ЛЕТНЯЯ ШКОЛА
CTF 2021

Разбор задач CryptoCTF 2020

http://10.10.10.19:8080/s/eQK4y9924FlNXgR

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, algbraic	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/

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 ≡ b % m). **Класс вычетов** – множество всех чисел, сравнимых с a по модулю m.

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

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

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

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

Encoding

Летняя Школа CTF

Abbot

```python
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)
```

```python
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

```python
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)
```

```python
def us_(num, den):

    text_ = ""
    resultDen, resultNum = den, num
    r = int(resultNum / resultDen)
    text_ += chr(r)
    result = frac(resultNum, resultDen) - r
    resultDen, resultNum = result.numerator, result.denominator
    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_
```

```python
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


```python
def comb(n, k):
    if k > n :
        return 0
    k = min(k, n - k)
    u = reduce(operator.mul, range(n, n - k, -1), 1)
    d = reduce(operator.mul, range(1, k + 1), 1)
    return u // d

def encrypt(msg, n, k):
    msg = bytes_to_long(msg.encode('utf-8'))
    print("msg", msg)
    if msg >= comb(n, k):
        return -1
    m = ['1'] + ['0' for i in range(n - 1)]
    for i in range(1, n + 1):
        if msg >= comb(n - i, k):
            m[i-1]= '1'
            msg -= comb(n - i, k)
            print("sub", comb(n - i, k), n, i, k)
            k -= 1
    m = int(''.join(m), 2)

    print("encryption_res_1", bin(m))
    i, z = 0, [0 for i in range(n - 1)]
    c = 0
    while (m > 0):
        if m % 4 == 1:
            c += 3 ** i
            m -= 1
        elif m % 4 == 3:
            c += 2 * 3 ** i
            m += 1
        m //= 2
        i += 1
    print("enc_res_2", c)
    return c

#enc = encrypt(flag, n, k)
#print('enc =', enc)

enc = 5550332817876280162274999855997378479609235817133438293571677699650886802
```

Amsterdam

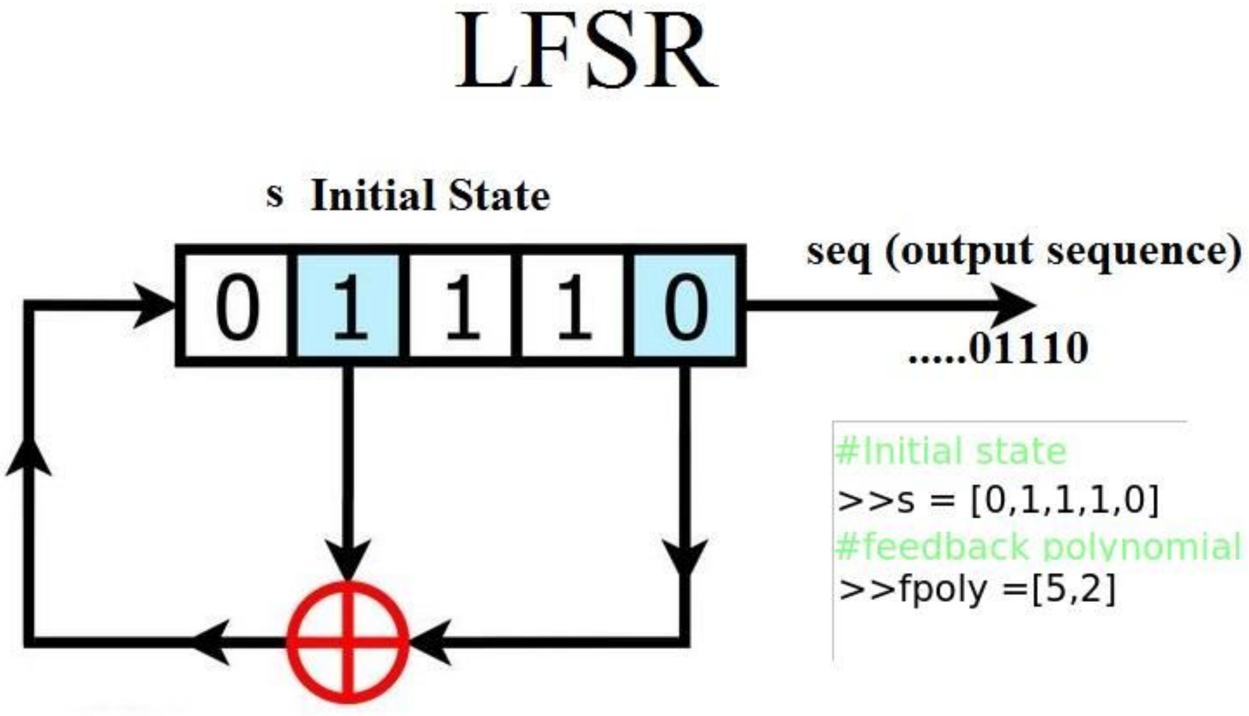

```python
def comb(n, k):
    if k > n :
        return 0
    k = min(k, n - k)
    u = reduce(operator.mul, range(n, n - k, -1), 1)
    d = reduce(operator.mul, range(1, k + 1), 1)
    return u // d

def encrypt(msg, n, k):
    msg = bytes_to_long(msg.encode('utf-8'))
    print("msg", msg)
    if msg >= comb(n, k):
        return -1
    m = ['1'] + ['0' for i in range(n - 1)]
    for i in range(1, n + 1):
        if msg >= comb(n - i, k):
            m[i-1]= '1'
            msg -= comb(n - i, k)
            print("sub", comb(n - i, k), n, i, k)
            k -= 1
    m = int(''.join(m), 2)

    print("encryption res 1", bin(m))
    i, z = 0, [0 for i in range(n - 1)]
    c = 0
    while (m > 0):
        if m % 4 == 1:
            c += 3 ** i
            m -= 1
        elif m % 4 == 3:
            c += 2 * 3 ** i
            m += 1
        m //= 2
        i += 1
    print("enc_res_2", c)
    return c

#enc = encrypt(flag, n, k)
#print('enc =', enc)

enc = 555033281787628016227499985599737847960923581713343829357167769965088680
```

```python
def dec_(enc):
    arr = []
    i = 0
    while enc > 0:
        m = enc % 3
        arr.append(m)
        enc //= 3
        i += 1
    c = 0

    for m in arr[::-1]:
        c *= 2
        if m == 1:
            c += 1
        elif m == 2:
            c -= 1
    return c
```

```python
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
```

```python
def decrypt(msg, n, k):
    pt = 0

    for i in range(2, n + 1):
        if msg[i-1] == '1':
            pt += comb(n - i, k)
            k -= 1

    return long_to_bytes(pt)


msg_ = dec_(enc)
print(msg_)
msg_ = bin(msg_)[2:]
n = len(msg_)

for k in range(n):
    pt = decrypt(msg_, n, k)
    if b'CCTF{' in pt:
        print(k, pt)
        break

# b'..:: CCTF{With_Re3p3ct_for_Sch4lkwijk_dec3nt_Encoding!} ::..'
```

Shift registers




```
#Initial state
>>s = [0,1,1,1,0]
#feedback polynomial
>>fpoly =[5,2]
```

Heaven

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

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

def born_to_die(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


david = len(seventh_seal)
elf = seventh_seal
lord = BitArray(bytes=bytes(open('flag.jpg', 'rb').read())).bin
bilbo = len(lord)
matthew = 0
princess_leia = ''
destiny = bilbo // david
apocalypse = bilbo % david
for i in range(32):
    elf = born_to_die(elf)
while matthew < destiny:
    princess_leia += matthew_effect(elf, lord[matthew * david : (matthew + 1) * david])
    elf = born_to_die(elf)
    matthew += 1
princess_leia += matthew_effect(elf[:apocalypse], lord[matthew * david :])
res = open('flag.enc', 'wb')
res.write(bytes(int(princess_leia[i : i + 8], 2) for i in range(0, bilbo, 8)))
```

Летняя Школа CTF

Heaven

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

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

def born_to_die(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


david = len(seventh_seal)
elf = seventh_seal
lord = BitArray(bytes=bytes(open('flag.jpg', 'rb').read())).bin
bilbo = len(lord)
matthew = 0
princess_leia = ''
destiny = bilbo // david
apocalypse = bilbo % david
for i in range(32):
    elf = born_to_die(elf)
while matthew < destiny:
    princess_leia += matthew_effect(elf, lord[matthew * david_: (matthew + 1) * david])
    elf = born_to_die(elf)
    matthew += 1
princess_leia += matthew_effect(elf[:apocalypse], lord[matthew * david_:])
res = open('flag.enc', 'wb')
res.write(bytes(int(princess_leia[i_: i + 8], 2) for i in range(0, bilbo, 8)))
```

```python
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

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	.H..яЫ.C........
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.C...
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	....

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

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

stream = "11000111001010110000110001110010"

keys = [
    "1100011100101011000"
    "0110001110010"
]
```

```python
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

Летняя Школа CTF

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 • `00` : No units; width:height pixel aspect ratio = Ydensity:Xdensity • `01` : Pixels per inch (2.54 cm) • `02` : Pixels per centimeter

https://asecuritysite.com/information/jpeg?file=new_logo4.jpg

Входящие (24) - otra... Sci-Hub: устраняя пре... CryptoCTF CTFtime.org / otrada WiFi传书 084.pdf () - 393.pdf CPA11.pdf MEPhI (Cryptology & ...

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

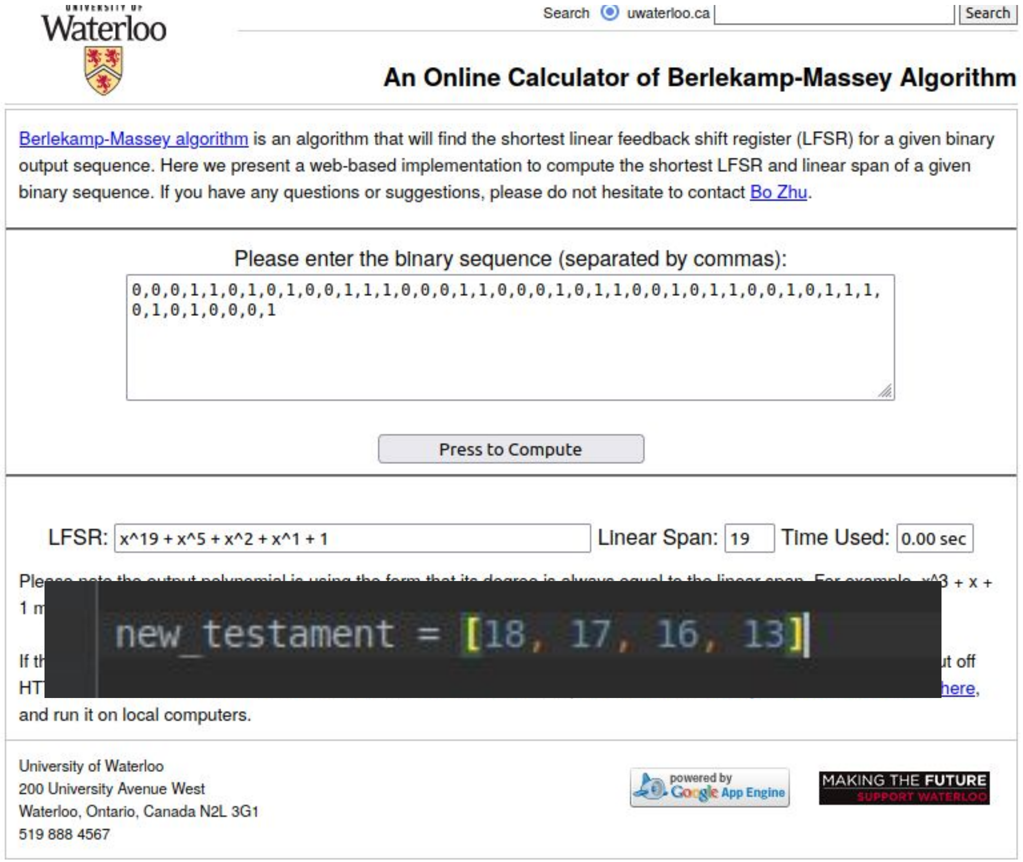
```
[00000000] FF D8 FF E0 00 10 4A 46 49 46 00 01 01 01 00 48   ......JFIF.....H
[00000016] 00 48 00 00 FF DB 00 43 00 01 01 01 01 01 01 01   .H.....C........
[00000032] 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01   ................
[00000048] 01 01 01 01 01 02 01 01 01 01 01 01 02 02 02 02   ................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

```python
key = "11000111001010110000"
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 = "FFD8FFE000104a4649460001"

#111100000000011111100100010000111110100110100110101110010000110011111
#11000111001010110000_11000111001010101100__11000111001010110__1100011
#11111111101100011l1_11111110000000000000__00010000010010100__0110010
#f   f   d   8   _   f   e   0   0   _   1   0   4   a   _   6   4

#111100000000011111100100010000111110100110100110101110010000110011111
#110001110010101100001100011100101011000011000111001010110000110001111
#111111111011000111111111110000000000000000001000001001010010001100100
#f   f   d   8   f   f   e   0   0   0   1   0   4   a   4   6   4
```

Heaven

2)

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

ct = BitArray(bytes=bytes(open('flag.enc', 'rb').read())).bin
msg_len = len(ct)
ctr = 0
pt = ''
add = "0001011001011001011101010001" + "_" * 100000
#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"

print(ct[400:])
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[400:])
print(pt[400:])
print(tobyte(pt)[400:])

k = "FFD8FFE000104a4649460001"
```




```
CCTF{0Ne_k3y_t0_rU1e_7hem_A11_
     4Nd_7o_d3crYp7_th3_fl4g!}
```

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


```python
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


```python
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 = [636730424634282684150787505024636846878192530834301373045541
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


```python
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 = [636730424634282684150787505024636846878192530834301373045411
enc = 154657917005376465967753276253676484467260782425419406781078357
```

```python
def LLL_for_knapsack_problem(vector, w):
    K = 100000000
    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 = [6367304246342826841507875050246368468781925308343013730454179 41, 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


```python
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 … d]

```python
    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 76
cipher = [-3459749918754130611, -3459749918754138177, -3459749918754137803, -3459749918754138385,
          -3459749918754138025, -3459749918754138097, -3459749918754138073, -3459749918754138245,
          -3459749918754138183, -3459749918754138445, -3459749918754137991, -3459749918754138597,
```

Mad hat


```python
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:
```

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

```python
    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 76
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

```python
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:

$p_1 = 37$

$p_2 = 75$

```python
    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 76
cipher = [-3459749918754130611, -3459749918754138177, -3459749918754137803, -3459749918754138385,
          -3459749918754138025, -3459749918754138097, -3459749918754138073, -3459749918754138245,
          -3459749918754138183, -3459749918754138445, -3459749918754137991, -3459749918754138597,
```

Mad hat


```python
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_&_bri11iant_Paley!}}}}}}}}}}}}}}}}}}}}}
```

```python
import numpy as np
R = IntegerModRing(257)

p1 = 37
p2 = 75

matrices = [Matrix(R, keygen(p1, 1)[0]), Matrix(R, keygen(p2, 1)[0]),
            Matrix(R, keygen(p1, 2)[0]), Matrix(R, keygen(p2, 2)[0])]
inv_matrices = [a.inverse() for a in matrices]
d0 = -3459749918754130000


for m in inv_matrices:
    for i in range(257):
        cipher_ = Matrix(R, [cipher]) + Matrix(R, [[-d0 + i] * 76])
        cipher_ = cipher_ * m
        flag = ""
        b = True
        for c in range(76):
            c_ = cipher_[0, c]
            if c_ < 0 or c_ >= 256:
                b = False
                break
            c_ = c_
            flag += chr(c_)
        if b and "CCTF{" in flag:
            print(flag)
            #break

```

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()
```

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

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

sage: L = B(L)

sage: L.parent()
```

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

Летняя Школа CTF

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
```

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

p, a, b – не известны

$f(x) \equiv x^3 + ax + b \ \% \ p$

f(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
```

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

p, a, b – не известны

$f(x) \equiv x^3 + ax + b \% p$

$f(flag) = c$

Мы можем получить f(x) для любого x

$f(0) \equiv b \ \% p$ – получаем b

$f(1) \equiv 1 + a + b \% p$ – получаем a

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

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

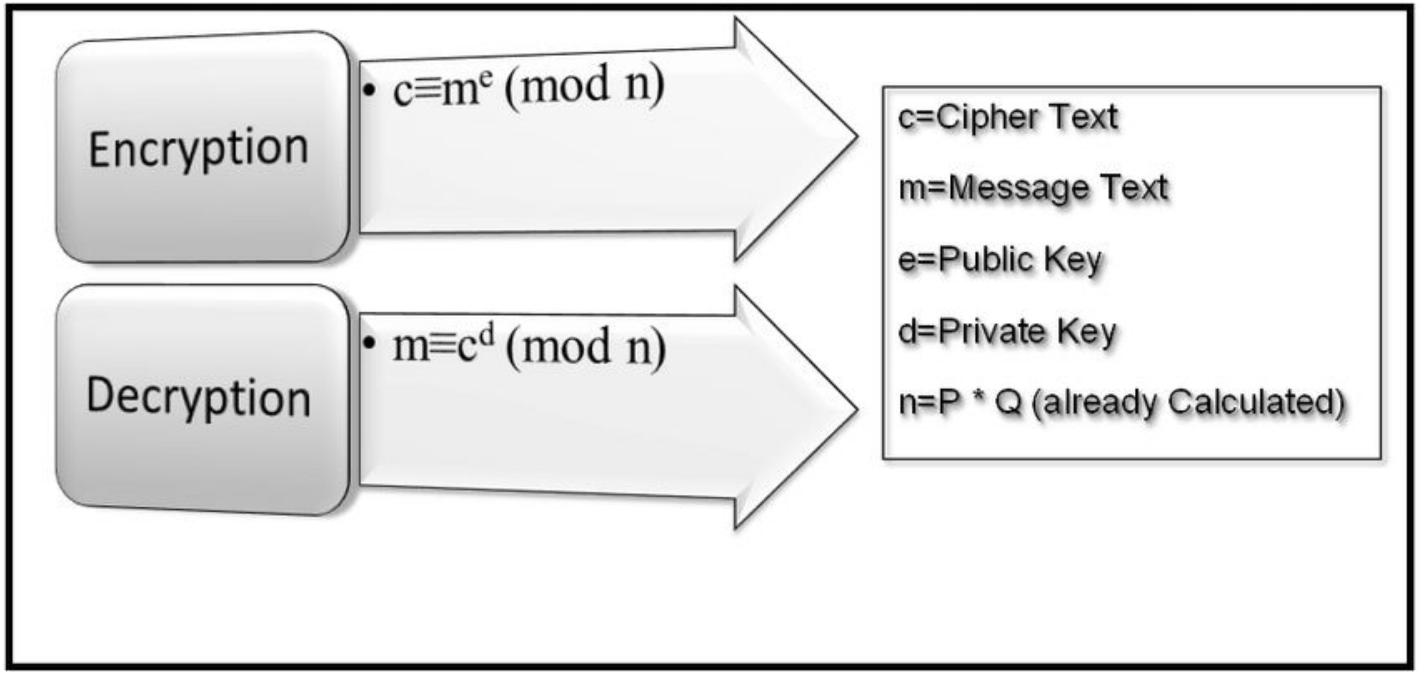
НОД $(c_1 – f(x_1), c_2 – f(x_2))$ делится на p

Решаем сравнение:

$x^3 + ax + b – c \equiv 0 \% p$

```python
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


```python
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


```python
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))
```

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

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

res = 146084741329523523288970807173254643084383226233198474284479339432

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


```python
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

$c \equiv m^e \ \% \ p*q*r*s$
$c\_ = m^e = k * (p*q*r*s) + c$

$m^e \equiv c \ \% \ s$

$d \equiv e^{-1} \ \% \ s$
$c^d \equiv m^{ed} \equiv m \ \% \ s$

```python
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'CCTF{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.yp.toc.tf 8001`

```python
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

```python
def hq_prng(x, p, g):
    rng = 0
    for _ in range(getRandomNBitInteger(14)):
        x = pow(g, x, p)
    for i in range(nbit):
        x = pow(g, x, p)
        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(2*r*s)
    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) = 117667582947026307482709850318214820165964980495414423711608614681075036546959

#(p, g, n) = (68396932999729141946282927360590169590631231980913314894620521363257317833167L, 1114840
```

Butterfly effect

```python
def hq_prng(x, p, g):
    rng = 0
    for _ in range(getRandomNBitInteger(14)):
        x = pow(g, x, p)
    for i in range(nbit):
        x = pow(g, x, p)
        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(2*r*s)
    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) = 117667582947026307482709850318214820165964980495414423711608614681075036546959

#(p, g, n) = (68396932999729141946282927360590169590631231980913314894620521363257317833167L, 1114840
```

Type some Sage code below and press Evaluate.

```
Zp = Zmod(68396932999729141946282927360590169590631231980913314894620521363257317833167)
a = Zp(11148408907716636563689390048104567047599159688378384611325239859308138541650)
print(a.multiplicative_order())
```

Evaluate

```
31337
```

Butterfly effect

```python
def hq_prng(x, p, g):
    rng = 0
    for _ in range(getRandomNBitInteger(14)):
        x = pow(g, x, p)
    for i in range(nbit):
        x = pow(g, x, p)
        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),
    u, v = gmpy2.next_prime(r**2 + s**2), gmpy2.next_
    e, n = 0x10001, u * v
    return e, n

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

#encrypt(flag, e, n) = 11766758294702630748270985031８

#(p, g, n) = (6839693299972914194628292736059016959O6
```

```python
import gmpy2
from Crypto.Util.number import bytes_to_long, getRandomNB

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_
    u, v = gmpy2.next_prime(r**2 + s**2), gmpy2.next_prim
    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:
```

```python
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()
```

https://blog.cryptohack.org/cryptoctf2020

Decent RSA

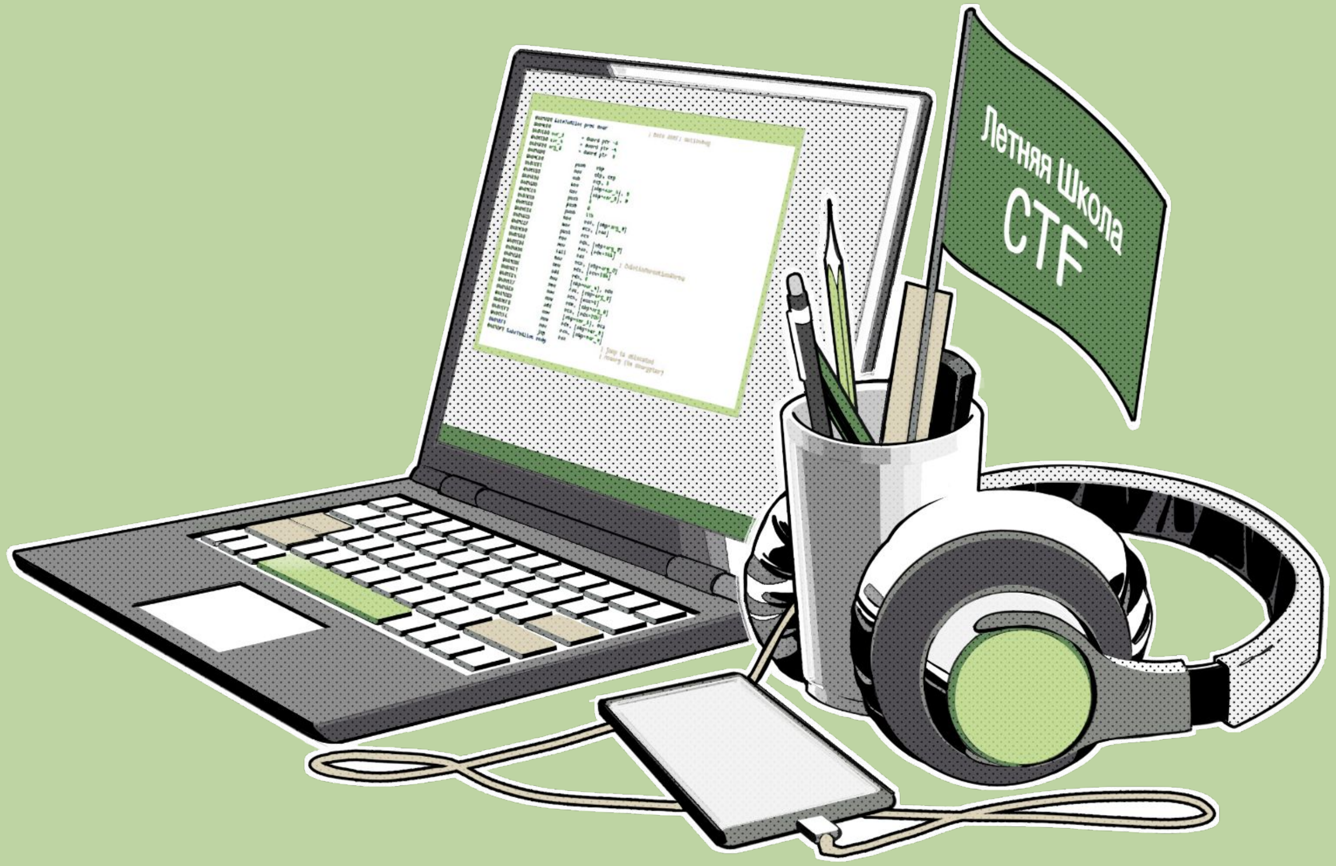

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA/Ug8rlEPci1UXMsT
+UDo
y8DfxbTHX/3BK2oU+FPWiJf+EiUBM2x4ep04qZ1SO9Pmqj/WH9skMrF1J/LX
uY3l
fjvJCh0DXa9VUyX2dAJidja9Ior7GpFwwjYdKh+OETNV+2/CcX4RiPvj+8Apm
edW
gn4Fxaeivki+f/UwDa+ws1fTUzmI325v8yvcryHhbgeUWiF85EP6HFAavTsVPl
xb
LikVMAB1fuzDbqqJvW2u138w6b2FH3WrezYF6tbAyZej2HX46phwDm9C7MX
YJ/sU
oS+E8P7S1jMTCWjfwMCOKU3SFGrkWtXuTaoMZ2nZ+HVfJV8xJOjWez1OxQ
5P3F1w
GQIDAQAB
-----END PUBLIC KEY-----
```

Decent RSA

-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA/Ug8rlEPci1UXMsT
+UDo
y8DfxbTHX/3BK2oU+FPWiJf+EiUBM2x4ep04qZ1SO9Pmqj/WH9skMrF1J/LX
uY3l
fjvJCh0DXa9VUyX2dAJidja9Ior7GpFwwjYdKh+OETNV+2/CcX4RiPvj+8Apm
edW
gn4Fxaeivki+f/UwDa+ws1fTUzmI325v8yvcryHhbgeUWiF85EP6HFAavTsVPl
xb
LikVMAB1fuzDbqqJvW2u138w6b2FH3WrezYF6tbAyZej2HX46phwDm9C7MX
YJ/sU
oS+E8P7S1jMTCWjfwMCOKU3SFGrkWtXuTaoMZ2nZ+HVfJV8xJOjWez1OxQ
5P3F1w
GQIDAQAB
-----END PUBLIC KEY-----


```python
from Crypto.PublicKey import RSA
from Crypto.Util.number import long_to_bytes, bytes_to_long

flag = bytes_to_long(open("flag.enc","rb").read())
key = RSA.import_key(open("mykey.pem").read())
n = Integer(key.n)

poly = sum(e * x^i 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

8-977-872-43-31