q - 1 = 1 \% P$$

$$1) e \equiv 1 \% (q-1)$$
$$c \equiv m \% q$$

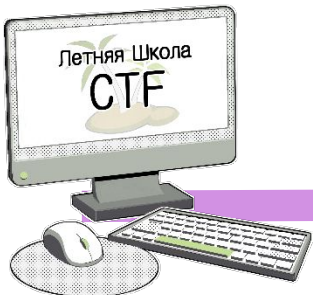
$$1) e \equiv 1 + P \% (q-1)$$
$$c = m^{1+(q-1)/2} \% q$$
$$c = m * 1^{1/2} \% q$$
$$c = \pm m \% q$$

$$q = \gcd(c \pm m, n)$$



# Butterfly effect

```
387 def hq_prng(x, p, g):
388     rng = 0
389     for _ in range(getRandomNBitInteger(14)):
390         x = pow(g, x, p)
391     for i in range(nbit):
392         x = pow(g, x, p)
393         if x < (p-1) // 2:
394             rng += 2**i - 1
395         elif x > (p-1) // 2:
396             rng -= 2**i + 1
397         else:
398             rng ^= 2**(i + 1)
399     if rng <= 0:
400         return -rng
401     return rng
402
403 def keygen(p, g):
404     r, s = hq_prng(getRandomNBitInteger(nbit), p, g), hq_prng(getRandomNBitInteger(nbit), p, g)
405     u, v = gmpy2.next_prime(r**2 + s**2), gmpy2.next_prime(2*r*s)
406     e, n = 0x10001, u * v
407     return e, n
408
409 def encrypt(msg, e, n):
410     return pow(bytes_to_long(msg.encode('utf-8')), e, n)
411
412 #encrypt(flag, e, n) = 117667582947026307482709850318214820165964980495414423711608614681075036546959
413
414 #(p, g, n) = (68396932999729141946282927360590169590631231980913314894620521363257317833167L, 1114840
415
```



# Butterfly effect

```
387 def hq_prng(x, p, g):
388     rng = 0
389     for _ in range(getRandomNBitInteger(14)):
390         x = pow(g, x, p)
391     for i in range(nbit):
392         x = pow(g, x, p)
393         if x < (p-1) // 2:
394             rng += 2**i - 1
395         elif x > (p-1) // 2:
396             rng -= 2**i + 1
397         else:
398             rng ^= 2**(i + 1)
399     if rng <= 0:
400         return -rng
401     return rng
402
403 def keygen(p, g):
404     r, s = hq_prng(getRandomNBitInteger(nbit), p, g), hq_prng(getRandomNBitInteger(nbit), p, g)
405     u, v = gmpy2.next_prime(r*2 + s*2), gmpy2.next_prime(2*r*s)
406     e, n = 0x10001, u * v
407     return e, n
408
409 def encrypt(msg, e, n):
410     return pow(bytes_to_long(msg.encode('utf-8')), e, n)
411
412 #encrypt(flag, e, n) = 117667582947026307482709850318214820165964980495414423711608614681075036546959
413
414 #(p, g, n) = (68396932999729141946282927360590169590631231980913314894620521363257317833167L, 1114840
415
```



Type some Sage code below and press Evaluate.

```
1 Zp = Zmod(68396932999729141946282927360590169590631231980913314894620521363257317833167)
2 a = Zp(11148408907716636563689390048104567047599159688378384611325239859308138541650)
3 print(a.multiplicative_order())
4
```

Evaluate

31337



# Butterfly effect

```
387 def hq_prng(x, p, g):
388     rng = 0
389     for _ in range(getRandomNBitInteger(14)):
390         x = pow(g, x, p)
391     for i in range(nbit):
392         x = pow(g, x, p)
393         if x < (p-1) // 2:
394             rng += 2**i - 1
395         elif x > (p-1) // 2:
396             rng -= 2**i + 1
397         else:
398             rng ^= 2**(i + 1)
399     if rng <= 0:
400         return -rng
401     return rng
402
403 def keygen(p, g):
404     r, s = hq_prng(getRandomNBitInteger(nbit), p, g)
405     u, v = gmpy2.next_prime(r**2 + s**2), gmpy2.next_prime(r**2 + s**2)
406     e, n = 0x10001, u * v
407     return e, n
408
409 def encrypt(msg, e, n):
410     return pow(bytes_to_long(msg.encode('utf-8')), e, n)
411
412 #encrypt(flag, e, n) = 117667582947026307482709850318214
413
414 #(p, g, n) = (68396932999729141946282927360590169590631231
415             11148408907716636563689390048104567047599159
416             17442100312381003338179023622183785651571732
```

```
import gmpy2
from Crypto.Util.number import bytes_to_long, getRandomNBitInteger

nbit = 256
def hq_prng(x, p, g):
    rng = 0
    global gs
    for i in range(nbit):
        x = gs[x % 31337]
        if x < (p-1) // 2:
            rng += 2**i - 1
        elif x > (p-1) // 2:
            rng -= 2**i + 1
        else:
            rng ^= 2**(i + 1)
    if rng <= 0:
        return -rng
    return rng

def keygen(p, g):
    r, s = hq_prng(getRandomNBitInteger(nbit), p, g), hq_prng(getRandomNBitInteger(nbit), p, g)
    u, v = gmpy2.next_prime(r**2 + s**2), gmpy2.next_prime(r**2 + s**2)
    e, n = 0x10001, u * v
    return e, n

def encrypt(msg, e, n):
    return pow(bytes_to_long(msg.encode('utf-8')), e, n)

#encrypt(flag, e, n) = 117667582947026307482709850318214

(p, g, n) = (68396932999729141946282927360590169590631231
            11148408907716636563689390048104567047599159
            17442100312381003338179023622183785651571732

print(gmpy2.is_prime(p))

#sage:
```

```
ord = 31337
gs = [0] * 31337
prns = [0] * 31337
curr = 1
for i in range(31337):
    gs[i] = curr
    curr = curr * g % p
print("done1")
for i in range(0, 31337):
    prns[i] = hq_prng(i, p, g)
print("done2")
prns.sort()

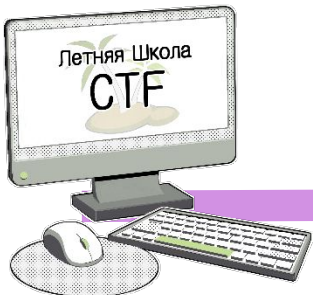
for i in range(0, 31337):
    print(i)
    lef = i+1
    rig = 31336
    mid, best = 0, 0
    while lef <= rig:
        mid = (lef + rig) // 2
        if (prns[i]*prns[i] + prns[mid] * prns[mid]) * 2 * prns[i] * prns[mid] >= n:
            best = mid
            rig = mid - 1
        else:
            lef = mid + 1

if best == 0:
    continue

for j in range(best-1, min(best+30, 31337)):
    u = prns[i] * prns[i] + prns[j] * prns[j]
    v = 2 * prns[i] * prns[j]

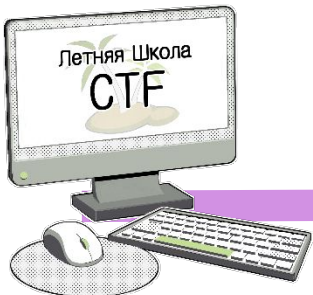
    u = gmpy2.next_prime(u)
    v = gmpy2.next_prime(v)
    if u * v == n:
        print(u, v)
        exit()
exit()
```

<https://blog.cryptohack.org/cryptoctf2020>



# Decent RSA

```
-----BEGIN PUBLIC KEY-----  
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA/Ug8rIEPci1UXMsT  
+UDo  
y8DfxbTHX/3BK2oU+FPWiJf+EiUBM2x4ep04qZ1SO9Pmqj/WH9skMrF1J/LX  
uY3I  
fjvJCh0DXa9VUyX2dAJidja9Ior7GpFwwjYdKh+OETNV+2/CcX4RiPvj+8Apm  
edW  
gn4Fxaeivki+f/UwDa+ws1fTUzmI325v8yvcryHhbgeUWiF85EP6HFAavTsVPI  
xb  
LikVMAB1fuzDbqqJvW2u138w6b2FH3WrezYF6tbAyZej2HX46phwDm9C7MX  
YJ/sU  
oS+E8P7S1jMTCWjfwMCOKU3SFGrkWtXuTaoMZ2nZ+HVfJV8xJOjWez1OxQ  
5P3F1w  
GQIDAQAB  
-----END PUBLIC KEY-----
```



# Decent RSA

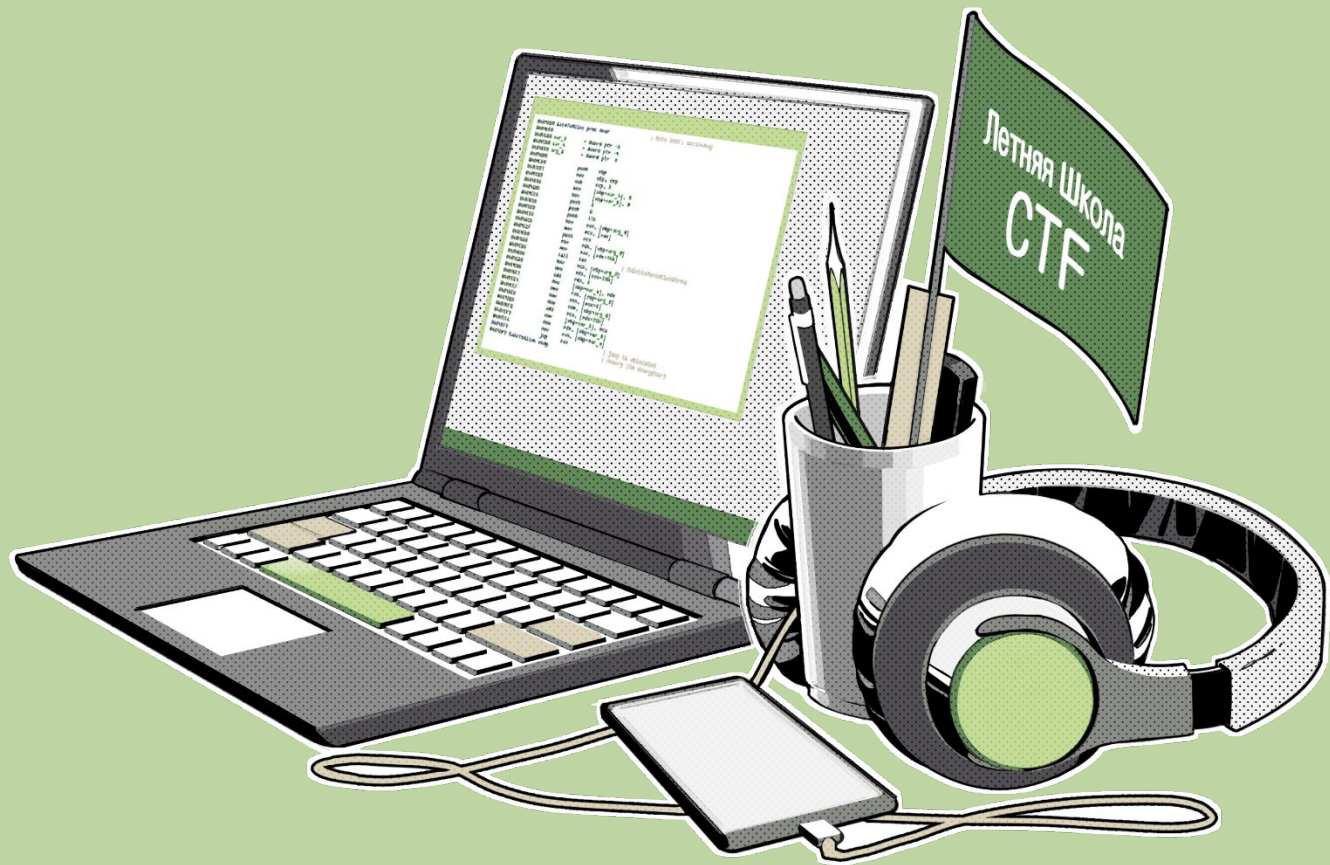
```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA/Ug8rIEPci1UXMST
+UDo
y8DfxbTHX/3BK2oU+FPWiJf+EiUBM2x4ep04qZ1SO9Pmqj/WH9skMrF1J/LX
uY3I
fjvJCh0DXa9VUyX2dAJidja9Ior7GpFwwjYdKh+OETNV+2/CcX4RiPvj+8Apm
edW
gn4Fxaeivki+f/UwDa+ws1fTUzmI325v8yvcrYHhbgeUWiF85EP6HFAavTsVPI
xb
LikVMAB1fuzDbqqJvW2u138w6b2FH3WrezYF6tbAyZej2HX46phwDm9C7MX
YJ/sU
oS+E8P7S1jMTCWifwMCOKU3SFGGrkWtXuTaoMZ2nZ+HVfJV8xJOjWez1OxQ
5P3F1w
GQIDAQAB
```

```
-----END PUBLIC KEY-----
flag = bytes_to_long(open("flag.enc", "rb").read())
key = RSA.import_key(open("mykey.pem").read())
n = Integer(key.n)

poly = sum(e * xi for i,e in enumerate(Integer(key.n).digits(11)))
(p, _), (q, _) = poly.factor_list()
p, q = p(x=11), q(x=11)
assert p*q == n

d = inverse_mod(key.e, (p-1)*(q-1))
print(long_to_bytes(pow(flag, d, n)))
```

Official writeups



# ЛЕТНЯЯ ШКОЛА CTF 2021



Слонкина Ирина  
Сергеевна



[otrada.nsk@gmail.com](mailto:otrada.nsk@gmail.com)



8-977-872-43-31

