这次re至少能看懂

RE

again!

打开后发现一个py打包的exe和bin2.pyc

反编译一下第一个文件,说是要用这个文件去解密bin2,但是看不懂QAQ

但是二进制形式打开bin2，观察可以发现出现大量一样的

a405b5d321e446459d8f9169d027bd92

怀疑是异或，所以

```python
import struct

def xor_with_key(input_file, key, output_file):

    with open(input_file, 'rb') as fin, open(output_file, 'wb') as fout:
        while True:
            chunk = fin.read(32)
            if len(chunk) == 0:
                break
            data = bytearray(len(chunk))
            for i in range(len(chunk)):
                data[i] = chunk[i] ^ ord(key[i % len(key)])
            fout.write(data)

# 使用方法
key = "a405b5d321e446459d8f9169d027bd92"
xor_with_key('bin2', key, 'output.bin')
```

然后得到的文件再二进制打开一下，是winPE，就ida打开

主函数:

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  __int64 v3; // rax
  int key[6]; // [rsp+20h] [rbp-18h] BYREF

  sub_140001020("plz input your flag:");
  sub_140001080("%32s",flag);
  key[0] = 0x1234;
  key[1] = 0x2341;
  key[2] = 0x3412;
  key[3] = 0x4123;
  encode(key);
  i = 0i64;
  while ( flag[i] == *(_DWORD *)((char *)&check + i * 4) )
  {
    if ( ++v3 >= 8 )
    {
      sub_140001020("Congratulations!");
```

```
      return 0;
    }
  }
  sub_140001020("Wrong!try again...");
  return 0;
}
```

看起来就是flag以dword来加密然后比较,encode里面是一个TEA变体，直接逆就好了

```
#include<iostream>

using namespace std;

int main(void) {

    unsigned int s6;
    unsigned int delta;
    unsigned int s5;
    unsigned int s4; // r15d
    unsigned int s3; // r14d
    unsigned int s2; // ebp
    unsigned int s1; // esi
    unsigned int s0; // r11d
    unsigned int v11; // ebx
    unsigned int s7; // [rsp+40h] [rbp+8h]

    unsigned int flag[8] = { 0x506FB5C3, 0xB9358F45, 0xC91AE8C7, 0x3820E280, 0xD13ABA83, 0x975CF554, 0x4352036B, 0x1CD20447 };
    unsigned int key[4] = {
        0x00001234, 0x00002341, 0x00003412, 0x00004123
    };
    unsigned int key_[4] = {
        0x00001234, 0x00002341, 0x00003412, 0x00004123
    };

    s7 = flag[7];
    s1 = flag[1];
    s2 = flag[2];
    s3 = flag[3];
    s4 = flag[4];
    s5 = flag[5];
    s6 = flag[6];
    s0 = flag[0];
    delta = 0;
    delta += 0x7937B99E * 12;
    for (int i = 0; i < 12; i++) {
        v11 = key[(delta >> 2) & 3];

        s7 -= (((s6 ^ key[(delta >> 2) & 3 ^ 3]) + (delta ^ flag[0])) ^ (((16 * s6) ^ (flag[0] >> 3)) + ((s6 >> 5) ^ (4 * flag[0]))));
        s6 -= ((s5 ^ key[(delta >> 2) & 3 ^ 2]) + (delta ^ s7)) ^ (((16 * s5) ^ (s7 >> 3)) + ((s5 >> 5) ^ (4 * s7)));
        s5 -= ((delta ^ s6) + (s4 ^ key[(delta >> 2) & 3 ^ 1])) ^ (((16 * s4) ^ (s6 >> 3)) + ((s4 >> 5) ^ (4 * s6)));
        //flag[0] = s0;
```

```
        s4 -= ((delta ^ s5) + (s3 ^ v11)) ^ (((16 * s3) ^ (s5 >> 3)) + ((s3 >> 5) ^ (4 * s5)));
        s3 -= ((delta ^ s4) + (s2 ^ key[(delta >> 2) & 3 ^ 3])) ^ (((16 * s2) ^ (s4 >> 3)) + ((s2 >> 5) ^ (4 * s4)));
        s2 -= ((delta ^ s3) + (s1 ^ key[(delta >> 2) & 3 ^ 2])) ^ (((16 * s1) ^ (s3 >> 3)) + ((s1 >> 5) ^ (4 * s3)));
        s1 -= ((delta ^ s2) + (flag[0] ^ key[(delta >> 2) & 3 ^ 1])) ^ (((16 * flag[0]) ^ (s2 >> 3)) + ((flag[0] >> 5) ^ (4 * s2)));
        //s0 = flag[0] - (((delta ^ s1) + (s7 ^ v11)) ^ (((16 * s7) ^ (s1 >> 3)) + ((s7 >> 5) ^ (4 * s1))));
        flag[0] -= (((delta ^ s1) + (s7 ^ v11)) ^ (((16 * s7) ^ (s1 >> 3)) + ((s7 >> 5) ^ (4 * s1))));
        delta -= 0x7937B99E;
    }



    //flag[0] = s0;
    flag[7] = s7;
    flag[1] = s1;
    flag[2] = s2;
    flag[3] = s3;
    flag[4] = s4;
    flag[5] = s5;
    flag[6] = s6;
    for (int i = 0; i < 8; i++)
        cout << hex << flag[i] << endl;
}
//6d616768
//74627b65
//695f6165
//5f615f73
//64726168
//636e655f
//74707972
//7d6e6f69
//hgame{btea_is_a_hard_encryption}
```

change

直接ida打开
主函数如下

```
sub_7FF787FD21E0(v10, "am2qasl", envp);
  v6 = std::shared_ptr<__ExceptionPtr>::operator=((__int64)v7, v10);
  sub_7FF787FD2280(a, v6);
  sub_7FF787FD1410(std::cout, "plz input your flag:");
  sub_7FF787FD10F0(std::cin, &flag);
  sub_7FF787FD29A0((__int64)a, (__int64)b, (__int64)&flag);
  for ( i = 0; i < 24; ++i )
  {
    last = byte_7FF787FD8000[i];
    if ( last != *(char *)sub_7FF787FD2960((__int64)b, i) )
    {
      sub_7FF787FD1410(std::cout, "sry,try again...");
```

```
      std::string::~string(b);
      sub_7FF787FD2780(a);
      std::string::~string(v10);
      return 0;
    }
  }
  sub_7FF787FD1410(std::cout, "Congratulations!");
  std::string::~string(b);
  sub_7FF787FD2780(a);
  std::string::~string(v10);
  return 0;
```

化简一下

```
sub_7FF787FD10F0(std::cin, &flag);
  sub_7FF787FD29A0((__int64)a, (__int64)b, (__int64)&flag);
  for ( i = 0; i < 24; ++i )
  {
    last = byte_7FF787FD8000[i];
    if ( last != *(char *)sub_7FF787FD2960((__int64)b, i) )
```

那就完了，sub_7FF787FD29A0肯定是加密

```
_QWORD *__fastcall sub_7FF787FD29A0(_QWORD *a, _QWORD *b, _QWORD *flag)
{
  char *v3; // rax
  char v4; // al
  char *v5; // rax
  int i; // [rsp+20h] [rbp-58h]
  unsigned int Duration; // [rsp+28h] [rbp-50h]
  unsigned int v9; // [rsp+30h] [rbp-48h]
  unsigned __int64 v10; // [rsp+48h] [rbp-30h]
  unsigned __int64 v11; // [rsp+58h] [rbp-20h]

  std::shared_ptr<__ExceptionPtr>::operator=((__int64)b, flag);
  for ( i = 0; i < (unsigned __int64)unknown_libname_20(b); ++i )
  {
    if ( i % 2 )
    {
      sub_7FF787FD2D20(sub_7FF787FD3670);
      v11 = unknown_libname_20(a);
      v9 = *(char *)sub_7FF787FD2960(a, i % v11);
      v5 = (char *)sub_7FF787FD2960(b, i);
      beep(*v5, v9);
    }
    else
    {
      sub_7FF787FD2D20(sub_7FF787FD3650);
      v10 = unknown_libname_20(a);
      Duration = *(char *)sub_7FF787FD2960(a, i % v10);
      v3 = (char *)sub_7FF787FD2960(b, i);
      beep(*v3, Duration);
    }
    *(_BYTE *)sub_7FF787FD2960(b, i) = v4;
```

```
  }
  return b;
}
```

根据奇数位和偶数位来分别加密：

奇数位直接和key异或
偶数位异或完加10

```python
def pxor(nums, *keys):
    if isinstance(nums, str):
        nums = [ord(char) for char in nums]
    result = cal([])

    if not keys:
        for i in range(len(nums)):
            result.append(nums[i] ^ i)

    for key in keys:
        if isinstance(key, int):
            for num in nums:
                result.append(num ^ key)
        elif isinstance(key, str) or isinstance(key, bytes):
            for i, num in enumerate(nums):
                xor_char = key[i % len(key)]
                result.append(num ^ ord(xor_char))
        elif isinstance(key, list):
            for i in range(len(nums)):
                result.append(nums[i] ^ key[i])

    print(result)
    return result

last = [0x13, 0x0A, 0x5D, 0x1C, 0x0E, 0x08, 0x23, 0x06,
        0x0B, 0x4B, 0x38, 0x22, 0x0D, 0x1C, 0x48, 0x0C,
        0x66, 0x15, 0x48, 0x1B, 0x0D, 0x0E, 0x10, 0x4F,
        ]
a = [0x61, 0x6D, 0x32, 0x71, 0x61, 0x73, 0x6C]
c = ''
for i in range(len(last)):
        if (i%2):
                c += chr(last[i])
        else:
                c += chr(last[i]-10)
pxor(c,a)
#hgame{ugly_Cpp_and_hook}
```

crackme2

这个题把我坑惨了……

ida打开之后反编译有标红，对应位置的指令是：mov　byte ptr ds:0, 1
于是准备先不管，去看主体逻辑

```
  sub_1400035C4("%50s", v6);
  MEMORY[0] = 1;
  v3 = sub_14000105C(v6);
  v4 = "right flag!";
  if ( !v3 )
    v4 = "wrong flag!";
  puts(v4);
```

很明显v6是flag，v3是最后的校验信息1或0

sub_14000105C打开后是换表base64

解密后发现是hgame{th1s_i5_fake_fl4g}假的，那哪里有问题呢

既然不能正确反编译主函数，那就尝试直接读汇编指令

```
.text:00000001400034EB 078                call    cs:GetCurrentProcess
.text:00000001400034F1 078                mov     rcx, rax        ; ProcessHandle
.text:00000001400034F4 078                lea     rax, [rsp+78h+arg_10]
.text:00000001400034FC 078                mov     [rsp+78h+ReturnLength], rax ; ReturnLength
.text:0000000140003501 078                mov     r9d, 8          ; ProcessInformationLength
.text:0000000140003507 078                lea     r8, [rsp+78h+ProcessInformation] ; ProcessInf
.text:000000014000350F 078                lea     edx, [r9-1]     ; ProcessInformationClass
.text:0000000140003513 078                call    NtQueryInformationProcess
.text:0000000140003518 078                cmp     [rsp+78h+ProcessInformation], 0FFFFFFFFFFFFFF
.text:0000000140003521 078                jz      short loc_14000359B
.text:0000000140003523 078                lea     r9, [rsp+78h+flOldProtect] ; lpflOldProtect
.text:000000014000352B 078                mov     edx, 6000h      ; dwSize
.text:0000000140003530 078                mov     r8d, 40h ; '@'  ; flNewProtect
.text:0000000140003536 078                lea     rcx, sub_14000105C ; lpAddress
.text:000000014000353D 078                call    cs:VirtualProtect
.text:0000000140003543 078                xor     r8d, r8d
.text:0000000140003546 078                xor     edx, edx
.text:0000000140003548
.text:0000000140003548     loc_140003548:                         ; CODE XREF: main+AE↓j
.text:0000000140003548 078                lea     rax, sub_14000105C
.text:000000014000354F 078                lea     r9, unk_140006000
.text:0000000140003556 078                mov     cl, [rax+rdx]
.text:0000000140003559 078                xor     cl, [rdx+r9]
.text:000000014000355D 078                lea     rax, sub_14000105C
.text:0000000140003564 078                mov     [rax+rdx], cl
.text:0000000140003567 078                inc     r8d
.text:000000014000356A 078                inc     rdx
.text:000000014000356D 078                movsxd  rax, r8d
.text:0000000140003570 078                cmp     rax, 246Ah
.text:0000000140003576 078                jb      short loc_140003548
.text:0000000140003578 078                lea     r9, [rsp+78h+flOldProtect] ; lpflOldProtect
.text:0000000140003580 078                mov     r8d, [rsp+78h+flOldProtect] ; flNewProtect
.text:0000000140003588 078                mov     edx, 6000h      ; dwSize
.text:000000014000358D 078                lea     rcx, sub_14000105C ; lpAddress
.text:0000000140003594 078                call    cs:VirtualProtect
```

在这一段发现了virtalprotect函数，好家伙！原来是加密了。

阅读后可知，是把加密函数和一个位置异或加密了，运行解密脚本：

```
start = 0x14000105C
loop = 0x246a
j = 0
for i in range(start,loop+start):
    patch_byte(i,get_wide_byte(i)^get_wide_byte(0x140006000+j))
    j += 1
```

在用u+c+p组合拳然后tab反汇编后是多项式方程，用z3求解即可：

```
from z3 import *
def pcheck(f,flag):
```

```
    print(f.check())
    while(f.check()==sat):
        condition = []
        m = f.model()
        p=""
        for i in range(len(flag)):
            p+=chr(int("%s" % (m[flag[i]])))
            condition.append(flag[i]!=int("%s" % (m[flag[i]])))
        print(p)
        f.add(Or(condition))

a1 = [BitVec('a1[%i]' % i,8) for i in range(32)]

v1 = a1[25]
v2 = a1[21]
v3 = a1[31]
v4 = a1[29]
v5 = a1[0]
v6 = a1[23]
v7 = a1[8]
v8 = a1[28]
v9 = a1[12]
v10 = a1[3]
v11 = a1[2]
v19 = a1[30]
v15 = a1[18]
v16 = a1[24]
v27 = a1[11]
v17 = a1[26]
v30 = a1[14]
v40 = a1[7]
v26 = a1[20]
v37 = 2 * v26
v42 = a1[22]
v28 = a1[1]
v25 = a1[27]
v21 = a1[19]
v23 = a1[16]
v31 = a1[13]
v29 = a1[10]
v41 = a1[5]
v24 = a1[4]
v20 = a1[15]
v39 = a1[17]
v22 = a1[6]
v18 = a1[9]
v33 = 2 * v41
v38 = 2 * v16
v32 = 2 * v18
v35 = v25 + v30
v34 = 2 * v31
v12 = v10 + 2 * (v31 + 4 * (v29 + v17)) + v31 + 4 * (v29 + v17)
v36 = 3 * v21
v13 = v6 + v1 + 8 * v6 + 4 * (v8 + 2 * v27)

s = Solver()
```

```
for a in a1:
    s.add(a >= 32, a <= 126)
s.add(a1[0]==ord('h'))
s.add(a1[1]==ord('g'))
s.add(a1[2]==ord('a'))
s.add(a1[3]==ord('m'))
s.add(a1[4]==ord('e'))
s.add(a1[5]==ord('{'))
s.add(a1[31]==ord('}'))

s.add(v18 +  201 * v24 +  194 * v10 +  142 * v20 +  114 * v39 +  103 * v11 +  52 * (v17 + v31) +  ((v9 + v23) << 6) +  14 * (v21 + 4 * v25 + v25) +  9 * (v40 + 23 * v27 + v2 + 3 * v1 + 4 * v2 + 4 * v6) +  5 * (v16 + 23 * v30 + 2 * (v3 + 2 * v19) + 5 * v5 + 39 * v15 + 51 * v4) +  24 * (v8 + 10 * v28 + 4 * (v42 + v7 + 2 * v26)) +  62 * v22 +  211 * v41 +  212 * v29 == 296473)
s.add(207 * v41 +  195 * v22 +  151 * v40 +  57 * v5 +  118 * v6 +  222 * v42 +  103 * v7 +  181 * v8 +  229 * v9 +  142 * v31 +  51 * v29 +  122 * (v26 + v20) +  91 * (v2 + 2 * v16) +  107 * (v27 + v25) +  81 * (v17 + 2 * v18 + v18) +  45 * (v19 + 2 * (v11 + v24) + v11 + v24) +  4 * (3 * (v23 + a1[19] + 2 * v23 + 5 * v4) + v39 + 29 * (v10 + v1) + 25 * v15) +  26 * v28 +  101 * v30 +  154 * v3 == 354358)

s.add(And((177 * v40 +  129 * v26 +  117 * v42 +  143 * v28 +  65 * v8 +  137 * v25 +  215 * v21 +  93 * v31 +  235 * v39 +  203 * v11 +  15 * (v7 + 17 * v30) +  2* (v24 +  91 * v9 +  95 * v29 +  51 * v41 +  81 * v20 +  92 * v18 +  112 * (v10 + v6) +  32 * (v22 + 2 * (v1 + v23)) +  6 * (v2 + 14 * v16 + 19 * v15) +  83 * v5 +  53 * v4 +  123 * v19) +  v17 +  175 * v27 +  183 * v3 == 448573)
    ,  (113 * v19 +  74 * v3 +  238 * v6 +  140 * v2 +  214 * v26 +  242 * v8 +  160 * v21 +  136 * v23 +  209 * v9 +  220 * v31 +  50 * v24 +  125 * v10 +  175 * v20 +  23 * v39 +  137 * v22 +  149 * v18 +  83 * (v4 + 2 * v30) +  21 * (9 * v29 + v16) +  59 * (4 * v27 + v17) +  41 * (v1 + v41) +  13 * (v7 + 11 * (v40 + v15) + 6 * v42 + 4 * (v28 + 2 * v11) + v28 + 2 * v11 + 17 * v5) +  36 * v25 == 384306)
    ,  (229 * v21 +  78 * v1 +  v2 +  v9 +  133 * v27 +  74 * v6 +  69 * v26 +  243 * v7 +  98 * v28 +  253 * v8 +  142 * v25 +  175 * v31 +  105 * v41 +  221 * v10 +  121 * v39 +  218 * (v19 + v29) +  199 * (v24 + v30) +  33 * (v40 + 7 * v17) +  4 * (27 * v20 + 50 * v11 + 45 * v18 + 19 * (v3 + v42) + v16 + 16 * v23 + 52 * v4) +  195 * v22 +  211 * v5 +  153 * v15 == 424240)
    ,  (181 * v25 +  61 * v2 +  65 * v21 +  58 * v31 +  170 * v29 +  143 * v24 +  185 * v10 +  86 * v11 +  97 * v22 +  235 * (v23 + v27) +  3* (53 * v41 +  74 * (v8 + v3) +  13 * (v42 + 6 * v9) +  11 * (v39 + 7 * v20) +  15 * (v18 + 4 * v17) +  v7 +  35 * v1 +  29 * v15) +  4 * (57 * v6 + 18 * (v5 + v37) + v28 + 17 * v16 + 55 * v30) +  151 * v40 +  230 * v4 +  197 * v19 == 421974)
    ,  (209 * v21 +  249 * v30 +  195 * v2 +  219 * v25 +  201 * v39 +  85 * v18 +  213 * (v17 + v31) +  119 * (v11 + 2 * v41) +  29 * (8 * v24 + v40 + 4 * v27 + v27) +  2* (v8 +  55 * (2 * v29 + v19) +  3 * (v10 + 39 * v9 + 2 * (v6 + 20 * v20) + 35 * v7) +  4 * (v5 + 31 * v42 + 28 * v3) +  26 * v28 +  46 * (v37 + v16) +  98 * v1) +  53 * v23 +  171 * v15 +  123 * v4 == 442074)
```

```
77      ,  ( 162 * v19 +  74 * v5 +  28 * v27 +  243 * v42 +  123 * v28 +  73 *
v8 +  166 * v23 +  94 * v24 +  113 * v11 +  193 * v22 +  122 * (v6 + 2 *
v7) +  211 * (v10 + v25) +  21 * (v17 + 7 * v41) +  11 * (v4 + 23 * (v16 +
v39) + 2 * (v40 + 5 * v30 + 2 * (2 * v18 + v29) + 2 * v18 + v29)) +  5 *
(46 * v9 + 26 * v20 + 4 * (v31 + 2 * v21) + v15 + 27 * v2 + 10 * v1) +  36
* (v3 + 5 * v26) == 376007)
78      ,  (63 * v19 +  143 * v5 +  250 * v6 +  136 * v2 +  214 * v40 +  62 *
v26 +  221 * v42 +  226 * v7 +  171 * v28 +  178 * v8 +  244 * v23 +  ((v9
<< 7)) +  150 * v31 +  109 * v29 +  70 * v41 +  127 * v20 +  204 * v39 +
 121 * v22 +  173 * v18 +  69 * (v25 + v30 + v27) +  74 * (v16 + 2 * v15 +
v15) +  22 * (7 * v24 + v17 + 10 * v11) +  40 * (v1 + 4 * v21 + v21) +  81
* v10 +  94 * v4 +  84 * v3 == 411252)
79      ,  (229 * v15 +  121 * v4 +  28 * v30 +  206 * v16 +  145 * v27 +  41 *
v1 +  247 * v6 +  118 * v26 +  241 * v28 +  79 * v8 +  102 * v25 +  124 *
v23 +  65 * v9 +  68 * v31 +  239 * v17 +  148 * v24 +  245 * v39 +  115 *
v11 +  163 * v22 +  137 * v18 +  53 * (v5 + 2 * v29) +  126 * (v40 + 2 *
v10) +  38 * (v7 + v21 + 4 * v7 + 6 * v41) +  12 * (v2 + 16 * v42) +  109 *
v20 +  232 * v3 +  47 * v19 == 435012)
80      ,  (209 * v21 +  233 * v40 +  93 * v1 +  241 * v2 +  137 * v8 +  249 *
v17 +  188 * v29 +  86 * v24 +  246 * v10 +  149 * v20 +  99 * v11 +  37 *
v22 +  219 * v18 +  17 * (v6 + 10 * v25) +  49 * (v5 + 3 * v3 + 4 * v28 +
v28) +  5 * (16 * v39 + 11 * (v41 + 2 * v27 + v27) + 12 * v7 + v31 + 30 *
v16 + 27 * v19) +  18 * (v23 + 2 * (v4 + v26 + 2 * v4) + v4 + v26 + 2 * v4)
+  24 * v9 +  109 * v42 +  183 * v30 +  154 * v15 == 392484)
81      ,  (155 * v15 +  247 * v40 +  157 * v28 +  119 * v23 +  161 * v17 +
 133 * v20 +  85 * v22 +  229 * (v7 + v24) +  123 * (2 * v31 + v42) +  21 *
(v41 + 12 * v30) +  55 * (v9 + v5 + v18 + 2 * v5) +  15 * (v3 + 16 * v10 +
9 * v21) +  2* (v2 +  115 * v29 +  111 * v16 +  26 * v6 +  88 * v8 +  73 *
v39 +  71 * v11 +  28 * (v26 + 2 * (v25 + 2 * v1)) +  51 * v27 +  99 * v4 +
 125 * v19) == 437910)
82      ,  (220 * v3 +  200 * v4 +  139 * v15 +  33 * v5 +  212 * v30 +  191 *
v16 +  30 * v27 +  233 * v1 +  246 * v6 +  89 * v2 +  252 * v40 +  223 *
v42 +  19 * v25 +  141 * v21 +  163 * v9 +  185 * v17 +  136 * v31 +  46 *
v24 +  109 * v10 +  217 * v39 +  75 * v22 +  157 * v18 +  125 * (v11 + v19)
+  104 * (v33 + v20) +  43 * (v28 + 2 * v29 + v29) +  32 * (v8 + v7 + 2 *
v8 + 2 * (v23 + v26)) == 421905)
83      ,  (211 * v24 +  63 * v15 +  176 * v5 +  169 * v16 +  129 * v27 +  146
* v40 +  111 * v26 +  68 * v42 +  39 * v25 +  188 * v23 +  130 * v9 +
((v31 << 6)) +  91 * v41 +  208 * v20 +  145 * v39 +  247 * v18 +  93 *
(v22 + v17) +  71 * (v6 + 2 * v11) +  103 * (v8 + 2 * v30) +  6 * (v21 + 10
* v28 + 28 * v7 + 9 * v29 + 19 * v2 + 24 * v1 + 22 * v3) +  81 * v10 +  70
* v4 +  23 * v19 == 356282)
84      ,  (94 * v42 +  101 * v2 +  152 * v40 +  200 * v7 +  226 * v8 +  211 *
v23 +  121 * v24 +  74 * v11 +  166 * v18 +  (((v6 + 3 * v28) << 6)) +  41
* (4 * v9 + v21) +  23 * (v39 + 11 * v41) +  7 * (v20 + 10 * v25 + 2 * v12
+ v12) +  3 * (78 * v30 + 81 * v16 + 55 * v27 + 73 * v1 + 4 * v26 + v15 +
85 * v3 + 65 * v19) +  62 * v22 +  88 * v5 +  110 * v4 == 423091)
85      ,  (133 * v22 +  175 * v15 +  181 * v30 +  199 * v16 +  123 * v27 +
 242 * v1 +  75 * v6 +  69 * v2 +  153 * v40 +  33 * v26 +  100 * v42 +
 229 * v7 +  177 * v8 +  134 * v31 +  179 * v29 +  129 * v41 +  14 * v10 +
 247 * v24 +  228 * v20 +  92 * v11 +  86 * (v9 + v32) +  94 * (v23 + v21)
+  37 * (v17 + 4 * v3) +  79 * (v25 + 2 * v28) +  72 * v5 +  93 * v39 +
 152 * v4 +  214 * v19 == 391869)
```

```
86      ,  (211 * v24 +  213 * v18 +  197 * v40 +  159 * v25 +  117 * v21 +
 119 * v9 +  98 * v17 +  218 * v41 +  106 * v39 +  69 * v11 +  43 * (v2 +
v29 + 2 * v2) +  116 * (v4 + v10 + v37) +  5 * (v42 + 9 * v23 + 35 * v20 +
37 * v31) +  11 * (v16 + 13 * v27 + 5 * v5 + 8 * v30) +  6 * (29 * v28 + 25
* v8 + 38 * v22 + v15 + 13 * v1 + 10 * v3) +  136 * v7 +  142 * v6 +  141 *
v19 == 376566)
87      ,  (173 * v3 +  109 * v15 +  61 * v30 +  187 * v1 +  79 * v6 +  53 *
v40 +  184 * v21 +  43 * v23 +  41 * v9 +  166 * v31 +  193 * v41 +  58 *
v24 +  146 * v10 +  ((v20 << 6)) +  89 * v39 +  121 * v11 +  5 * (v17 + 23
* v8) +  7 * (29 * v18 + v29 + 4 * v7) +  13 * (3 * v42 + v16 + 7 * v26 +
13 * v2) +  3 * (v4 + 83 * v5 + 51 * v27 + 33 * v22 + 8 * (v19 + 4 * v28) +
18 * v25) == 300934)
88      ,  (78 * v1 +  131 * v5 +  185 * v16 +  250 * v40 +  90 * v26 +  129 *
v42 +  255 * v28 +  206 * v8 +  239 * v25 +  150 * v10 +  253 * v39 +  104
* v22 +  58 * (v2 + 2 * v7) +  96 * (v15 + v31) +  117 * (v9 + 2 * v4) +
 27 * (v17 + 8 * v18 + v18) +  19 * (v23 + 3 * v21 + 4 * v29 + v29) +  7 *
(22 * v41 + 3 * (v11 + 11 * v24) + v3 + 29 * v6 + 14 * v27) +  109 * v20 +
 102 * v30 +  100 * v19 == 401351)
89      ,  (233 * v19 +  71 * v5 +  209 * v27 +  82 * v6 +  58 * v26 +  53 *
v25 +  113 * v23 +  206 * v31 +  39 * v41 +  163 * v20 +  222 * v11 +  191
* v18 +  123 * (v7 + v40) +  69 * (v9 + 2 * v22 + v22) +  9 * (v3 + 8 * v24
+ 7 * (3 * v1 + v28) + 5 * v16 + 19 * v30) +  4 * (v15 + 26 * v17 + 61 *
v29 + 43 * v42 + 49 * v2 + 32 * v4) +  10 * (7 * (v8 + v36) + v39 + 12 *
v10) == 368427)
90      ,  (139 * v30 +  53 * v5 +  158 * v16 +  225 * v1 +  119 * v6 +  67 *
v2 +  213 * v40 +  188 * v28 +  152 * v8 +  187 * v21 +  129 * v23 +  54 *
v9 +  125 * v17 +  170 * v24 +  184 * v11 +  226 * v22 +  253 * v18 +  26 *
(v29 + v41) +  97 * (v4 + 2 * v25) +  39 * (5 * v26 + v27) +  21 * (v39 + 8
* v42) +  12 * (17 * v10 + v31 + 15 * v7 + 12 * v19) +  165 * v20 +  88 *
v15 +  157 * v3 == 403881)
91      ,  (114 * v3 +  61 * v27 +  134 * v40 +  62 * v42 +  89 * v9 +  211 *
v17 +  163 * v41 +  66 * v24 +  201 * (v7 + v18) +  47 * (5 * v16 + v22) +
 74 * (v4 + v31) +  142 * (v2 + v28) +  35 * (v20 + 6 * v26) +  39 * (v15 +
6 * v30) +  27 * (v25 + 9 * v23 + 8 * v6) +  4 * (v21 + 63 * v19 + 2 * (v1
+ 12 * (v10 + v5) + 8 * v11 + 26 * v29)) +  10 * (v8 + 4 * v39 + v39) ==
382979)
92      ,  (122 * v25 +  225 * v21 +  52 * v23 +  253 * v9 +  197 * v17 +  187
* v31 +  181 * v29 +  183 * v41 +  47 * v20 +  229 * v39 +  88 * v22 +  127
* (v10 + v32) +  37 * (v7 + 3 * v3) +  (((v11 + 2 * v30 + v30) << 6)) +  7
* (21 * v8 + v27 + 18 * (v4 + v1 + v38)) +  6 * (23 * v24 + v26 + 17 * v2 +
39 * v6) +  10 * (v5 + 11 * v28 + 21 * v42) +  149 * v19 +  165 * v40 +
 121 * v15 == 435695)
93      ,  (165 * v20 +  223 * v4 +  249 * v5 +  199 * v1 +  135 * v2 +  133 *
v26 +  254 * v42 +  111 * v7 +  189 * v28 +  221 * v25 +  115 * v21 +  186
* v9 +  79 * v41 +  217 * v24 +  122 * v11 +  38 * v18 +  109 * (v34 + v29)
+  14 * (v8 + 17 * v40 + 8 * (v6 + v38)) +  4 * (11 * (5 * v30 + v39) + 6 *
(v10 + 2 * v22) + v27 + 52 * v17 + 50 * v23) +  229 * v15 +  86 * v3 +  234
* v19 == 453748)
94      ,  (181 * v25 +  94 * v42 +  125 * v1 +  226 * v26 +  155 * v7 +  95 *
v21 +  212 * v17 +  91 * v31 +  194 * v29 +  98 * v24 +  166 * v11 +  120 *
v22 +  59 * v18 +  32 * (v9 + v8) +  158 * (v6 + v5) +  101 * (v41 + v19) +
 63 * (v4 + 2 * v23) +  67 * (v28 + 2 * v20) +  11 * (v39 + 10 * v16 + 11 *
v10) +  39 * (v30 + 4 * (v2 + v15)) +  233 * v40 +  56 * v27 +  225 * v3 ==
358321)
```

```
    ,  (229 * v21 +  135 * v4 +  197 * v15 +  118 * v5 +  143 * v16 +  134 * v6 +  204 * v40 +  173 * v26 +  81 * v7 +  60 * v28 +  58 * v8 +  179 * v23 +  142 * v9 +  178 * v17 +  230 * v31 +  148 * v29 +  224 * v41 +  194 * v24 +  223 * v10 +  87 * v20 +  200 * v39 +  233 * v11 +  49 * v22 +  127 * v35 +  31 * (4 * v27 + v18) +  42 * (v1 + 6 * v2) +  109 * v42 +  75 * v3 +  165 * v19 == 456073)
    ,  (41 * v4 +  253 * v3 +  163 * v15 +  193 * v30 +  155 * v16 +  113 * v27 +  131 * v6 +  55 * v2 +  21 * v40 +  53 * v26 +  13 * v8 +  201 * v25 +  237 * v9 +  223 * v31 +  95 * v24 +  194 * v20 +  62 * v39 +  119 * v11 +  171 * v22 +  135 * v18 +  69 * (v10 + 3 * v28) +  211 * (v1 + v29) +  4 * (43 * v7 + v42 + 40 * v17) +  6 * (v5 + 33 * v41 + 20 * (2 * v19 + v21) + 24 * v23) == 407135)
    ,  (111 * v19 +  190 * v3 +  149 * v4 +  173 * v28 +  118 * v23 +  146 * v29 +  179 * v10 +  51 * v20 +  49 * v39 +  61 * v11 +  125 * v22 +  162 * v18 +  214 * v35 +  14 * (v34 + v24) +  178 * (v41 + v16) +  11 * (4 * v9 + v21 + 17 * v42) +  65 * (v26 + v17 + 2 * v26 + 2 * v5) +  4 * (v7 + 38 * v15 + 4 * v13 + v13 + 8 * v40 + 43 * v2) == 369835)
    ,  (27 * v27 +  223 * v6 +  147 * v26 +  13 * v21 +  35 * (v17 + 7 * v4) +  57 * (v19 + v32 + 3 * v11) +  11 * (v1 + 17 * (v9 + v5) + 10 * v16 + 3 * v31) +  2* (53 * v23 +  v25 +  38 * v15 +  43 * v42 +  115 * v29 +  61 * v22 +  111 * (v10 + v40) +  14 * (v20 + v7 + 2 * v7 + 8 * v28) +  109 * v2 +  100 * v41 +  63 * v8) +  93 * v39 +  251 * v30 +  131 * v3 == 393303)
    ,  (116 * v9 +  152 * v29 +  235 * v20 +  202 * v18 +  85 * (v8 + 3 * v11) +  221 * (v16 + v40) +  125 * (v33 + v24) +  7 * (19 * v4 + 9 * (v10 + 2 * v25) + v2 + 33 * v3 + 32 * v19) +  3 * (71 * v39 + 43 * v22 + 32 * (v17 + v26) + 15 * (v5 + v6 + 2 * v23) + v28 + 74 * v31 + 48 * v42) +  10 * (v21 + 11 * v30 + 16 * v15) +  136 * v7 +  106 * v1 +  41 * v27 == 403661)
    ,  (127 * v4 +  106 * v15 +  182 * v30 +  142 * v5 +  159 * v16 +  17 * v1 +  211 * v6 +  134 * v2 +  199 * v7 +  103 * v28 +  247 * v23 +  122 * v9 +  95 * v41 +  62 * v10 +  203 * v39 +  16 * v11 +  41 * (6 * v42 + v25) +  9 * (22 * v24 + v20 + 27 * v31 + 28 * v40) +  10 * (v8 + v22 + v36 + 8 * v17 + 2 * (v22 + v36 + 8 * v17) + 13 * v29) +  6 * (23 * v27 + v26) +  213 * v18 +  179 * v3 +  43 * v19 == 418596)))
s.add(149 * v19 +  v1 +  133 * v22 +  207 * v41 +  182 * v26 +  234 * v7 +  199 * v8 +  168 * v21 +  58 * v10 +  108 * v20 +  142 * v18 +  156 * (v9 + v25) +  16 * (v29 + 6 * v31) +  126 * (v17 + 2 * v39) +  127 * (v4 + 2 * v27 + v40) +  49 * (v30 + 4 * v16) +  11 * (v5 + 22 * v11) +  5 * (v15 + v42 + 45 * v24 + 50 * v28) +  109 * v2 +  124 * v6 +  123 * v3 == 418697)
pcheck(s,a1)
#hgame{SMC_4nd_s0lv1ng_equ4t1Ons}
```