序列密码的代数攻击

李浩然

PB13210007

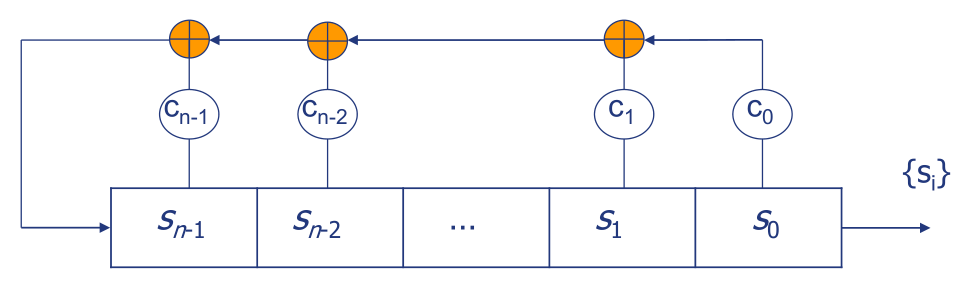
# 实验目的

1. 理解利用LFSR和滤波函数f生成伪随机序列的原理。
2. 理解代数攻击的原理。
3. 构造LFSR和f，并得到一定长度的密钥流。以此为基础进行代数攻击得到初始密钥。

# 实验原理

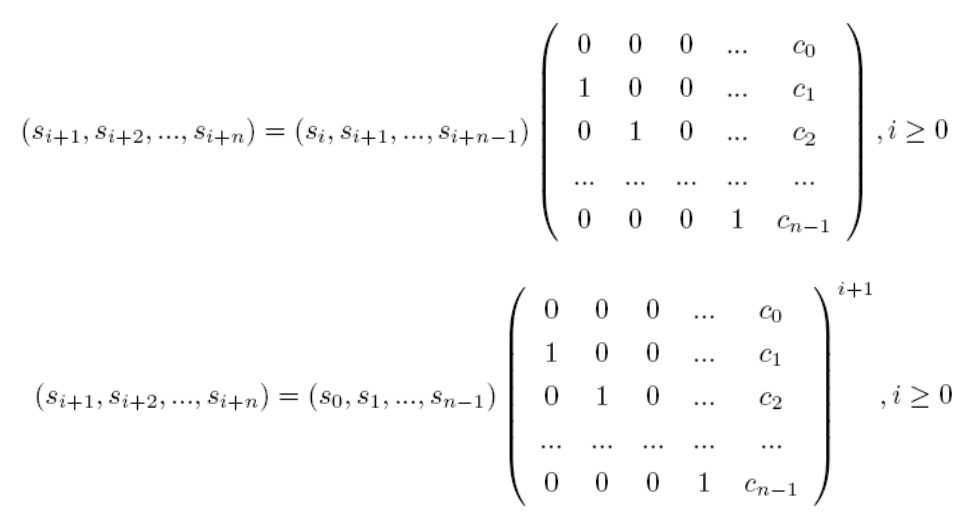
本实验主要包括两个方面：密钥流的生成和对该密钥流的代数攻击。

## 线性反馈移位寄存器（LFSR）



LFSR包括两个部分：当前状态和反馈系数。LFSR通过将当前状态和反馈系数求内积得到新的一位状态。

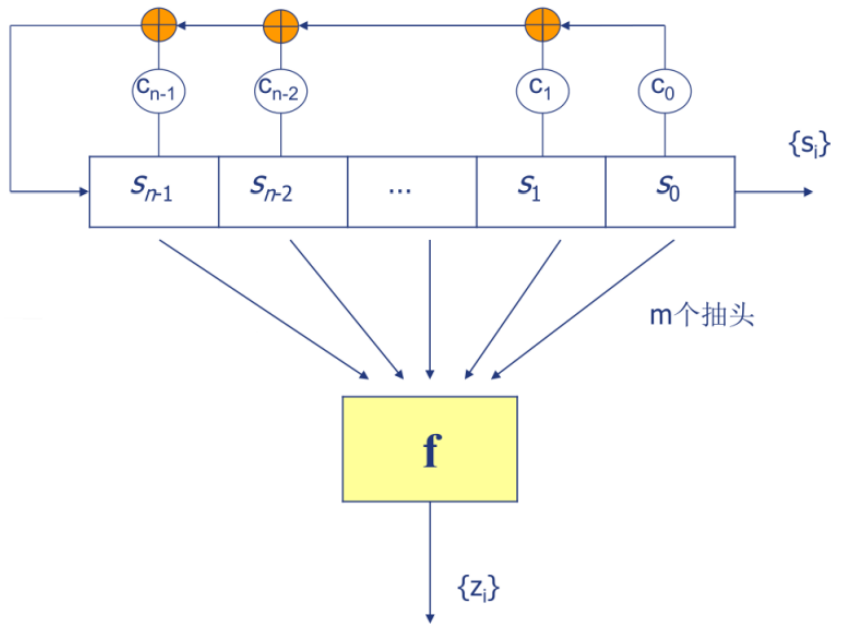
内积操作可以用矩阵来形式化地描述为：



## 滤波函数

因为LFSR生成的序列具有高度线性性，若是直接拿来作为密钥无法保证安全。所以最终的密钥流一般会利用非线性运算综合多个这种序列，或者对LFSR中不同时刻的状态计算一个f值得到密钥序列。这个f就是滤波函数。

这个f可以是LFSR中所有比特位的任意组合的加和运算，它将n比特作为输入，得到1比特的输出。



*（其中zi即为最终的到的密码序列）*

## 代数攻击

假设，攻击者可以了解整个系统，并且得到了一定长度的密钥序列。这时候若想要得到初始密钥，只需要得到LFSR密码序列的任意长度为n的连续一段即可逆推得到初始密钥。但由于f是非线性的，重密码序列反推LFSR的状态是有很大难度的。这也是这种流密码的安全保障的来源。

然而，如果f函数的设计不具备：平衡、高的代数次数、高的非线性度、高的相关免疫阶、良好的雪崩特性这些性质，那么攻击者可以通过将一个次数较低的函数g乘到f上，使得得到的多项式次数远低于f的次数。这样再对方程线性化，然后就可以通过解方程很快地攻破这个系统。

# 实验内容

## 攻击类型

本实验中主要讨论攻击函数g的次数不大于2，并且f和g的乘积恒等于0的情况。

### 函数g的次数为1

如果f能被表示为：，其中f1表示次数不超过1且不恒为1的多项式，f’可以是任意函数。

那么构造g=f1+1，则有：



其中，所以有：



又这是在GF(2)上的运算，所以有：



### 函数g的次数为2

如果f能被表示为：，其中f1和f2表示次数不超过1且不恒为1的多项式，f’和f’’可以是任意函数。

那么构造g=(f1+1)( f2+1)，则有



由g是1次函数的讨论可知，最后这两项的第一个括号中的式子均恒等为零，所以有：fg≡0。

## 程序实现

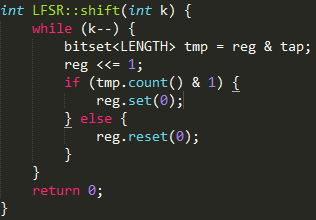
本次实验的程序主要分四个模块：LFSR、滤波函数、高斯消元和主函数。这些模块各自负责相对独立的工作，每个都被创建为了独立的类，由主函数协调调度。

### LFSR

LFSR模块使用两个bitset类型的变量表示LFSR当前的存储状态和反馈系数。

不同于之前的定义，程序中将LFSR中的比特位从左到右递增编号。这样新产生的密码位会出现在新状态的0位上，比较符合计算机的存储逻辑。

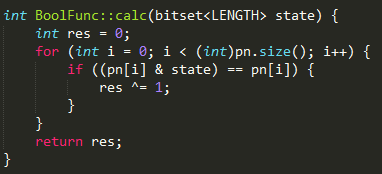
主要的移位更新操作如下图所示：



### 滤波函数

本次实验中会涉及到两个滤波函数：f, g。并且这两个滤波函数的定义域还有较大区别。

f函数的作用是“正向”生成密码序列，所以是知道LFSR中每一位的具体取值的。这样一来，f的运算过程就相对简洁：



但是g函数是攻击者构造出来想要借此来破译出当前确切状态的。所以g的定义域是最终要求的未知量。

但之前提到过，LFSR是线性的变换，所以未来的任意时刻都可以用初始状态和转移矩阵若干次幂的乘积来表示。因此，g的定义域是一个矩阵，表示每个变量是初始状态中各个分量的怎样的线性组合。

具体实现中，还要考虑到逻辑与运算上的分配律等，较为复杂：

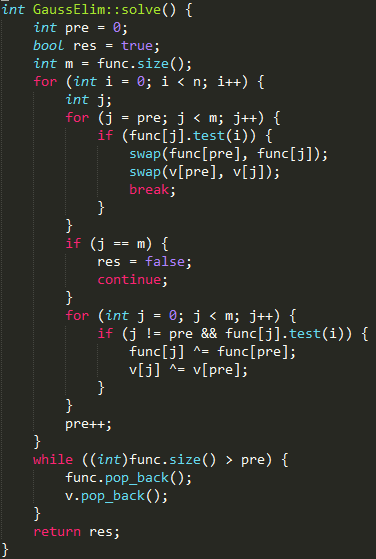


大致流程是用两个bitset类型变量tmp[pre]和tmp[now]滚动记录上一个分配律后的结果和存储当前进行中的结果。

左半部分先定位到连续的与运算中的第一个分量记录到tmp[pre]中。右半部份继续往下找其他分量。

### 高斯消元

高斯消元利用的是基本的找对角线的算法。时间复杂度为O(n3)。但因为利用了bitset的紧凑存储结构，所以在实际效率上会有6到8倍的常数优化效果。具体实现如下：

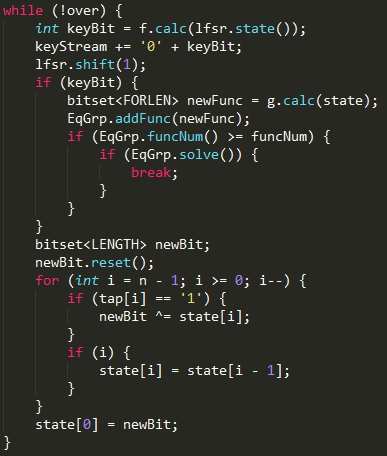
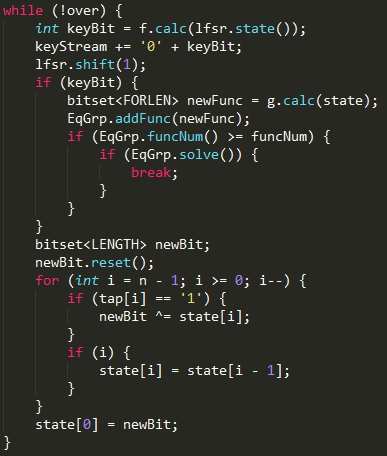


### 主函数

主函数的算法流程如下：

1. 读入配置文件中：LFSR的长度、反馈系数、f函数和g函数。
2. 随机生成一个长度和LFSR相同的密钥作为LFSR的初始状态。
3. 生成一个新的LFSR状态位，并计算它对应的初始状态的线性表示。
4. 计算当前LFSR状态的f值得到新的密码位。
5. 如果这个密码位为1，则计算当前LFSR的状态在g函数上的值，得到一组初始状态的线性关系，将其加入方程组中。
6. 如果方程组的数量不少于未知数的数量，那么开始解方程，否则回到步骤3。
7. 如果方程组有多解则回到步骤3，否则说明解出了初始密钥，输出推出程序。

循环部分如下图所示：



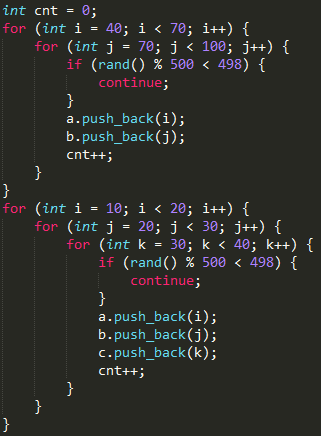
左半部分生成新的状态位和密钥比特，再尝试解方程。右半部份维护更新每个状态位对应的初始状态的线性表示。

# 实验结果

实验中用到的LFSR反馈系数经过验证能产生周期最大的密码序列。

|  |  |
| --- | --- |
| LFSR长度 | 反馈系数的多项式表示 |
| 30 | X30+ X6 +X4 +X+1 |
| 50 | X50+ X49 +X24 +X23+1 |
| 100 | X100+ X63 +1 |

实验中的f函数利用函数生成器随机生成若干3~4次和一个10次项，算法如下：



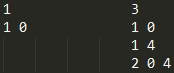
其中a, b, c容器里对应的位置合在一起表示了一组随机出来的项。

这项的特点是，二次项在输出的时候都会再乘上x4，三次项则会再乘上x0。这样保证了最后得到的f满足上一节分析的形式：每个项至少包含x4 , x0两个变量中的一个，从而可以用这两个变量如上一节描述的方法构造二次多项式g，使得fg=0。

对于一次g函数的构造，采取了直接将上述数据中四次项删去的做法，留下的项都有x0这个因子，符合上节描述的情况。

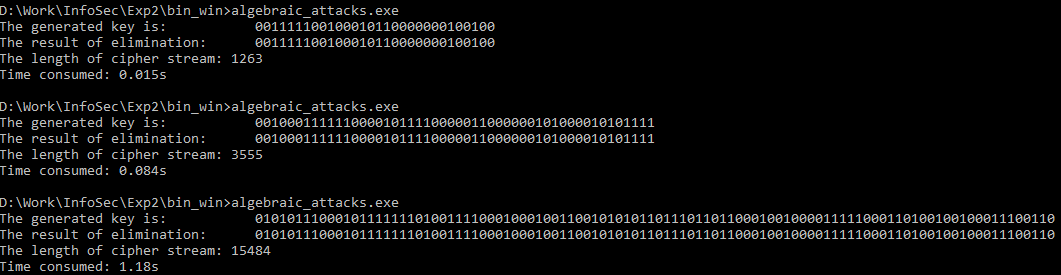
具体数据可以在sample文件夹中查看，里面包含了LFSR，f，g三者的信息。描述滤波函数的方法是先指定这个函数有多少项相亦或，接下来的几行第一个数字指定这个项由几个变量相与而成，紧接在后面的就是对应的这几个变量。例如：

g函数只有两种形式：g=x0+1和g=(x0+1)(x4+1)=x0x4+x0+x4+1，他们分别对应的输入形式是：



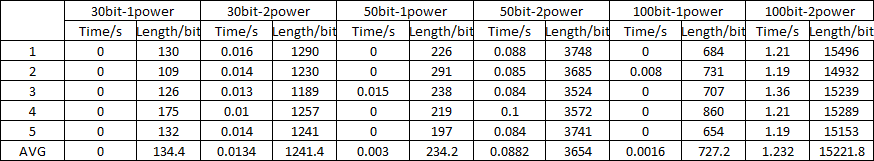
其中常数项因为存在于所有g函数中，故不在输入中描述，直接作用到运算中。

运行结果如下：



这是程序在30，50，100三个不同LFSR长度、f函数包含15~20个三次项或四次项和一个10次项、g函数最高次数为2的情况下的表现。可以看出，最慢 的情况也只需要1s多一点的时间并且仅需不到1K的密码流。

以下是不同LFSR，在g函数分别是1次和2次的时候的几组测试数据：



可以看出，虽然整体时间空间效率都很高，但是g函数的次数的提升会极大地影响时空效率，从1次函数到2次函数的改变会导致10倍以上的效率下降。

# 实验结论

流密码的安全性很大程度上依赖于f函数的选择，若是部分（或全部）项之间含有公因子，那么就会面临代数攻击的严重威胁。从本实验可以看出，糟糕的f函数选择以及合理的g函数选择可以在短短1秒的时间内攻破100bit长的序列密码。

所以，在设计滤波函数的时候，应该尽量选择次数高的、相互之间没有公因子的项。

# 实验总结

1. 随着f函数项数的增多，攻破系统的时间会降低。反之如果项数过少，很可能会存在无解的情况。这是因为，每一项都有一个等于0和不等于0的概率，因为LFSR的生成序列随机，所以每个单一变量的上述概率均为1/2。但是多个变量相与的时候，等于0的概率就指数上升了。可以证明，这个时候，多个变量相互亦或可以提高不等于0的概率。又因为只有在密钥位不等于零的时候才得到一个方程，所以这样增大了能够解出方程的概率。
2. 在新加入方程的时候动态地（当方程个数等于未知数个数的时候）消元，可以保证方程组始终维持一个类梯形的结构，还能动态地剔除线性相关的方程，节约空间。这样一来，之后的消元就会逐渐退化到平方，甚至是线性量级。期望情况是最终的总复杂度趋近于三方的量级，而不是三方乘上消元次数。