Crypto

Coppersmith-I

这道题直接算小根算不出来，需要设置参数epsilon=0.01，可以算到248位，剩下的5位就只能枚举了。

```
N =
1355006465745825112398457647103117692608019989824295006801719198234311788995264635
662158342343833313744450933639692188109069917845693402705109367591835044965842259
376149400863297753258933074539190558302709866011520021913684315272852853136699793
58099782497422114870417519470053198217401297960844455029559146309
h =
91857802455816883663891963609077758613549763881820953361542065028229216863148 5
t = 248
h = h << 253
for i in range(2**(253-t)):
    PR.<x> = PolynomialRing(Zmod(N), implementation='NTL')
    f = h + (i<<t) + x
    try:
        x0 = f.small_roots(X = 2**t, beta = 0.49, epsilon=0.01)[0]
        print(x0)
        break
    except:
        print(i)
```

将上面输出放进下面的代码解出明文。

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


N =
1355006465745825112398457647103117692608019989824295006801719198234311788995264635
662158342343833313744450933639692188109069917845693402705109367591835044965842259
376149400863297753258933074539190558302709866011520021913684315272852853136699793
58099782497422114870417519470053198217401297960844455029559146309
e = 65537
c =
4176395681864014555663222972062637265692187585650738901485557539650249865945021132
372707455174227923542563489585428645912494105007504106589885091362424355022591722
584326765028467290882782027507217604511606686537460199656957218448195876716029255
51448624324524027931677927410810126647175483982178300855471710099
h =
918578024558168836638919636090777586135497638818209533615420650282292168631485
p = (
    (h << 253)
    + (20 << 248)
    + 12757858752378535708139819660175106458980802485374460654460439738856801589
)
q = N // p
print(N % p)
d = gmpy2.invert(e, (p - 1) * (q - 1))
```

```
m = pow(c, d, N)
print(long_to_bytes(m))  # 0xGame{8f4c17cb-442a-49bd-830a-d16af225a5c5}
```

RNG

这道题比较像逆向，RNG先通过种子生成初始状态，再用twist处理，最后用一系列位运算得到随机数。

twist操作会损失原数据的，在逆向时只能获得原始mt除了seed以外的数，所以应用mt[1]反推seed。同时逆向得到的seed不一定是唯一的，可能会出错，只能多试几次。

```python
from Crypto.Util.number import inverse
from random import getrandbits
from pwn import *


class RNG:
    def __init__(self, seed):
        self.mt = [0] * 624
        self.mt[0] = seed
        self.mti = 0
        for i in range(1, 624):
            self.mt[i] = self._int32(
                1812433253 * (self.mt[i - 1] ^ (self.mt[i - 1] >> 30)) + i
            )

    def _int32(self, x):
        """
        Convert a integer to a 32-bit integer."""
        return int(0xFFFFFFFF & x)

    def extract(self):
        if self.mti == 0:
            self.twist()
        y = self.mt[self.mti]
        y = y ^ y >> 11
        y = y ^ y << 7 & 2636928640
        y = y ^ y << 15 & 4022730752
        y = y ^ y >> 18
        self.mti = (self.mti + 1) % 624
        return self._int32(y)

    def twist(self):
        for i in range(0, 624):
            y = self._int32(
                (self.mt[i] & 0x80000000) + (self.mt[(i + 1) % 624] & 0x7FFFFFFF)
            )
            self.mt[i] = (y >> 1) ^ self.mt[(i + 397) % 624]

            if y % 2 != 0:
                self.mt[i] = self.mt[i] ^ 0x9908B0DF


def reverse_seed(x: int) -> int:
    x = (x - 1) * inverse(1812433253, 2**32) % 2**32
    x ^= x << 30 & 0xFFFFFFFF
    return x


def reverse_twist(twist):
    reverse_twist = [0] * 624
    for i in range(623, -1, -1):
        k = twist[i] ^ twist[(i + 397) % 624]
        if (k & 0x80000000) >> 31 == 1:  ##末位1和0判断
            k = k ^ 0x9908B0DF
            lowbit = 1
            highbit = (k & 0x40000000) >> 30
            reverse_twist[i] = highbit << 31
            reverse_twist[(i + 1) % 624] = (
                reverse_twist[(i + 1) % 624] + ((k & 0x3FFFFFFF) << 1) + lowbit
            )
            if i != 623:
                twist[(i + 1) % 624] = reverse_twist[(i + 1) % 624]
        elif (k & 0x80000000) >> 31 == 0:
            lowbit = 0
            highbit = (k & 0x40000000) >> 30
            reverse_twist[i] = highbit << 31
            reverse_twist[(i + 1) % 624] = (
                reverse_twist[(i + 1) % 624] + ((k & 0x3FFFFFFF) << 1) + lowbit
            )
            if i != 623:
                twist[(i + 1) % 624] = reverse_twist[(i + 1) % 624]
    return reverse_twist


def xorshift(y):
    y ^= y >> 18
    temp = y
    while temp:
        temp <<= 15
        temp &= 4022730752
        y ^= temp
    temp = y
    while temp:
        temp <<= 7
        temp &= 2636928640
        y ^= temp
    temp = y
    while temp:
        temp >>= 11
        y ^= temp
    return y


def test():
    result = []
    seed = getrandbits(32)
    print("Seed:", seed)
    rng = RNG(seed)
    mt = rng.mt.copy()
    for _ in range(624):
        result.append(rng.extract())
    mt_ = result.copy()
    # mt = input().split(", ")
    # mt = [int(i) for i in mt]

    for i in range(len(mt_)):
        mt_[i] = xorshift(mt_[i])

    if rng.mt == mt_:
        print("Success")
    else:
        print("Failed")

    # mt__ = untwist(mt_)
    mt__ = reverse_twist(mt_)
    if mt[1:] == mt__[1:]:
        print("Success")
    else:
        print("Failed")

    seed = reverse_seed(mt__[1])

    result_ = []
    rng = RNG(seed)
    for _ in range(624):
        result_.append(rng.extract())

    if result == result_:
        print("Success")
    else:
        print("Failed")


def main():
    result = io.recvuntil(b"\n", drop=True).decode().strip()
    result = io.recvuntil(b"\n", drop=True).decode().strip()
    result = result[1:-1]
    mt_ = result.split(", ")
    mt_ = [int(i) for i in mt_]

    for i in range(len(mt_)):
        mt_[i] = xorshift(mt_[i])

    mt__ = reverse_twist(mt_)
    seed = reverse_seed(mt__[1])
    print(str(seed).encode())

    io.sendlineafter(b"[+] seed = ?\n>", str(seed).encode())

    result = io.recvuntil(b"\n", drop=True).decode().strip()
    print(result)  # 0xGame{2569bd55-a14d-46d8-81f5-e1397e4be7bc}


ip = "118.195.138.159"
port = 10006
io = remote(ip, port)
```

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

SIDH

这道题实测用sageMath9.3去参数后算不出来，必须安装sageMath10.4

这题题目给的代码可以直接拿来用，传了参数就有了

```
┌──(root㉿Spreng)-[~]
└─# nc 118.195.138.159 10009
[+] RA = (116053925917084542011324769615777776587074207943195123562016477777*i +
29853625691926097630473026940814722005987892402785323248602784145,
48589849634044362335499881707258110110225972146727020812467785107*i +
93950513117315108194013046943470249031829772504602597292364689469)
[+] Give me RB:
>
53994948819107287718103790613608198624099860495406021066938911313,361159892384820
81321343318658725623161289831461278943951578131227,685493623274193035583850296536
95812698659503364632641474133172065,732377180386966106735287144704810023656457445
55489751669283393573
[+] Tell me the secret
>
51043981074163751579860189255783611925910678911900493322626381728,119031559959345
839581900659518118581713074678821401592863666418251
[+] flag:0xGame{4179c8c3-db69-4fb0-bd14-ef6c76ddb973}
```

Reverse

MineSweeper

Unity逆向工具：dnSpy、AssetRipper

先用dnspy打开Assembly-CSharp.dll Update() 打通第二关会给flag

下面用C语言呈现flag的解密过程：

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

// 加密函数
char *crypt(const char *Game_key)
{
    unsigned char Key[] = "This is: True_KEY!for #0xgAmE_Unity~Cryption";
    int num = 0;
    // unsigned char array[] =
"E!>\u0008W1\tMBEBD]ZKKRV\u0016DfEl@WD35Qu\rX\u0015q\u0011\u001B\u000B\u0008v\u00
04O\\h<";
    unsigned char array[] =
"E!>\x08W1\tMBEBD]ZKKRV\026DfEl@WD35Qu\rX\x15q\x11\x1B\x0B\x08v\x04O\\h<";
```

```
    for (int i = 0; i < 44; i++)
    {
        num = (num + (int)Game_key[i % strlen(Game_key)]) % 44;
        unsigned char b = Key[num];
        Key[num] = Key[i];
        Key[i] = b;
    }

    for (int j = 43; j >= 0; j--)
    {
        array[j] ^= Key[j];
    }

    // 将加密后的数组转换为字符串并返回
    char *encrypted = (char *)malloc(45); // 分配足够的内存来存储加密后的数据和空字符
    memcpy(encrypted, array, 44);
    encrypted[44] = '\0'; // 确保字符串以空字符结尾
    return encrypted;
}

int main()
{
    char *enc_data = crypt("0xoX0XOxOXoxGAME");
    printf("%s\n", enc_data); // 0xGame{36ecd059-b3e7-73c8-fa80-0a2abef3c757}
    free(enc_data);            // 释放分配的内存
    return 0;
}
```

相关数据一个是key = "0xoX0XOxOXoxGAME"
还有一个array用AssetRipper打开resources找到<TextAsset>enc：
`E!>\u0008W1\tMBEBD]ZKKRV\u0016DfEl@WD35Qu\rX\u0015q\u0011\u001B\u000B\u0008v\u0004O\\h<`

后面还有提示Maybe your flag is hidden here

PyPro

先使用pyinstxtractor提取PyPro.exe，再用在线网站反编译PyPro.pyc，得到源码一部分：对输入的flag加密

不难猜测后面的内容应该是对比密文是否一致，用记事本打开得到这一行：

```
format error?
z@2e8Ugcv8lKVhL3gkv3grJGNE3UqkjlvKqCgJSGRNHHEk98Kd0wv6s60GpAUsU+8Qu
```

密文应是2e8Ugcv8lKVhL3gkv3grJGNE3UqkjlvKqCgJSGRNHHEk98Kd0wv6s60GpAUsU+8Q，解密即可

```
import base64
from Crypto.Cipher import AES
from Crypto.Util.number import long_to_bytes

key = 0x554B134A029DE539438BD18604BF114


def PKCS5_pad(data):
    if len(data) < 48:
        length = 48 - len(data)
        return data.ljust(48, length.to_bytes())


def main():
    cipher = AES.new(long_to_bytes(key, 16), AES.MODE_ECB)
    c = "2e8Ugcv8lKVhL3gkv3grJGNE3UqkjlvKqCgJSGRNHHEk98Kd0wv6s60GpAUsU+8Q"
    c = base64.b64decode(c.encode("utf-8"))
    m = cipher.decrypt(c)
    print(m.decode("utf-8"))


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

Tea2.0

本题中的data在程序执行中被回调函数修改过，即先用处理后的key1对data先tea加密一次，再对输入的flag tea2.0加密。

只需对data先tea加密，再tea2-de解密即可，解密依然是反转加密函数的代码

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

void tea(unsigned int *a1, int *a2)
{
    int result;      // eax
    int i;           // [esp+D0h] [ebp-38h]
    int v4;          // [esp+DCh] [ebp-2Ch]
    unsigned int v5; // [esp+F4h] [ebp-14h]
    unsigned int v6; // [esp+100h] [ebp-8h]

    v6 = *a1;
    v5 = a1[1];
    v4 = 0;
    for (i = 0; i < 32; ++i)
    {
        v4 -= 1640531527;
        v6 += (v4 + v5) ^ (a2[1] + (v5 >> 5)) ^ (*a2 + 16 * v5);
        v5 += (v4 + v6) ^ (a2[3] + (v6 >> 5)) ^ (a2[2] + 16 * v6);
    }
    *a1 = v6;
    a1[1] = v5;
}

int tea_de(uint32_t *a1, int a2)
{
    uint32_t v4;
    uint32_t v5;
    uint32_t v6;

    v6 = *a1;
    v5 = a1[1];
    v4 = -1914802624;
    for (int i = 0; i < 64; ++i)
    {
        v5 -= (*(unsigned int *)(a2 + 4 * ((v4 >> 11) & 3)) + v4) ^ (v6 + ((v6 >> 5) ^ (16 * v6)));
        v4 += 1640531527;
        v6 -= (*(unsigned int *)(a2 + 4 * (v4 & 3)) + v4) ^ (v5 + ((v5 >> 5) ^ (16 * v5)));
    }
    a1[0] = v6;
    a1[1] = v5;
    return 4;
}

int main()
{
    char v1;
    char v2;
    int i;
    unsigned char key1[] = {
        0x45, 0x12, 0x00, 0x00, 0x98, 0x32, 0x00, 0x00,
        0x56, 0x47, 0x00, 0x00, 0x63, 0x14, 0x00, 0x00};
    unsigned char key2[] = {
        0x12, 0x45, 0x00, 0x00, 0x32, 0x98, 0x00, 0x00,
        0x47, 0x56, 0x00, 0x00, 0x14, 0x63, 0x00, 0x00};
    unsigned char data[] = {
        0x60, 0xC3, 0x8D, 0x01, 0x57, 0x54, 0x83, 0xD5,
        0xCB, 0x2D, 0xEE, 0x8B, 0xEE, 0x2D, 0xBB, 0x92,
        0x54, 0xAD, 0xF4, 0xFD, 0x2D, 0x8C, 0x3F, 0x04,
        0xA9, 0x32, 0xA2, 0x61, 0xD1, 0xF4, 0x15, 0x0F,
        0x79, 0x49, 0xEA, 0x16, 0xDA, 0xF6, 0x2B, 0x7C,
        0x32, 0xFA, 0xD5, 0xDC, 0x19, 0x08, 0x45, 0x76};

    unsigned int *k = (unsigned int *)key1;
    // 处理密钥
    for (i = 0; i < 4; ++i)
    {
        *(k + i) ^= 0xABCDu;
    }

    for (i = 0; i < 6; ++i)
    {
        tea(data + 8 * i, key1);
    }

    for (i = 0; i < 6; ++i)
        tea_de(data + 8 * i, key2);

    printf("%44s\n", data); // 0xGame{a7961e4b-c809-f340-412e-91abd2c9b535}

    return 0;
}
```