第3章 对称密码体系

本章先介绍对称密码体系的概念、结构和特点，之后详细阐述对称密码体系的两类算法：流密码和分组密码。流密码部分重点说明其构造算法和工作原理；分组密码部分主要介绍 Feistel 密码结构。然后通过介绍一些有代表性的加密算法，如 DES、AES 和 IDEA，强化读者对对称加密算法的理解。最后介绍分组密码的常用工作模式。

3.1 对称密码体系概述

对称密码算法(Symmetric Algorithm)也称传统密码算法、单钥密码算法，它包括许多数据加密方法。公钥密码技术出现之前，对称密码系统已被使用了多年。对称密码体系的基本模型如图 3-1 所示，其基本特征是：数据加密和解密使用同一个密钥。在算法公开的前提下所有秘密都在密钥中，因此密钥本身应该通过另外的秘密信道传递。对称密码体系的安全性依赖于两个因素：其一，加密算法强度至少应该满足当敌手已知算法时通过截获密文不能导出明文或者发现密钥，更高的要求是敌手即使拥有部分密文以及相应明文段落也不能导出明文或者发现密钥系统；其二，发送方和接收方必须以安全的方式传递和保存密钥副本，对称加密的安全性取决于密钥的保密性而不是算法的机密性。

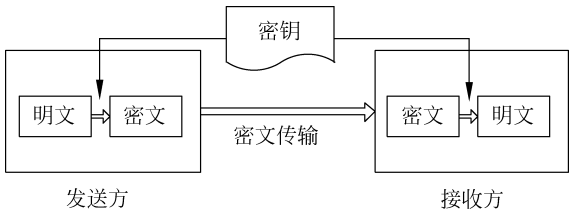

图 3-1 对称密码体系的基本模型

对称加密算法可以分成两类：一类为流算法，是一次只对明文中单个位(有时为字节)加密或解密的运算；另一类为分组算法，是一次只对明文的一组固定长度的字节加密或解密的运算。现代计算机密码算法一般采用的都是分组算法，一般分组的长度为 64 位，这是由于这个长度大到足以防止分析破译，但又小到足以方便使用。

对称算法的加密和解密表示为

$$E_k(M)=C$$

$$D_k(C)=M$$

常用的对称加密体系有五个组成部分。

(1) 明文 M：原始数据信息。

(2) 加密算法 E_k：以密钥 k 为参数，对明文 M 进行多种置换和转换的规则和步骤，结果即为密文。

(3) 密钥 k：加密与解密算法的参数，直接影响对明文进行变换的结果。

(4) 密文 C：对明文进行变换的结果。

(5) 解密算法 D_k：加密算法的逆变换，以密文 C 为输入、密钥 k 为参数，变换结果为明文。

对称密码体系的优点是算法实现的效率高、速度快，缺点是密钥的管理过于复杂。如果按照上述方法，任何一对发送方和接收方都有他们各自商议的密钥，那么很明显，假设有 N 个用户进行对称加密通信，则需要产生 $N(N-1)$ 把密钥，每个用户要记住或保留 $N-1$ 把密钥，当 N 很大时，记住是不可能的，而保留起来又会引起密钥泄露可能性的增加。常用的对称加密算法有 DES、AES 和 IDEA 等。

3.2 流密码

3.2.1 流密码简介

香农证明了“一次一密”密码体制在理论上是不可破译的，促使人们长期以来一直寻求某种能仿效“一次一密”密码的密码体制，流密码就是所寻求的方法之一。流密码也称序列密码(Stream Cipher)，具有实现简单、便于硬件实施、加解密处理速度快、没有或只有有限的错误传播等特点。因此在实际应用中，特别是在专用或机密机构中保持着优势，典型的应用领域包括世界军事、无线通信、外交通信。

流密码加密时先将文本、声音、图像等原始明文转换成 0-1 串，然后将它与密钥流逐位异或生成密文流传送给接收者，接收者将密文流与相同的密钥流逐位异或恢复出明文。在流密码中，将明文分成一定长度的分组，分别对各个分组用同一密钥流序列的不同部分进行加密产生相应的密文。相同的明文分组因为处于明文序列中的不同位置，所对应的密钥流位也不同，因此会加密成不同的密文组。

流密码系统可以用一个六元组(M, C, K, Z, E_k, D_k)来描述。其中，M 表示明文空间，C 表示密文空间，K 表示密钥空间，Z 表示密钥流生成算法，E_k 和 D_k 表示密钥流与明文(密文)的加密和解密规则，通常为异或运算。对密钥 $k \in K$，由 Z 确定一个密钥流。流密码系统加解密的原理如图 3-2 所示。

流密码的加密是用一个密钥流序列(伪随机)和明文序列相异或来产生密文，解密过程相同，即用同一密钥流序列和密文序列相异或来获得明文。

设二进制序列 $M = M_1 M_2 \cdots M_i \cdots M_n$ 是明文序列，$M_i \in \mathrm{GF}(2)$，$i \geqslant 1$；二进制序列 $C_1 C_2 \cdots C_i \cdots C_n$ 作为密文比特流，$C_i \in \mathrm{GF}(2)$，$i \geqslant 1$；序列 $k = k_1 k_2 \cdots k_i \cdots k_n$ 作为密钥流序列，同样 $k_i \in \mathrm{GF}(2)$，$i \geqslant 1$。加密过程可表示为 $C_i = M_i \oplus k_i$，解密操作表示为 $M_i = C_i \oplus k_i$，$i \geqslant 1$。$\oplus$表示按位异或运算。

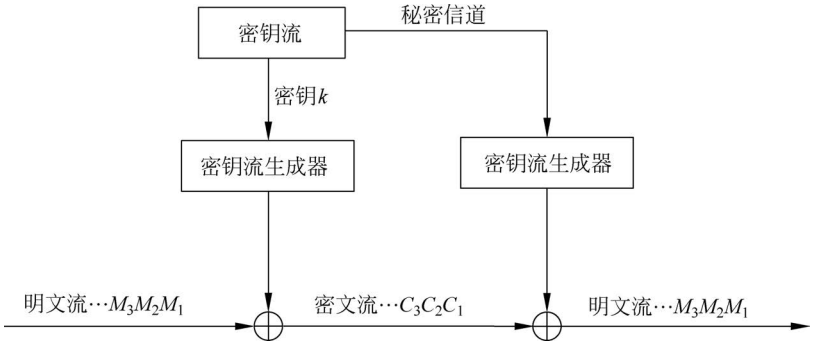

图 3-2　流密码系统加解密的原理

密钥流序列的伪随机性决定了流密码的安全性。当密钥流序列是由无记忆离散的二进制均匀分布信源产生的随机序列时,产生该序列的密码就是“一次一密”密码,在理论上它已经被证明是安全的。但是,用产生真正随机序列的方法来产生完全相同的随机序列几乎是不可能的。实际使用“一次一密”密码时要求信息的收发双方必须持有加解密信息所用的密钥流副本。已经证明仅当密钥数目与明文数目至少一样多时,“一次一密”才是完全保密的,即密钥必须至少和明文一样长且不重复使用。而很长的密钥序列不便于存储和分配,流密码的设计就是研究如何用一个短的密钥生成一个周期长且安全的密钥流。利用这种方法能够重复产生相同的密钥序列,但产生的序列不是真正的随机序列,而是伪随机的,这就要求伪随机序列满足真随机序列的一些随机特性。要实现保密通信,只需要在信息的发送方和接收方之间传送一个短的密钥。

3.2.2　流密码的结构

根据明文和密文的消息流与密钥流序列的关系,可将流密码分为同步流密码和自同步流密码两类。

1. 同步流密码

在同步流密码中,密钥流的产生独立于明文和密文。发送方和接收方只要有相同的密钥和初始内部状态,就能产生相同的密钥流。同步流密码加密结构如图 3-3 所示。

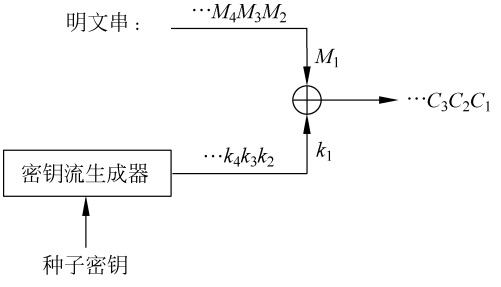

图 3-3　同步流密码加密结构

由于密钥流与明文串无关,所以同步流密码中的每个密文字符 c_i 不依赖于之前的明文 $m_{i-1},\cdots,m_1$。所以同步流密码的一个重要特点就是无错误传播:在传输期间一个密

文字符被改变只影响该符号的恢复，不会对后继的符号产生影响。但是，在同步流密码中发送方和接收方必须是同步的，用同样的密钥且该密钥操作在同样的位置时才能保证正确解密。如果在传输过程中密文字符有插入或删除导致同步丢失，密文与密钥流将不能对齐，导致无法正确解密。要正确还原明文，密钥流必须再次同步。与同步流密码相反，自同步流密码有错误传播现象，但可以自行实现同步。

2. 自同步流密码

在自同步流密码中，密钥流的产生与之前已经产生的若干密文有关，其加密结构如图 3-4 所示。其中，密钥流 k_i 的生成过程如图 3-5 所示。

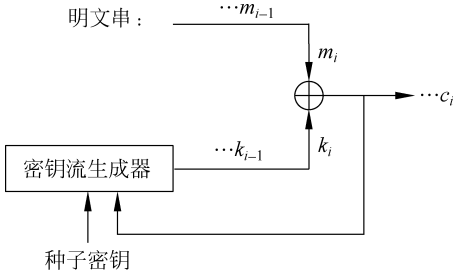

图 3-4　自同步流密码加密结构

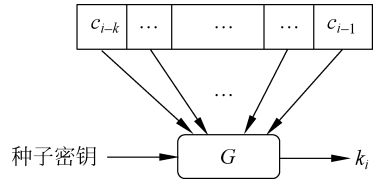

图 3-5　同步流密码的密钥流生成过程

用函数表示为

$$\sigma_i = F(\sigma_{i-1}, c_{i-1}, \cdots, c_{i-k})$$
$$k_i = G(\sigma_i, k)$$
$$c_i = E(k_i, m_i)$$

其中，k 是种子密钥；σ_i 是密钥流生成器的内部状态（初始状态记为 σ_0）；F 是状态转移函数；G 是生成密钥流的函数；E 是自同步流密码的加密变换，它是 k_i 与 m_i 的函数。

由此可见，如果自同步流密码中某一符号出现传输错误，则将影响到它之后 k 个符号的解密运算，亦即，自同步流密码有错误传播现象。等到该错误移出寄存器后寄存器才能恢复同步，因而一个错误至多影响 k 个符号。在 k 个密文字符之后，这种影响将消除，密钥流自行实现同步。由于密文流参与了密钥流的生成，使得密钥流的理论分析复杂化，目前的流密码研究结果大部分都是关于同步流密码的，因为这些流密码的密钥流的生成独立于消息流，从而使它们的理论分析成为可能。

3.2.3　反馈移位寄存器与线性反馈移位寄存器

如前所述，序列密码的安全强度取决于密钥流生成器生成的密钥流的安全性（如周期、游程分布、线性复杂度等）。有多种产生同步密钥流生成器的方法，最普遍的是使用一种称为线性反馈移位寄存器（Linear Feedback Shift Register，LFSR）的设备。

采用 LFSR 作为基本部件的主要原因如下：

（1）LFSR 的结构非常适合硬件实现；

（2）LFSR 的结构便于使用代数方法进行理论分析；

（3）产生的序列的周期可以很大；

(4) 产生的序列具有良好的统计特性。

1. 反馈移位寄存器

图 3-6 所示为一个反馈移位寄存器的流程图,信号从左到右。a 表示存储单元,取值为 0 或 1,a_i 的个数 n 称为反馈移位寄存器的级。在某一时刻,这些级的内容构成该反馈移位寄存器的一个状态,共有 2^n 个可能的状态,每一个状态对应于 F 上的一个 n 维向量,用$(a_1,a_2,\cdots,a_n)$表示。函数 f 是一个 n 元布尔函数,称之为反馈函数。

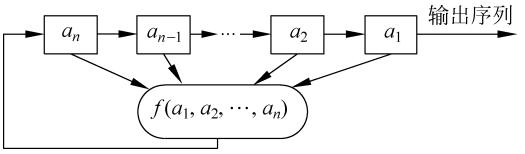

图 3-6　反馈移位寄存器流程图

在主时钟确定的周期区间上,每一级存储器 a_i 都将其存储内容向下一级 a_{i-1} 传递,最右一级存储器的内容作为该时刻的输出,根据该时刻寄存器的状态计算 $f(a_1,a_2,\cdots,a_n)$作为最左一级寄存器在下一时刻的内容。显然,一个反馈移位寄存器的逻辑功能完全由该移位寄存器的反馈函数所标志。

2. 线性反馈移位寄存器

如果反馈函数形如 $f(a_1,a_2,\cdots,a_n)=c_na_1\oplus c_{n-1}a_2\oplus\cdots\oplus c_1a_n$,其中,系数 $c_i=0$,1,这里的加法运算为模 2 加,乘法运算为普通乘法,则称该反馈函数是 $a_1,a_2,\cdots,a_n$ 的线性函数,对应的反馈移位寄存器称为线性反馈移位寄存器,用 LFSR 表示。否则,称为非线性反馈移位寄存器(Non-Linear Feedback Shift Register,NLFSR)。LFSR 的示意图如图 3-7 所示。

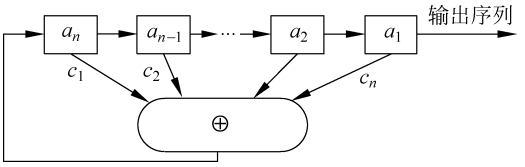

图 3-7　线性反馈移位寄存器流程图

显然,根据 LFSR 中反馈函数的系数 $c_i(i=1,\cdots,n)$取值的不同,这样的反馈函数有 2^n 种。令 $a_i(t)$表示 t 时刻第 i 级寄存器的内容,则第 $t+1$ 时刻寄存器的内容为:

$$a_i(t+1)=a_{i+1}(t),\quad i=1,2,\cdots,n-1$$

$$a_n(t+1)=c_na_1(t)\oplus c_{n-1}a_2(t)\oplus\cdots\oplus c_1a_n(t)$$

通常称多项式 $c_nx^n+c_{n-1}x^{n-1}+\cdots+c_1x^1+1$ 为上述 LFSR 的特征多项式。

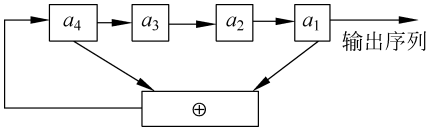

图 3-8　4 级线性反馈移位寄存器示例

LFSR 的特征多项式与它的反馈函数是一一对应的,如果知道了 LFSR 的特征多项式,便立即可以求得该移位寄存器的反馈函数,反之亦然。

例 3-1　图 3-8 为一个 4 级线性反馈移位寄

存器，状态转移关系为

$$a_i(t+1)=a_{i+1}(t),\quad i=1,2,3$$

$$a_4(t+1)=a_1(t)\oplus a_4(t)$$

假设初始状态为$(a_1,a_2,a_3,a_4)=(0,1,1,0)$，则可根据反馈函数计算出该线性反馈移位寄存器在各时刻的所有状态，如表 3-1 所示。

表 3-1　各时刻的所有状态

t	a_4	a_3	a_2	a_1	t	a_4	a_3	a_2	a_1
0	0	1	1	0	8	1	1	1	0
1	0	0	1	1	9	1	1	1	1
2	1	0	0	1	10	0	1	1	1
3	0	1	0	0	11	1	0	1	1
4	0	0	1	0	12	0	1	0	1
5	0	0	0	1	13	1	0	1	0
6	1	0	0	0	14	1	1	0	1
7	1	1	0	0	15	0	1	1	0

由计算过程可见，在 $t=15$ 时刻该寄存器的状态恢复至 $t=0$ 时刻的状态，因此之后的状态将开始重复。这个移位寄存器输出的序列就是 011001000111101011001000111101……，序列的周期为 15，也称该移位寄存器的周期为 $15(=2^4-1)$。

为了从直观上描述这一 LFSR 的状态转移情况，可以使用一些方框以及连接这些方框的箭头组成的图形，即为状态转移图，如图 3-9 所示。

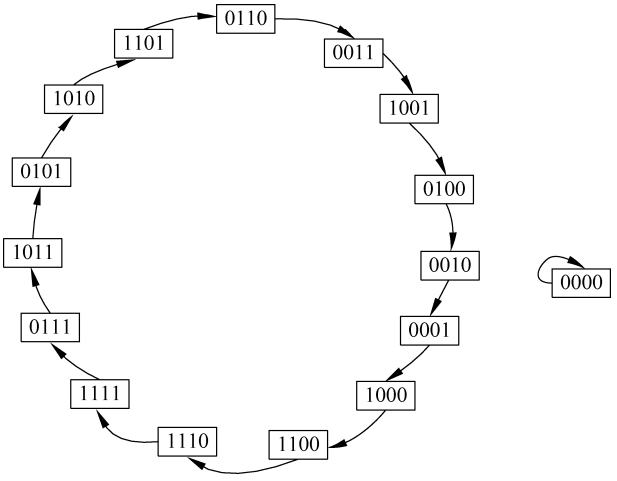

图 3-9　状态转移图

例 3-2　图 3-10 所示是一个特征多项式为 x^3+x+1 的线性反馈移位寄存器。

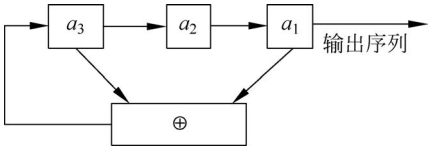

图 3-10　3 级线性反馈移位寄存器示例

根据特征多项式可知,该 LFSR 的反馈函数为 $f(a_1,a_2,a_3)=a_1\oplus a_3$。假设初始状态为$(a_1,a_2,a_3)=(1,1,1)$,其状态转移图如图 3-11 所示。

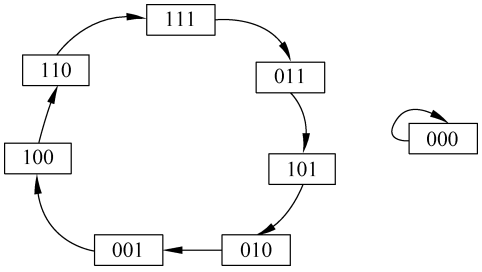

图 3-11　3 级 LFSR 的状态转移图

在状态转移图中,从初始状态开始,沿着箭头所指示的路径依次取出最右边的分量便得到该 LFSR 的输出序列:1110100 1110100……,周期为 7($=2^3-1$)。3 级移位寄存器的所有可能状态数为 $2^3=8$(含状态(0,0,0)),而本例中从初始状态(1,1,1)开始可以得到所有非(0,0,0)的状态。

若以状态转移图中任一状态作为初始状态,沿箭头所指示的路径依次取出最右边的分量还可得到另外 6 个序列:1101001 1101001……;1010011 1010011……;0100111 0100111……;1001110 1001110……;0011101 0011101……;0111010 0111010……。全部 7 个序列取自同一个状态转移图,将这 7 个序列之一经过适当的移位可以得到其余任一序列,称这 7 个序列是移位等价的。

例 3-3　图 3-12 所示为一个 4 级 LFSR,其特征多项式为 $x^4+x^3+x^2+x+1$。

① 如取初始状态为$(a_1,a_2,a_3,a_4)=(1,1,1,1)$,则其状态转移图如图 3-13 所示。

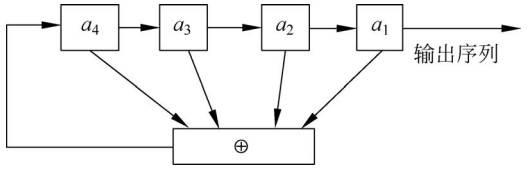

图 3-12　4 级 LFSR

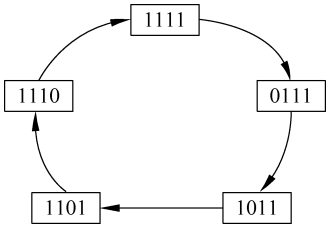

图 3-13　初始状态转移图一

对应的输出序列为 11110 11110……,周期为 5。

② 如取初始状态为$(a_1,a_2,a_3,a_4)=(0,0,0,1)$,则其状态转移图如图 3-14 所示。

对应的输出序列为 00011 00011……,周期为 5。

③ 如取初始状态为$(a_1,a_2,a_3,a_4)=(1,0,1,0)$,则其状态转移图如图 3-15 所示。

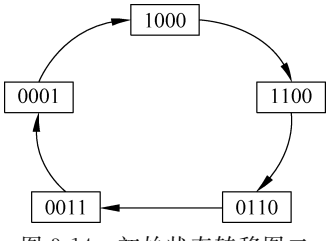

图 3-14　初始状态转移图二

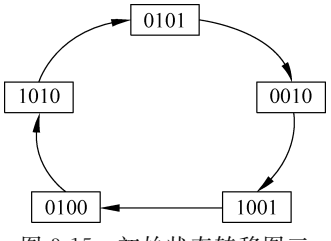

图 3-15　初始状态转移图三

对应的输出序列为 10100 10100……，周期为 5。

以上 15 个状态连同状态(0,0,0,0)即为 4 级移位寄存器所有可能的 $2^4=16$ 个状态。

3.2.4　*m* 序列及其伪随机性

1. m 序列与最长线性移位寄存器

根据 LFSR 的状态转移图可以看出，一个 n 级 LFSR 序列的周期最大只能为 2^n-1；有的 LFSR 序列周期可能远小于这个值，但也有的 LFSR 序列的周期可以达到这个值。如果以 GF(2)上 n 次多项式 $c_nx^n+c_{n-1}x^{n-1}+\cdots+c_1x+1$ 为连接多项式的 n 级 LFSR 所产生的非零序列的周期为 2^n-1，则称这个序列为 n 级最大周期线性移位寄存器序列，简称 m 序列。显然，如果一个 n 级 LFSR 产生了 m 序列，则该 LFSR 的状态转移图仅由两个圈构成，其中一个是由全零状态构成的长度为 1 的圈，另一个是由全部其余 2^n-1 个状态构成的长度为 2^n-1 的圈。换句话说，如果一个 n 级 LFSR 输出的非零序列是 m 序列，则其余 2^n-2 个非零序列也是 m 序列，它们一起构成了一个移位等价类。这里把产生周期为 2^n-1 的 m 序列的 n 级 LFSR 称为最长线性移位寄存器。

显而易见，一个序列是否是 m 序列应当与产生这一序列的 LFSR 的连接多项式有密切的关系。事实上，已证明下面的结论是正确的：如果以 GF(2)上 n 次多项式 $c_nx^n+c_{n-1}x^{n-1}+\cdots+c_1x+1$ 为连接多项式的 n 级 LFSR 所产生的非零序列是 m 序列，则 $c_nx^n+c_{n-1}x^{n-1}+\cdots+c_1x+1$ 必为 GF(2)上的不可约多项式。这一结果表明一个 LFSR 为最长移位寄存器的必要条件是它的连接多项式为不可约多项式。

但是，连接多项式为不可约的假设尚不足以保证该 LFSR 输出的非零序列是 m 序列。例如，对于 GF(2)上 4 次不可约多项式 $x^4+x^3+x^2+x+1$，对应的 LFSR 的非零序列周期为 5，而非 2^4-1。关于 m 序列的充分必要条件是：以 GF(2)上 n 次多项式 $c_nx^n+c_{n-1}x^{n-1}+\cdots+c_1x+1$ 为连接多项式的 n 级 LFSR 所产生的非零序列是 m 序列$\Leftrightarrow$ $c_nx^n+c_{n-1}x^{n-1}+\cdots+c_1x+1$ 为 GF(2)上的 n 次本原多项式。

因此，一个 n 级 LFSR 为最长移位寄存器的充要条件是它的连接多项式为 GF(2)上的 n 次本原多项式。例如，多项式 x^3+x+1 是 GF(2)上的 3 次本原多项式，可以验证以此为连接多项式的 LFSR 的输出序列均为 m 序列。在实践中，经常使用 GF(2)上的本原三项式，这是因为它只需要一个抽头，所以电路设计最简单。

由以上的讨论可知，本原多项式的概念不论是在理论上还是实践上均起重要作用。由代数学的知识知道，当 2^n-1 为素数时，GF(2)上的每一个 n 次不可约多项式均为 n 次本原多项式。形如 2^n-1 的素数称为梅森(Mersenne)数，如 $n=2,3,5,7,13$ 等。

2. m 序列的安全性

LFSR 的输出序列就是流密码的密钥流。一个 LFSR 可以输出足够长的二进制位以匹配明文的二进制位。要解密密文，只需要运行具有相同初始状态的 LFSR 即可。对于给定的整数 n，由于 m 序列是周期最长的，是否可以用 m 序列直接作为密钥流实现流密码？或者说将 LFSR 作为密码序列产生器，安全吗？下面通过一个简单的例子来描述 m 序列直接在流密码中使用时可能遇到的问题。

例 3-4　m 序列的破译。假设在某个流密码体制中，其密钥流生成器是一个 5 级

LFSR,如图 3-16 所示。设攻击者得到密文串 10110 10111 和相应的明文串 01100 11111,并知道该流密码体制中的 LFSR 是 5 级的。因此,攻击者可计算出相应使用的密钥流为 11010 01000。

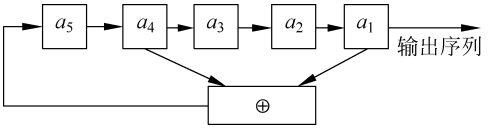

图 3-16　5 级 LFSR

由于

$$a_6 = c_5a_1 + c_4a_2 + c_3a_3 + c_2a_4 + c_1a_5$$
$$a_7 = c_5a_2 + c_4a_3 + c_3a_4 + c_2a_5 + c_1a_6$$
$$a_8 = c_5a_3 + c_4a_4 + c_3a_5 + c_2a_6 + c_1a_7$$
$$a_9 = c_5a_4 + c_4a_5 + c_3a_6 + c_2a_7 + c_1a_8$$
$$a_{10} = c_5a_5 + c_4a_6 + c_3a_7 + c_2a_8 + c_1a_9$$

攻击者可根据密钥流建立如下方程组:

$$0 = c_5 1 + c_4 1 + c_3 0 + c_2 1 + c_1 0 \quad (1)$$
$$1 = c_5 1 + c_4 0 + c_3 1 + c_2 0 + c_1 0 \quad (2)$$
$$0 = c_5 0 + c_4 1 + c_3 0 + c_2 0 + c_1 1 \quad (3)$$
$$0 = c_5 1 + c_4 0 + c_3 0 + c_2 1 + c_1 0 \quad (4)$$
$$0 = c_5 0 + c_4 0 + c_3 1 + c_2 0 + c_1 0 \quad (5)$$

由(5)知 $c_3=0$;从而由(2)知 $c_5=1$;从而由(4)知 $c_2=1$;从而由(1)知 $c_4=0$;从而由(3)知 $c_1=0$。这样,攻击者就知道了该 LFSR 的具体结构。

攻击者利用这一结构和已经掌握的部分密钥流,就可以计算出之后所有的密钥序列。因此,相应的流密码毫无安全性可言。

上述分析可以推广到一般情况:对于一个 n 级的 LFSR,如果攻击者可以得到 $2n$ 个明文-密文对,就可以获得该 LFSR 的具体代数结构。

3. Golomb 随机性假设

流密码的安全性取决于密钥流的安全性,要求密钥流序列有好的随机性,以使密码分析者对它无法预测。也就是说,即使截获其中一段,也无法推测后面是什么。如果密钥流是周期的,要完全做到随机性是困难的。严格地说,这样的序列不可能做到随机,只能要求截获比周期短的一段密钥流时不会泄露更多信息,这样的序列称为伪随机序列。之所以把这种序列称为伪随机序列,是因为产生这种序列是按照完全确定的方式进行的,而不像随机序列那样元素的出现是随机的。

下面先说明游程的概念:设$\{a_i\}=(a_1a_2a_3\cdots)$为 0、1 序列,例如 00110111,其前两个数字是 00,称为 0 的 2 游程;接着是 11,是 1 的 2 游程;再下来是 0 的 1 游程和 1 的 3 游程。

Golomb 提出了度量伪随机序列随机性的三条规则,称为 Golomb 随机性假设。简单起见,假设所述伪随机序列为二元序列 $a_1,a_2,\cdots,a_i,\cdots,a_n,\cdots(a_i\in\{0,1\})$,其周期为 n。

(1) 在每一周期内,0 的个数与 1 的个数近似相等。

(2) 在每一周期内,长度为 i 的游程数占游程总数的 $1/2^i$。

(3) 定义自相关函数为 $C(\tau)=\sum_{i=1}^{n}(-1)^{a_i+a_{i+\tau}}$，这是一个二值函数：

$$C(\tau)=\begin{cases}n, & \tau\equiv 0 \bmod n\\ c, & \text{其他}\end{cases}$$

其中，c 为一个常数。

例如，在周期为 7 的序列 1110010……中，每一周期内有 4 个 1，3 个 0，4 个游程，且有 2 个长度为 1 的游程、1 个长度为 2 的游程、1 个长度为 3 的游程。自相关函数为：

$$C(\tau)=\begin{cases}7, & \tau\equiv 0 \bmod n\\ -1, & \text{其他}\end{cases}$$

m 序列是满足 Golomb 随机性假设的典型代表。

4. m 序列的伪随机性

m 序列之所以在电子工程的许多领域获得广泛的应用，主要是由于它具有与随机序列类似的性质，满足 Golomb 的三个随机性假设。下面不加证明地给出其相应的结论，感兴趣的读者可以自己尝试给出其正确性证明。

(1) 在 n 级 m 序列的一个周期段内，1 出现的次数恰为 2^{n-1}，0 出现的次数恰为 $2^{n-1}-1$。

(2) 在 n 级 m 序列的一个周期段内，游程总数为 2^{n-1}；长为 $k(1\leqslant k\leqslant n-2)$ 的 0-游程(或 1-游程)数为 2^{n-2-k}；长为 $n-1$ 的游程只有 1 个，为 0-游程；长为 n 的游程也只有 1 个，为 1-游程。

(3) 自相关函数是二值的，且为

$$C(\tau)=\begin{cases}2^n-1, & \tau\equiv 0 \bmod (2^n-1)\\ -1, & \text{其他}\end{cases}$$

需要说明的是，对于密钥流序列而言，Golomb 随机性假设只是判别二元周期序列的随机性标准的必要条件，并不是充分的。这是因为，由其中很小一部分就可简单地确定出整个密钥序列。

5. 线性复杂度

除了要求伪随机序列具有长周期、满足 Golomb 的三个随机性假设外，还要求适用于流密码的伪随机序列满足高的线性复杂度(Linear Complexity)。给定二元序列 $a_1,a_2,\cdots,a_i,\cdots,a_n,\cdots(a_i\in\{0,1\})$，其线性复杂度定义为能够输出该序列的最短线性移位寄存器的级数。

例如，给定序列 011 011……，连接多项式为 x^2+x+1 的 LFSR 可以生成该序列，连接多项式为 x^3+1 的 LFSR 也可以生成该序列。但连接多项式为 $x+1$ 的 LFSR 则无法做到这一点，所以，该序列的线性复杂度为 2。序列线性复杂度的概念在密码学中的重要意义是通过以下结论体现出来的：如果序列的线性复杂度为 $l(l\geqslant 1)$，则只要知道序列中任意相继的 $2l$ 位，就可确定整个序列。由此可见，序列线性复杂度是流密码安全性的重要指标，作为密钥流序列，其线性复杂度很小是不安全的。通常认为一个安全的密钥流应该满足以下三个基本条件：周期充分长；随机统计特性好(即基本满足 Golomb 的随机性假设)；线性复杂度大。这里长周期一般指不少于 10^{16}，而线性复杂度为序列长度的一半是比较合适的。

3.2.5 线性移位寄存器的非线性组合

由于直接使用 LFSR 的 m 序列是不安全的,因此线性移位寄存器序列不能直接作为密钥流使用。如果在 LFSR 的基础上加入非线性化的手段,便可产生适合于流密码应用的密钥序列。

为了使密钥流生成器输出的二元序列尽可能复杂,应保证其周期尽可能大、线性复杂度和不可预测性尽可能高,常使用多个 LFSR 来构造二元序列,称每个 LFSR 的输出序列为驱动序列。这样 LFSR 作为驱动源以其输出推动了一个非线性组合函数所决定的电路产生出非线性序列。实际上,这就是所谓的非线性前馈序列生成器。线性移位寄存器用来保证密钥流的周期长度,非线性组合函数用来保证密钥流的各种密码性能,以抗击各种可能的攻击。许多专用流密码算法都是用这种方法构成,通常可将这种方法分为三类:滤波生成器、组合生成器和钟控生成器。

1. 滤波生成器

滤波生成器(Filter Generator)是一种常见的密钥流生成器,由一个 n 级线性移位寄存器和一个 $m(m<n)$ 元非线性滤波函数组成,滤波函数的输出为密钥流序列,工作模式如图 3-17 所示。

这里 g 为一个 m 元布尔函数,其输入由 LFSR 中的一部分抽头构成,其输出构成整个生成器的输出。

2. 组合生成器

在序列密码设计和分析中,组合生成器(Combinatorial Generator)也有着广泛的应用,它由若干线性移位寄存器 $\mathrm{LFSR}_i(i=1,\cdots,n)$ 和一个非线性组合函数组成,组合函数的输出构成密钥流序列。组合生成器工作模式如图 3-18 所示。

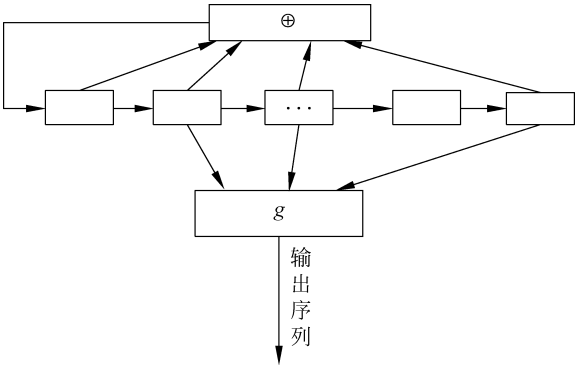

图 3-17 滤波生成器工作模式

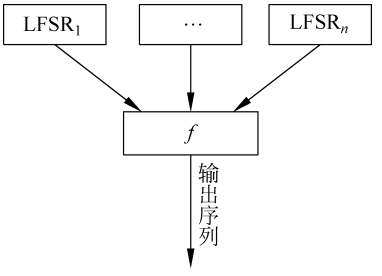

图 3-18 组合生成器工作模式

其中,$\mathrm{LFSR}_i(i=1,\cdots,n)$ 为 n 个级数分别为 $r_1,r_2,\cdots,r_n$ 的线性移位寄存器,相应的移位寄存器序列为 $\{a_{i_j}\}(i=1,\cdots,n)$。函数 $f(x_1,x_2,\cdots,x_n)$ 是 n 元布尔函数。令 $k_j=f(a_{1_j},a_{2_j},\cdots,a_{n_j})$,则 $\{k_j\}$ 即为该组合生成器的输出。

使用组合生成器可以极大地提高序列的周期。事实上,如果 $r_1,r_2,\cdots,r_n$ 两两互素,函数 $f(x_1,x_2,\cdots,x_n)$ 与各变元均有关,则 $\{k_j\}$ 的周期为 $\prod_{i=1}^{n}(2^{r_i}-1)$。

3. 钟控生成器

钟控方法设计密钥流生成器的基本思想是：用一个或多个移位寄存器来控制另一个或多个移位寄存器的时钟，这样的序列生成器称为钟控生成器（Clock-Controlled Generator）。最终的输出称为钟控序列，基本模型如图 3-19 所示。

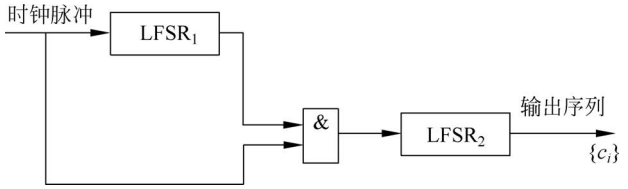

图 3-19　钟控序列生成器基本模型

假设 $LFSR_1$ 和 $LFSR_2$ 分别输出序列$\{a_k\}$和$\{b_k\}$。当 $LFSR_1$ 输出 1 时，移位时钟脉冲通过与门使 $LFSR_2$ 进行一次移位，从而生成下一位。当 $LFSR_1$ 输出 0 时，移位时钟脉冲无法通过与门影响 LFSR，因此 LFSR 重复输出前一位。

例如，假设 $LFSR_1$ 输出周期序列 10101 10101……，$LFSR_2$ 输出周期为 3 的序列 a_0，a_1，a_2，a_0，a_1，a_2，…，则上述钟控生成器输出的钟控序列为 a_0，a_0，a_1，a_1，a_2，a_0，a_0，a_1，a_1，a_2，…，周期为 5。

交错停走式生成器也是一种钟控生成器。这个生成器使用了 3 个不同级数的移位寄存器，如图 3-20 所示。

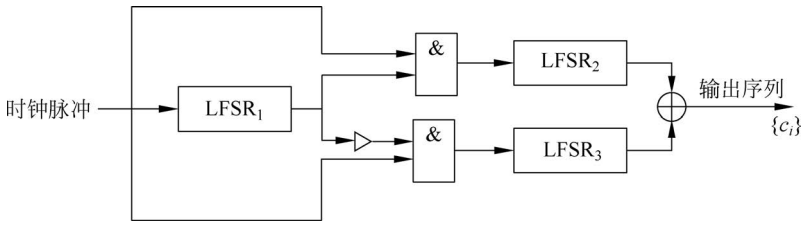

图 3-20　交错停走式生成器

当 $LFSR_1$ 的输出是 1 时，$LFSR_2$ 被时钟驱动；当 $LFSR_1$ 的输出是 0 时，$LFSR_3$ 被时钟驱动。最后，$LFSR_2$ 的输出与 $LFSR_3$ 的输出做异或运算即为这个交错停走式生成器的输出，输出的序列具有长周期和大的线性复杂度。

除利用 LFSR 的良好结构设计伪随机序列生成器之外，还有其他一些基于数论或有限域的知识构造的伪随机序列。这些生成器所依赖的数学工具可对它们的周期、随机统计特性、线性复杂度等进行理论分析。

3.3　分组密码

3.3.1　分组密码概述

在许多密码系统中，分组密码是系统安全的一个重要组成部分。分组密码易于构造

伪随机数生成器、流密码、消息认证码(MAC)和散列函数等,还可进而成为消息认证技术、数据完整性机制、实体认证协议以及单钥数字签字体制的核心组成部分。

分组密码是将明文消息编码表示后的数字序列 $x_0,x_1,\cdots,x_i,\cdots$ 划分成长为 n 的组 $x=(x_0,x_1,\cdots,x_{n-1})$,各组(长为 n 的量)分别在 $k=(k_0,k_1,\cdots,k_{t-1})$ 控制下变换成等长的输出数字序列 $y=(y_0,y_1,\cdots,y_{m-1})$(长为 m 的量),其加密函数 $E:V_n\times K\to V_m$,V_n 为 n 维明文空间,V_m 为 m 维密文空间,K 为密钥空间,如图 3-21 所示。

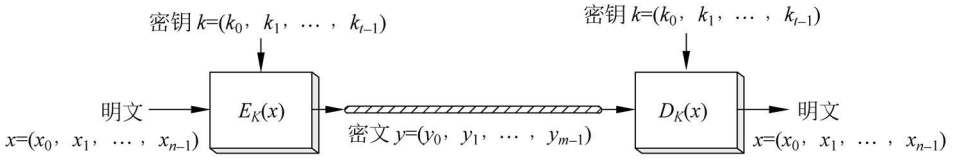

图 3-21 分组密码框图

分组密码与流密码的不同之处在于:输出的每一位数字不是只与相应时刻输入的明文数字有关,而是与一组长为 n 的明文数字有关。在相同密钥下,分组密码对长为 n 的输入明文组所实施的变换是等同的,所以只需研究对任一组明文数字的变换规则。这种密码实质上是字长为 n 的数字序列的代换密码。通常取 $m=n$。若 $m>n$,则为有数据扩展的分组密码;若 $m<n$,则为有数据压缩的分组密码。下面介绍设计分组密码时的一些常用方法。

1. 代换

设明文和密文的分组长都为 n 比特,则明文的每一个分组都有 2^n 个可能的取值。为使加密运算可逆,明文的每一个分组都应产生唯一的一个密文分组,这样的变换是可逆的,称明文分组到密文分组的可逆变换为代换。不同可逆变换的个数有 $2^n!$ 个。

图 3-22 表示 $n=4$ 的代换密码的一般结构,4 比特输入产生 16 个可能输入状态中的一个,由代换结构将这一状态映射为 16 个可能输出状态中的一个,每一输出状态由 4 个比特表示。加密映射和解密映射可由代换表来定义,如表 3-2 所示。这种方法是定义分组密码最简单的常用形式。

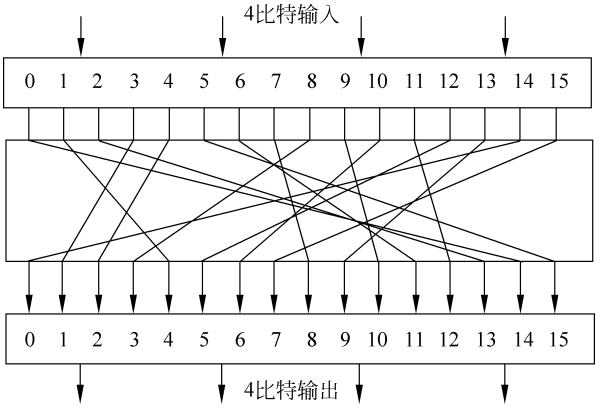

图 3-22 $n=4$ 的代换结构

表 3-2　$n=4$ 对应的代换表

明文	密文	明文	密文
0000	1110	1000	0011
0001	0100	1001	1010
0010	1101	1010	0110
0011	0001	1011	1100
0100	0010	1100	0101
0101	1111	1101	1001
0110	1011	1110	0000
0111	1000	1111	0111

2. 扩散和混淆

扩散和混淆是由信息论创始人香农提出的设计密码系统的两个基本方法，目的是抗击敌手对密码系统的统计分析。如果敌手知道明文的某些统计特性，如消息中不同字母出现的频率、可能出现的特定单词或短语，而且这些统计特性以某种方式在密文中反映出来，那么敌手就有可能得出加密密钥或其一部分，或者得出包含加密密钥的一个可能的密钥集合。在香农称之为理想密码的密码系统中，密文的所有统计特性都与所使用的密钥独立。

所谓扩散，就是将明文的统计特性散布到密文中去，实现方式是使得明文的每一位影响密文中多位的值，即密文中每一位均受明文中多位影响。例如，对英文消息 $M=m_1m_2m_3\cdots$，对字符 c_n 的加密操作为：

$$c_n=m_n \oplus_{26} m_{n+1} \oplus_{26} m_{n+2} \oplus \cdots \oplus_{26} m_{n+k}$$

上式表示密文字母 c_n 由明文中连续 k 个字母进行模 26 相加所得。这样使密文中各字母出现的频率特征较为平均，敌手无从猜测其中的关键短语或者词汇。单纯的扩散较容易被敌手攻破。混淆试图使密文的统计特征与密钥取值的关系尽量复杂，实现混淆的常用方法是代换。要注意的是线性变换起不到有效的混淆效果。

流密码通过反馈设计加入了一些扩散，但它主要依赖于混淆。分组密码算法既用到了扩散，也用到了混淆。E. Schaefer(1996)为教学目的提出了一个简化的 DES(S-DES)加密算法，这个算法非常具体地说明了分组密码中如何实施扩散和混淆。

S-DES 加密算法以 10 位密钥和 8 位明文分组为输入，产生 8 位分组密文输出。其解密算法用同一密钥对 8 位密文分组产生原来的明文分组。图 3-23 给出了 S-DES 算法流程。

S-DES 加密算法步骤如下。

(1) 对输入的 8 位明文分组 m 执行初始置换 IP(m)。

(2) 执行一个包含置换、替代操作且依赖于密钥的变换 f_{k_1}。

(3) 将数据进行左右 4 位两半部分交换 SW。

(4) 结果再执行 f_{k_2}。

(5) 最后执行 IP^{-1} 产生逆密文输出 c，并且

$$c=\mathrm{IP}^{-1}(f_{k_2}(\mathrm{SW}(f_{k_1}(\mathrm{IP}(m)))))$$

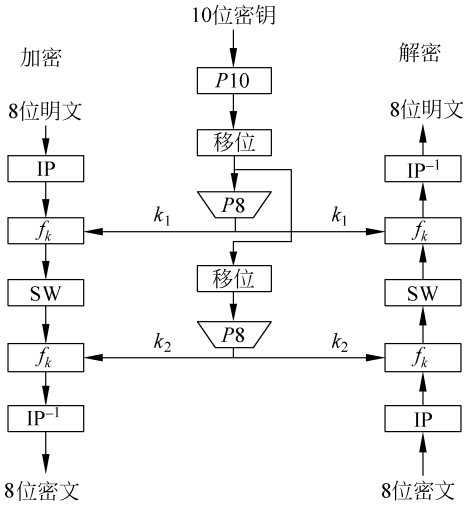

图 3-23　S-DES 算法流程

解密是加密的逆变换,即

$$m = \mathrm{IP}^{-1}(f_{k_1}(\mathrm{SW}(f_{k_2}(\mathrm{IP}(c)))))$$

其中,密钥 k_1、k_2 都是 8 位子密钥,由 10 位输入密钥产生。

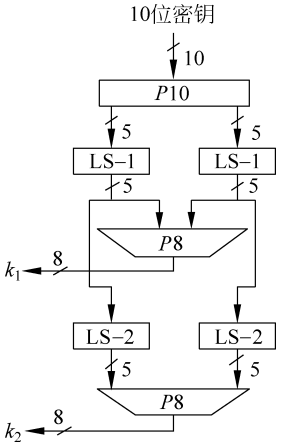

图 3-24　S-DES 密钥生成

1) S-DES 密钥的生成

S-DES 密钥是一个 10 位码,由发送、接收双方共享。两个 8 位子密钥由 10 位输入密钥按照图 3-24 所示的流程生成。子密钥 k_1、k_2 生成过程如下。

(1) 对 10 位码 m 中进行 10 阶置换 $P10$。置换 $P10$ 定义为

$$P10 = \begin{pmatrix} 1,2,3,4,5,6,7,8,9,10 \\ 3,5,2,7,4,10,1,9,8,6 \end{pmatrix}$$

即对 $m=(b_1,b_2,b_3,b_4,b_5,b_6,b_7,b_8,b_9,b_{10})$,$m_1=P10(m)=(b_3,b_5,b_2,b_7,b_4,b_{10},b_1,b_9,b_8,b_6)$。

例如,$P10(1010000010)=(1000001100)$。

(2) 将 m_1 看作左右两个 5 位码 m_{1L} 和 m_{1R},即 $m_1=m_{1L}\parallel m_{1R}$,分别循环左移一次(LS−1,LS 表示左移循环),输出 m_2 仍为 10 位码。如上例,$m_2=\mathrm{LS}-1(m_{1L})\parallel \mathrm{LS}-1(m_{1R})=(0000111000)$。

(3) 10 位转 8 位置换。$P8(b_1,b_2,b_3,b_4,b_5,b_6,b_7,b_8,b_9,b_{10})=(b_6,b_3,b_7,b_4,b_8,b_5,b_{10},b_9)$得子密钥 k_1。如上例,$k_1=P8(m_2)=P8(0000111000)=(10100100)$。

(4) 将 m_2 看作左右两个 5 位码 m_{2L} 和 m_{2R},即 $m_2=m_{2L}\parallel m_{2R}$,分别循环左移二次(LS−2),输出 m_3 仍为 10 位码。如上例,$m_3=\mathrm{LS}-2(m_{2L})\parallel \mathrm{LS}-2(m_{2R})=(0010000011)$。

(5) 对 m_3 做 10 位转 8 位置换 $P8$,得子密钥 $k_2=P8(m_3)$。如上例,$k_2=P8(m_3)=P8(0010000011)=(01000011)$。

2）S-DES 加密操作

S-DES 加密操作首先执行一个 8 位的置换 IP：

$$\mathrm{IP}=\begin{pmatrix}1,2,3,4,5,6,7,8\\2,6,3,1,4,8,5,7\end{pmatrix}$$

即 $\mathrm{IP}(b_1,b_2,b_3,b_4,b_5,b_6,b_7,b_8)=(b_2,b_6,b_3,b_1,b_4,b_8,b_5,b_7)$。将 IP 上下两行互换，即得 IP 的逆置换 IP^{-1}，$\mathrm{IP}^{-1}(\mathrm{IP}(x))=x$。加密操作中最复杂的是 f_k，f_k 是若干置换和代换的组合。如图 3-25 所示，$f_k(L,R)=(L\oplus P4((S_0\parallel S_1)(E/P(R)\oplus \mathrm{Key})))\parallel R$，其中：

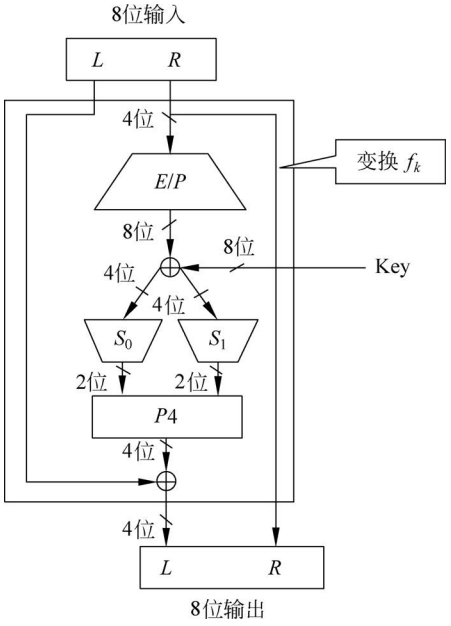

图 3-25　变换 f_k 的细节

① L、R 分别为输入字节的左半 4 位和右半 4 位。

② E/P 为一个 4 位到 8 位的扩展变换：$E/P(b_1,b_2,b_3,b_4)=(b_4,b_1,b_2,b_3,b_2,b_3,b_4,b_1)$。

③ ⊕是按位异或运算。

④ S_0、S_1 分别为 4 位到 2 位的变换盒，S_0 作用于 8 位字节的左 4 位，S_1 作用于 8 位字节的右 4 位。S_0、S_1 定义如下：

$$S_0=\begin{bmatrix}1&0&3&2\\3&2&1&0\\0&2&1&3\\3&1&3&2\end{bmatrix}\qquad S_1=\begin{bmatrix}0&1&2&3\\2&0&1&3\\3&0&1&0\\2&1&0&3\end{bmatrix}$$

变换盒 S 的操作过程是将 4 位输入码的第 0、3 位作为 2 位数值 i，第 1、2 位作为 2 位数值 j，则 S 盒中元素 S_{ij} 即为输出。例如，$S_0(1110)=3=(11)$。

① $P4$ 是一个 4 位置换：$P4(b_1,b_2,b_3,b_4)=(b_2,b_4,b_3,b_1)$。

② ‖表示两个位串的拼接。

S-DES是一个对称分组加密的简化模型,它揭示了设计分组密码算法的基本模式和框架。但是,S-DES没有实际应用价值。由于算法是公开的,所有秘密都在密钥中。即假设攻击者掌握S-DES算法细节(包括IP、E/P、S_0、S_1、$P4$的值),已知明文$m=(b_1,b_2,b_3,b_4,b_5,b_6,b_7,b_8)$和对应密文$c=(p_1,p_2,p_3,p_4,p_5,p_6,p_7,p_8)$,则通过穷举攻击,遍历$2^{10}=1024$个可能密钥,一定可以找出正确的加密密钥。

现代分组密码算法应满足以下要求。

(1) 分组长度n要足够大,使分组代换字母表中的元素个数2^n足够大,防止明文穷举攻击法奏效。DES、IDEA分组长度$n=64$,AES的分组长度$n=128$。

(2) 密钥空间要足够大,尽可能消除弱密钥并使所有密钥的加密强度相同,但是为便于密钥管理,密钥又不能过长。DES的密钥长度为56位,但被认为太短。AES的密钥长度为128位,目前被认为是安全的。

(3) 由密钥确定置换的算法要足够复杂,充分实现明文与密钥的扩散和混淆,能抗击各种已知的密码分析攻击。使得敌手除穷举攻击外没有其他捷径可循。

(4) 加密和解密运算简单,易于软件和硬件高速实现。通常的考虑是分组长度应该是2的幂,如取32、64、128、256等,密码运算的基本操作是加、与、或、异或、移位和矩阵变换等。

(5) 通常不考虑数据位扩展或者压缩,即输入明文和输出密文具有相同长度。

(6) 差错传播尽可能小。

扩散和混淆成功地实现了分组密码的本质属性,因而成为设计现代分组密码的基础。

3.3.2 Feistel密码结构

1. Feistel加密结构

由图3-23,S-DES算法对8位明文的加密过程分为两个阶段:第一阶段对明文作置换IP,然后施以具有扩散和混淆效果的代换处理f_{k_1};第二阶段对上阶段输出交换左、右半部,然后再次施以f_{k_2},最后实施IP^{-1}。其中每个阶段的模式基本一致。将实施一次f_{k_i}称为一轮,则下一轮的输入和上一轮输出的关系是:

$$L_2=R_1$$
$$R_2=L_1\oplus F(R_1,k_2)$$

这种交换左、右半部,其中下一轮右半部为上一轮左半部和依赖于上轮右半部与子密钥的变换值,下一轮左半部为上一轮右半部的框架模式称为Feistel加密结构。

S-DES仅实施了两轮具有Feistel结构特征的处理,一般的Feistel加密过程需要实施n轮迭代,其中第i轮迭代关系如下:

$$L_i=R_{i-1}$$
$$R_i=L_{i-1}\oplus F(R_{i-1},k_i)$$

其中,k_i是第i轮用的子密钥,由加密密钥k得到。

Feistel网络中每轮结构都相同,总是依赖于上轮右半部数据与子密钥经过变换函数

$F(R_{i-1},k_i)$处理后，其结果与上轮左半部数据进行异或运算，这就是前面介绍的代换操作。代换过程完成后，再交换左、右两半数据，这一过程称为置换。这种结构是香农提出的代换-置换网络 SPN(Substitution-Permutation Network)的特有形式。

S-DES 中增加了开始的置换 IP 和最后结束的逆置换 IP^{-1}。在完整的 DES 中也是这样处理的。

Feistel 网络的实现与以下参数和特性有关。

(1) 分组大小。分组越大则安全性越高，但加密速度就越慢。分组密码设计中最为普遍使用的分组大小是 64 比特或者 128 比特。

(2) 密钥大小。密钥越长则安全性越高，但加密速度就越慢，现在普遍认为 64 比特或更短的密钥长度是不安全的，通常使用 128 比特的密钥长度。

(3) 轮数。单轮结构远不足以保证安全性，多轮结构有足够的安全性。典型的轮数取为 16。

(4) 子密钥产生算法。该算法的复杂性越大，则密码分析的困难性就越大。

(5) 轮函数 F。轮函数 F 的复杂性越大，密码分析的困难性也越大。

(6) 算法分析难度。算法结构清晰，解释没有二义性，则容易分析算法的复杂性和抗攻击能力。

2. Feistel 解密结构

Feistel 解密过程本质上和加密过程一样，算法使用密文作为输入，但使用子密钥 k_i 的次序与加密过程相反，即第 1 轮使用 k_n，第 2 轮使用 k_{n-1}，…，最后一轮使用 k_1。这一特性保证了解密和加密可采用同一算法。

图 3-26 的左、右图分别表示 16 轮 Feistel 结构的加、解密过程，加密过程由上而下，解密过程由下而上。为清楚起见，加密算法每轮的左右两半用 LE_i 和 RE_i 表示，解密算法每轮的左右两半用 LD_i 和 RD_i 表示。图中右边标出了解密过程中每一轮的中间值与左边加密过程中间值的对应关系，即加密过程第 i 轮的输出是 $LE_i \| RE_i$（$\|$ 表示连接），解密过程第 16-i 轮相应的输入是 $LD_i \| RD_i$。

加密过程的最后一轮执行完后，左右两半输出再经交换，因此密文是 $RE_{16} \| LE_{16}$。解密过程取以上密文作为同一算法的输入，即第 1 轮输入是 $RE_{16} \| LE_{16}$。下面证明解密过程第 1 轮的输出等于加密过程第 16 轮输入左右两半的交换值。

在加密过程中：

$$LE_{16} = RE_{15}$$
$$RE_{16} = LE_{15} \oplus F(RE_{15},k_{16})$$

在解密过程中：

$$LD_1 = RD_0 = LE_{16} = RE_{15}$$
$$RD_1 = LD_0 \oplus F(RD_0,k_{16}) = RE_{16} \oplus F(RE_{15},k_{16})$$
$$= [LE_{15} \oplus F(RE_{15},k_{16})] \oplus F(RE_{15},k_{16}) = LE_{15}$$

所以解密过程第 1 轮的输出为 $LE_{15} \| RE_{15}$，等于加密过程第 16 轮输入左右两半交换后的结果。容易证明这种对应关系在 16 轮中每轮都成立。

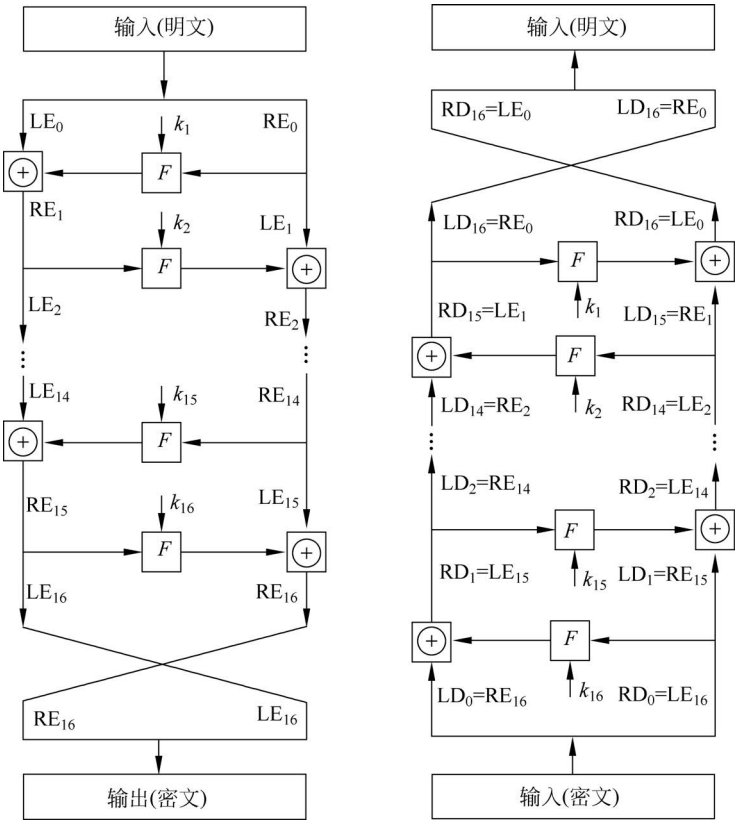

图 3-26　Feistel 加、解密过程

通常有 $LD_i=RE_{16-i}$，$RD_i=LE_{16-i}$，$i=1,2,\cdots,16$。明文 $m=(LE_0 \parallel RE_0)$。这里并不要求 F 是可逆函数，事实上 F 的构造任意，以上过程仍然成立，但是从密码强度考虑，函数 F 的构造是关键。

3.4 DES

3.4.1 DES 算法简介

DES(Data Encryption Standard，数据加密标准)是 IBM 的研究成果，是数据加密算法(Data Encryption Algorithm，DEA)的规范描述，在 1997 年被美国政府正式采纳。值得注意的是，IBM 在美国国家标准局(NBS)第二次发布征集公告后，所提交的候选算法是在其早先开发的 Lucifer 算法基础上修改和发展而来的，密钥长度为 112，但是公布的 DES 算法的密钥长度为 56。无论如何，DES 算法成为使用最广泛的密钥系统之一，在各个领域特别是在金融领域数据安全保护中广泛应用，与此同时，对 DES 安全性的研究也在不断继续。

1997 年一个研究小组经过 4 个月努力，在 Internet 上搜索了 3×10^{16} 个密钥，找出了 DES 的密钥。同年美国国家标准与技术研究所(NIST)宣布 1998 年 12 月以后美国政府

不再使用 DES，并且发出征集 AES（高级加密标准）的通知。1998 年 5 月美国研究机构 EFF（Electronic Frontier Foundation）宣布用一台价值 20 万美元的计算机改装的专用解密系统，花费 56h 破译了 56 位密钥的 DES。2000 年 10 月 2 日，NIST 公布了新的 AES，DES 作为标准正式结束。尽管如此，学习 DES，对于掌握分组密码的基本理论和设计思想仍然有重要参考价值。同时在非机密级的许多应用中，DES 仍在广泛使用。

3.4.2　DES 算法设计思想

DES 算法对明文以 64 位为分组单位进行分组，最后一组若不足 64 位，以 0 补齐。之后对每组 64 位的数据进行加密。DES 中密钥长度为 56 位，输出 64 位密文分组，其加密算法框图如图 3-27 所示。图的左边是明文的加密处理过程，该过程分三个阶段。

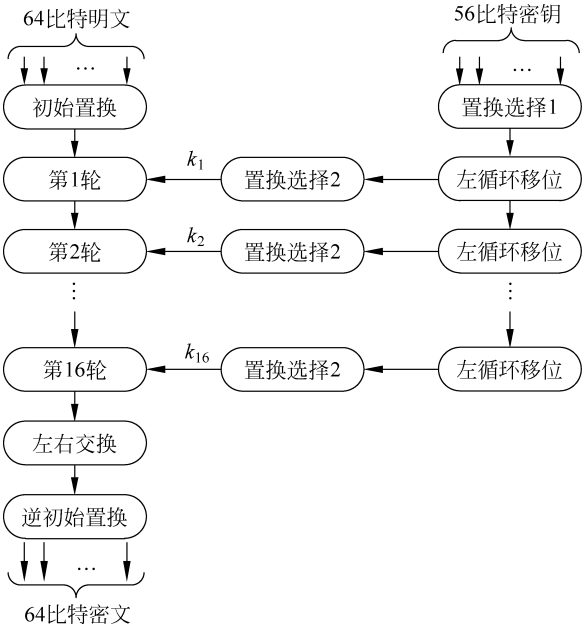

图 3-27　DES 加密算法框图

第一阶段：64 位明文进行初始置换 IP，用于重排明文分组的 64 比特数据。

第二阶段：顺序经过 16 轮功能相同的变换，每一轮变换的输入是上轮变换的输出和右部 56 位初始密钥生成的 48 位子密钥 k_i。对加密过程中每一轮，子密钥分发前都需进行左循环移位。因此 $k_1,\cdots,k_{16}$ 各不相同。

第三阶段：将第 16 轮变换输出的 64 位码进行左右 32 位变换，然后对其进行逆初始置换。所得结果即为 64 位密文分组。

除初始置换和逆初始置换外，DES 的结构和 Feistel 图的密码结构完全相同。

DES 算法中每轮处理细节如图 3-28 所示。结合图 3-27 可知，DES 的 16 轮循环处理流程具有严格的 Feistel 密码结构。由图 3-28 的左部，第 i 轮处理过程为：F 函数的输入为前一轮输出的右半部分 R_{i-1} 和当前轮密钥 k_i。F 函数的输出将与加密左半部分输入位 L_{i-1} 进行 XOR 操作产生 R_i。和 Feistel 网络一样，每轮变换可由以下公式表示：

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus F(R_{i-1}, k_i)$$

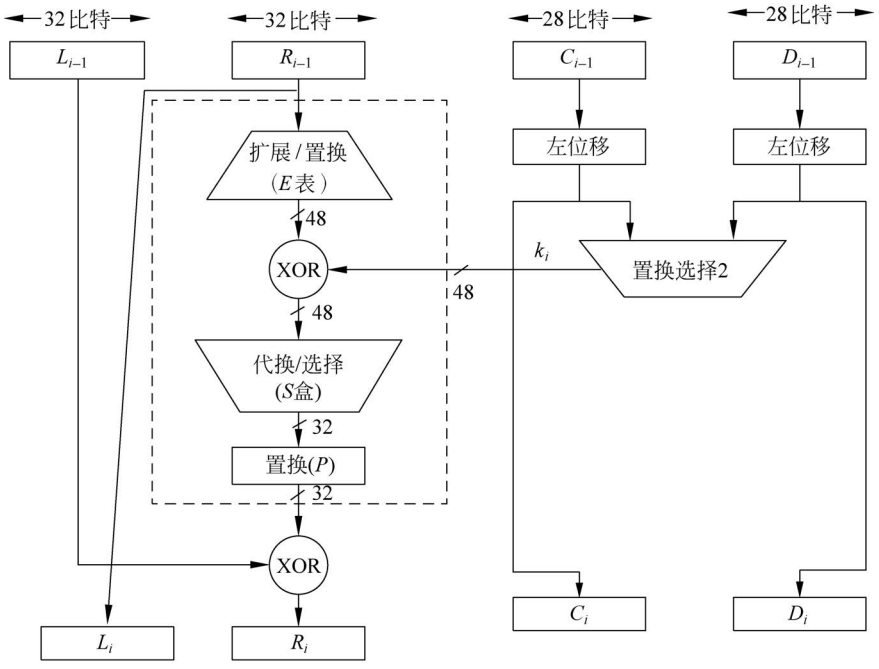

图 3-28　DES 算法单轮处理过程

3.4.3　DES 算法内部结构

DES 的基本构造元件为初始置换与逆初始置换、F 函数以及密钥生成,其中 F 函数涉及 E 盒扩展置换、S 盒置换和 P 盒置换。

1. 初始置换与逆初始置换

1) 初始置换 IP

初始置换 IP 的功能是把输入的 64 位明文数据,通过一个 8×8 的置换矩阵按位进行重新组合,如表 3-3 所示。初始置换是线性变换,它使明文发生位置上的变换。

表 3-3　初始置换 IP

IP							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

设 x 是分块后的 64 位明文数据块,置换后 $x_0 = \mathrm{IP}(x) = L_0R_0$,这里 L_0 和 R_0 都是

32 位。初始置换后效果如图 3-29 所示。

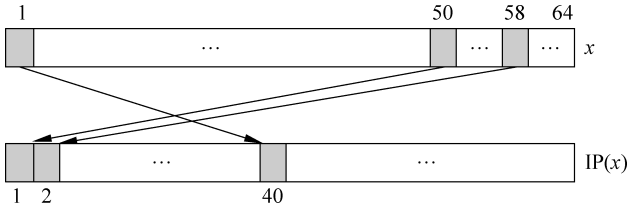

图 3-29　初始置换中位交换的示例

2）逆初始置换 IP^{-1}

在加密中，经过 16 次迭代运算后得到 L_{16} 和 R_{16}，将此作为输入进行逆初始置换，即得到密文输出。逆置换正好是初始置换的逆运算。其逆置换的规则如表 3-4 所示。

表 3-4　逆初始置换 IP^{-1}

IP^{-1}							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

逆初始置换后效果如图 3-30 所示。

值得注意的是，初始置换和逆初始置换都没有增加 DES 的安全性。尽管人们不是很清楚这两种置换存在的真正原理，但看上去他们的初衷是以字节形式排列明文和密文，以方便 8 位数据总线的数据读取。8 位数据总线是 20 世纪 70 年代初期最新的寄存器大小。

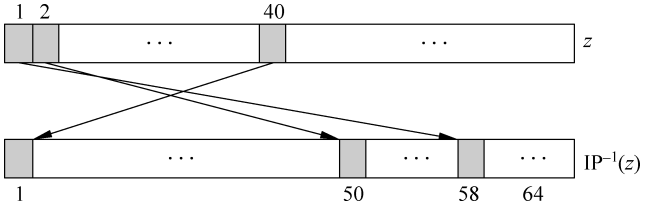

图 3-30　逆初始置换中位置换的示例

2. 函数计算 $F(R_{i-1},k_i)$

正如前文所述，F 函数在 DES 的安全性中发挥着重要作用，F 函数实现原理如图 3-31 所示。

1）E 盒扩展置换

首先将输入分成 8 个 4 位的分组，然后将每个分组扩展为 6 位，从而将 32 位的输入扩展为 48 位。这个过程在 E 盒中进行，E 盒是一种特殊的置换。第一个分组包含的位为(1,2,3,4)，第二个分组包含的位为(5,6,7,8)，以此类推。

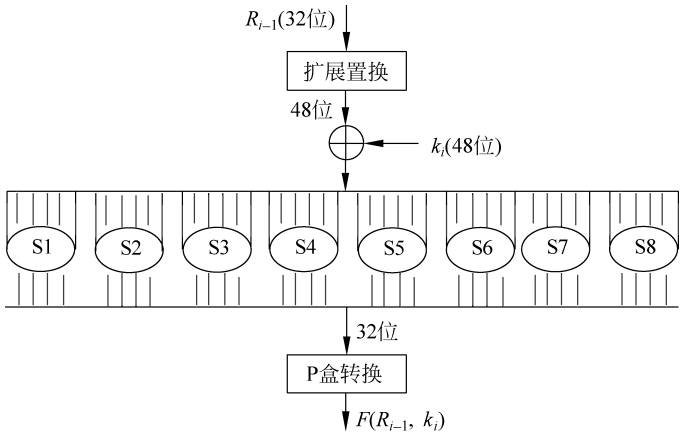

图 3-31　F 函数实现原理

图 3-32 显示了将 4 位扩展为 6 位的过程。

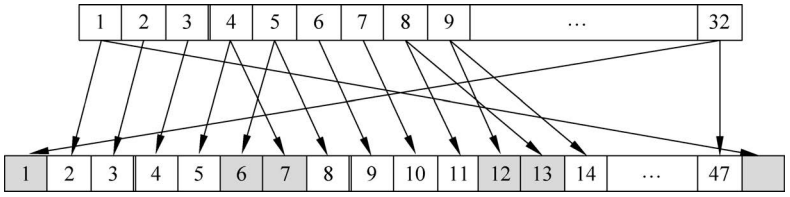

图 3-32　E 盒扩展置换的置换示例

从表 3-5 可知,32 个输入位中正好有 16 个输入位在输出中出现了两次。但是任意一个输入位都不会在同一个 6 位的输出分组出现两次。扩展盒增加了 DES 的扩散行为,因为某些输入位会影响两个不同的输出位置。

表 3-5　E 盒扩展置换表

E					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

2) S 盒置换

将 E 盒扩展得到的 48 位结果与当前轮密钥 k_i 进行 XOR 操作,并将 8 个 6 位长的分组送入 8 个不同的替换盒中,这个替换盒也称 S 盒,如表 3-6 所示,每个 S 盒都是一个查找表,它将 6 位的输入映射为 4 位的输出。

每个 S 盒包含 $2^6=64$ 项,可以表示为一个 4 行 16 列的表格。每项是一个 4 位的值。图 3-33 列出了表格的读取方式:每个 6 位输入中最重要的位(MSB)和最不重要的位

(LSB)将选择表行,而 4 个内部位则选择列。该表中每个项的整数 0,1,…,15 表示的是 4 位值对应的十进制的值。

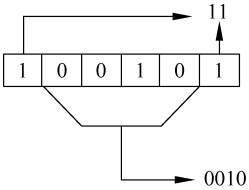

图 3-33　使用 S 盒 1 对输入 1001012 进行解码的示例

从密码学强度来讲,S 盒是 DES 的核心,因为 S 盒在密码中引用了非线性,即

$$S(a) \oplus S(b) \neq S(a \oplus b)$$

S 盒中的非线性构造元件也是 DES 算法中唯一的非线性元素,并提供了混淆。有了这些特征,DES 算法可以抵御各种高级的数学攻击,尤其是差分密码分析的攻击。

例 3-5　S 盒的输入 $b=(100101)_2$ 表示行 $11_2=3$(即第 4 行),以及列 $0010_2=2$(即第 3 列)。如果将输入 b 送入 S 盒 1,则输出为 $S_1(37=100101_2)=8=1000_2$。

例 3-6　求出用 DES 的 8 个 S 盒(见表 3-6)将 48 比特串 70a990f5fc36 压缩置换输出的 32 比特串(用十六进制写出每个 S 盒的输出)。

解:比特串 70a990f5fc36 用二进制表示为 011100 001010 100110 010000 111101 011111 110000 110110,每 6 比特一组共 8 组,分别用 8 个 S 盒变换如下:

$$S_1(011100)=S_1(00,1110)=S_1(0,14)=0=0000=0$$

$$S_2(001010)=S_2(00,0101)=S_2(0,5)=11=1011=\mathrm{b}$$

$$S_3(100110)=S_3(10,0011)=S_3(2,3)=9=1001=9$$

$$S_4(010000)=S_4(00,1000)=S_4(0,8)=1=0001=1$$

$$S_5(11101)=S_5(11,1110)=S_5(3,14)=5=0101=5$$

$$S_5(11101)=S_5(11,1110)=S_5(3,14)=5=0101=5$$

$$S_7(110000)=S_7(10,1000)=S_7(2,8)=10=1010=\mathrm{a}$$

$$S_8(110110)=S_8(10,1011)=S_8(2,11)=13=1101=\mathrm{d}$$

故 8 个 S 盒的输出为 00001011 10010001 01011000 10101101,即 0b9158ad。

表 3-6　S 盒置换表

S_1	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S_2	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	13	7	0	9	3	4	6	10	2	8	5	14	13	11	15	1
	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

续表

S_4	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S_5	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	6	11	12	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

3) P 盒置换

S 盒置换后产生 32 位输出,该输出再经过表 3-7 定义的 P 盒置换进行按位置换,产生的结果即为函数 $F(R_i, k_i)$的输出。

表 3-7　P 盒置换表

P							
16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

与初始置换 IP 及其逆初始置换 IP^{-1} 不同,P 盒置换将扩散引入 DES 中,因为每个 S 盒的 4 位输出都会进行置换,使得每位在下一轮中会影响多个不同的 S 盒。由扩充带来的扩散、S 盒与 P 盒置换可以保证,在第 5 轮结束时每个位都是每个明文位与每个密钥位的函数。这种行为也称雪崩效应。

3. 子密钥 k_i 的产生

图 3-34 显示了实际密钥生成过程。DES 的输入密钥通常是 64 位,由于 DES 算法规定,其中每第 8 个位都作为前面 7 位的一个奇偶校验位,不参与 DES 运算。

故经过表 3-8 所示的初始 PC-1(置换选择 1)置换后密钥的实际可用位数由 64 位压缩为 56 位,可以说 DES 是一个 56 位的密码,而不是 64 位的。

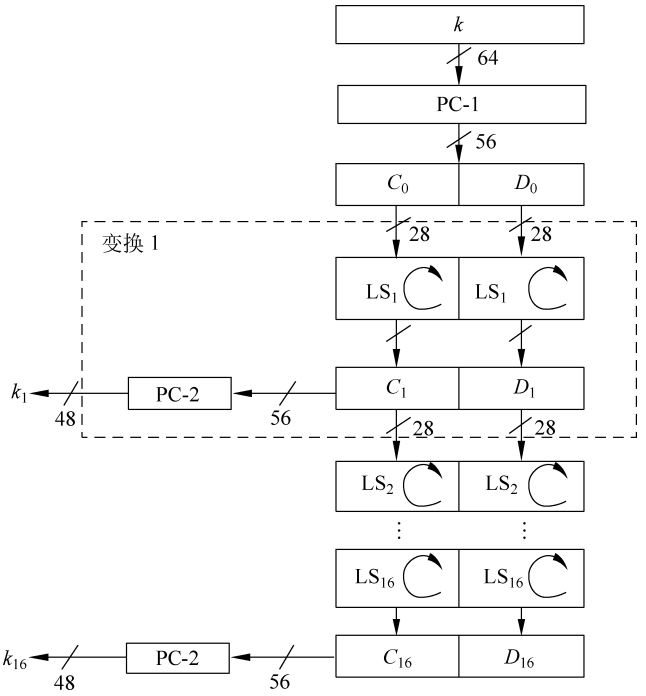

图 3-34　DES 加密的密钥生成过程

表 3-8　初始密钥置换 PC-1

PC-1							
57	49	41	33	25	17	9	1
58	50	42	34	26	18	10	2
59	51	43	35	27	19	11	3
60	52	44	36	63	55	47	39
31	23	15	7	62	54	46	38
30	22	14	6	61	53	45	37
29	21	13	5	28	20	12	4

长度均为 28 位的左右两部分将周期性地向左移动 1 位或 2 位(即循环移位),而移动的具体位数则取决于轮数 i,如表 3-9 所示,其规则如下:

表 3-9　循环左移位数

轮	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
位数	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

(1) 在 $i=1,2,9,16$ 轮中,左右两部分向左移动 1 位。

(2) 在 $i\neq1,2,9,16$ 的其他轮中,左右两部分向左移动 2 位。

这里需要注意的是,循环移动位置的总数为 $4\times1+12\times2=28$,这样会使得 $C_0=C_{16}$ 和 $D_0=D_{16}$,此结果对解密密钥的生成非常有用。

移位后的结果作为求下一轮子密钥的输入,同时也作为表 3-10 的 PC-2(置换选择 2)

的输入,即密钥的左右两部分需要再次根据 PC-2 进行按位置换。C_i 和 D_i 总共有 56 位,而 PC-2 忽略了其中的 8 位,得到 48 位的本轮子密钥 k_i,作为函数 $F(R_{i-1},k_i)$的输入。

表 3-10　PC-2 的轮密钥置换

PC-2							
14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

4. DES 算法解密过程

DES 算法的优势之一是其解密过程与加密过程在本质上是完全相同的。这主要是因为 DES 基于 Feistel 网络。与加密相比,解密过程中只有密钥生成顺序逆转了,即解密的第一轮需要子密钥 k_{16},第二轮需要子密钥 k_{15},……,最后一轮用 k_1,算法本身并没有任何变化。

3.4.4　DES 算法的安全性

1. DES 算法的安全强度

在 DES 算法被提出后不久,针对 DES 密码强度的批评主要围绕以下两个方面。

(1) DES 的密钥空间太小,该算法很脆弱,易受蛮力攻击。

IBM 提议的原始密码的密钥长度为 128 位,而将它减少为 56 位的做法很令人怀疑。此外,DES 算法所使用的密钥 k_i 是各次迭代中递推产生的,这种相关性也必然降低了密码体制的安全性。

DES 蛮力攻击也为硬件开销的不断下降提供了很好的学习案例。EFF(Electronic Frontier Foundation)于 1998 年构建的硬件机器 Deep Crack,使用蛮力攻击可以在 56 小时内破解 DES,其平均搜索时间为 15 天。在 2006 年,来自德国波鸿大学和基尔大学一个研究小组基于商业集成电路构建的 COPACOBANA 机器破解 DES 的官方平均搜索时间不到 7 天。

总之,56 位的密钥大小已经不足以保证当今机密数据的安全性。因此,对大多数应用程序而言,单重 DES 只能用于要求短期安全性(比如几小时)的应用或被加密数据价值较低的情况。不过,多重 DES 仍然很安全,尤其是三重 DES。

(2) DES 算法中 S 盒的设计准则是保密的,所以有可能已经存在利用 S 盒数学属性的分析攻击,只是此攻击只有 DES 的设计者知道。

尽管 DES 算法自公布之日起就经历了许多很强的分析攻击,但至今还没发现能高效破解它的攻击方式。1990 年,Eli Biham 和 Adi Shamir 发现了差分密码分析(DC),这是一种非常强大的攻击方式,理论上它可以破解任何分组密码。1993 年,Mitsuru Matsui 公布了一种与 DC 相关但又不同的分析攻击,即线性分析攻击(LC)。与差分密码分析类

似，这种攻击的有效性很大程度上取决于 S 盒的结构。

然而事实证明，DES 的 S 盒可以很好地抵抗住了这些攻击。

2. DES 算法的安全管理

1）避开 DES 算法漏洞

在 DES 密钥 k_i 的使用、管理及密钥更换的过程中，应绝对避开 DES 算法的应用误区，即绝对不能把 k_i 的第 8、16、24、64 位作为有效数据位，来对 k_i 进行管理。从上述 DES 算法的描述中知道，每个字节的第 8 位作为奇偶校验位以确保密钥不发生错误，这 8 位不参与 DES 运算。因此，如果采用定期更换 DES 密钥 k_i 的办法来进一步提高系统的安全性和可靠性，忽略了上述应用误区，那么，更换新密钥将是徒劳的。所以更换密钥一定要保证新 k_i 与旧 k_i 真正的不同，即除了第 8、16、24、64 位以外其他位数据发生了变化，这样才能保证 DES 算法安全可靠地发挥作用。

2）DES 算法存在弱密钥

在 DES 算法中存在 12 个半弱密钥和 4 个弱密钥。由于在子密钥的产生过程中，密钥被分成了两个部分，如果这两个部分密钥是全 0 或全 1，那么每轮产生的子密钥都是相同的，当密钥是全 0 或全 1，或者一半是 1 或 0 时，就会产生弱密钥或半弱密钥，DES 算法的安全性就会变差。因此，在设定密钥时应避免弱密钥或半弱密钥的出现。

3.4.5　多重 DES

1. 二重 DES

为了提高 DES 的安全性，并利用实现 DES 的现有软硬件，可将 DES 算法在多密钥下多重使用。二重 DES 是多重使用 DES 最简单的形式，如图 3-35 所示。其中明文为 P，密文为 C，两个加密密钥为 k_1 和 k_2。

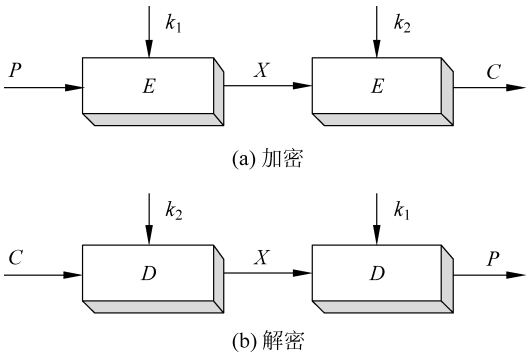

图 3-35　二重 DES

加密过程为 $C=E_{k_2}[E_{k_1}[P]]$。解密时，以相反顺序使用两个密钥 $P=E_{k_1}[E_{k_2}[C]]$。

对于任意 56 位密钥 k_1、k_2，是否能够找出 56 位密钥 k_3，使得 $E_{k_3}[P]=E_{k_2}[E_{k_1}[P]]$？换言之，是否存在等价于二重 DES 的单重 DES 算法？研究表明，这是不可能的。但是由于 DES 本身的安全性有限，对二重 DES 有以下一种称为中途相遇攻击的攻击方案，这种攻击不依赖于 DES 的任何特性，因而可用于攻击任何分组密码。其基本思想如下：

设 $C=E_{k_2}[E_{k_1}[P]]$,则令 $X=E_{k_1}[P]=D_{k_2}[C]$。如果知道明文-密文对(M,C),可以用如下方法进行攻击(找出密钥):

(1) 用 2^{56} 个所有可能密钥 k_1 对 P 加密,将结果按照递增序存入一个表 T 中。

(2) 用 2^{56} 个所有可能的 k_2 对 C 解密,在表 T 中查找与 C 解密结果相匹配的项。

(3) 如果找到匹配项,则记下相应的 k_1 和 k_2。

(4) 最后再用一新的明文-密文对(P',C')检验上面找到的 k_1 和 k_2,即 C'是否等于 $E_{k_2}[E_{k_1}[P']]$,如果相等则攻击有效。

对已知的明文 P,二重 DES 能产生 2^{64} 个可能的密文,而可能的密钥个数为 2^{112} 个。因此就平均情况而言,对一个已知的明文,有 $2^{112}/2^{64}=2^{48}$ 个密钥可产生已知的密文。而再经过另外一对明文密文的检验,误报率将下降到 $2^{48-64}=2^{-16}$。所以在实施中途相遇攻击时,如果已知两个明文密文对,则找到正确密钥的概率为 $1-2^{-16}$。

2. 三重 DES

为了抵抗中途相遇攻击,又提出图 3-36 所示的三重 DES(TDES):$C=E_{k_3}[D_{k_2}[E_{k_1}[M]]]$。其中第二步的解密操作的目的就是阻断中途相遇攻击。其使用了三个不同的密钥,TDES 密钥的有效长度为 168,它以 DES 为基础,但安全性大大增加。1999 年,三重 DES 被合并到数据加密标准中(FIPS PUB 46-3)。许多基于 Internet 的应用采用了三重 DES,其中包括 PGP 和 S/MIME。考虑到实现效率和开销,实际应用中使用的是两个密钥的三重 DES,即在三重 DES 中令 $k_1=k_3$。

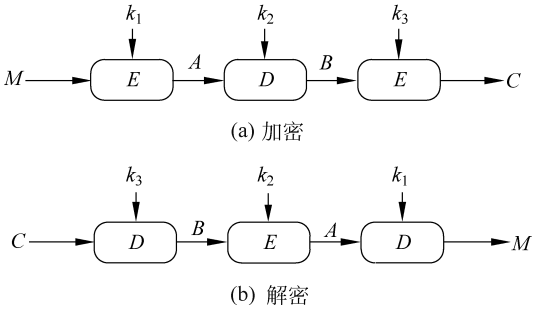

图 3-36 三重 DES

3.5 AES

3.5.1 AES 算法简介

随着未来计算机能力的提高,DES 加密技术已经难以满足长期高安全性的加密要求。NIST 于 1997 年公开征集新一代加密算法,希望能找到一种新的加密算法,选定的算法最终会形成 AES(Advanced Encryption Standard,高级加密标准)。新的加密算法要求满足四个基本条件:运算速度要比三重 DES 更快;安全强度不低于三重 DES;数据分

组长度采用 128 位；密钥可支持 128/192/256 位等多种长度。

1998 年 8 月，在首届 AES 会议上公布了 15 个候选算法。1999 年 8 月，最终确定了五个算法作为候选方案进一步提交讨论，候选算法包括 RC6、Rijndael、Twofish、MARS、Serpent。最后在 2000 年 10 月，选定由比利时的 Joan Daemen 和 Vincent Rijmen 提出的 Rijndael 算法作为 AES 的标准算法。Rijndael 算法是一个分组密码算法，其分组长度和密钥长度相互独立，都可以改变，分组长度可以支持 128 位、192 位、256 位的明文分组长度。NIST 对算法进行了一定的修改以简化其复杂度，修改后的算法只提供 128 位的明文分组长度，能符合当时的各种主流应用环境。

最终形成的 AES 算法是一个对称密钥的分组密码，是一个采用迭代的反复重排方式来实现加解密的演算法，以 128 位(16 字节)分组大小加密和解密数据。算法采用的密钥长度有 128、192、256 位三种，分别形成 AES-128、AES-192、AES-256 系统。AES 标准已成为 NIST 用于加密电子数据的规范，由于采用了新的加密算法，这样可更好地保护金融、电信和政府数字信息的安全。

3.5.2　AES 算法设计思想

Rijndael 算法分组大小和密钥大小都可以为 128、192 或 256 位。根据 AES 标准要求，只有分组长度为 128 位的 Rijndael 才称为 AES 算法。本节只讨论分组长度为 128 位的 Rijndael 的标准版本。图 3-37 显示了 AES 输入/输出参数。

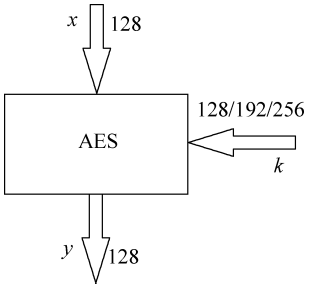

图 3-37　AES 输入/输出参数

AES 算法同时支持三种密钥长度。一般而言，加密轮数 N_r 取决于密钥长度 l_k，两者之间关系为 $N_r=6+l_k/32$。表 3-11 给出了两者的关系。

表 3-11　AES 轮函数与密钥关系

参　数	轮　函　数		
	AES-128	**AES-192**	**AES-256**
密钥长度 N_r	128	192	256
轮数 l_k	10	12	14

与 DES 不同，AES 未采用 Feistel 结构。Feistel 网络在每轮迭代中并没有加密整个分组，例如单轮 DES 只加密了 64/2＝32 位。而 AES 在一次迭代中就加密了所有 128 位。这也是为什么 AES 的轮数比 DES 少的原因。

AES 加密框图如图 3-38 所示。其中明文用 x 表示，密文用 y 表示，轮数用 N_r 表

示。其轮函数是由三个不同的可逆变换组成的。每个变换的设计遵循“宽轨迹策略”。所谓宽轨迹策略,是指抗线性分析和差分分析的一种策略,其实现思想体现在轮函数中的三种功能层。

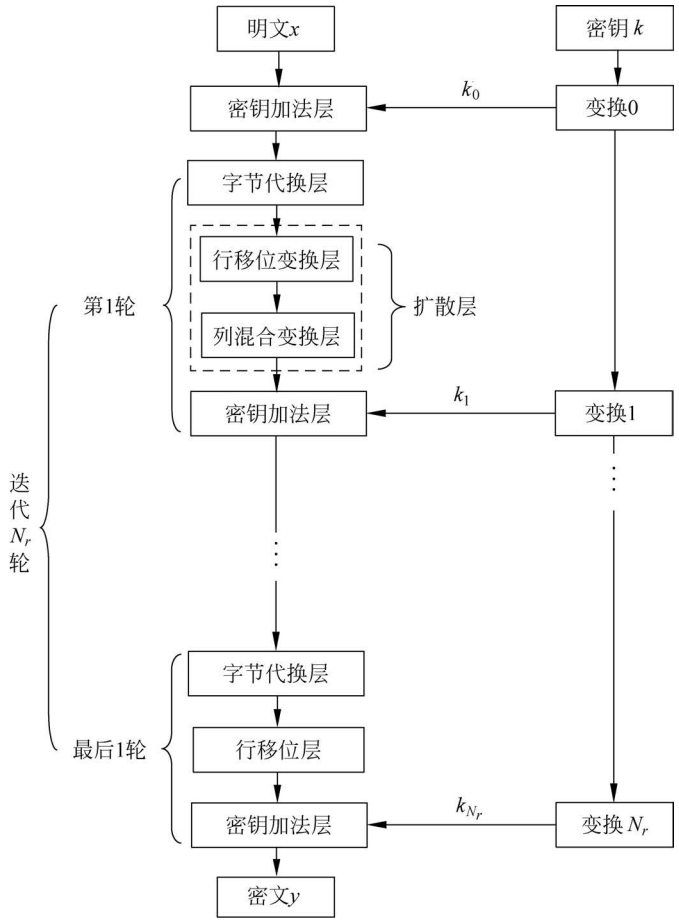

图 3-38　AES 加密框图

1) 非线性层(字节代换层)

将具有最优的“最坏情况非线性特性”的 S 盒并行使用,即状态中的每个元素都使用具有特殊数学属性的查找表进行非线性变换。这种方法将混淆引入数据中,即它可以保证对单个状态位的修改,可以迅速传播到整个数据路径中。这就导致线性逼近和差分分布表中的各项趋近于均匀分布,为抵御差分和线性攻击提供了安全性。

2) 线性混合层(扩散层)

为所有状态位提供扩散。它由两个子层组成,每个子层都执行线性操作。

(1) 行移位变换(ShiftRows)层:在位级别进行数据置换。

(2) 列混合变换(MixColumns)层:是一个混淆操作,它合并(混合)了长度为 4 个字节的分组。

3) 密钥加法层

128 位轮密钥(或子密钥)来自于密钥生成中的主密钥,它将与状态进行相加(异或)操作。

与 DES 类似,AES 密钥的生成也从原始 AES 密钥中计算出轮密钥或子密钥(k_0,k_1,…,k_n)。在 AES 算法中,除第一轮外,其他每轮都是由三种功能层组成。此外,最后一轮 N_r 并没有使用列混合变换,而这种方式使得加密方案和解密方案正好对称。

3.5.3　AES 算法相关知识

在进一步描述各层的内部功能前,首先定义一系列基本概念。

1. AES 算法的基本概念

1) 字节

AES 算法中的基本运算单位是字节(byte),即作为一个整体的 8 位二进制序列。算法的输入数据、输出数据和密钥都以字节为单位,明文、密钥和密文数据串被分隔成一系列 8 个连续比特的分组,并形成字节数组。假设一个 8 字节的分组 b 是由字节序列{b_7,b_6,b_5,b_4,b_3,b_2,b_1,b_0}所组成的,则可将 b_i 看作一个 7 次多项式 $b(x)$的系数,即

$$b(x)=b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x^1+b_0$$

式中,$b_i\in\{0,1\}$。

例如,{01010111}表示成多项式为 $x^6+x^4+x^2+x+1$。

也可以使用十六进制符号来表示字节值,将每 4 比特表示成一个符号便于记忆。例如,{01010111}={57}。

2) 字节数组

输入序列按字节划分,表示形式如下:

$$\{a_0a_1a_2a_3a_4a_5a_6a_7a_8a_9a_{10}a_{11}a_{12}a_{13}a_{14}a_{15}\}$$

假设将 128 位输入串{i_0,i_1,i_2,i_3,i_4,…,i_{127}}划分成字节,字节和字节内的比特按照如下方式排序:

$$a_0=\{i_0,i_1,i_2,i_3,i_4,i_5,i_6,i_7\}$$

$$a_1=\{i_8,i_9,i_{10},i_{11},i_{12},i_{13},i_{14},i_{15}\}$$

$$\cdots$$

$$a_{15}=\{i_{120},i_{121},i_{122},i_{123},i_{124},i_{125},i_{126},i_{127}\}$$

一般式表示:$a_n=\{i_{8n},i_{8n+1},i_{8n+2},i_{8n+3},i_{8n+4},i_{8n+5},i_{8n+6},i_{8n+7}\}$,其中 $0\leqslant n\leqslant 15$。

3) 状态

AES 算法的运算都是在一个二维字节数组上完成的,这个数组称为状态(State)。当输入的明文序列转换成字节数组后,进一步将一维的字节数组内容转换为二维排列,就形成了状态矩阵。一个状态矩阵由 4 行组成,每一行包括 N_b 个字节,N_b 的值等于分组长度除以 32。状态矩阵用 $\boldsymbol{s}$ 表示,每一个字节的位置由行号 r(范围是 $0\leqslant r<4$)和列号 c(范围是 $0\leqslant c<N_b$)唯一确定,记为 $s_{r,c}$ 或 $s[r,c]$。在 AES 标准中状态矩阵参数 $N_b=$

4,即 $0 \leqslant r < 4$。

算法在加密和解密的初始阶段将输入字节数组{$in_0\ in_1\ in_2\ in_3\ in_4 \cdots in_{15}$}复制到如图 3-39 所示的状态矩阵中。加密或解密的运算都在该状态矩阵上进行,最后的计算结果输出并复制到输出字节数组{$out_0\ out_1\ out_2\ out_3\ out_4 \cdots out_{15}$}中。

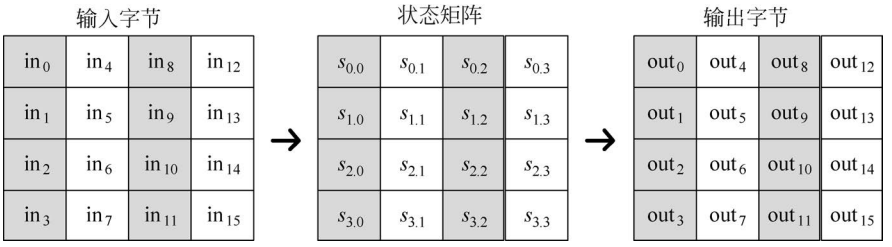

图 3-39　AES 加解密状态矩阵运算

在加密和解密的初始阶段,输入数组 in 转化成状态矩阵的公式如下:

$$s[r,c] = \text{in}[r + 4c]$$

式中,$0 \leqslant r < 4$,$0 \leqslant c < N_b$。

在加密和解密的完成阶段,状态矩阵将按照下述规则转换,结果存储在输出数组 out 中:

$$\text{out}[r + 4c] = s[r,c]$$

式中,$0 \leqslant r < 4$,$0 \leqslant c < N_b$。

4) 状态矩阵列数组

状态矩阵中每一列的 4 个字节是一个 32 位长的字,行号 r 是这个字中每个字节的索引,因此状态矩阵可以看作 32 位(列)长的一维数组,列号 c 是该数组的列索引。由此上例中的状态可以看作 4 个字组成的数组,表示如下:

$$\omega_0 = [s_{0,0}, s_{1,0}, s_{2,0}, s_{3,0}]$$
$$\omega_1 = [s_{0,1}, s_{1,1}, s_{2,1}, s_{3,1}]$$
$$\omega_2 = [s_{0,2}, s_{1,2}, s_{2,2}, s_{3,2}]$$
$$\omega_3 = [s_{0,3}, s_{1,3}, s_{2,3}, s_{3,3}]$$

2. 有限域基本数学运算

AES 算法的基本思想是基于置换和代替变换的演算方法。其中,置换是对数据的重新排列,而代替则是用数据替换另一个。AES 算法中的所有字节按照每 4 位表示成有限域 GF(2^8)中的一个元素。这个有限域元素可以进行加减法和乘法运算,但是这些运算不同于代数中使用的运算。下面介绍有限域相关算法的基本数学概念。

1) 加减法

在有限域中,多项式的加法运算定义为两个元素对应多项式相同位置指数项相应系数的“加法”。简单地说,有限域 GF(2^8)的加法是按位进行异或 XOR 运算(记为$\oplus$),即模 2 加。多项式减法与多项式加法的规则相同。例如:

$$\{57\} + \{83\} = (01010111)_2 \oplus (10000011)_2 = (11010100)_2 = \{D4\}$$

以多项式表示加法的计算过程如下:

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$$

2) 乘法

有限域 GF(2^8)的乘法运算也可以用多项式表示。乘法运算很容易造成溢出问题,解决的方法是多项式相乘后再模一个不可分解的多项式。因此 AES 算法在有限域 GF(2^8)上的乘法(记为·)定义为多项式的乘积再模一个幂次数为 8 的不可约多项式 $m(x)$,模 $m(x)$确保了所得结果是次数小于 8 的二元多项式,因此可以用一字节表示。$m(x)$的内容为

$$m(x)=x^8+x^4+x^3+x+1$$

或用十六进制表示该多项式为{01}{1B}。

例如,{57}·{83}$=(x^6+x^4+x^2+x+1)\cdot(x^7+x+1)$

$$=(x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+x+1)\bmod(x^8+x^4+x^3+x+1)$$

$$=x^7+x^6+1=(11000001)_2=\{C1\}$$

3) x 乘法

用多项式 x 乘以 $b(x)$,即 $x\cdot b(x)$,其结果可以表示为

$$b_7x^8+b_6x^7+b_5x^6+b_4x^5+b_3x^4+b_2x^3+b_1x^2+b_0x$$

将上述结果模 $m(x)$即可求得结果。如果得到的结果序列中:

(1) $b_7=0$,则不会出现结果溢出问题,该结果即是模运算后的正确形式;

(2) $b_7=1$,则会出现结果溢出问题,该乘积结果需要减去 $m(x)$,即求此乘积结果与 $m(x)$的异或。

由此得出,x(即$\{00000010\}_2$ 或$\{02\}_{16}$)乘以 $b(x)$可先对 $b(x)$在字节内左移一位(最后一位补 0),若 $b_7=1$,则再与$\{1B\}_{16}$(其二进制为 00011011)做逐比特异或来实现,该操作记为 $b=x\text{time}(a)$。x 的幂乘运算可以通过重复应用 $x\text{time}()$实现。而任意常数乘法可以通过对中间结果相加实现。

例如,{57}·{13}={FE},因为

$$\{57\}\cdot\{02\}=x\text{time}(\{57\})=\{AE\}$$

$$\{57\}\cdot\{04\}=x\text{time}(\{AE\})=\{47\}$$

$$\{57\}\cdot\{08\}=x\text{time}(\{47\})=\{8E\}$$

$$\{57\}\cdot\{10\}=x\text{time}(\{8E\})=\{07\}$$

所以{57}·{13}={57}·({01}⊕{02}⊕{10})={57}⊕{AE}⊕{07}={FE}。

4) 系数在有限域 GF(2^8)的特殊多项式运算

给定字符向量转换为系数在有限域 GF(2^8)中的多项式 $a(x)=a_3x^3+a_2x^2+a_1x+a_0$,它可以用$[a_0,a_1,a_2,a_3]$形式表示。该多项式与有限域元素定义中使用的多项式操作不同,此处的系数本身就是有限域元素,即是字节(byte)而不是比特(bit),系数本身可以用另一个有限域多项式表示。特殊的 4 项多项式的乘法可使用不同的模多项式 $M(x)=x^4+1$。系数在有限域 GF(2^8)中的多项式运算主要有乘法和乘以 x 两种。

(1) 给定多项式 $b(x)=b_3x^3+b_2x^2+b_1x+b_0$,计算 $a(x)$与 $b(x)$相乘。令 $d(x)=a(x)\otimes b(x)=d_3x^3+d_2x^2+d_1x+d_0$,即

$$d_0=a_0\cdot b_0\oplus a_3\cdot b_1\oplus a_2\cdot b_2\oplus a_1\cdot b_3$$

$$d_1=a_1\cdot b_0\oplus a_0\cdot b_1\oplus a_3\cdot b_2\oplus a_2\cdot b_3$$

$$d_2 = a_2 \cdot b_0 \oplus a_1 \cdot b_1 \oplus a_0 \cdot b_2 \oplus a_3 \cdot b_3$$
$$d_3 = a_3 \cdot b_0 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \oplus a_0 \cdot b_3$$

其向量表示为

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

由于多项式 $M(x)=x^4+1$ 在 GF(2^8)下可能不是可约多项式,因此如果任意给定一个4项多项式,在模 $M(x)$下不一定存在一个对应的乘法反多项式。

在AES算法中,对多项式 $b(x)$,这种乘法运算只限于乘一个固定的有逆元的多项式 $a(x)=a_3x^3+a_2x^2+a_1x+a_0$,即存在 $a(x)a^{-1}(x)=1 \bmod (x^4+1)$关系。

(2) 计算 $b(x)$乘以 x。$c(x)=x \otimes b(x)$定义为 x 与 $b(x)$的模 $M(x)$乘法,即 $c(x)=x \otimes b(x)=b_2x^3+b_1x^2+b_0x+b_3$。其矩阵表示中,除 $a_1=01$ 外,其他所有 $a_i=00$,即

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 00 & 00 & 00 & 01 \\ 01 & 00 & 00 & 00 \\ 00 & 01 & 00 & 00 \\ 00 & 00 & 01 & 00 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

因此,x(或 x 的幂)模乘多项式相当于对字节构成的向量进行字节循环移位。

3.5.4 AES算法内部结构

为了理解数据在AES内部的传递方式,首先假设状态 A(即128位的数据路径)是由16个字节 $A_0, A_1, \cdots, A_{15}$ 按照 4×4(字节)的矩阵方式组成。

A_0	A_4	A_8	A_{12}
A_1	A_5	A_9	A_{13}
A_2	A_6	A_{10}	A_{14}
A_3	A_7	A_{11}	A_{15}

从下面的内容可知,AES操作的元素是当前状态矩阵的行或列。同样,密钥字节也是以矩阵方式排列,其行数为4,列数可以为4(128位的密钥)、6(192位的密钥)或8(256位的密钥)。

1. 字节代换

字节代换是一个非线性可逆变换,针对状态矩阵中的每个字节,利用替代表(S盒)进行运算,表示为SubBytes(State),也称仿射变换,其计算过程主要由两个变换复合而成。两个变换的内容如下:

(1) 选取有限域 GF(2^8)上的乘法逆运算,其中元素{00}映射到它自身。

(2) 应用如下算法完成一有限域 GF(2^8)上的仿射变换

$$b' = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus b_{(i+8) \bmod 8} \oplus c_i$$

算法中当 $0 \leqslant i < 8$ 时，b_i 是字节的第 i 个比特，c_i 是值为{63}(即{01100011})的字节 c 数组的第 i 个比特。算法描述中以 b' 表示该变量将用右侧的值更新。

首先，将字节数据看成 GF(2^8)上的元素，进行模 $M(x)$ 运算映射到自己的乘法逆，0 映射到自身；其次，作 GF(2^8)的仿射变换，该变换过程可逆。预先将 GF(2^8)上的每个元素通过查表作 SubBytes 变换，形成 S 盒。与 DES 算法中 S 盒不同的是，AES 算法中的 S 盒具有一定的代数结构，而 DES 算法中是人为指定构造的。

以矩阵的形式，S 盒的仿射变换可表示为

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

S 盒的仿射变换在状态矩阵上的变换功能如图 3-40 所示。

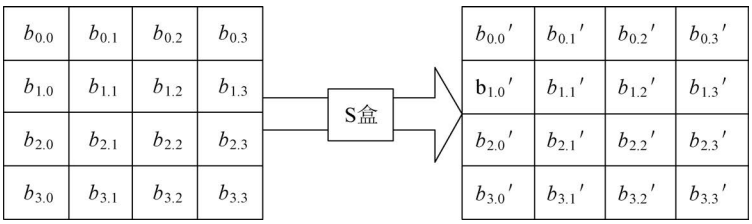

图 3-40　S 盒的仿射变换

实际运算时，此函数可以通过查表快速获得对应变换值，字节变换的变换值如表 3-12 所示。例如，$b_{1,1}$={53}，查字节变换值表，找到第 5 列第 3 行，对应数值为{ed}，表示经过字节替代变换后 $b'_{1,1}$={ed}。

表 3-12　S 盒中字节 x，y 的替代值(十六进制格式)

x	y															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**a**	**b**	**c**	**d**	**e**	**f**
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2

续表

x	y															
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

2. 行移位变换

行移位变换是在状态矩阵的行上进行的。状态阵列的后 3 行分别以 c_i 为移位大小循环移位。其中,第 0 行 $c_0=0$,即保持不变;第 1 行循环移位 c_1 字节;第 2 行循环移位 c_2 字节;第 3 行循环移位 c_3 字节。运算结果是将行中的字节移向较低位,最低位的字节循环移动至行的最高位。128 位状态矩阵的行移位变换操作如图 3-41 所示。

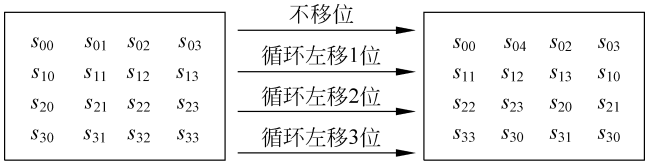

图 3-41　128 位状态矩阵的行移位变换操作

偏移量 c_i 与分组长度 N_b 有关,不同分组长度的行移位变换偏移量如表 3-13 所示。

表 3-13　不同分组长度的行移位变换偏移量

N_b	c_1	c_2	c_3
4	1	2	3
6	1	2	3
8	1	3	4

行移位变换实现了字节在每一行的扩散,很自然地想到字节在列中也需要扩散。

3. 列混合变换

列混合变换在状态矩阵上,按照每一列分别进行运算,并将每一列看作有限域 4 次多项式,即将状态的列看作 GF(2^8)上的多项式 $a(x)$与多项式 $c(x)$相乘,计算结果对固定多项式 $M(x)=x^4+1$ 取模。多项式 $c(x)$表示为

$$c(x)=\{03\}x^3+\{01\}x^2+\{01\}x+\{02\}$$

其中,系数是用十六进制表示的,并且 $c(x)$与 x^4+1 互素。

令 $b(x)=c(x)\otimes a(x) \bmod (x^4+1)$,由于 $x^1 \bmod (x^4+1)=x^{1 \bmod 4}$,故有

$$b_0=c_0\cdot a_0\oplus c_3\cdot a_1\oplus c_2\cdot a_2\oplus c_1\cdot a_3$$

$$b_1 = c_1 \cdot a_0 \oplus c_0 \cdot a_1 \oplus c_3 \cdot a_2 \oplus c_2 \cdot a_3$$
$$b_2 = c_2 \cdot a_0 \oplus c_1 \cdot a_1 \oplus c_0 \cdot a_2 \oplus c_3 \cdot a_3$$
$$b_3 = c_3 \cdot a_0 \oplus c_2 \cdot a_1 \oplus c_1 \cdot a_2 \oplus c_0 \cdot a_3$$

写成矩阵表示形式为

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 01 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

列混合变换的过程如图 3-42 所示。

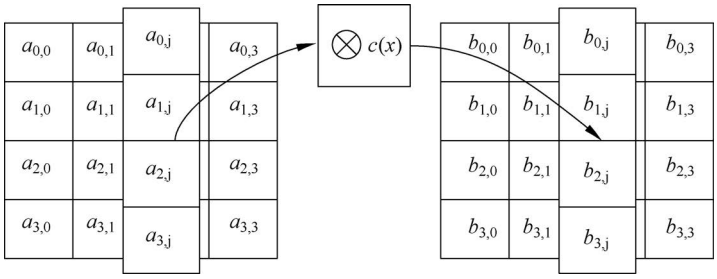

图 3-42　列混合变换的过程

例 3-7　在 AES 列混合变换中，模多项式 $M(x)=x^8+x^4+x^3+x+1$，请根据下面的状态矩阵，填写下面表格空格的值，并写出计算过程。

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{00} & s_{01} & s_{02} & s_{03} \\ s_{10} & s_{11} & s_{12} & s_{13} \\ s_{20} & s_{21} & s_{22} & s_{23} \\ s_{30} & s_{31} & s_{32} & s_{33} \end{bmatrix} = \begin{bmatrix} s'_{00} & s'_{01} & s'_{02} & s'_{03} \\ s'_{10} & s'_{11} & s'_{12} & s'_{13} \\ s'_{20} & s'_{21} & s'_{22} & s'_{23} \\ s'_{30} & s'_{31} & s'_{32} & s'_{33} \end{bmatrix}$$

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

→

	40	A3	4C
37		70	9F
94	E4		42
ED	A5	A6	

解：

$s'_{00}=02 \cdot 87 \oplus 03 \cdot 6E \oplus 01 \cdot 46 \oplus 01 \cdot A6$
$=00001110 \oplus 00011011 \oplus 11011100 \oplus 01101110 \oplus 01000110 \oplus 10100110$
$=0100011=47$

$s'_{11}=01 \cdot F2 \oplus 02 \cdot 4C \oplus 03 \cdot E7 \oplus 01 \cdot 8C$
$=11110010 \oplus 10011000 \oplus 11001110 \oplus 00011011 \oplus 11100111 \oplus 10001100$
$=11010100=D4$

$s'_{22}=01 \cdot 4D \oplus 01 \cdot 90 \oplus 02 \cdot 4A \oplus 03 \cdot D8$
$=01001101 \oplus 10010000 \oplus 10010100 \oplus 101100000 \oplus 00011011 \oplus 11011000$

$=00111010=3A$

$s'_{33}=03\cdot 97\oplus 01\cdot EC\oplus 01\cdot C3\oplus 02\cdot 95$

$=00101110\oplus 00011011\oplus 10010111\oplus 11101100\oplus 11000011\oplus$

$00101010\oplus 00011011$

$=10111100=BC$

故空格处分别填47,D4,3A,BC。

4. 轮密钥加变换

在轮密钥加变换中,用轮密钥与状态矩阵按比特进行异或(XOR)操作。轮密钥是通过主密钥生成的子密钥,为了便于计算,轮密钥的长度等于分组长度,每一个轮密钥由 N_b 个字节组成。轮密钥加变换的过程如图3-43所示。

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$		$b_{0,0}$	$b_{0,1}$	$b_{0,2}$	$b_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$		$b_{1,0}$	$b_{1,1}$	$b_{1,2}$	$b_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$	=	$b_{2,0}$	$b_{2,1}$	$b_{2,2}$	$b_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$		$b_{3,0}$	$b_{3,1}$	$b_{3,2}$	$b_{3,3}$

图3-43 轮密钥加变换的过程

例3-8 设AES算法分组长度为128,输入的明文 $M=$32 6C A8 F6 42 31 8C D6 43 72 64 E0 98 89 07 C3,密钥 $k=$A3 61 89 B5 54 12 D8 90 F4 14 FC AB 81 70 AE 3F。求AES的第一轮输出。

解:考虑到扩展密钥的前 N_k 个字是由密码密钥填充的,即初始轮密钥为 k。明文与初始轮密钥加得:91 0D 21 43 16 23 54 46 B7 66 98 4B 19 F9 A9 FC

经过字节代换为:D0 B1 F4 8A 21 EB 4F AA 0E F6 3B F7 BD 84 C5 DF

行位移变换得:D0 B1 F4 8A EB 4F AA 21 3B F7 0E F6 DF BD 84 C5

列混合变换得:79 E2 9C 43 8F D0 2D 0C 17 D7 D5 08 1E 18 B0 C1

轮密钥加变换得:A3 54 F4 81 61 12 14 70 89 D8 FC AE B5 90 AB 3F

即得第一轮输出:DA B6 68 C2 EE C2 39 7C 9E 0F 29 A6 AB 88 1B FE

5. 密钥扩展算法

下面以128位的密钥长度介绍密钥扩展算法,其他192位和256位的密钥长度所对应的密钥编排存在一定的相似性。AES的密钥扩展算法是面向字的,其中1个字=32位。对长度为128位的密钥而言,它对应的轮数 $N_r=10$,并得到11个子密钥,且每个密钥的长度均为128位。AES轮密钥(子密钥)k_i 的计算是递归的,即为了得到子密钥 k_i,子密钥 k_{i-1} 必须是已知的,以此类推。

11个子密钥存储在元素为 $W[0],\cdots,W[43]$ 的密钥扩展数组 $W[i]$ 中。子密钥的计算方式如图3-44所示。元素 $k_0,\cdots,k_{15}$ 表示原始AES密钥对应的字节。

首先,需要注意的是,第一个子密钥 k_0 为原始AES密钥,即原始密钥直接被复制到扩展密钥数组 $W[i]$ 的前4个元素中。从图中可知,子密钥 $W[4i]$($i=1,\cdots,10$)最左边字

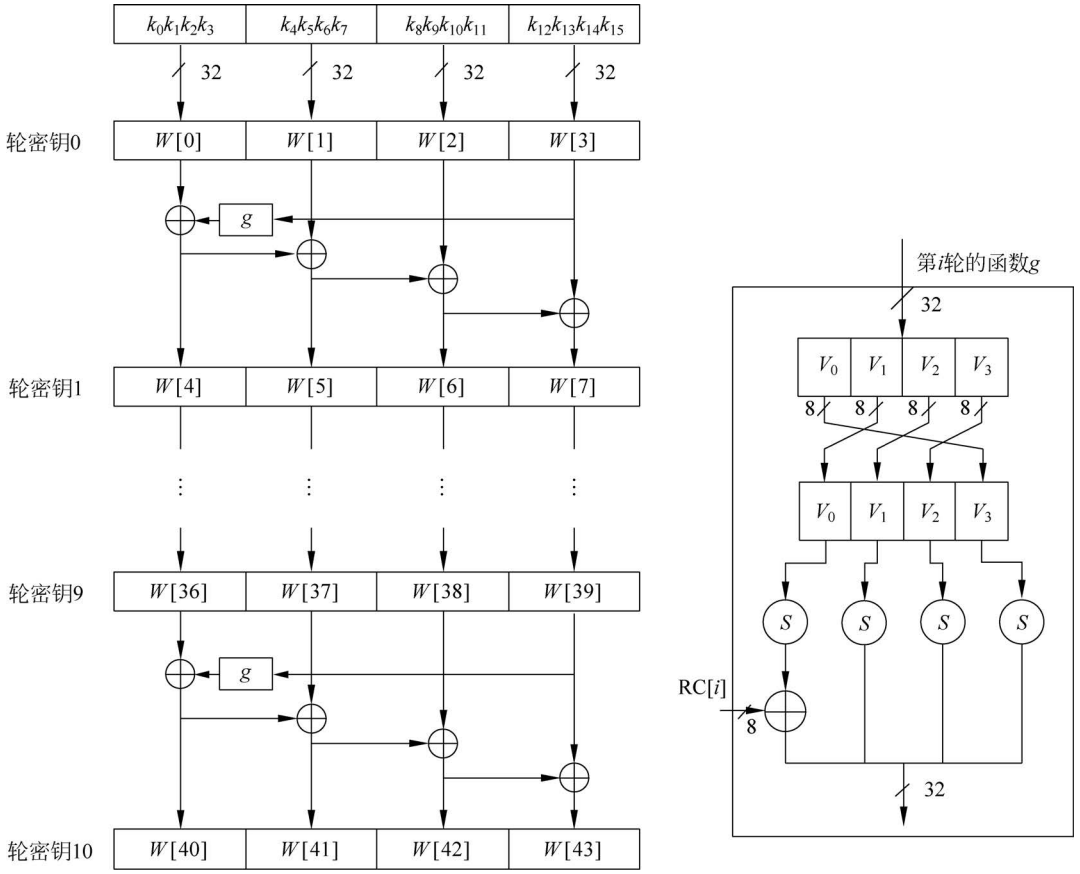

图 3-44　128 位密钥大小的 AES 密钥编排

的计算方式为：

$$W[4i]=W[4(i-1)]+g(W[4i-1])$$

这里的 g 表示的是一个输入和输出均为 4 字节的非线性函数。子密钥其余的三个字是通过递归计算得到的：

$$W[4i+j]=W[4i+j-1]+W[4(i-1)+j]$$

其中，$i=1,\cdots,10$；$j=1,2,3$。函数 g 首先将 4 个输入字节翻转，并执行一个按字节的 S 盒代换，最后与轮系数 RC 相加。轮系数是 $GF(2^8)$ 域中的一个元素，即为一个 8 位的值。轮系数只与函数 g 最左边的字节相加。而且轮系数每轮都会改变，其变换规则为：

$$RC[1]=x^0=(00000001)_2$$

$$RC[2]=x^1=(00000010)_2$$

$$RC[3]=x^2=(00000100)_2$$

$$\vdots$$

$$RC[10]=x^9=(00110110)_2$$

函数 g 的目的有两个：第一，增加密钥编排中的非线性；第二，消除 AES 中的对称

性。这两种属性都是抵抗某些分组密码攻击所必要的。

6. AES 解密

AES 的解密过程是加密过程的逆运算。解密算法中各个变换的操作顺序与加密算法不同,但是加密和解密算法中的密钥生成形式是一致的。AES 算法的若干性质保证了可以构造一个等价的解密算法,解密时各个变换的操作顺序与加密时相反(由逆变换取代原来的变换)。解密算法中使用的变换依次为逆列混合变换、逆行位移变换、逆字节变换和轮密钥加变换,变换作用在密文序列对应的状态矩阵上。具体的解密过程如图 3-45 所示。

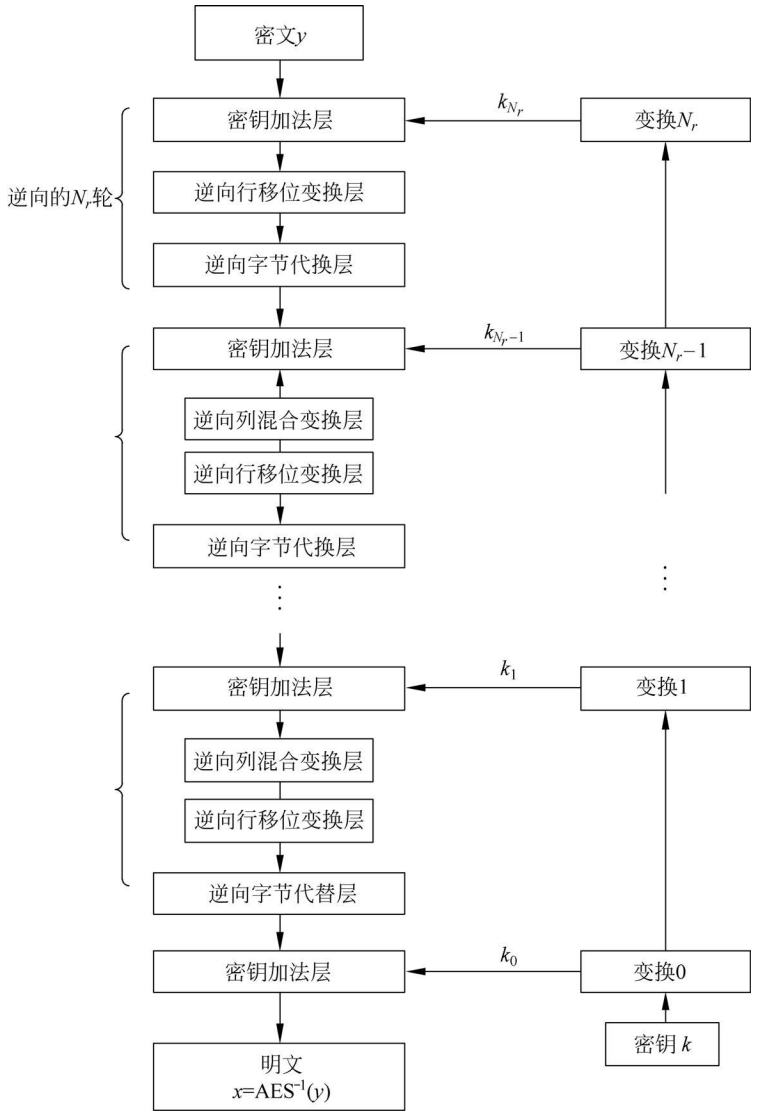

图 3-45　AES 解密过程

3.5.5　AES 算法的安全性

对于 Rijndael 算法的安全性讨论，通过对以下已知的攻击方法来进行分析。

(1) 穷举攻击法：Rijndael 算法中最短的密钥长度是 128 比特，如果用穷举法则需要 2^{128} 次，计算量是非常大的，故使用目前技术的穷举法是无效的。

(2) 差分攻击法：这种攻击法是利用大量已知的明文/密文对之间的差异来推测出密钥。对于 Rijndael 算法，当轮数超过三轮，若存在 $1/2^{n-1}$（n 位分组块的）长度可预测性的差异，这种攻击法就可以推测出密钥的位元值，在 Rijndael 算法中已经证明 Rijndael 经过四轮运算后，存在不超过 2^{-150} 的可预测性差异，5 轮运算后不超过 2^{-300} 的可预测差异，因此可以说这种攻击法对 Rijndael 算法是无效的。

(3) 线性攻击法：这种攻击法利用大量收集到的明文/密文对，根据其中可预测的相对关系推导出一个代数展开式，只要能找出相对关系数超过 $2^{n/2}$ 的线性轨迹，就可以找出密钥值。Rijndael 算法已经被证明执行 4 轮运算后不存在相关系数大于 2^{-75} 的线性轨迹；在执行 8 轮后，其相关系数大于 2^{-150} 的相关系数也不存在，因此可以说这种攻击法对 Rijndael 算法是无效的。

(4) 平方攻击法：这种攻击是一种明文攻击，攻击强度不依赖于 S 盒列混合矩阵和密钥扩散准则的选取，但迄今为止这种攻击法只能够对经过不超过 6 次轮运算的 Rijndael 算法有效，计算量为 $2^{73}+2^{64}$，因此可以说这种攻击法对 Rijndael 算法是无效的。

(5) 内插攻击法：这种攻击法是攻击者利用加密的输入与输出配对，建立一些多项式。如果加密的元件有一个乘法的代数展开式，并且与管理的复杂度结合在一起时，这种攻击便是可行的。攻击的方式是如果攻击者建立的代数展开式的阶度很小，只需要一些加密法的输入和输出配对就可以得到代数展开式的各项系数。但是，Rijndael 算法中的 GF(2^8)域是非常复杂的，因此可以说这种攻击法对 Rijndael 算法是无效的。

从以上分析可以看出，Rijndael 算法对已知的攻击具有较好的免疫性，安全性比较高。

3.6　IDEA

3.6.1　IDEA 算法简介

1990 年，瑞士联邦技术学院的 Xuejia Lai 和 James Massey 提出一个取代 DES 的对称密码算法，称为 PES(Proposed Encryption Standard，建议的加密标准)。为提高对差分密码攻击的抵抗能力，设计者对 PES 经过两次修改，于 1992 年将修改后的 PES 改名为 IDEA(International Data Encryption Algorithm，国际数据加密算法)。类似于 DES，IDEA 算法也是一种数据块加密算法，它设计了一系列加密轮次，每轮加密都使用从完整的加密密钥中生成的一个子密钥。与 DES 的不同处在于：它采用软件实现和采用硬件实现同样快速。由于 IDEA 是在美国之外提出并发展起来的，避开了美国法律上对加密

技术的诸多限制,因此,有关IDEA算法和实现技术的文献都可以自由出版和交流,极大地促进了IDEA的发展和完善。

IDEA的分组长度是64位,密钥长度是128位,是已公开的可用算法中速度快且安全性强的分组加密算法,不但具有极强的抗攻击能力,而且具有良好的应用前景。目前,PGP使用IDEA作为其分组加密算法;安全套接字层SSL也将IDEA包含在其加密算法库SSLRef中;IDEA算法专利的所有者Ascom公司也推出了一系列基于IDEA算法的安全产品,包括基于IDEA的Exchange安全插件、IDEA加密芯片、IDEA加密软件包等。IDEA算法的应用和研究正在不断走向成熟。

3.6.2 IDEA算法设计思想

分组密码的强度,主要是通过混淆和扩散来实现的,IDEA算法所依据的设计思想是"混合使用来自不同代数群中的运算"。

1. 混淆

算法的"混淆"性由连续使用作用于一对16比特子块的三种"不相容"的群运算获得,这三种16位的基本运算如下。

(1) 异或运算⊕,即按位做不进位加法运算,规则为

$$1 \oplus 0 = 0 \oplus 1 = 1, \quad 0 \oplus 0 = 1 \oplus 1 = 0$$

(2) 模2^{16}加运算⊞,即16位无符号整数做加法运算,规则为:

$$X \boxplus Y \equiv (X + Y) \bmod (2^{16})$$

(3) 模$(2^{16}+1)$乘运算⊙,即16位整数做乘法运算,规则为:

$$X \odot Y \equiv (X \times Y) \bmod (2^{16}+1)$$

注意:模$(2^{16}+1)$整数乘法$a \odot b$的实现公式为

$$(ab) \bmod (2^n+1) = \begin{cases} ((ab) \bmod 2^n) - ((ab) \operatorname{div} 2^n) \cdots ((ab) \bmod 2^n) \geqslant ((ab) \operatorname{div} 2^n) \\ ((ab) \bmod 2^n) - ((ab) \operatorname{div} 2^n) + 2^n + 1 \cdots ((ab) \bmod 2^n) \leqslant ((ab) \operatorname{div} 2^n) \end{cases}$$

例如,(04A5)⊕(B605)≡(B2A0)

(74A5)⊞(B60B)≡(2AB0)

(0000)⊙(8000)≡$(2^{16} \times 2^{15}) \bmod (2^{16}+1) \equiv (2^{15}+1) \equiv$(8001)

在三种不同的群运算中,要特别注意$2^{16}+1$整数乘法运算⊙,这里除了将16位的全零子块处理为2^{16}外,其余16位的子块均按通常处理成一个整数的二进制形式。

三种运算结合起来使用可对算法的输入提供复杂的变换,从而使得对IDEA的密码分析比对仅使用异或运算的DES更为困难。

2. 扩散

IDEA加密算法中的"扩散"性是由称为乘加(Multiplication/Addition,MA)结构的基本单元实现的,如图3-46所示。该结构的输入是两个16位的子段和两个16位的子密钥,输出也为两个16位的子段。这一结构在算法中重复使用8次,除获得了非常有效的扩散效果之外,还保证了加解密过程的相似性。

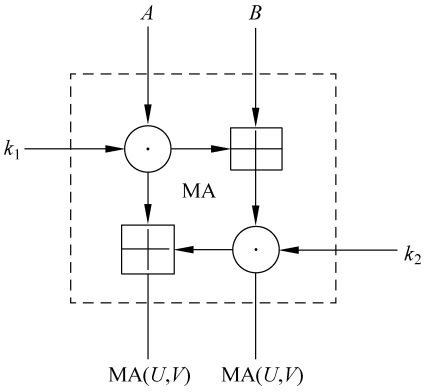

图 3-46　MA 运算结构

MA 结构定义：设 A、B、k_1、k_2 是 4 个 16 位输入码，‖ 表示位串的拼接，则 32 位输出码 MA(A,B,k_1,k_2)可表示为：

$$MA(A,B,k_1,k_2)=((A\odot k_1)\boxplus(((A\odot k_1)\boxplus B)\odot k_2))\,\|\,(((A\odot k_1)\boxplus B)\odot k_2)$$

以上表达式中的括号不能省略，因为三种运算中的任何两个都相互不可分配，也不可结合。

这里 MA 运算结构作为 IDEA 的主要模块，相当于分组加密算法中的 S 盒作用。

3.6.3　IDEA 算法内部结构

整个算法包括数据加密过程、子密钥产生、数据解密过程三部分。

1. IDEA 迭代过程

IDEA 加密算法的总体结构如图 3-47 所示。这里加密函数有两类输入：待加密明文和密钥。其中输入的 64 位明文数据块 x 被分成 4 个 16 位子块 x_1,x_2,x_3,x_4，即 $x=x_1x_2x_3x_4$。这 4 个分组作为算法的第 1 轮输入，整个加密过程总共进行 8 轮循环，每轮循环由 64 位码 W_{i1},W_{i2},W_{i3},W_{i4} 和 6 个 16 位子密钥 $Z_{(i-1)*6+1}$,$Z_{(i-1)*6+2}$,$Z_{(i-1)*6+3}$,$Z_{(i-1)*6+4}$,$Z_{(i-1)*6+5}$,$Z_{(i-1)*6+6}$ 作为输入，产生 64 位输出 $W_{(i+1)1}$,$W_{(i+1)2}$,$W_{(i+1)3}$,$W_{(i+1)4}$，这里 $i=1,2,\cdots,8$。最后再经过一次输出变换得到 4 个 16 位子分组密文 y_1,y_2,y_3,y_4，将 4 个子分组重新连接起来就可生成密文 y，即 $y=y_1y_2y_3y_4$。输入的密钥 Z 长度为 128 位，通过子密钥生成器产生 52 个 16 位子密钥。

IDEA 加密算法详细迭代过程如图 3-48 所示，其中每轮迭代运算分为以下两部分。

第一部分是变换运算，其方法是采用加法及乘法运算将 4 个 16 位的子明文分组与 4 个子密钥混合，产生 4 个 16 位的输出。

第二部分是用于产生扩散性的乘加(MA)运算。MA 运算生成 2 个 16 位的输出。MA 的输出再与变换运算的输出采用 XOR 运算产生 4 个 16 位的最后输出，这 4 个 16 位的最后输出即成为下一轮运算的原始输入。在这 4 个最后结果中的第 2、3 个输出是经位置互换而得到的，这样处理的目的是对抗差分分析攻击。

下面是 IDEA 算法具体的变换过程。

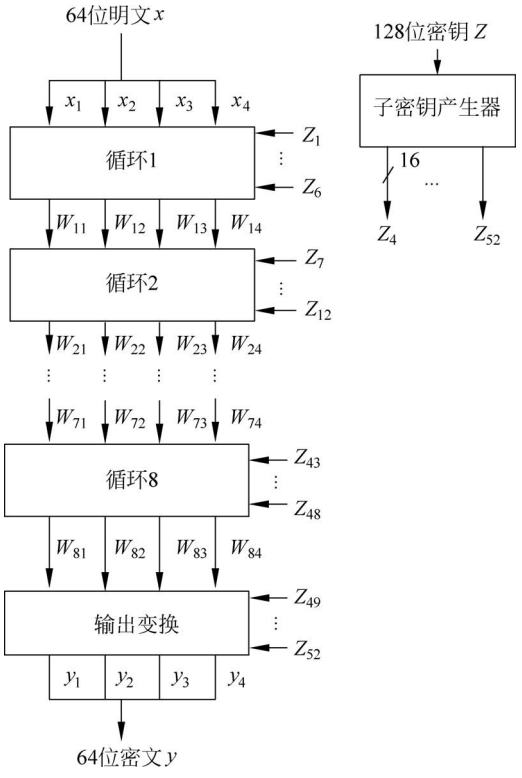

图 3-47　IDEA 加密算法的总体结构

① x_1 和第 1 个子密钥 $Z_1^{(1)}$ 做乘法运算。

② x_2 和第 2 个子密钥 $Z_2^{(1)}$ 做加法运算。

③ x_3 和第 3 个子密钥 $Z_3^{(1)}$ 做加法运算。

④ x_4 和第 4 个子密钥 $Z_4^{(1)}$ 做乘法运算。

⑤ 将①和③的结果相异或。

⑥ 将②和④的结果相异或。

⑦ 将⑤的结果与 $Z_5^{(1)}$ 相乘。

⑧ 将⑥和⑦的结果相加。

⑨ 将⑧的结果与 $Z_6^{(1)}$ 相乘。

⑩ 将⑦和⑨的结果相加。

⑪ 将①和⑨的结果相异或。

⑫ 将③和⑨的结果相异或。

⑬ 将②和⑩的结果相异或。

⑭ 将④和⑩的结果相异或。

第一轮的输出是 4 个子块,即⑪、⑫、⑬和⑭的结果。将中间两个块交换(最后一轮除外)后,就是下一轮的输入。

在经过 8 轮运算之后,IDEA 使用一个输出变换对 8 轮迭代的结果进行运算,输出变换在进行运算前将第 2 个输入与第 3 个输入进行互换,这实际上是将第 8 轮迭代运算最

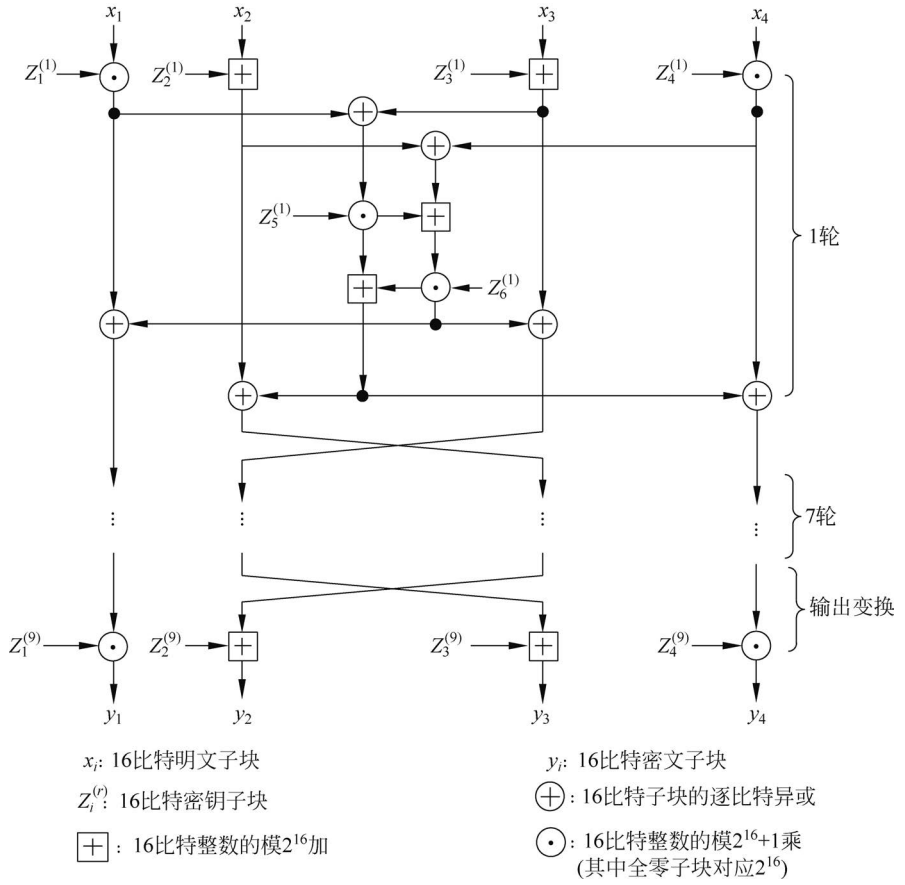

图 3-48　IDEA 加密算法详细迭代过程

后所做的互换抵消了，这种特殊安排的目的在于可以使用与加密算法相同结构的解密算法进行解密，从而简化了设计及使用 IDEA 算法的复杂性。输出变换的过程如下：

① x_1 和 $Z_1^{(9)}$ 相乘。

② x_2 和 $Z_2^{(9)}$ 相加。

③ x_3 和 $Z_3^{(9)}$ 相加。

④ x_4 和 $Z_4^{(9)}$ 相乘。

将这 4 步运算的结果连到一起就是最后产生的正式密文。

2. IDEA 密钥生成过程

在加密之前，IDEA 需要通过密钥扩展(Key Expansion)将 128 位的密钥扩展为 52 个子密钥。加密算法进行迭代操作时，每轮需要 6 个子密钥，另外还需要 4 个额外子密钥用于输出变换。

整个密钥的扩展的过程如表 3-14 所示，为了能够看清楚 8 轮迭代的关系，在表中用粗线条将每轮进行了区别。将 128 位密钥按位编号为(0,1,2,…,127)，表 3-14 中单元格的内容表示 128 位密钥子段的范围。例如，16～31 表示(16,17,18,…,31)，121～8 表示

(121,122,…,127,1,2,…,8)。

表 3-14 IDEA 的密钥扩展过程

r	k_1	k_2	k_3	k_4	k_5	k_6
1	0～15	16～31	32～47	48～63	64～79	80～95
2	96～111	112～127	25～40	41～56	57～72	73～88
3	89～104	105～120	121～8	9～24	50～65	66～81
4	82～97	98～113	114～1	2～17	18～33	34～49
5	75～90	91～106	107～122	123～10	11～26	27～42
6	43～58	59～74	100～115	116～3	4～19	20～35
7	36～51	52～67	68～83	84～99	125～12	13～28
8	29～44	45～60	61～76	77～92	93～108	109～124
9	22～37	38～53	54～69	70～85	—	—

下面详细说明密钥扩展过程。首先,将 128 位密钥 k 表述为 $k=b_0b_1\cdots b_{127}$。

第 1 轮:将此 128 位密钥分成 8 段,依次得 8 个子密钥:

$Z_1^{(1)}=b_0b_1\cdots b_{15}$,$Z_2^{(1)}=b_{16}b_{17}\cdots b_{31}$,…,$Z_6^{(1)}=b_{80}b_{81}\cdots b_{95}$,$Z_1^{(2)}=b_{96}b_{97}\cdots b_{111}$,$Z_2^{(2)}=b_{112}b_{113}\cdots b_{127}$,即将 8 段中的前 6 段密钥用于第 1 轮加密,后 2 段用于第 2 轮。

第 2 轮:将 k 循环左移 25 位,得 $b_{25}b_{26}\cdots b_{127}b_0b_1\cdots b_{24}$。

将此左移后的 128 位密钥再分成 8 段,前 4 段用于第 2 轮的子密钥:$Z_3^{(2)}$,$Z_4^{(2)}$,$Z_5^{(2)}$,$Z_6^{(2)}$;后 4 段用于构成第 3 轮子密钥的前半部分:$Z_1^{(3)}$,$Z_2^{(3)}$,$Z_3^{(3)}$,$Z_4^{(3)}$。

第 3 轮:再次把第 2 轮的 k 循环左移 25 位,同样分成 8 段,前 2 段用于构造第 3 轮子密钥;后 6 段用于下一轮。

以此继续操作,当循环左移了 5 次之后,已经生成了 48 个子密钥,还有 4 个额外的子密钥需要生成,再次把 k 循环左移 25 位,选取划分出来的 8 个 16 位子密钥的前 4 个作为那 4 个额外加密密钥。至此,供加密使用的 52 个子密钥生成完毕。

3. IDEA 的解密过程

IDEA 的解密过程本质上与加密过程相同,所不同的是参与迭代运算的解密子密钥的生成方式。解密密钥和加密密钥有一个对应关系,其子密钥 $k_i^{(r)}$ 是从加密密钥 $Z_i^{(r)}$ 导出的。导出关系如下:

$$(k_1^{(r)},k_2^{(r)},k_3^{(r)},k_4^{(r)})=(z_1^{(10-r)^{-1}},-z_3^{(10-r)},-z_2^{(10-r)},z_4^{(10-r)^{-1}}),\quad r=2,\cdots,8$$

$$(k_1^{(r)},k_2^{(r)},k_3^{(r)},k_4^{(r)})=(z_1^{(10-r)^{-1}},-z_2^{(10-r)},-z_3^{(10-r)},z_4^{(10-r)^{-1}}),\quad r=1,9$$

$$(k_5^{(r)},k_6^{(r)})=(z_5^{(9-r)},z_6^{(9-r)}),\qquad r=1,\cdots,8$$

这里 Z^{-1} 表示 $Z \bmod (2^{16}+1)$乘法的逆,即 $Z\odot Z^{-1}\equiv 1 \bmod (2^{16}+1)$;$-Z$ 表示 $Z \bmod 2^{16}$ 加法的逆,即 $Z\odot -Z\equiv 0 \bmod 2^{16}$。

以上解密密钥导出的公式描述如下。

(1) 第 $r(r=1,2,\cdots,9)$轮解密的前 4 个子密钥是由加密过程第$(10-r)$轮的前 4 个子密钥导出的,其中变换阶段被记为循环 9。解密密钥的第 1 个和第 4 个子密钥取为相

应的第 1 个和第 4 个加密子密钥模 $2^{16}+1$ 乘法逆元。第 2 个和第 3 个解密子密钥取法为：当轮数 $r=2,\cdots,8$ 时，取为相应的第 3 个和第 2 个加密子密钥的模 2^{16} 加法逆元；当 $r=1$ 和 9 时，取为相应的第 2 个和第 3 个加密子密钥的模 2^{16} 加法逆元。

(2) 第 $r(r=1,2,\cdots,8)$轮解密的后两个子密钥等于加密过程的第$(9-r)$轮的后两个子密钥。

3.6.4　IDEA 算法的安全性

IDEA 算法的密钥长度为 128 位。自 1991 年提出至今，设计者尽最大努力使该算法不受差分密码分析的影响，数学家已证明 IDEA 算法在其 8 轮迭代的第 4 轮之后便不受差分密码分析的影响。假定穷举法攻击有效，那么即使设计一种每秒可以试验 10 亿个密钥的专用芯片，并将 10 亿片这样的芯片用于此项工作，仍需 10^{13} 年才能解决问题；若用 10^{24} 片这样的芯片，有可能在一天内找到密钥，不过人们还无法找到足够的硅原子来制造这样一台机器。目前，尚无公开发表的试图对 IDEA 进行密码分析的文章。因此，目前看 IDEA 是非常安全的，并且 IDEA 数据比较 RSA 算法加、解决速度快得多，又比 DES 算法要相对安全得多。

3.7　中国商用密码算法 SM4

3.7.1　SM4 算法简介

2012 年 3 月，中国国家密码管理局(National Cryptographic Administration)正式公布了包含 SM4 分组密码算法在内的 6 项密码行业标准，SM4 的正式名称是“基于 SM1 算法的分组密码算法”(Block Cipher Algorithm Based on SM1)。SM1 算法是中国自主研发的分组密码算法，SM4 算法在其基础上进行了改进和优化。

3.7.2　SM4 算法设计思想

与 DES 和 AES 算法类似，SM4 算法是一种分组密码算法。其分组长度为 128bit，密钥长度也为 128bit。加密算法与密钥扩展算法均采用 32 轮非线性迭代结构，以字(32 位)为单位进行加密运算，每次迭代运算均为一轮变换函数 F。SM4 算法加/解密算法的结构相同，只是使用轮密钥相反，其中解密轮密钥是加密轮密钥的逆序。

SM4 密码算法的基本运算有模 2 加和循环移位。

(1) 模 2 加：记为$\oplus$，为 32 位逐比特异或运算。

(2) 循环移位：$<<<i$，把 32 位字循环左移 i 位。

3.7.3　SM4 算法内部结构

1. SM4 算法基本构件

1) S 盒

S 盒是以字节为单位的非线性替换，其密码学作用是混淆，它的输入和输出都是 8 位

的字节。设输入字节为 a,输出字节为 b,则 S 盒的运算可表示为

$$b = S(a)$$

S 盒的替换规则如表 3-15 所示。

表 3-15　SM4 密码算法的 S 盒

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	D6	90	E9	FE	CC	E1	3D	B7	16	B6	14	C2	28	FB	2C	05
1	2B	67	9A	76	2A	BE	04	C3	AA	44	13	26	49	86	06	99
2	9C	42	50	F4	91	EF	98	7A	33	54	0B	43	ED	CF	AC	62
3	E4	B3	1C	A9	C9	08	E8	95	80	DF	94	FA	75	8F	3F	A6
4	47	07	A7	FC	F3	73	17	BA	83	59	3C	19	E6	85	4F	A8
5	68	6B	81	B2	71	64	DA	8B	F8	EB	0F	4B	70	56	9D	35
6	1E	24	0E	5E	63	58	D1	A2	25	22	7C	3B	01	21	78	87
7	D4	00	46	57	9F	D3	27	52	4C	36	02	E7	A0	C4	C8	9E
8	EA	BF	8A	D2	40	C7	38	B5	A3	F7	F2	CE	F9	61	15	Al
9	E0	AE	5D	A4	9B	34	1A	55	AD	93	32	30	F5	8C	Bl	E3
A	1D	F6	E2	2E	82	66	CA	60	C0	29	23	AB	0D	53	4E	6F
B	D5	DB	37	45	DE	FD	8E	2F	03	FF	6A	72	6D	6C	5B	51
C	8D	1B	AF	92	BB	DD	BC	7F	11	D9	5C	41	1F	10	5A	D8
D	0A	C1	31	88	A5	CD	7B	BD	2D	74	D0	12	B8	E5	B4	B0
E	89	69	97	4A	0C	96	77	7E	65	B9	F1	09	C5	6E	C6	84
F	18	F0	7D	EC	3A	DC	4D	20	79	EE	5F	3E	D7	CB	39	48

2) 非线性变换 τ

非线性变换 τ 是以字为单位的非线性替换,它由 4 个 S 盒并置构成。设输入为 $A=(a_0,a_1,a_2,a_3)$(4 个 32 位的字),输出为 $B=(b_0,b_1,b_2,b_3)$(4 个 32 位的字),则

$$B=(b_0,b_1,b_2,b_3)=\tau(A)=(S(a_0),S(a_1),S(a_2),S(a_3)) \tag{3-1}$$

3) 线性变换 L

线性变换 L 以字为处理单位,其输入输出都是 32 位的字,它的密码学作用是扩散。

设 L 的输入为字 B,输出为字 C,则

$$C=L(B)=B\oplus(B<<<2)\oplus(B<<<10)\oplus(B<<<18)\oplus(B<<<24) \tag{3-2}$$

4) 合成变换 T

合成变换 T 由非线性变换 τ 和线性变换 L 复合而成,数据处理的单位是字。设输入为字 X,则先对 X 进行非线性 τ 变换,再进行线性 L 变换,记为

$$T(X)=L(\tau(X)) \tag{3-3}$$

由于合成变换 T 是非线性变换 τ 和线性变换 L 的复合,所以它综合起到混淆和扩散的作用,从而可提高密码的安全性。

2. 轮函数

轮函数由上述基本构件组成。设轮函数 F 的输入为 4 个 32 位字(X_0,X_1,X_2,X_3),共 128 位,轮密钥为一个 32 位的字 rk,输出也是一个 32 位的字,由下式给出:

$$F(X_0,X_1,X_2,X_3,\mathrm{rk})=X_0\oplus T(X_1\oplus X_2\oplus X_3\oplus \mathrm{rk})$$

根据式(3-3),有

$$F(X_0,X_1,X_2,X_3,\mathrm{rk})=X_0\oplus L(\tau(X_1\oplus X_2\oplus X_3\oplus \mathrm{rk}))$$

记 $B=(X_1\oplus X_2\oplus X_3\oplus \mathrm{rk})$,根据式(3-1)和式(3-2),有

$$F(X_0,X_1,X_2,X_3,\mathrm{rk})=X_0\oplus[S(B)]\oplus[S(B)<<<2]\oplus[S(B)<<<10]\oplus$$
$$[S(B)<<<18]\oplus[S(B)<<<24]$$

轮函数的结构如图 3-49 所示。

3. 加密算法

加密算法采用 32 轮迭代结构,每轮使用一个轮密钥。

设输入的明文为 4 个字(X_0, X_1, X_2, X_3)(128 比特长),输入的轮密钥为 rk_i($i=0,1,\cdots,31$),共 32 个字。输出的密文为 4 个字(Y_0,Y_1,Y_2,Y_3)(128 比特长),加密算法可描述如下:

$$X_{i+4}=F(X_i,X_{i+1},X_{i+2},X_{i+3},\mathrm{rk}_i)$$
$$=X_i\oplus T(X_{i+1}\oplus X_{i+2}\oplus X_{i+3}\oplus \mathrm{rk}_i)\quad(i=0,1,\cdots,31)$$

为了与解密算法需要的顺序一致,同时也与人们的习惯顺序一致,在加密算法之后还需要一个反序处理 R:

$$(Y_0,Y_1,Y_2,Y_3)=(X_{35},X_{34},X_{33},X_{32})=R(X_{32},X_{33},X_{34},X_{35})$$

加密算法的框图如图 3-49 所示。

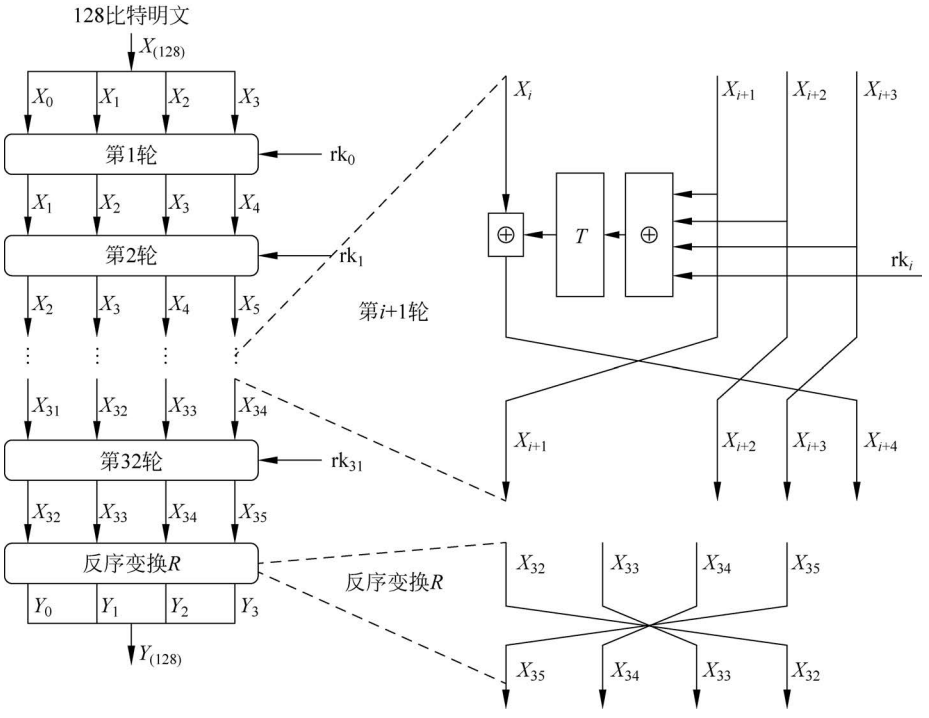

图 3-49　SM4 加密算法和轮函数结构图

4. 解密算法

解密算法与加密算法相同,只是轮密钥的使用顺序相反,解密轮密钥是加密轮密钥的逆序。

算法的输入为密文(Y_0, Y_1, Y_2, Y_3)和轮密钥 rk_i($i=31,30,\cdots,1,0$),输出为明文(X_0,X_1,X_2,X_3)。为了便于与加密算法对照,解密算法中仍然用 X_i 表示密文,于是可得到如下的解密算法。

$$X_i = F(X_{i+4}, X_{i+3}, X_{i+2}, X_{i+1}, \mathrm{rk}_i)$$
$$= X_{i+4} \oplus T(X_{i+3} \oplus X_{i+2} \oplus X_{i+1} \oplus \mathrm{rk}_i) \quad (i = 31,30,\cdots,1,0)$$

与加密算法之后需要一个反序处理同样的道理,在解密算法之后也需要一个反序处理 R:

$$(X_0, X_1, X_2, X_3) = R(X_3, X_2, X_1, X_0)$$

5. 密钥扩展算法

SM4 算法采用 32 轮的迭代加密结构,拥有 128 位加密密钥,一共使用 32 轮密钥,每一轮的加密使用 32 位的一个轮密钥。SM4 算法的特点使得它需要使用一个密钥扩展算法,在加密密钥当中产生 32 个轮密钥。在这个密钥的扩展算法当中有常数 FK、固定参数 CK 这两个数值,利用这两个数值便可完成它的这一个扩展算法。

1) 常数 FK

FK 为系统参数,其中每一项都为 32 位的字,表示为:$\mathrm{FK}=(\mathrm{FK}_0,\mathrm{FK}_1,\mathrm{FK}_2,\mathrm{FK}_3)$,在密钥扩展中使用的常数设置如下:

$$\mathrm{FK}_0 = (\mathrm{A3B1BAC6}), \quad \mathrm{FK}_1 = (\mathrm{56AA3350})$$
$$\mathrm{FK}_2 = (\mathrm{677D9197}), \quad \mathrm{FK}_3 = (\mathrm{B27022DC})$$

2) 固定参数 CK

CK 用于密钥扩展算法,一共使用有 32 个固定参数 CK_i,其中每一项 CK_i 都为 32 位的字,其产生规则如下:

设 $\mathrm{ck}_{i,j}$ 为 CK_i 的第 j 字节($i=0,1,\cdots,31$; $j=0,1,2,3$),即 $\mathrm{CK}_i=(\mathrm{ck}_{i,0},\mathrm{ck}_{i,1},\mathrm{ck}_{i,2},\mathrm{ck}_{i,3})$,则 $\mathrm{ck}_{i,j}=(4i+j)\times 7 \pmod{256}$。

这 32 个固定参数如下(十六进制):

```
00070e15,1c232a31,383f464d,545b6269,
70777e85,8c939aa1,a8afb6bd,c4cbd2d9,
e0e7eef5,fc030a11,181f262d,343b4249,
50575e65,6c737a81,888f969d,a4abb2b9,
c0c7ced5,dce3eaf1,f8ff060d,141b2229,
30373e45,4c535a61,686f767d,848b9299,
a0a7aeb5,bcc3cad1,d8dfe6ed,f4fb0209,
10171e25,2c333a41,484f565d,646b7279。
```

3) 密钥扩展算法

加密密钥:SM4 算法的加密密钥长度为 128 比特,将其分为四项,其中每项都为 32 位的字,可设输入的加密密钥为 $\mathrm{MK}=(\mathrm{MK}_0,\mathrm{MK}_1,\mathrm{MK}_2,\mathrm{MK}_3)$。

轮密钥：由加密密钥通过密钥扩展算法生成，其中每项都为 32 位的字，可设输出的轮密钥为 $rk_i(i=0,1,\cdots,30,31)$。

密钥扩展算法可描述如下，其中 $K_i(i=0,1,\cdots,34,35)$为中间数据：

(1) $(K_0,K_1,K_2,K_3)=(MK_0\oplus FK_0,MK_1\oplus FK_1,MK_2\oplus FK_2,MK_3\oplus FK_3)$

(2) For $i=0,1,\cdots,31$ Do

$$rk_i=K_{i+4}=K_i\oplus T'(K_{i+1}\oplus K_{i+2}\oplus K_{i+3}\oplus CK_i)$$

说明：其中的 T'变换与加密算法轮函数中的 T 基本相同，只将其中的线性变换 L 修改为以下的 L'：

$$L'(B)=B\oplus(B<<<13)\oplus(B<<<23)$$

从密钥扩展算法中可以发现，密钥扩展算法与加密算法在算法结构方面类似，同样都采用了 32 轮类似的迭代处理。

3.7.4　SM4 算法的安全性

SM4 算法的设计目标是提供高安全性、高效率和易于实现的分组密码方案。它采用 128 位密钥和 128 位分组大小，通过 32 轮的迭代结构和一系列的置换、代换和异或等基本运算来实现加密和解密操作，已通过了多种密码学安全性分析和评估，被广泛认可和接受。

SM4 算法的主体运算是非平衡 Feistel 网络，有很高的灵活性，所采用的 S 盒可以根据需要进行替换，以应对突发性的安全威胁。算法的 32 轮迭代采用串行处理，这与 AES 中每轮使用代换和混淆并行地处理整个分组有很大不同。此外，需要特别注意的是，密钥扩展算法采用了非线性变换 T，这个措施将极大地增强密钥扩展的安全性，SM4 算法的这个特点与 AES 加密算法类似，而 DES 的密钥生产算法并没有采用这种类似措施。

SM4 算法的发布标志着中国在商用密码算法领域的自主研发和国际化进程。它已成为中国政府和企事业单位的标准加密算法，并在各个领域得到广泛应用，包括金融、电子支付、电子政务、物联网等。SM4 算法的发布也体现了中国在信息安全领域的重视和发展。作为一种自主可控的密码算法，SM4 在保护国家信息资产、提升信息安全能力方面发挥着重要作用。同时，SM4 算法也向国际密码学界展示了中国在密码学研究和应用方面的贡献和实力。总的来说，SM4 密码算法是中国自主研发的分组密码算法，具有高安全性、高效率和易于实现的特点，被广泛应用于各个领域并成为中国的标准加密算法。

3.8　分组密码的工作模式

分组密码是最基本的密码技术之一，其处理消息的长度是固定的，如 DES 为 64 位，AES 为 128 位，但是在实际中需要处理的消息通常是任意长的，且要求密文尽量不确定，同时需要采用适当的方式来隐蔽明文的统计特性、数据的格式等，以提高整体的安全性。这些要求分组密码自身不能做到，因此，引出了如何利用分组密码处理任意长度消息的问题，解决这个问题的技术就是分组密码的工作模式。1980 年 12 月，FIPS 81 标准化了为

DES开发的五种工作模式。这些工作模式适合任何分组密码。本节以DES为例介绍分组密码主要的五种工作模式。

3.8.1 电码本模式

对给定的随机密钥k,每一块明文对应固定的密文块,即相同的明文组蕴含着相同的密文组,类似电码本中的码字,因此又称电码本(Electronic Code Book,ECB)模式,如图3-50所示。电码本(ECB)模式是分组密码最简单的工作模式。加密算法和解密算法定义如下。

加密算法:$C_i=E_k(M_i)(i=1,2,\cdots,m)$

解密算法:$M_i=D_k(C_i)(i=1,2,\cdots,m)$

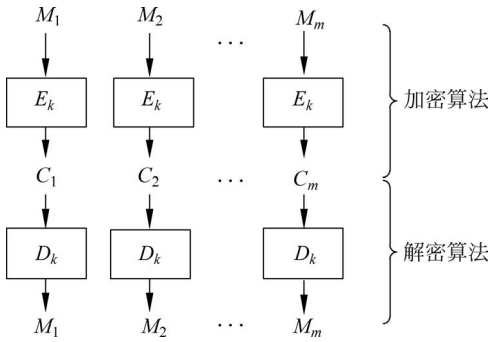

图3-50 ECB工作模式

上述算法中,$M=M_1M_2\cdots M_m$表示明文;$C=C_1C_2\cdots C_m$表示相应的密文。每个分组块M_i长度为64位,如果最后一个分组不够64位,则需要填充0或1。

ECB模式的优点是可并行运算、速度快和易于标准化。缺点是分组加密不能隐蔽数据格式;不能抵抗组的重放、嵌入、删除等攻击;加密长度只能是分组的倍数。因此,ECB模式仅适用于短数据加密,如果需要安全地传递DES密钥,ECB是最合适的模式。

3.8.2 密码分组链接模式

为了解决ECB的安全缺陷,可以让重复的明文分组产生不同的密文分组,密码分组链接(Cipher Block Chaining,CBC)模式就可满足这一要求。CBC模式主要基于两种思想。第一,所有分组的加密都链接在一起,其中各分组所用的密钥相同。加密时输入的是当前的明文分组和上一个密文分组的异或,这样使得密文分组不仅依赖当前明文分组,而且还依赖前面所有的明文分组。因此,加密算法的输入不会显示出与这次的明文分组之间的固定关系,所以重复的明文分组不会在密文中暴露出这种重复关系。第二,加密过程使用初始量(IV)进行了随机化。图3-51是CBC工作模式示意图。

加密算法和解密算法定义如下。

加密算法:

$$C_i=E_k(C_{i-1}\oplus M_i)(i=1,2,\cdots,m)$$
$$C_0=\mathrm{IV}$$

解密算法:

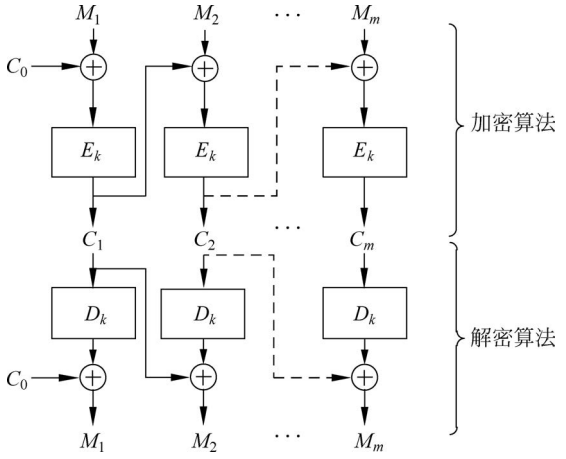

图 3-51　CBC 工作模式示意图

$$M_i = D_k(C_i) \oplus C_{i-1} (i = 1, 2, \cdots, m)$$
$$C_0 = \mathrm{IV}$$

CBC 模式的优点是引入了收发双方相互可公开的随机初始量 $C_0 = \mathrm{IV}$，为使安全性最高，IV 应像密钥一样被保护，可使用 ECB 加密模式来发送 IV。保护 IV 的原因：如果敌手篡改 IV 中的某些比特，则接收方收到的 M_1 中相应的比特也发生了变化。

如果加密算法 E 是伪随机的，则输出具有一定的随机性，避免了 ECB 模式的缺点，隐蔽了明文的数据格式，在一定程度上能防止数据窜改。缺点是会出现错误传播，密文 C_i 在传输中发生错误不仅影响 C_i 的正确译文，还会影响其后 C_{i+1} 的正确解密。CBC 模式不能纠正传输中的同步差错，即传输中增加或丢失一个或多个比特所引起的密文组边缘的错乱。CBC 模式是应用最广、影响也最大的一个工作模式，适合加密长度大于 64 位的消息，但消息长度只能是分组长度的倍数，不能是任意长度的消息；此外，CBC 模式还可以用来实现报文的完整性认证和用户的身份认证。

3.8.3　密码反馈模式

上述的 CBC 模式非常适用于大量信息的加密，然而在实时(Real-Time)通信的应用中，如果接收方收到发送方传送的密文后，想立即解密时，就不适用了，此时效率就变成非常重要的问题，这种情况下密码反馈(Cipher Feedback，CFB)模式加密就派上用场了。

利用 CFB 模式思想是：把分组密码当作流密码使用，即 CFB 模式可将 DES 分组密码置换成流密码。流密码具有密文和明文长度一致、运行实时的性质，这样数据可以在比分组小得多的单元里进行加密。如果需要发送的每个字符长为 8 比特，就应使用 8 比特密钥来加密每个字符。如果长度超过 8 比特，则造成浪费。但是要注意，由于 CFB 模式中分组密码是以流密码方式使用，所以加密和解密操作完全相同，因此无法适用于公钥密码系统，只能适用于对称密钥密码系统。

加密算法和解密算法定义如下：设原分组密码长度为 l，以流密码方式传送的单元长度为 s，一般取 $s = 8$ 且 $1 < s < l$。如图 3-52 所示，明文 M 被划分成明文组 $M_1 M_2 \cdots M_i \cdots$

M_m,其中 M_i(1≤i<m)都为 s 比特。

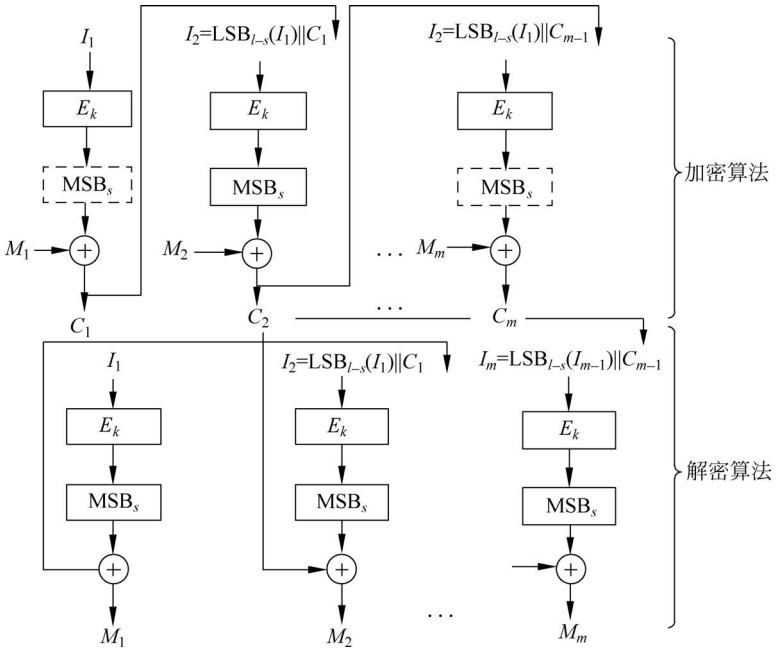

图 3-52　CFB 工作模式

MSB$_s$(X)：表示从自变量 X 中选择最左侧(最高有效位)s 比特输出。

LSB$_{l-s}$(X)：表示把自变量 X 的内容向左移位 s 比特(最左边的 s 比特丢失了)。

‖：表示串联。

加密算法：$C_1=M_1\oplus\text{MSB}_s(E_k(\text{IV}))$

$$C_i=M_i\oplus\text{MSB}_s(E_k(\text{LSB}_{l-s}(I_{i-1})\,\|\,C_{i-1}))(i=2,3,\cdots,m)$$

解密算法：$M_1=C_1\oplus\text{MSB}_s(E_k(\text{IV}))$

$$M_i=C_i\oplus\text{MSB}_s(E_k(\text{LSB}_{l-s}(I_{i-1})\,\|\,C_{i-1}))(i=2,3,\cdots,m)$$

加密时,加密算法的输入是 64 比特移位寄存器,其初值为某个初始量 IV。加密算法输出的最左(最高有效位)s 比特与明文的第一个单元 M_1 进行异或,产生出密文的第一个单元 C_1,并传送该单元。然后将移位寄存器的内容左移 s 位,并将 C_1 送入移位寄存器最右边(最低有效位)s 位。这一过程继续到明文的所有单元都被加密为止。

解密时,将收到的密文单元与加密函数的输出进行异或。注意这时仍然使用加密算法而不是解密算法,原因如下：$C_1=M_1\oplus\text{MSB}_s(E_k(\text{IV}))$；$M_1=C_1\oplus\text{MSB}_s(E_k(\text{IV}))$。可证明以后各步也有类似的这种关系。

CFB 模式也需要一个初始量 IV,无须保密,但对每条消息必须有一个不同的 IV。

CFB 模式的缺点：

(1) 对信道错误较敏感,且会造成错误传播。

(2) 数据加密的速率被降低。

CFB 模式的优点：

(1) 可以处理任意长度的消息,能适应用户不同数据格式的需要。

(2) 可实现自同步功能。

(3) 具有有限步的错误传播,除能获得保密性外,还可用于认证。

(4) 具备 CBC 模式的优点。

该模式适应于数据库加密、无线通信加密等对数据格式有特殊要求或密文信号容易丢失或出错的应用环境。

3.8.4　输出反馈模式

输出反馈(Output Feedback,OFB)模式的结构类似于 CFB 模式,也把分组密码置换成流密码的形式进行加密处理,如图 3-53 所示。不同之处在于 OFB 模式将加密算法的输出反馈到移位寄存器,而 CFB 模式是将密文单元反馈到移位寄存器,即 OFB 模式加密后的初始量没有与明文进行异或操作。事实上对于第一组明文,初始量加密以后作为第二组明文的输入并且再与第一组明文进行异或操作。后续的加密操作都发生在异或之前。

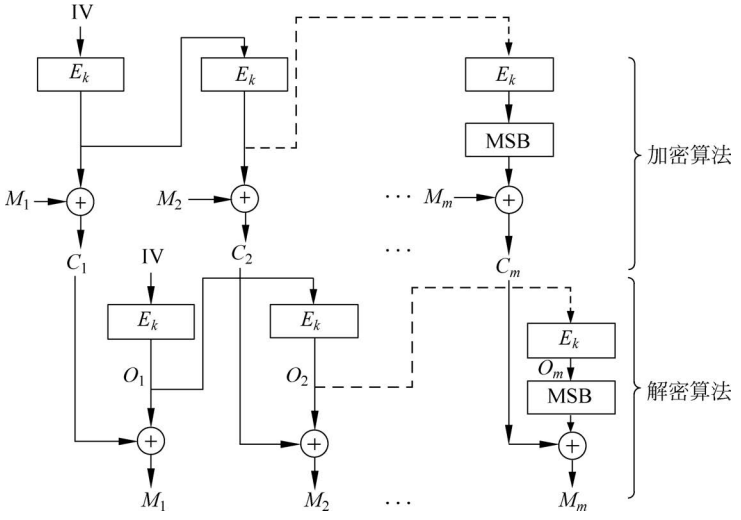

图 3-53　OFB 工作模式

加密算法和解密算法定义如下。

加密算法:

$$O_0 = \mathrm{IV}$$

$$O_i = E_k(O_{i-1}) \quad C_i = M_i \oplus O_i \quad (i = 1, 2, \cdots, m-1)$$

$$O_m = E_k(O_{m-1}) \quad C_m = M_m \oplus \mathrm{MSB}_s(E_k(O_m))$$

解密算法:

$$O_0 = C_0$$

$$O_i = E_k(O_{i-1}) \quad M_i = C_i \oplus O_i \quad (i = 1, 2, \cdots, m-1)$$

$$O_m = E_k(O_{m-1}) \quad M_m = C_m \oplus \mathrm{MSB}_s(E_k(O_m))$$

OFB 模式的初始量 IV 的要求同 CFB 模式相同,也无须保密,对每条消息也必须选择不同的 IV。

OFB 模式的优点:

(1) 错误传播小,如 C_1 中的 1 比特错误只导致 M_1 中的 1 比特错误,后面各明文单元则不受影响;而在 CFB 中,C_1 也作为移位寄存器的输入,因此它的 1 比特错误会影响解密结果中各明文单元的值。

(2) 消息长度是任意的,可以预处理,并且可在线处理(随时处理明文)等。

OFB 模式的引入是为了克服 CBC 和 CFB 这两种模式中存在的错误传播问题。OFB 模式虽然不存在错误传播问题,但对密文是否被篡改难以进行检测,因此比 CFB 模式更易遭受攻击,好在 OFB 模式多在同步信道中运行,对手难以知道消息的起止点而使篡改攻击不易奏效。OFB 模式不具有自同步能力,系统必须保持严格的同步,否则难以解密。在实际应用中,比其他模式更适用于不太稳定的信道(噪声通道)上加密,如人造卫星通信中的加密。

3.8.5 计数器模式

CBC 模式和 CFB 模式都存在这样一个问题:不能以随机顺序来访问加密的数据,因为当前密文数据块的解密依赖于前面的密文块。而这个问题对于很多应用来说,特别是数据库的应用,是很难接受的。为此,又出现了另一种工作模式,即计数器(Counter,CTR)模式。

CTR 模式本质上和 OFB 模式很类似,都是将分组密码变为流密码,如图 3-54 所示。两者的区别在于:CTR 模式通过递增一个加密计数器来产生连续的密钥流,密钥流的产生之间是没有关联的,而是由计数器来提供;而 OFB 模式中的密钥流是逐级反馈的。CTR 模式中计数器可以是任意保证不产生长时间重复输出的函数,但使用一个普通的计数器是最简单和最常见的做法。

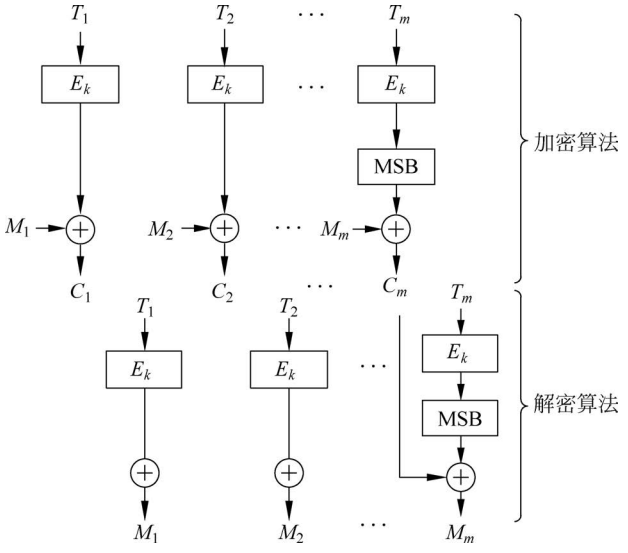

图 3-54 CTR 工作模式

加密算法和解密算法如下定义。

加密算法:

$$C_i = M_i \oplus E_k(T_i) \quad (i = 1,2,\cdots,m-1)$$
$$C_m = M_m \oplus \mathrm{MSB}_{|M_m|}(E_k(T_m))$$

解密算法：

$$M_i = C_i \oplus E_k(T_i) \quad (i = 1,2,\cdots,m-1)$$
$$M_m = C_m \oplus \mathrm{MSB}_{|M_m|}(E_k(T_m))$$

CTR 模式需要计数器序列 $T_1,\cdots,T_i,\cdots,T_m$，通过对计数器序列调用分组加密算法得到密钥流，然后和明文异或得到密文。对计数器序列的要求是两两不同，而且不仅是在一个消息的操作中，而且是在同一密钥的所有操作中均要求所用计数器序列两两不同。

也许有人会问，为什么需要这么多模式？CTR 模式最吸引人的一个特点就是可以并行化，因为 CTR 模式不需要任何反馈，这与 OFB 或 CFB 模式完全不同。所以可以让两个分组密码引擎同时并行工作，即让两个引擎同时使用第一个分组密码加密计数器值 CTR1 和 CTR2。等这两个分组密码引擎完成后，一个引擎将继续加密值 CTR3，而另一个引擎则继续加密 CTR4，如此循环。这种方案的加密速率是单个实现方式的两倍。当然，也可以同时运行多个分组密码引擎，这也会使加密速率按比例增加。对吞吐率要求严格的应用，并行化的加密模式非常合适。CTR 模式被广泛用于 ATM 网络安全和 IPSec 应用中。

习题 3

1. 分别画出并说明流密码流程和分组密码流程。

2. 三级线性反馈移位寄存器在 $c_3 = 1$ 时可有 4 种线性反馈函数，设其初始状态为 $(a_1,a_2,a_3) = (1,0,1)$，求各线性反馈函数的输出序列及周期。

3. 设 n 级线性反馈移位寄存器的特征多项式为 $p(x)$，初始状态为 $(a_1,a_2,\cdots,a_{n-1},a_n) = (1,0,1)$，证明输出序列的周期等于 $p(x)$ 的阶。

4. 设 $n=4$，$f(a_1,a_2,a_3,a_4) = a_1 \oplus a_4 \oplus a_2a_3$，初始状态为 $(a_1,a_2,a_3,a_4) = (1,1,0,1)$，求此非线性反馈移位寄存器的输出序列及周期。

5. 已知流密码的密文串 1010110110 和相应的明文串 0100 010001，而且已知密钥流是使用三级线性反馈移位寄存器产生的，试破译该密码系统。

6. 给定密钥 $k = (0111111101)$，按照 S-DES 密码处理过程以手工方式对明文字节 (01000001) 进行加密和解密。要求写出每个函数处理后的中间结果。

7. 说明 DES 中每个子密钥的第一个 24 位来自初始密钥的同一个 28 位子集，而其后的第二个 24 位来自初始密钥的不相交的 28 位子集。

8. 画出 DES 解密算法的流程图（注意：输入是密文，输出是明文）。

9. 在 IDEA 算法中，已知明文 M 和密钥 k 分别为

$M =$ 10101010 11100110 01010101 00001111 11001100 00110011 10011001 01100110

k = 00000000 11111111 00000000 11111111 11111111 00001111 11110000 11111111
00001111 11110000 11111111 00000000 00001111 11111111 11110000 00001111

求第一轮的输出和第二轮的输入。

10. 为什么 IDEA 算法中的乘法操作是模($2^{16}+1$),而不是简单的模 2^{16}?

11. 以 F5 为例说明 S 盒的替代操作(不通过查表,而通过代数运算)。

12. 在 8 比特 CFB 模式中,如果在密文字符中出现 1 比特的错误,那么该错误能传播多远?

13. 比较 CBC 模式和 CFB 模式的异同。

14. 请设计一种一次加密一个明文字节(例如加密来自远程的击键)的 OFB 模式方案。使用的分组密码为 AES,对每个新的明文字节都执行一个分组密码操作。请绘出你的方案框图,请特别留意你给出的框图中使用的位长度。