2020CISCN初赛-crypto-lfsr writeup

原创

lnjoy 于 2020-09-15 21:40:35 发布 519 收藏 5
分类专栏： CTF WriteUp 文章标签： 信息安全 密码学 lfsr

本文链接：https://blog.csdn.net/qq_43531895/article/details/108599913

第十三届全国大学生信息安全竞赛初赛，密码学的lfsr详解。

> 原题文件：https://wwe.lanzous.com/iDF95gntw5a

本题考查线性反馈移位寄存器。

初始状态，反馈函数，输出序列都已给出，已知n是100。

输出序列即题给的output.txt。
反馈函数是代码形式，需要经过分析后转化为表达式形式。
通过分析反馈函数可以发现，初始状态即是输出序列的前100位倒置。

lfsr源代码如下，

```
def lfsr(state, mask):
    feedback = state & mask
    feed_bit = bin(feedback)[2:].count("1") & 1  #当feedback中有奇数个1时，feed_bit为1
    output_bit = state & 1  #取state的最后一位
    state = (state >> 1) | (feed_bit << (N-1))  #state右移一位，将feed_bit反馈到state的第一位
    return state, output_bit
```

其中feed_bit是反馈位，当feedback中有奇数个1时，feed_bit为1。
因为0异或任何数等于任何数本身，不影响最后结果，则当feedback中有奇数个1时，最低位向最高位依次做异或运算的结果是1。

因此反馈位feed_bit就是将当前状态的每一位与mask中对应位做位与，并将每一位计算结果依次异或。

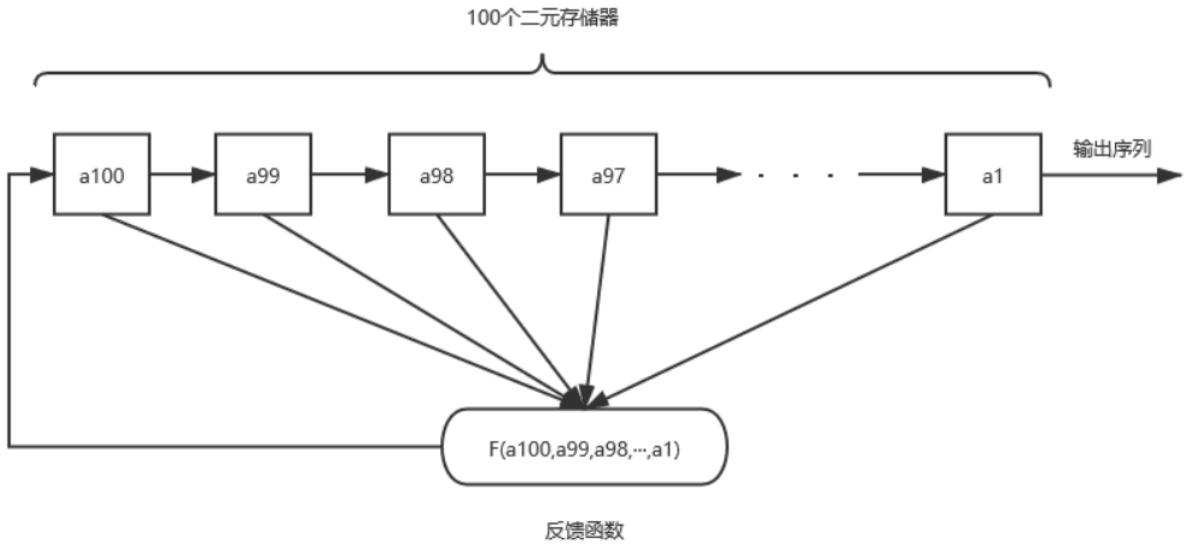
设lfsr每一位依次为a1到a100，mask中每一位依次为b1到b100，可得反馈函数如下：

$$feed\_bit = b\quad \&a\quad \oplus b\ \&a\ \oplus b\ \&a\ \oplus b\ \&a\ \oplus$$

output_bit是输出位，取lfsr当前状态的最后一位输出。

state表示当前状态，每次state右移一位，并将feed_bit反馈到state的第一位。

整个lfsr流程如下图：



因此我们由输出序列可以知道lfsr初始状态，和其后的每一个反馈位。

本题将flag设为mask，即要求mask。

根据反馈函数，可知mask的每一位都是反馈函数的系数。
而反馈函数的值与参数都已知，我们可以通过列100个反馈函数表达式，利用矩阵解线性方程组，计算得到100个系数的值。

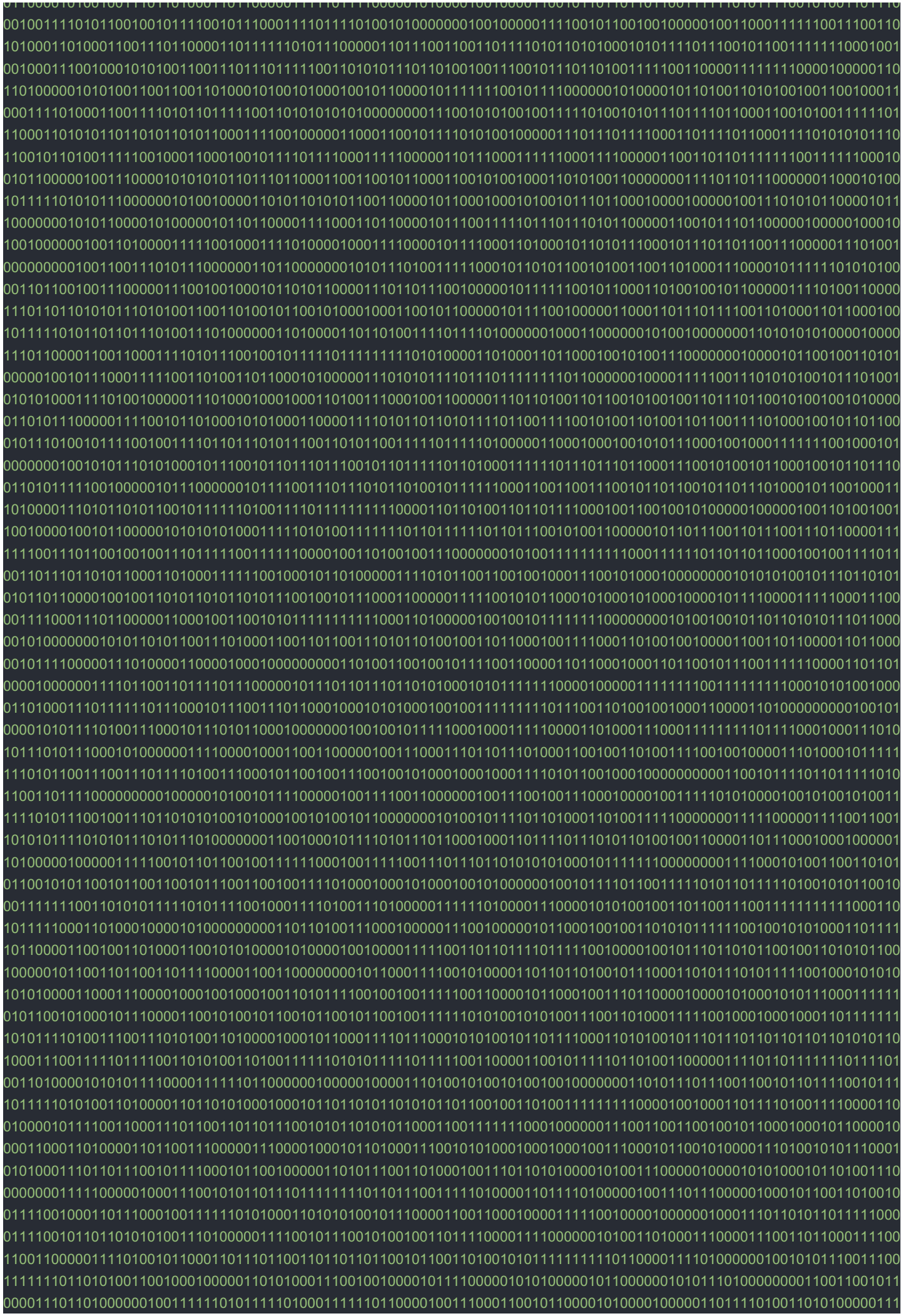
由于在有限域GF(2)内，只有0和1，此时乘法相当于异或，加法相当于与运算。
因此可直接将公式中的异或（⊕）替换为乘法（*），将与运算（&）替换为加法（+）。

计算原理如下（c表示输出序列）：

sage脚本为：

```
from sage.all_cmdline import *
import hashlib

output = '01001100111011110111101101010011100101001010000111111011011110101111001111101001001011101110011011101100110000101001001110110100011011000001111101111000001010000100100001100101101101101100111111010111100101001101110
```



```
0000000010000010110101001010111110110011011011100001110011100101001010010100011101100100111001111110001001001011
0100111101100110110111001011110010000111011111010101000010001101110011111010110011100010001111111011100110100011
1101100011010011100010100011100111101011111010111011000001110110111011100001101111010100010000111100110111011000
1110010111010000010001011011001101010101100001001011000000010001001100111001001111001011011100111101101010111101
1000000111110000011011100100010011110010110000001111110111000100111010100110010111100010000110101011110011010110
0110010110110100101110010000000000010010101010100011000001101111010000000101001000101001010111111100110111001111
0100111010101000110000010010011011111000111100001110001011000110110101001100101111110101100011011000011111110110
0000000010000100100100111010010001111000000101110110110010011001100001010101100010110000010110101110001111100111
0110101010100111001001001000100000011010011010101111100101001010000011001111101111111100001011111001100011101110
1011100111111011110000011101101101011101100111101010001000011101111111100010011011000110000011010100000101001001
0100111011100000011111110001000010011010101001111010111100011110110110111000000111101010110001100000100001101101
0011010101111110001101101100101101101001000100000100110111111010011010011111010110110111101001000100010111101100
11010111011111101001111000010000110010110111111001010100000001011111010000001111110100000000101101111111110010011010
1101011100101101001011010001100010101101100000101000001110110101010100001011100010000111010011101110000000010000
0000000101111001110000100000001101100110111011100000011110111000111011111000110000110010001001110101010110101011
1001101001101101101010001101100011011110100111001111000010001001100001000000110101001001100011100000000010110001
10100010001000101001001000110001011101000100011111010010101000010000001100100111010111010000000100010000011010110
10011111010111000111100100000001010111010101111101101001000110100000011010111111100100111110110111011111101110000010
10111100010001001001100000000110011100001100100111010011111011001010001101111111100001011010100001110000101011010
0101110111001001100000000110110101011001110000011001111100001101110100101101111100110001010011011001010110111000
1000100001100011111011100111000011010001001100001110001100010110111101000101010110110010011111010011100111000011
1000101010001100111011010110100110101010100111001111000101110101110010000110110011011010101011101010110011100000
0001011010111011101000000101110111110010000000000101001110011111011001110001100010111001010010110011100001000011
11110111110110011100010101110001100000110000000000001000011111101001010111010101110001010000100000001001111110101
11110010010111010110010100111000010111100000000010001011101001001101001100110001000100000110011010010111010011011
011110101000110000101001111101110000000101101101100000000100101100110100101100111001010100010001011010001010011110
10011010100011000111000010011111100100011001111001011100011001000111111111101011110100111101001001101110110011111
1100011001011100001101111100111010110110\
    10000110101010110101111010111011110111001010011111110001011001000000110001001010010110010111100011010011001010
01'

list1 = [int(i) for i in list((output[100:200]))]
y = vector(GF(2),list1) #值向量
list2 = []
for i in range(100):
    list2.append([int(j) for j in list(reversed(output[i:i+100]))])
x = matrix(GF(2),list2) #参数矩阵
mask = x.solve_right(y) #解方程x*mask=y
mask = ''.join([str(i) for i in list(mask)])
flag = int(mask,2)
print(flag)
```

得到结果

则flag为 `flag{856137228707110492246853478448}`