第3章 流 密 码

3.1 伪随机序列的生成

从上一章的讨论可以发现,完善保密的局限性在于需使用足够长的随机比特串作为密钥,而用自然的方法产生的随机比特流难以保证效率和安全性。因此人们更倾向于寻找一种短密钥扩展方法,使加密时能够生成近似随机比特流。这就是现代密码学的基础。

3.1.1 一般方式

大多数的计算机均支持生成随机数,如标准C语言库函数中的rand( )函数可产生介于0～65 535之间的任意一个伪随机(即看似均匀)数,这个伪随机函数需要一个“种子”(seed)作为输入,之后就可产生一个比特流的输出。这类伪随机序列发生器一般都建立在线性同余生成器(linear congruential generator)的基础之上,它可以根据以下关系式产生一系列数。

$$x_n \equiv ax_{n-1} + b \pmod{m}$$

设x_0是初始值,a、b和m是关系式中的参数。但是这类伪随机序列发生器所产生的伪随机序列无法满足密码学的要求,较适合以实验为目的的情况,因为它们依然是可被预知的比特流。对于已经知道的任何多项式同余生成器都存在潜在的不安全性。

为产生不可预知的比特流作为输入源,常用两种比特流创建方法,分别是利用单向函数和数论中的难解问题(intractable problem)。

单向函数是指那些在已知y和$y = f(x)$的情况下依然不能求解x的函数。假设存在一个单向函数f和一个随机的输入参数s,$y_j = f(s+j)$,$j=1,2,3,\cdots$,若令b_j是y_j的最低有效比特,那么序列$b_0,b_1,\cdots$将是一个伪随机的比特序列。运用这种方法的最具代表性的加密算法就是DES安全散列算法。

而解决数论中的难解问题的方法中人们最常用的密码安全伪随机序列生成器之一是平方剩余伪随机序列生成器(Blum-Blum-Shub,BBS),即人们所说的二次剩余方程生成器。BBS是一种流行的产生安全的伪随机数的方法,是由它的研制者的名字(Blum L,Blum M和Shub M)命名的。该算法首先产生两个大的素数p和q,它们都同余于3模4,设$n=pq$,且存在一个随机的整数x与n互素,则为了初始化BBS生成器,设初始输入是$x_0 \equiv x^2 \pmod{n}$,BBS通过如下过程产生一个随机的序列$b_0,b_1,\cdots$满足

① $x_j \equiv x_{j-1}^2 \pmod{n}$;

② b_j 是 x_j 的最低有效比特。

BBS 生成器的安全性是基于分解 n 的难度,很可能是不可预测的,但是它的计算速度慢,在有些情况下人们可能更在意加密的效率而非安全性。因此在保证应有的安全性时可以减少 k 个 x_j 中的最低有效比特,只要 $k \leqslant \log_2 \log_2 n$,这样应该也能保证加密的安全性。

3.1.2 线性反馈移位寄存序列

在很多加密情景中,人们需要考虑到存储开销、计算效率与安全性的平衡。线性反馈移位寄存器(linear feedback shift registers,LFSR)具有满足该场景的多项优点:高计算性能、低实现开销以及统计性良好的生成序列。例如,可用如下一个线性递归关系表示 LFSR。

$$x_{n+5} \equiv x_n + x_{n+2} (\mathrm{mod}\ 2) \tag{3-1}$$

并设置式中变量的初始值为

$$x_0 = 0, \quad x_1 = 1, \quad x_2 = 0, \quad x_3 = 0, \quad x_4 = 0 \tag{3-2}$$

则将式(3-2)代入式(3-1)并重复 31 次后即可得到如下序列。

01000 01001 01100 11111 00011 01110 10100 0

一般一个长度为 m 的线性递归关系(系数 $c_0, c_1, \cdots$ 是整数)可表示为

$$x_{n+m} \equiv c_0 x_n + c_1 x_{n+1} + \cdots + c_{m-1} x_{n+m-1} (\mathrm{mod}\ 2)$$

若给出了初始值(initial value,IV)为

$$x_0, \quad x_1, \quad \cdots, \quad x_{m-1}$$

则序列$\{x_n\}$的所有值都可以由递归计算出来,这个由 0 和 1 组成的结果序列可被用作加密的密钥。

这种方法可以用短的种子密钥生成一个周期非常长的加密密钥,这个长周期是相较于维吉内尔密码的改进。上例中,***IV*** 和线性递归关系式系数分别为|0,1,0,0,0|和|1,0,1,0,0|。这表示可以存储 10 比特产生 31 比特的伪随机数序列。对于精心选取的长为 31 的线性递归关系,任何非零 ***IV*** 均可产生一个周期为 $2^{31}-1=2\ 147\ 483\ 647$ 的序列,这是相当可观的。

人们可用被称为线性反馈移位寄存器的硬件以实现上述过程,自动而快速地产生随机序列。图 3-1 描述了一个简单的线性反馈移位寄存器的工作原理,它由三个寄存器和两个异或运算器组成。

在图 3-1 所示的线性反馈移位寄存器中,每增加一次计算,每个盒子中的比特被移到另一个盒子中作为输入,用异或运算表明其引入的比特的模 2 加法。比特 x_m 即输出和明文的下一个比特异或产生的密文。图 3-1 中,当 x_0, x_1, x_2 的初始值被确定后,这个运算器就可以高效率地产生随机比特流。

但是,这种加密方法容易受到已知明文的攻击。一旦得知一段连续的明文比特及其相对应的密文比特就能异或得到相应的密钥,再通过求解线性方程组就可以确定递推关系,因此由递推关系和部分密钥就可以计算出密钥的所有比特。

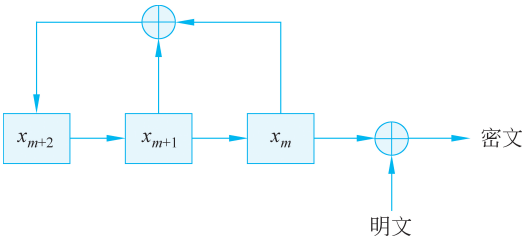


图 3-1　一个满足 $x_{m+2}=x_{m+1}+x_m$ 的线性反馈移位寄存器

3.2 流密码

流密码一般由初始化和密钥生成两个阶段组成，其主要包含密钥加载算法与迭代算法。初始化阶段首先调用密钥加载算法将短密钥 k（有时也被称为种子）以及可选的初始化向量 ***IV*** 加载到密码的内部状态中。然后多次调用迭代算法来更新内部状态，形成生成密钥流的初始状态 st。初始化阶段的目的是混淆，使攻击者无法确定 k、***IV*** 与 st 的关系。然后在密钥生成阶段重复调用迭代算法并输出近似随机的无限比特流。无 ***IV*** 的流密码应该像伪随机生成器一样，即当密钥 k 是均匀的时，生成的比特序列应该与一串独立均匀的比特无法区分。当流密码使用 ***IV*** 时，它应该像伪随机函数一样工作，也就是说，只需改变初始化向量 ***IV*** 的值而输入 k 保持不变，就能够生成一串序列与独立均匀的比特序列。

LFSR 输出比特之间的线性关系导致了采用该部件设计的流密码无法抵抗相关攻击、代数攻击等方法。为了阻止这些攻击，必须引入一些非线性，即使用秘密值的与操作而不仅是异或操作。以下是常用的几种增加非线性度的方法，在一些流密码方案中这些方法也会组合出现。

非线性反馈：使用高于一次的反馈函数作为反馈移位寄存器（feedback shift registers，FSR）。换句话说，如果 t 时刻状态为 $x_{m-1}^{(t)},\cdots,x_0^{(t)}$，那么下一时钟的状态为

$$x_i^{(t+1)}=x_{i+1}^{(t)},\quad i=0,\cdots,m-2$$

$$x_{m-1}^{(t+1)}=f(x_{m-1}^{(t)},\cdots,x_0^{(t)})$$

其中，f 为精心选取的非线性函数，其目的是使攻击者难以通过某时刻的部分状态计算出其他时刻的状态。使用此设计的流密码有 Grain 系列、Trivium 等。

非线性滤波：在输出序列中产生非线性。也就是寄存器状态仍然可以用 LFSR 更新，只是每个时钟的输出值是当前状态的一个非线性函数 g，而不仅是最右侧的寄存器值。采用此设计的流密码有 SNOW、ZUC 等。

非线性组合：以多个独立的 LFSR 输出作为一个非线性布尔函数 g 的输入，以各个 LFSR 的输出作为密钥流的输出。这个方式被称为非线性组合器。采用此设计的流密码有 E0。

当然，使用以上方法得到的密钥流还需要满足平衡性（0 与 1 的个数几乎相同）、长周期、高非线性度（与线性函数的输出序列相似性低）等性质。下面将介绍三种不同结构的流密码：基于非线性反馈移位寄存器的 Trivium、基于置换的 ChaCha20 和基于非线性滤波的 ZUC。

3.2.1 Trivium

2008 年欧洲完成的 eSTREAM 新型流密码项目的最终方案有 3 个适用于受限硬件环境,即 Grain v1、MICKEY 2.0 和 Trivium。其中,Trivium 流密码不仅结构简单而且硬件实现紧凑[10],其结构如图 3-2 所示。首先,方案的主要器件只有三个级数分别为 93、84、111 的 NFSR,记为 A,B,C。其次,Trivium 的输出序列仅为 6 个寄存器的异或,即在每个 FSR 中选取最右侧寄存器和另一个寄存器:$Z^{(t)}=A_{66}^{(t)}\oplus A_{93}^{(t)}\oplus B_{69}^{(t)}\oplus B_{84}^{(t)}\oplus C_{66}^{(t)}\oplus C_{111}^{(t)}$。

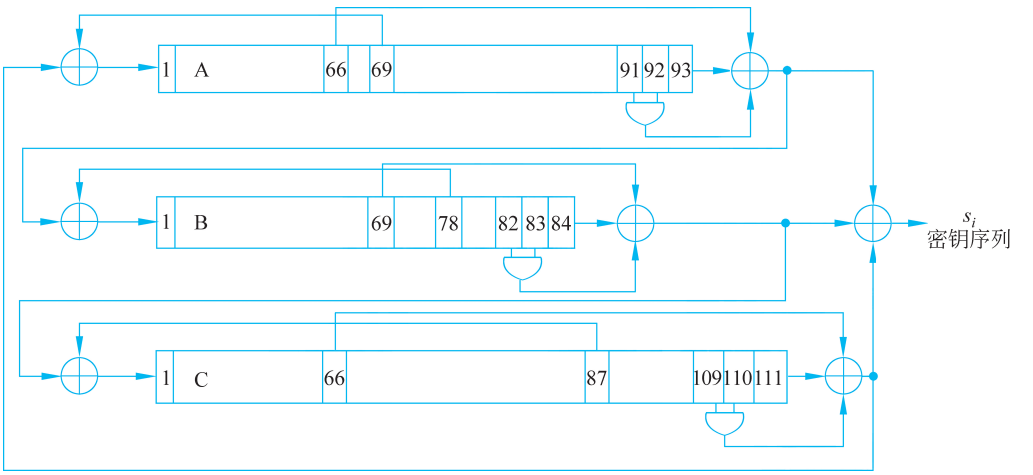


图 3-2 Trivium 内部结构图

Trivium 的内部是互相耦合的:每个时钟周期,每个 FSR 最左侧寄存器的更新值是由同一个 FSR 中的某一个寄存器和另一个 FSR 部分寄存器共同计算得到[11]。三个非线性反馈函数具体如下所示。

$$A_1^{(t+1)}=C_{66}^{(t)}\oplus C_{109}^{(t)}\wedge C_{110}^{(t)}\oplus C_{111}^{(t)}\oplus A_{69}^{(t)}$$
$$B_1^{(t+1)}=A_{66}^{(t)}\oplus A_{91}^{(t)}\wedge A_{92}^{(t)}\oplus A_{93}^{(t)}\oplus B_{78}^{(t)}$$
$$C_1^{(t+1)}=B_{69}^{(t)}\oplus B_{82}^{(t)}\wedge B_{83}^{(t)}\oplus B_{84}^{(t)}\oplus C_{87}^{(t)}$$

Trivium 的初始化算法接受 80 比特密钥和 80 比特 ***IV***。首先,密钥和 ***IV*** 分别加载到 A 和 B 的左 80 个寄存器中,对于 C 的 3 个最右寄存器被赋值为 1,其余寄存器设置为 0。然后运行 Next 4×288 次时钟周期(丢弃输出),并将结果状态作为初始状态。

流密码的侧信道攻击

侧信道攻击(side-channel attack,SCA)指攻击者利用密码算法与其所运行的物理设备交互过程中泄露的信息来展开攻击。近几十年来,该攻击相关研究主要集中在针对分组密码方案来提高攻击效果或提出新颖的抗攻击方法。2022 年,S. Kumar 等组合机器学习、混合整数线性规划、可满足性模理论开发了一个自动化框架[12],可在任何流密码或类似结构上执行 SCA。这种攻击不但可以恢复例如 Trivium 方案的内部状态,还能逆向得到其初始密钥。

3.2.2　ChaCha20

ChaCha 是 D. J. 伯恩斯坦(D. J. Bernstein)于 2008 年设计的流密码方案,本节介绍的 ChaCha20 为 Chacha 类中一个有 20 轮运算、96 比特随机数、256 比特密钥的特例方案[13]。ChaCha20 结合了 Poly1305 消息验证码以构造 TLS 协议中面向软件的高性能认证加密方案,旨在替代已经不安全的 RC4 算法。ChaCha20-Poly1305 是 OpenSSH、WireGuard、OTRv4 和比特币闪电网络中默认的关联数据认证加密(authenticated encryption with associated data,AEAD)方案[14]。ChaCha20 主要思想如下。

此算法的核心是一个精心选取的固定置换 P,其具有既高效又密码学高强度的特点。为了提高效率,它被设计为主要依赖在 32 比特字上运行的三个汇编级指令:模 2^{32} 加(addition),循环逐比特旋转(rotation)以及异或(Xor);P 也因此被称为基于 ARX 的设计。从密码学的角度来看,P 可以是一个“随机置换”的实例,并且基于 P 的构造方法可以用随机置换模型分析。与随机预言机模型相比,随机置换模型假设所有参与方都可以调用关于一个均匀置换 P 及其逆 P^{-1} 的预言机。就像在随机预言机模型中一样,计算 P 或 P^{-1} 的唯一方法就是直接询问这些预言机。

在 ChaCha20 中,512 比特的置换 P 用于构造一个伪随机函数 F,该函数根据 256 比特密钥值将 128 比特输入映射到 512 比特输出,如下所示。

$$F_k(x)=P(\text{const}\,||\,k\,||\,x)\boxplus\text{const}\,||\,k\,||\,x$$

其中,const 为 4 个 32 比特字 0x61707865,0x3320646e,0x79622d32,0x6b206574 组成的常数,$\boxplus$ 为逐字模加操作。如果 P 是随机置换,可以证明 F 就是一个伪随机函数。

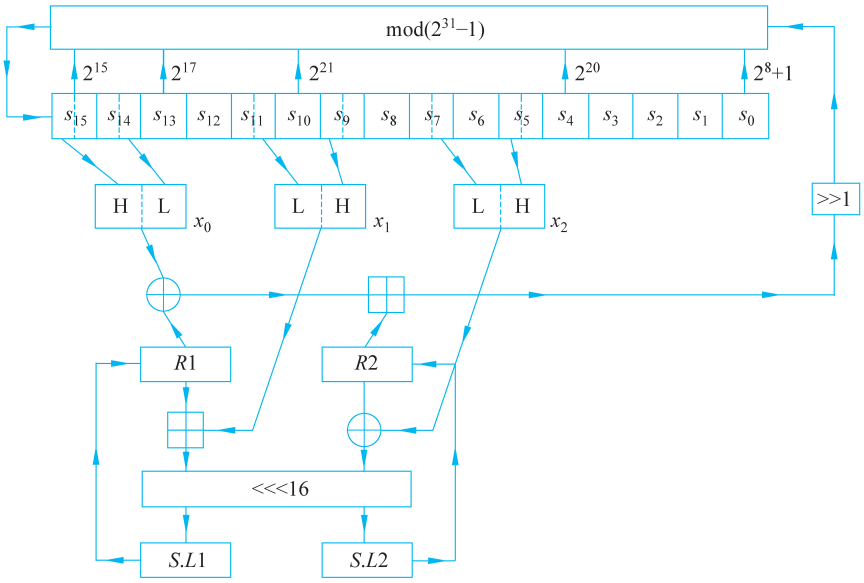
给定一个 256 比特种子 $\boldsymbol{s}$ 和一个初始向量 $\boldsymbol{IV}\in\{0,1\}^{64}$,ChaCha20 流密码输出是 $F_s(\boldsymbol{IV}||<0>),F_s(\boldsymbol{IV}||<1>),\cdots$,其中计数器 $<0>,<1>,\cdots$,均代表 64 比特整数。明文长度如果不是 512 的倍数时将填充 0,之后逐比特异或密钥流就能得到密文。

3.2.3　ZUC

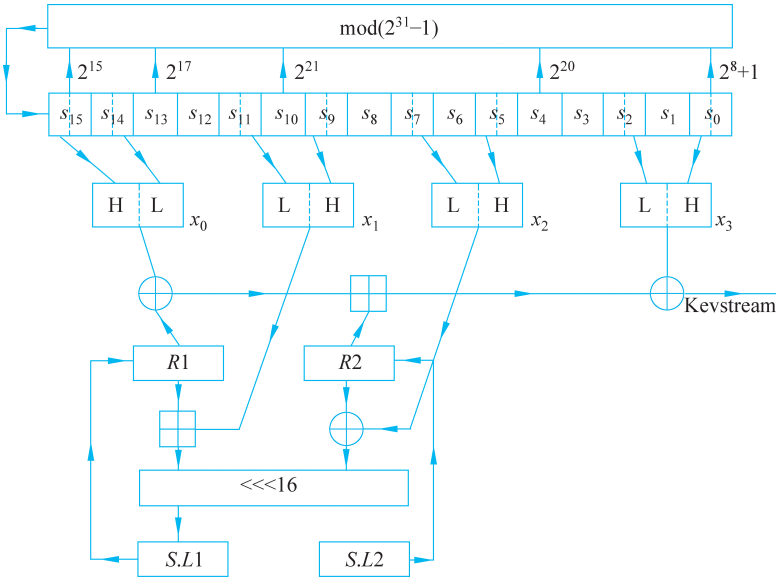
ZUC(祖冲之密码)是由冯登国院士团队研制的一种基于字的商用流密码算法,是 LTE (long term evolution)国际移动通信标准的核心部分。该算法包括祖冲之算法(ZUC)、加密算法(128-EEA3)和完整性算法(128-EIA3)三个部分[15]。

它以 128-bit 的初始密钥 ***KEY*** 和 128-bit 的初始向量 ***IV*** 作为输入,算法的输出为位宽 32-bit 的字序列。整个 ZUC 加密算法的运行按照工作性质可分为两个阶段:初始化阶段和工作阶段。在初始化阶段,算法运行 32 轮但不输出密钥流,目的是让内部状态和初始密钥的关系尽可能地无法预测,从而得到近似随机的内部状态。在工作阶段,算法每完成一次运算后输出一个 32-bit 的密钥字,其具体的执行过程会在后文中算法执行部分详细描述。ZUC 加密算法从逻辑上可以分为三层,从上到下依次为线性反馈移位寄存器(LFSR)层,比特重组(BR)层和非线性函数(F) 层,其基本结构如图 3-3 所示。方案生成的密钥流可以用来加密解密或实现消息完整性认证。

与大多数流密码的 LFSR 设计不同,ZUC 的 LFSR 基于有限域 GF($2^{31}-1$),其目的是在不牺牲效率的条件下提供一定的安全性。LFSR 为 16 个字的寄存器 $\boldsymbol{S}=(s_0,\cdots,$



(a) 初始化阶段



(b) 密钥流生成阶段

图 3-3　ZUC 内部结构图

s_{15}),每个寄存器s_i 有 31 比特,$t+1$ 时刻反馈函数输出字

$$s_{15}^{(t+1)} = 2^{15}\ s_{15}^{t} + 2^{17}\ s_{13}^{t} + 2^{21}\ s_{10}^{t} + 2^{20}\ s_{4}^{t} + (2^{8}+1)\ s_{0}^{t}$$

因其线性反馈函数为本原多项式,所以 LFSR 的周期为$(2^{31}-1)^{16}-1 \approx 2^{496}$。换句话说,ZUC 像其他流密码方案中的 LFSR 一样有序列周期长、统计特性好的优点。而且由于任意 $a \in \mathrm{GF}(2^{31}-1)$都可以表示为 $a = a_{31}2^{31} + a_{30}2^{30} + a_{0}2^{0}$,其中$a_i \in \mathrm{GF}(2)$。换句话

说，a 可以用向量表示，GF($2^{31}-1$)上的乘法可以用左移位和加法快速实现，因此，LFSR 层避开了 GF(2)上已经成熟的序列分析方法还能保证较高的实现效率。

BR 层有四个 32 比特寄存器 X_0，X_1，X_2，X_3，其中存储的每个值都是由 LFSR 中两个状态的各 16 比特信息级联而成，如下所示。

$$X_0 = s_{15H} \parallel s_{14L}$$
$$X_1 = s_{11L} \parallel s_{9H}$$
$$X_2 = s_{7L} \parallel s_{5H}$$
$$X_3 = s_{2L} \parallel s_{0H}$$

例如，两个 31 比特寄存器 s_{15} = 0x12345678，s_{14} = 0x1abcdef2，那么 X_0 = 0x2468def2。虽然重组操作仍然是一种线性变换，但经过重组后的输出序列能够更快地做出变化。

F 层旨在为保密算法提供混淆与扩散性的同时降低对移动通信设备的硬件资源占用。其 F 函数包含三个主要部件：两个 32 比特寄存器 $R1$、$R2$，两个 32 比特到 32 比特的线性变换 $L1$、$L2$ 以及两个 8 进 8 出 S 盒 $S0$ 和 $S1$。F 函数将 BR 层 3 个寄存器 X_0、X_1、X_2 压缩为一个 32 比特字 W，与此同时使用线性变换和 S 盒置换更新寄存器 $R1$、$R2$。两个线性变换在 S 盒变换操作前，实现信息跨 S 盒间的扩散，表达式如下。

$$L1(X) = X \oplus (X \lll_{32} 2) \oplus (X \lll_{32} 10) \oplus (X \lll_{32} 18) \oplus (X \lll_{32} 24)$$
$$L2(X) = X \oplus (X \lll_{32} 8) \oplus (X \lll_{32} 14) \oplus (X \lll_{32} 22) \oplus (X \lll_{32} 30)$$

其中，$X \lll_{32} k$ 表示对 32 比特字 X 循环左移 k 位，$\oplus$ 为逐比特异或。线性变换后的 32 比特输出再通过 4 个小的 S 盒进行局部混淆，这 4 个 S 盒由两个非线性 S 盒 $S0$ 和 $S1$（见表 3-1、表 3-2）组合而成，即

$$S(X) = S0(X \gg 24) \,||\, S1(X \gg 16 \wedge 0\text{xff}) \,||\, S0(X \gg 8 \wedge 0\text{xff}) \,||\, S1(X \wedge 0\text{xff})$$

表 3-1　S 盒 $S0$

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	3e	72	5b	47	ca	e0	00	33	04	d1	54	98	09	b9	6d	cb
01	7b	1b	f9	32	af	9d	6a	a5	b8	2d	fc	1d	08	53	03	90
02	4d	4e	84	99	e4	ce	d9	91	dd	b6	85	48	8b	29	6e	ac
03	cd	c1	f8	1e	73	43	69	c6	b5	bd	fd	39	63	20	d4	38
04	76	7d	b2	a7	cf	ed	57	c5	f3	2c	bb	14	21	06	55	9b
05	e3	ef	5e	31	4f	7f	5a	a4	0d	82	51	49	5f	ba	58	1c
06	4a	16	d5	17	a8	92	24	1f	8c	ff	d8	ae	2e	01	d3	ad
07	3b	4b	da	46	eb	c9	de	9a	8f	87	d7	3a	80	6f	2f	c8
08	b1	b4	37	f7	0a	22	13	28	7c	cc	3c	89	c7	c3	96	56
09	07	bf	7e	f0	0b	2b	97	52	35	41	79	61	a6	4c	10	fe
0a	bc	26	95	88	8a	b0	a3	fb	c0	18	94	f2	e1	e5	e9	5d
0b	d0	dc	11	66	64	sc	ec	59	42	75	12	f5	74	9c	aa	23
0c	0e	86	ab	be	2a	02	e7	67	e6	44	a2	6c	c2	93	9f	f1

续表

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
0d	f6	fa	36	d2	50	68	9e	62	71	15	3d	d6	40	c4	e2	0f
0e	8e	83	77	6b	25	05	3f	0c	30	ea	70	b7	a1	e8	a9	65
0f	8d	27	1a	db	81	b3	a0	f4	45	7a	19	df	ee	78	34	60

表 3-2　S 盒 $S1$

	0	1	2	3	4	5	6	7	8	9	0a	0b	0c	0d	0e	0f
0	55	c2	63	71	3b	c8	47	86	9f	3x	da	5b	29	aa	fd	77
1	8c	c5	94	0c	a6	1a	13	00	e3	a8	16	72	40	f9	f8	42
2	44	26	68	96	81	d9	45	3e	10	76	c6	a7	8b	39	43	e1
3	3a	b5	56	2a	c0	6d	b3	05	22	66	bf	dc	0b	fa	62	48
4	dd	20	11	06	36	c9	c1	cf	f6	27	52	bb	69	f5	d4	87
5	7f	84	4c	d2	9c	57	a4	bc	4f	9a	df	fe	d6	8d	7a	eb
6	2b	53	d8	5c	a1	14	17	fb	23	d5	7d	30	67	73	08	09
7	ee	b7	70	3f	61	b2	19	8e	4e	e5	4b	93	8f	5d	db	a9
8	ad	f1	ae	2e	cb	0d	fc	f4	2d	46	6e	1d	97	e8	d1	e9
9	4d	37	a5	75	5e	83	9e	ab	82	9d	b9	1c	e0	cd	49	89
0a	01	b6	bd	58	24	a2	5f	38	78	99	15	90	50	b8	95	e4
0b	d0	91	c7	ce	ed	0f	b4	6f	a0	cc	f0	02	4a	79	c3	de
0c	a3	ef	ea	51	e6	6b	18	ec	1b	2c	80	f7	74	e7	ff	21
0d	5a	6a	54	1e	41	31	92	35	c4	33	07	0a	ba	7e	0e	34
0e	88	b1	98	7c	f3	3d	60	6c	7b	ca	d3	1f	32	65	04	28
0f	64	be	85	9b	2f	59	8a	d7	b0	25	ac	af	12	03	e2	f21

其中，$X \gg k$ 表示对 32 比特字 X 右移 k 位。非线性函数 F 的伪代码如下。

$F(X_0, X_1, X_2)$

{

$W = (X_0 \oplus R1) \boxplus R2$

$W_1 = R1 \boxplus X_1$

$W_2 = R2 \oplus X_2$

$R1 = S(L1(W_{1L} || W_{2H}))$

$R2 = S(L2(W_{2L} || W_{1H}))$

return W

}

其中，$\boxplus$ 为模 2^{32} 加法。

加密算法(128-EEA3)和完整性算法(128-EIA3)均基于 ZUC 算法，前者使用对称密钥初始化，再用生成的密钥流逐比特异或明文进行加密。后者使用另一个密钥(即完整性

密钥)来初始化,生成密钥流后,将密钥流根据明文的值以累加到一个 32 比特的累加器中以生成最终的 MAC。

ZUC 密码研究新进展

冯登国教授(如图 3-4 所示)团队于 2016 年起草发布了国家标准《信息安全技术 祖冲之序列密码算法》的第一部分[16],此部分保证祖冲之序列密码算法使用的正确性,为国内企业正确研发祖冲之算法的相关设备提供指导。2021 年,冯教授团队继续完善 ZUC 算法标准的剩余部分,包括保密性算法[17]和完整性算法[18]。这两部分主要适用于基于祖冲之序列密码算法的保密性算法和完整性算法的相关产品的研制、检测和使用。

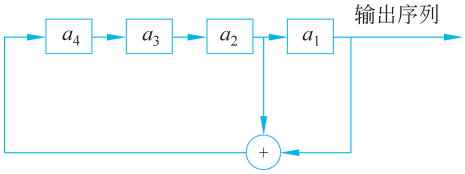
图 3-4　冯登国教授

3.3 小结

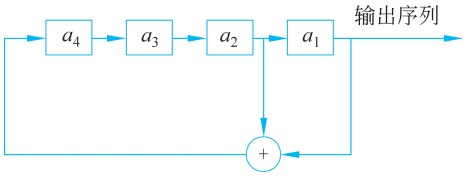
本章首先引入了多个伪随机序列生成器以替代保存足够长的完全随机密钥,介绍了伪随机序列生成的一般方式,接着讲解了伪随机序列生成的例子,也是研究最为广泛的 LFSR 及其序列,并阐述其优缺点。之后分析了流密码方案的工作流程,介绍了几种主要密码学器件。最后介绍了三种不同结构的流密码方案:Trivium、ChaCha20 以及国产 ZUC 密码。在实际应用中,往往精心设计的流密码方案可以用于存储及计算资源受限且需要高性能的设备中。

思　考　题

1. 下图是一个 4 级 LFSR,它的输出序列最小周期是多少?请给出一个初试状态并计算相应的一个周期的输出序列。



2. 下图是一个 4 级 LFSR,它的输出序列最大周期是多少?请给出一个初试状态并计算相应的一个周期的输出序列。



3. 对于一个仅用 LFSR 的流密码方案,已知用户从 t 时刻起发送的明文为 0100 1101 1001 10,并截获其密文为 0110 0011 1100 01。

(1) 请确定该 LFSR 的级数和 t 时刻的内部状态。

(2) 求 LFSR 的反馈系数并画出该 LFSR 的电路图。

4. 两个 LFSR 的线性移位寄存器 A、B,其级数分别为 2、3,它们由如下方法互相耦合而成:

$$A_2^{(t+1)} = B_1^{(t)} \oplus B_3^{(t)}$$

$$B_3^{(t+1)} = A_1^{(t)} \oplus A_2^{(t)}$$

求 $Z^{(t+1)} = A_1^{(t)} \oplus A_2^{(t)} \oplus B_1^{(t)} \oplus B_3^{(t)}$ 生成序列的最大周期。

5. 已知一个 GF(7)上的 5 级 LFSR $S=(s_0,\cdots,s_4)$,每个 s_i 取值为$\{1,\cdots,7\}$,如果反馈函数为本原多项式 $f(x)\equiv x^5+x^3+x^2+x+1 \pmod 7$,那么该寄存器输出序列最大周期是多少?