NPUCTF 2020 Crypto writeup

原创

Slightwindsec 于 2020-04-24 15:18:42 发布 2354 收藏 2

分类专栏： CTF 文章标签： python

本文链接：https://blog.csdn.net/qq_41956187/article/details/105731368

我的博客：https://blog.slight-wind.com/

认清形势，建立信心

题目中给了 $2 \mod n$, $4 \mod p$ 和 $8 \mod p$。令它们分别为 a,b,c 。

$4 = 2 * 2 \ldots$ 所以有 b ……

还差 e 可以直接枚举，虽然线性时间复杂度，保险起见还是用 C++ 来跑，几秒种就可以出结果。（其实这是个简单的DLP，使用我之前写过的DLP脚本也可以解决

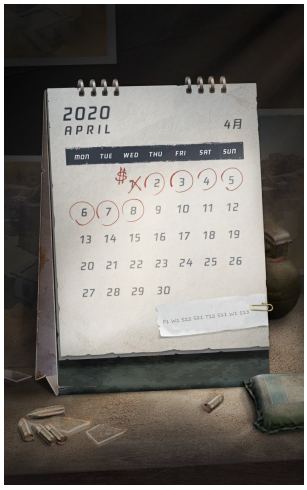
```
#include&lt;iostream>
#define LL long long
using namespace std;

int main(){
    LL e=1,X=2,N=527247002021197,a=128509160179202;
    while(e<10000000000){
        if(e%100000000==0)cout<<e<<endl;
        e++;
        X=X*2%N;
        if(X==a){
            cout<<"find"<<endl;
            cout<<e<<endl;
            return 0;
        }
    }
    return 0;
}
```

得到 e = 808723997 ， 正常解 RSA 就可以了。

```
from Crypto.Util.number import *
c = 169169912654178
e = 808723997
p = 18195301
q = 28977097
m = pow(c, inverse(e, (p-1)*(q-1)), p*q)
print(long_to_bytes(m))
```

这是什么觅□



图片上有四月的日历和小纸条，F1 W1 S22 S21 T12 S11 W1 S13，字母代表星期的首字母，其中 S 和 T 都出现两次，所以 S1 代表 SAT，S2 代表 SUN，每组最后一个数字即代表第几行，得到 3 1 12 5 14 4 1 18，对照字母表 calendar，flag{calendar}。

Classical Cipher

放在https://quipqiup.com/上解，试出来压缩包密码：the_key_is_atbash

解压得到一张图，变形猪圈很容易看出来（那个鸟和狗是啥真没见过.....）,根据题目名称里的classical还是很容易顺出来的，答案是 classicalcode，flag{classicalcode}

EzRSA

题目给了 (p-1) 和 (q-1) 的最小公倍数 gift，也就是 (p-1)*(q-1) 除以它们的最大公因数，也知道 p*q 是 2048 bit 的，phi = (p-1)*(q-1) 的 bit 长度只有可能是 2047 bit 或 2048 bit，因此可以从 bit 长度入手在很小的范围内枚举这个最大公因数，同时这个最大公因数还可以整除 gift，并尝试二分法来分解出 p，q 。得到 (p-1)*(q-1) = gift*8 是可行的。

~~在我的上一篇博客里有二分法分解的脚本，这里就不贴了.....~~

这时的问题是 GCD(e , phi) = 2，通过 c = pow(c , inverse(e/2 , phi) , n) = pow(m , 2 , n)，也就是一个 Rabin 加密。

Rabin 解密需要分别在有限域 GF§ 和 GF(q) 内对 c 开平方根，测试发现 p 和 q 都是除 4 余 1 的素数，所以麻烦一些，用 Tonelli–Shanks algorithm 来开平方根。

exp：

```
from Crypto.Util.number import *

n = 17083941230213489700426636484487738282426471494607098847295335339638177583685457921198569105417734668692072727759139358207667248703952436680183153327606147421932365889983347282046439156176685765143620637107347870401946946501620531665573668068349080410807996582297505889946205052879002028936125315312256470583622913646319779125559691270916064588684997382451412747432722966919513413709987353038375477178385125453567111965259721484997156799355617642131569095810304077131053588483057244340742751804935494087687363416921314041547093118565767609667033859583125275322077617576783247853718516166743858265291135353895239981121
c = 37389606391947379576676684143565005503596276451617922474669745529299929395507971435311181578387223323429323286927370576955078618335757508161263585164126047545413028829873269342924092339298957635079736446851837414357757312525158356579607212496060244403765822636515347192211817658170822313646743520831977673861869637519843133863288550058359429455052676323196728280408508614527953057214779165450356577820378810467527006377296194102671360302059901897977339728292345132827184227155061326328585640019916328847372295754472832318258636054663091475801235050657401857262960415898483713074139212596685365780269667500271108538319
e = 54722
```

```
p = 10602144899102139144455074937511527708084428174624884580256568055778500934195232048417556876370742493217203359751486160211417145917644027904576184669523178837607505045215492414126629093141354211063908179255064810624096655240681305939635835573718535488547445524857994619026615241614913761685579180561720615349 7
q = n//p
c = pow(c, inverse(e//2, (p-1)*(q-1)), n)

# Euler's criterion
def legendre_symbol(a, p):
    symbol = pow(a, (p - 1) // 2, p)
    if symbol == p - 1:
        return -1
    return symbol

# Tonelli–Shanks algorithm
def sqrt_mod(a, p):
    if a == 0 or legendre_symbol(a, p) != 1:
        return 0
    if p % 4 == 3:
        return pow(a, (p + 1) // 4, p)
    if p % 8 == 5:
        if pow(a, (p - 1) // 4, p) == 1:
            return pow(a, (p + 3) // 8, p)
        else:
            return (pow(2, (p - 1) // 4, p) * pow(a, (p + 3) // 8, p)) % p

    q = p - 1
    s = 0
    while q & 1 == 0:
        q >>= 1
        s += 1

    z = 2
    while legendre_symbol(z, p) != -1:
        z += 1

    m = s
    c = pow(z, q, p)
    t = pow(a, q, p)
    r = pow(a, (q + 1) // 2, p)

    while t != 1:
        t2 = t
        i = 0
        while t2 != 1 and i < m:
            t2 = pow(t2, 2, p)
            i += 1

        b = pow(c, 2 ** (m - i - 1), p)
        m = i
        c = (b * b) % p
        t = (t * c) % p
        r = (r * b) % p

    return r


r = sqrt_mod(c, p)
s = sqrt_mod(c, q)
a = inverse(p, q)
b = inverse(q, p)
```

```
b = inverse(q, p)
x = (a*p*s+b*q*r) % n
y = (a*p*s-b*q*r) % n
print(long_to_bytes(x % n))
print(long_to_bytes((-x) % n))
print(long_to_bytes(y % n))
print(long_to_bytes((-y) % n))
```

NPUCTF{diff1cult_rsa_1s_e@sy}

Mersenne_twister

MT19937

这题很显然是考梅森旋转伪随机数生成器（MT19937）的，正常的 MT19937 内部有 624 个状态（state），也就是至少得到 624 个输出才可以逆向出内部的 624 个状态。

从状态到输出的随机数，经历了一组异或运算，通常是异或左移异或右移，这是可逆的。

以这题的一组异或位移运算为例：

```
def Next(self , tmp):
    tmp ^= (tmp >> 11)
    tmp ^= (tmp << 7) & 0x9ddf4680
    tmp ^= (tmp << 15) & 0xefc65400
    tmp ^= (tmp >> 18) & 0x34adf670
    return tmp
```

方便起见，可以把四种（带遮罩(masked)和不带遮罩的左移和右移）写在一个文件里，方便下次调用：

```python
class unBitShift:
    def __init__(self):
        pass

    def RightXor(self,value, shift):
        i = 0
        while i * shift < 32:
            part_mask = ((0xffffffff << (32 - shift)) & 0xffffffff) >> (i * shift)
            part = value & part_mask
            value ^= part >> shift
            i += 1
        return value

    def RightXorMasked(self,value, shift, mask):
        i = 0
        while i * shift < 32:
            part_mask = ((0xffffffff << (32 - shift)) & 0xffffffff) >> (i * shift)
            part = value & part_mask
            value ^= (part >> shift) &amp;amp; mask
            i += 1
        return value

    def LeftXor(self,value, shift):
        i = 0
        while i * shift < 32:
            part_mask = ((0xffffffff >> (32 - shift)) & 0xffffffff) << (i * shift)
            part = value & part_mask
            value ^= part << shift
            i += 1
        return value

    def LeftXorMasked(self,value, shift, mask):
        i = 0
        while i * shift < 32:
            part_mask = ((0xffffffff >> (32 - shift)) & 0xffffffff) << (i * shift)
            part = value & part_mask
            value ^= (part << shift) & mask
            i += 1
        return value
```

writeup

这题是修改过的 MT19937 ， 内部只有 233 个状态，每获取一个随机数 key 会输出 4 个随机数，也就是 4 个状态运算出的结果。每次加密一个字符，先把这个字符 MD5 哈希，然后与 key 异或。

由于题目中保证了前 7 个字符为 'npuctf{' ，把它们每个字符 MD5 值与对应密文异或就可以得到 7 个 key ，就相当于我们可以推出前 28 个状态，距离 233 还很遥远…

既然 233 个状态不能全部逆出来，就只能尝试爆破 seed …

先通过第一个已知明文 'n' 得到第一个 key ，截取这个 key 的前四分之一长，并对 Next() 函数进行逆运算，得到第一个状态值：

```python
from hashlib import md5
from binascii import unhexlify
from unBitShift import unBitShift
from Crypto.Util.number import bytes_to_long
def XOR(s1, s2): return bytes([x1 ^ x2 for x1, x2 in zip(s1, s2)])
cip = b'cef4876036ee8b55aa59bca043725bf3'
assert len(cip)==32
cip = unhexlify(cip)
tmp = md5('n'.encode()).digest()
key = XOR(tmp, cip)
print(key)
#b'\xb5\x7f\x11:\xe2R+\xb1\xb0\xec\xa1G\xf0a8R'

def Last(tmp):
    unshift=unBitShift()
    tmp=unshift.RightXorMasked(tmp,18,0x34adf670)
    tmp=unshift.LeftXorMasked(tmp,15,0xefc65400)
    tmp=unshift.LeftXorMasked(tmp, 7,0x9ddf4680)
    tmp=unshift.RightXor(tmp,11)
    return tmp

x=bytes_to_long(key[:4])
print(Last(x))
# 778501557
```

所以现在只要去爆破一个 seed 可以使 state[0] == 778501557 （如果有很多这样的 seed 也可以再计算出第二个状态值加以限制，事实证明不需要：）估算一下时间复杂度 232*n，约 O(200n)。（ n 最后枚举到 1e9 的数量级）

一开始我当然用 Python 来写爆破，就硬枚举 seed，我的辣鸡电脑每分钟约可以跑多于 2e5 个 seed，跑了几十分钟放弃了。（现在带着最终结果来看(1.6e9)，要跑 5 天(8000 min)）

于是尝试用 C++ 来跑（正常 oi 评测机运行 C++ 可以以 5e8 每秒来估算），也就是每秒 2e6 个 seed，比 Python 快了约 600 倍！！也就是 13 分钟…

~~FBIWarning~~：600 只是针对这一例的估算，代码的实现方式可能也略有差异，可能有很强的特殊性，切勿以此为准。

由于是分段跑的，也没有计时，但差不多是在 20 分钟内跑完的。先算出来一个 seed 只能满足 state[0]，继续往后枚举，很快就找到了第二个 seed，也是这题真正的 seed，下面是 C++ 写的脚本：

```cpp
#include&lt;iostream>
#include&lt;cstring>
#include&lt;cstdio>
#define LL long long
using namespace std;
// seed1 = 1620671466
// seed2 = 1668245885
LL Tar=778501557,state[250];
LL mt19937(LL seed){
    memset(state,0,sizeof(state));
    state[0]=seed;
    for(LL i=0;i<232;i++){
        state[i+1]=1812433253 * (state[i] ^ (state[i] >> 27)) - i;
        state[i+1] &= 0xffffffff;
    }
    for(LL i=0;i<233;i++){
        LL y=(state[i] & 0x80000000) | (state[(i+1)%233] & 0x7fffffff);
        LL temp=y >> 1;
        temp ^= state[(i + 130) % 233];
        if(y & 1)temp ^= 0x9908f23f;
        state[i] = temp;
        return state[0];
    }
}
int main(){
    for(LL i=1;i<9000000000;i++){
        if(i%1000000==0)cout<<i<<endl;
        if(mt19937(i)==Tar){
            cout<<i<<endl;
            return 0;
        }
    }
    return 0;
}
```

得到的 seed2 经检验就是真正的 seed 啦。接下来的事情就很简单了，我们有了 seed ，就可以直接用题目中的 MT19937 进行计算还原所有的状态和随机数，随机数与密文逐个异或后的 MD5 与可显示字符的 MD5 值进行匹配就可以还原明文了。

```
from hashlib import md5
from binascii import hexlify, unhexlify
from Mersenne_twister import mt73991,XOR
from Crypto.Util.number import long_to_bytes

S=b''
for i in range(0x20,0x7f):
    S+=long_to_bytes(i)
S_list = []

def decrypt(key, cipher):
    tmp = XOR(key, unhexlify(cipher))
    for i in range(len(S)):
        if S_list[i] == tmp:
            return long_to_bytes(S[i])
    return b''

if __name__ == "__main__":
    for i in S:
        tmp = md5(chr(i).encode()).digest()
        S_list.append(tmp)
    seed = 1668245885
    random = mt73991(seed)
    ans = b''
    f = open("cipher.txt")
    cipher = f.read()
    L = len(cipher)
    for i in range(0, L, 32):
        cip = cipher[:32]
        key = b''.join([random.getramdanbits() for _ in range(4)])
        print("key=",key)
        ans += decrypt(key, bytes(str(cip), encoding="utf-8"))
        cipher = cipher[32:]
    print(ans)
```

> 2020.4.22:赛后出题人 shallow 师傅告诉我这题 seed 不用爆破可以推出来…已知明文的 “}” 刚好用到了第 103 个随机数。tql，我根本没想到能凑这么巧，我好菜啊… orz

programming-OI的梦

1 点到 n 点走 k 步的路径条数问题，很容易可以在网上找到类似的题目：https://blog.csdn.net/bbbbswbq/article/details/81177945

拿来代码改一改就可以用

```
#include <cstdio>
#include <iostream>
#include <cstring>
using namespace std;
const int N = 105;
const int mod = 10003;
struct node
{
    int a[N][N];
};
node c, ans;
int n;
node mul(node x, node y)
{
```

```
    node ans;
    memset(ans.a, 0, sizeof(ans.a));
    for (int k = 1; k <= n; k++)
    {
        for (int i = 1; i <= n; i++)
        {
            if (x.a[i][k])
                for (int j = 1; j <= n; j++)
                {
                    ans.a[i][j] += x.a[i][k] * y.a[k][j];
                    ans.a[i][j] %= mod;
                }
        }
    }
    return ans;
}
node Pow(node x, int m)
{
    node ans;
    memset(ans.a, 0, sizeof(ans.a));
    for (int i = 1; i <= n; i++)
        ans.a[i][i] = 1;
    while (m)
    {
        if (m & 1)
            ans = mul(ans, x);
        x = mul(x, x);
        m >>= 1;
    }
    return ans;
}
int main()
{
    freopen("yyh.in","r",stdin);
    freopen("out","w",stdout);
    int m, u, v, step;
    scanf("%d %d %d", &n, &m, &step);
    memset(c.a, 0, sizeof(c.a));
    while (m--){
        scanf("%d %d", &u, &v);
        c.a[u][v] = c.a[v][u] = 1;
    }

    ans = Pow(c, step);
    printf("%d\n", ans.a[1][n]);
    fclose(stdin);
    fclose(stdout);
    return 0;
}
```

flag：5174