CTF中的LFSR考点(一)

原创

沐一 · 林 于 2021-09-20 22:01:17 发布 569 收藏 8

分类专栏： 笔记 文章标签： ctf

本文链接：https://blog.csdn.net/xiao__1bai/article/details/120392307

CTF中的LFSR考点(一)

前提概要：

这是我在理解了 作者：道路结冰 的博客 深入分析CTF中的LFSR类题目（一） 下写的一次回顾和分析，只是在其中加上自己的见识和理解来加深印象。

> 博客地址：https://www.anquanke.com/post/id/181811#h2-0

.

.

前言：

LFSR（线性反馈移位寄存器）已经成为如今CTF中**密码学**方向题目的一个常见考点了，在今年上半年的一些国内赛和国际赛上，也出现了非常多的这类题目，但是其中绝大多数题目目前都没有writeups（或者writeups并没有做cryptanalysis，而是通过**爆破**的方法解决，这种思路只适用于部分类似去年强网杯出现的几道非常基础的LFSR类题目有效，对于绝大多数国际赛上的题目不仅是没有任何效果的，也是没有任何意义的，只有真正掌握了LFSR的密码学原理，才有可能在国际赛上解决一道高分值的LFSR类题目），网上针对这类考点的详细分析也不多，因此接下来我将通过几篇文章，对这类知识点进行一个详细的分析。

.

.

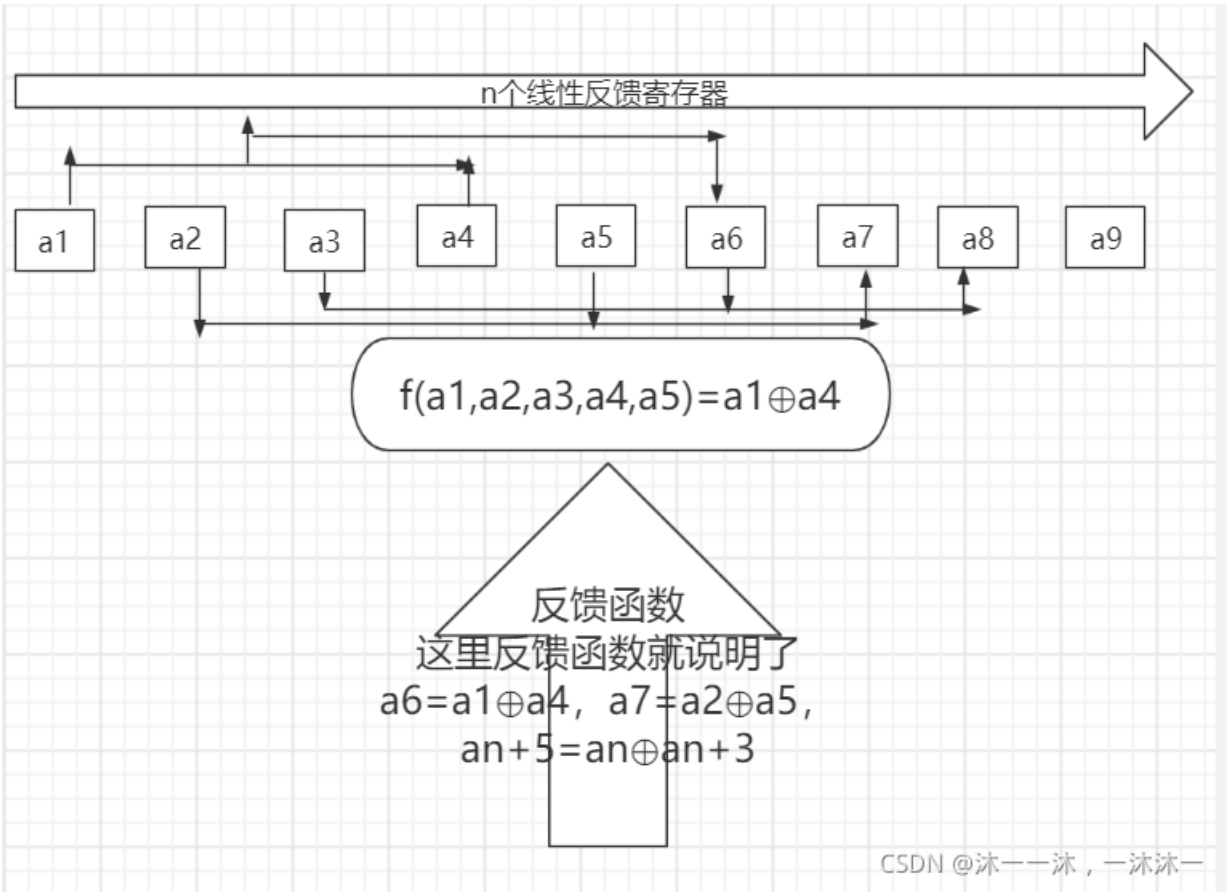
.

LFSR简介：

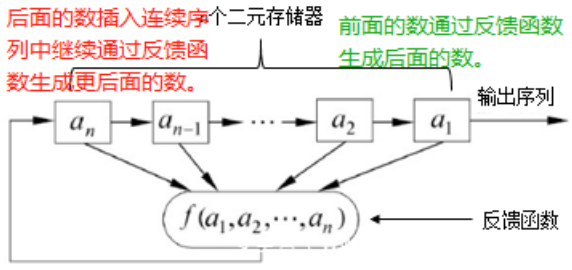
LFSR是属于FSR（**反馈移位寄存器**）的一种，除了LFSR之外，还包括NFSR（非线性反馈移位寄存器）。

附加：

反馈移位就是可以通过前面**已经存在**的寄存器中的值**反馈**出后面的的寄存器的值，通过不断移位对应不同的前寄存器值一直**反馈出**后面连续的后寄存器值。



FSR是流密码产生密钥流的一个重要组成部分，在GF(2)上的一个n级FSR通常由**n**个**二元存储器**和一个**反馈函数**组成，如下图所示：



.

如果这里的反馈函数是**线性**的，我们则将其称为LFSR，此时该反馈函数可以表示为：

$$f(a_1,a_2,\cdots,a_n)=c_na_1\oplus c_{n-1}a_2\oplus\cdots c_1a_n$$

其中cn=0或1，⊕表示异或（模二加），也就是说反馈函数必然为**a1……⊕……an**形式中的一部分。

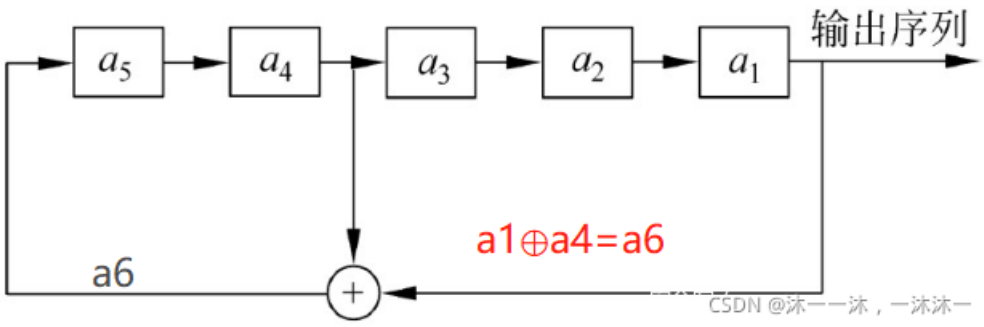
我们接下来通过一个例子来更直观的明确LFSR的概念，假设给定一个**5级**的LFSR，其**初始**状态（即a1到a5这5个二元存储器的值）为：

$$(a_1, a_2, a_3, a_4, a_5) = (1,0,0,1,1)$$

其反馈函数为：

$$f(a_1, a_2, a_3, a_4, a_5) = a_4 \oplus a_1$$

整个过程可以表示为下图所示的形式：



接下来我们来**计算**该LFSR的输出序列，输出序列的**前5位**即为我们的初始状态**10011**，第**6**位的计算过程如下:

$$a_6 = a_4 \oplus a_1 = 1 \oplus 1 = 0$$

第7位的计算过程如下:

$$a_7 = a_5 \oplus a_2 = 1 \oplus 0 = 1$$

由此类推，可以得到前31位的计算结果如下：

```
10011010010000101011110110001111
```

对于一个**n级**的LFSR来讲，其最大周期为**2^n-1**，因此对于我们上面的**5级**LFSR来讲，其最大周期为 **2^5-1=31**，再后面的输出序列即为前31位的**循环**。

注意：
这里的**级**就是能通过反馈函数生成**完整连续序列**的**最少**寄存器数，因为**a6=a1⊕a4**，所以a5必须包含在内，所以是5级寄存器。

通过上面的例子我们可以看到，对于一个LFSR来讲，我们目前主要关心三个部分：**初始状态**、**反馈函数**和**输出序列**。

那么对于CTF中考察LFSR的题目来讲也是如此，大多数情况下，我们在CTF中的考察方式都可以概括为：给出**反馈函数**和**输出序列**，要求我们**反推出初始状态**，初始状态即为我们需要提交的**flag**，另外大多数情况下，初始状态的**长度**我们也是已知的。

显然，这个反推并不是一个容易的过程，尤其当反馈函数十分复杂的时候，接下来我们就通过一些比赛当中出现过的具体的CTF题目，来看一下在比赛当中我们应该如何解决这类问题，由于不同题目之间难度差异会很大，所以我们先从最简单的题目开始，我将尽可能的用最通俗的语言和脚本来进行演示，在后面会逐渐提升题目的难度，同时补充相应的代数知识。

.

.

CTF例题演示

2018 CISCN 线上赛 oldstreamgame

题目给出的脚本如下：

```
flag = "flag{xxxxxxxxxxxxxxx}"
assert flag.startswith("flag{")
assert flag.endswith("}")
assert len(flag)==14

def lfsr(R,mask):
    output = (R << 1) & 0xffffffff
    i=(R&mask)&0xffffffff
    lastbit=0
    while i!=0:
        lastbit^=(i&1)
        i=i>>1
    output^=lastbit
    return (output,lastbit)

R=int(flag[5:-1],16)
mask = 0b10100100000010000000100010010100

f=open("key","w")
for i in range(100):
    tmp=0
    for j in range(8):
        (R,out)=lfsr(R,mask)
        tmp=(tmp << 1)^out
    f.write(chr(tmp))
f.close()
```

.

.

分析一下我们的已知条件（1）：

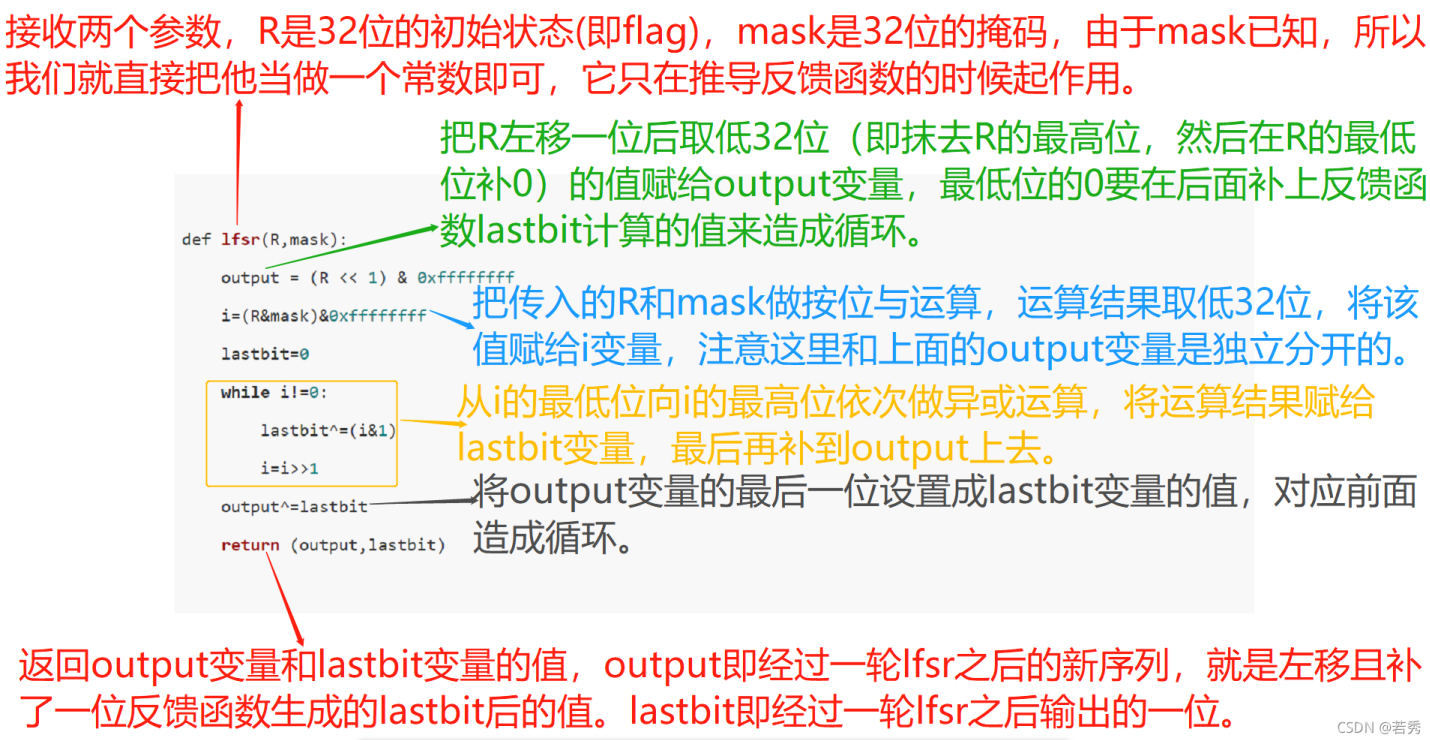
已知初始状态的长度为4个十六进制数，即32位，初始状态的值即我们要去求的flag，所以初步判断这里的 级 是32，那么最大周期就是 2^32-1 。

已知 反馈函数lfsr ，只不过这里的反馈函数是代码的形式，我们需要提取出它的数学表达式。 已知输出序列。

.

.

那么我们的任务很明确，就是通过分析`lfsr函数`，整理成`数学表达式`的形式求解即可，接下来我们一行一行的来分析这个函数：



.

.

通过上面的分析，我们可以看出在这道题的情境下，lfsr函数本质上就是一个**输入R输出lastbit的函数**，虽然我们现在已经清楚了**R是如何经过一系列运算得到lastbit**的，但是我们前面的反馈函数都是**数学表达式**的形式，我们能否将上述过程整理成一个表达式的形式呢？这就需要我们再进一步进行分析：

> mask只有第3、5、8、12、20、27、30、32这几位为1，其余位均为0。
> mask与R做按位与运算得到i，当且仅当R的第3、5、8、12、20、27、30、32这几位中也出现1时，i中才可能出现1，否则i中将全为0。
> lastbit是由i的最低位向i的最高位依次做异或运算得到的，在这个过程中，所有为0的位我们可以忽略不计（因为0异或任何数等于任何数本身，不影响最后运算结果），因此lastbit的值仅取决于i中有多少个1：当i中有奇数个1时，lastbit等于1；当i中有偶数个1时，lastbit等于0。
> 当R的第3、5、8、12、20、27、30、32这几位依次异或结果为1时，即R中有奇数个1，因此将导致i中有奇数个1；当R的第3、5、8、12、20、27、30、32这几位依次异或结果为0时，即R中有偶数个1，因此将导致i中有偶数个1。
> 因此我们可以建立出联系：lastbit等于R的第3、5、8、12、20、27、30、32这几位依次异或的结果。

.

.

将其写成数学表示式的形式，即为我们要求的**反馈函数**，反馈函数的形式也说明了初始位数要**32位**：

$$lastbit = R_3 \oplus R_5 \oplus R_8 \oplus R_{12} \oplus R_{20} \oplus R_{27} \oplus R_{30} \oplus R_{32}$$

.

.

.

然后我们看一下明文是怎么来的：

```
key=20FDEEF8A4C9F4083F331DA8238AE5ED083DF0CB0E7A83355696345DF44D7C186C1F459BCE135F1DB6C76775D5DCBAB7A783E48A203C
19CA25C22F60AE62B37DE8E40578E3A7787EB429730D95C9E1944288EB3E2E747D8216A4785507A137B413CD690C
```

最后八行代码，对 `flag`，它做了一百次循环，每次循环都产生一个结果，并将这个结果写到key里面去，所以 `key` 里面总共有一百个 `ASCII` 字符，共 `两百个` 16进制数。

.

补充：

题目之所以会出现 100 个ASCII字符是因为 4 循环次加内嵌的 8 位一次共 32 位循环往后产生的 4 个flag生成的加密字符共 8 个 16 进制数后，继续用这 4 个加密后的 flag 字符继续新一轮加密，就是多层加密。所以我们取 32 位即可，结果 flag 也是 4 个ASCII字符拆分出的 8 个 16 进制数。

.

```
f=open("key","w")
for i in range(100):
    tmp=0
    for j in range(8):
        (R,out)=lfsr(R,mask)
        tmp=(tmp << 1)^out
    f.write(chr(tmp))
f.close()
```

文件内是100个ASCII字符，这是前面提到的给反馈函数和生成的值反推初始值。
100个字符每个都是用lastbit补成的8位数生成的ASCII码对应的字符。

.

.

.

显然，`lastbit` 和 `R` 之间满足 `线性` 关系，那么接下来我们就可以开始求解了。

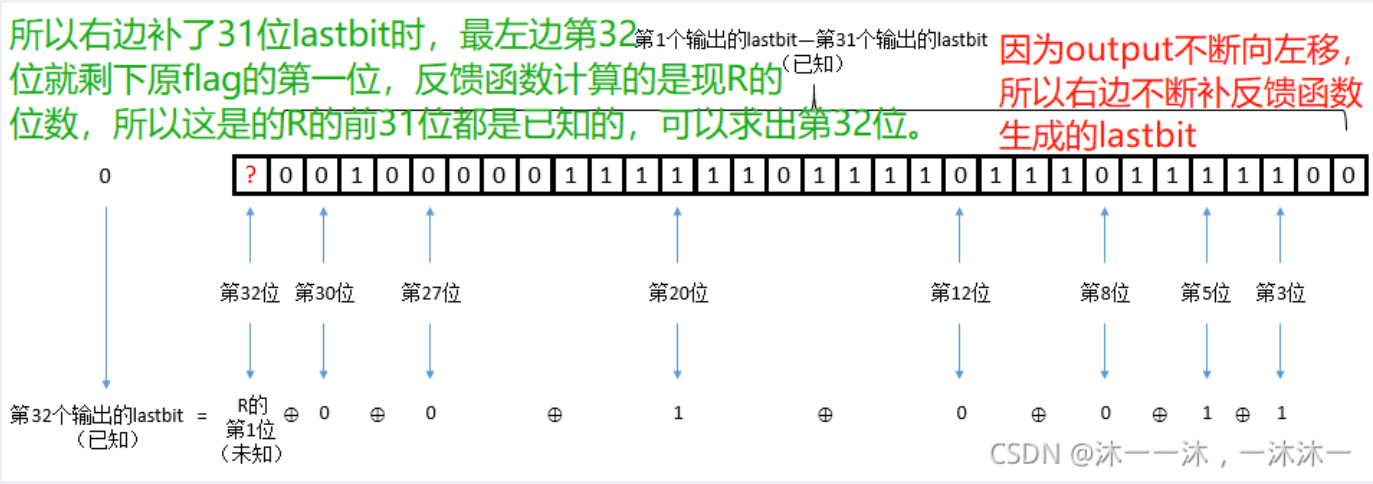
而且从这里我们可以知道，这种 `移位` 的CTF题目类型要 `反推32位` 的初始值要多少位 `输出序列` 呢，答案就是 `32位`，因为这种移位的题目是32位为一个循环，这就和我们前面说的 `2^32-1` 的循环不太同了，因为这里是 `移位` 操作。

```
32位Key值：00100000111111011110111011111000
```

解题的关键是发现在明文只剩 最后一位 时,可以通过最后的结果来求出排在 第32位 的第一位，然后就可以反推回去了。

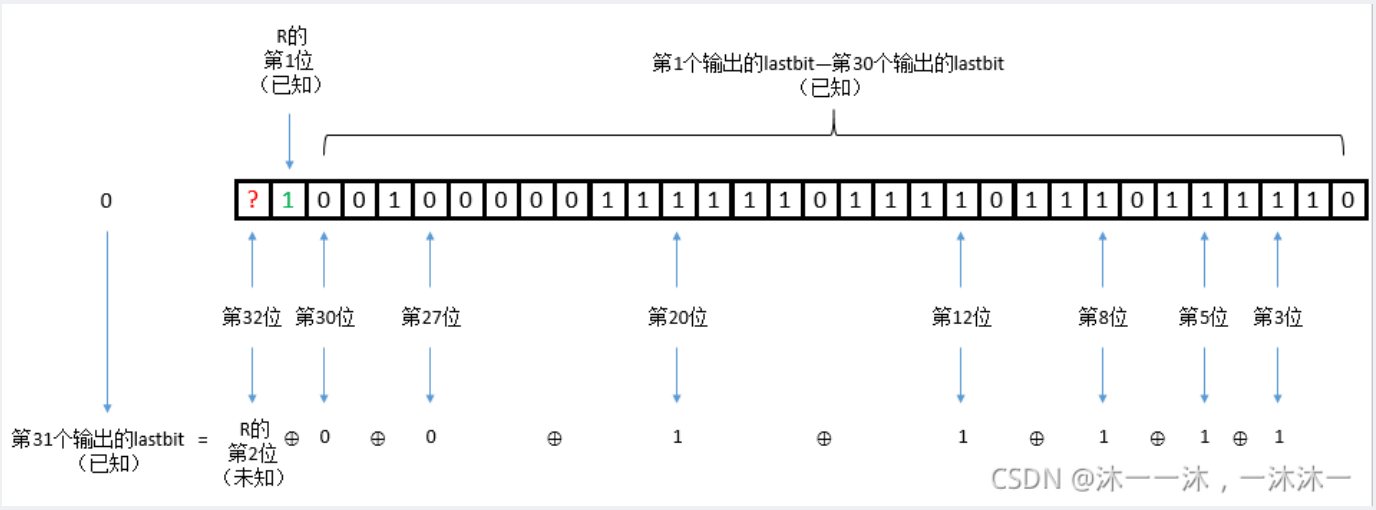
.

我们想象这样一个场景，当即将输出第32位lastbit时，此时R已经左移了31位，根据上面的数学表达式，我们有：



.

这样我们就可以求出R的第1位， 同样的方法 ，我们可以求出R的第2位：



以此类推 ，R的全部32位我们都可以依次求出了

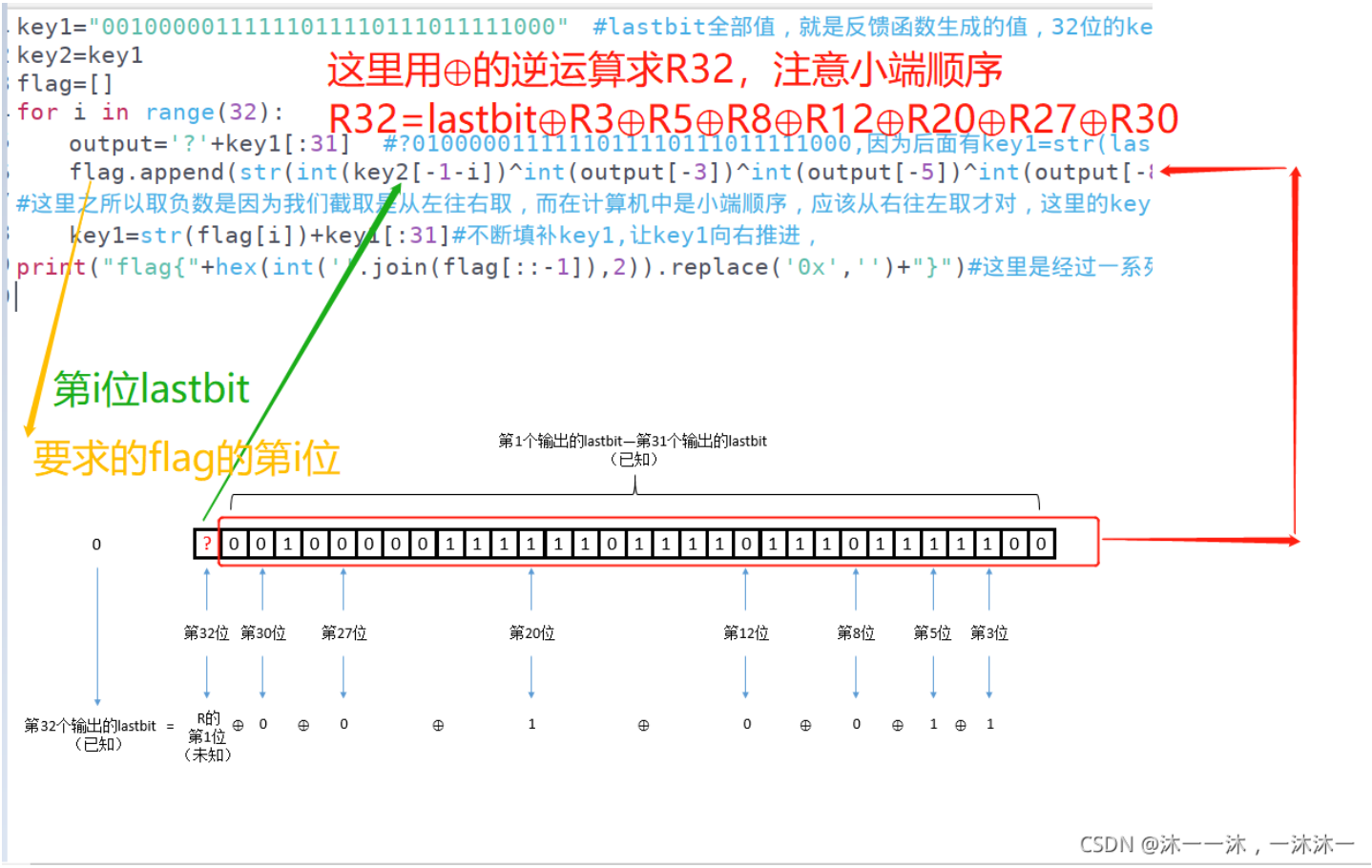
.

.

.

最终脚本，这里注意 小端顺序 的 反序 取位：

```
key1="00100000111111011110111011111000" #lastbit全部值，就是反馈函数生成的值，32位的key1
key2=key1
flag=[]
for i in range(32):
 output='?'+key1[:31] #?01000001111110111101110111111000,因为后面有key1=str(lastbit)+key1[:31]，key1不断填补，output不断取前31位，所以这里output每次把?定在第i位上，注意output和key1是独立的分开的。
 flag.append(str(int(key2[-1-i])^int(output[-3])^int(output[-5])^int(output[-8])^int(output[-12])^int(output[-20])^int(output[-27])^int(output[-30])))
#这里之所以取负数是因为我们截取是从左往右取，而在计算机中是小端顺序，应该从右往左取才对，这里的key2[-1-i]就是小端左到右的第i位，也就是前面分析的反馈函数生成值的第i位。flag值的原第i位,现在在第32位，可以通过⊕的可逆性来求，就是R32=lastbit ⊕ R3 ⊕ R5 ⊕ R8 ⊕ R12 ⊕ R20 ⊕ R27 ⊕ R30
 key1=str(flag[i])+key1[:31]#不断填补key1,让key1向右推进，
print("flag{"+hex(int(''.join(flag[::-1]),2)).replace('0x','')+"}")#这里是经过一系列操作，首先把flag变成小端顺序的[::-1]，然后就是转十六进制。
```



.

.

结果：

```
└─$ python 2.py
flag{926201d7}
```