第 3 章　对称密码体制

作为现代密码学，虽然算法更加复杂，但原理还是没变。只不过现代密码学的算法是针对二进制位而不是针对字母进行变换，实际上这只是字母表长度上的改变：从 26 个元素变为 2 个元素。大多数优秀算法的主要组成部分仍然是代换和置换的组合，如 DES 算法。

前面已经提到，对称密码体制就是在加密和解密时用到的密钥相同，或加密密钥和解密密钥之间存在着确定的转换关系。其实质是设计一种算法，能在密钥控制下，把 n 位明文简单而又迅速地置换成唯一的 n 位密文，并且这种变换是可逆的(解密)。

根据不同的加密方式，对称密码体制又有两种不同的实现方式，即分组密码和序列密码(或称流密码)。其中，分组密码是先把明文划分为许多分组，每个明文分组被当作一个整体来产生一个等长(通常情况下)的密文分组，通常使用的是 64 位或 128 位分组大小。而序列密码则每次加密数据流中的一位或一个字节。

3.1　分 组 密 码

分组密码体制是目前商业领域中比较重要而流行的一种加密体制，广泛地应用于数据的保密传输、加密存储等应用场合。分组密码对明文进行加密时，首先需要对明文进行分组，每组的长度都相同，然后对每组明文分别加密得到等长的密文。分组密码的安全性主要依赖于密钥，而不依赖于对加密算法和解密算法的保密，因此，分组密码的加密和解密算法可以公开。

1. 要求

分组密码算法实际上就是在密钥的控制下，简单而迅速地找到一个置换，用来对明文分组进行加密变换，一般情况下对密码算法的要求如下：

(1) 分组长度 m 足够大。当分组长度 m 较小时，分组密码很类似于某些古典密码，如维吉尼亚密码、希尔密码和置换密码。它仍然有效地保留了明文中的统计信息，这种统计信息将给攻击者留下可乘之机，攻击者可以有效地穷举明文空间，得到密码变换本身。

(2) 密钥空间足够大。分组密码的密钥所确定的密码变换只是所有置换中极小的一部分。如果这一部分足够小，攻击者可以有效地通过穷举密钥，确定所有的置换。到时，攻击者就可以对密文进行解密，以得到有意义的明文。

(3) 密码变换必须足够复杂,使攻击者除了穷举法攻击以外,找不到其他简洁的数学破译方法。

2. 基本思想

为了便于实现和分析,在设计分组密码时通常遵循以下两个基本思想:

(1) 扩散(diffusion)。将明文及密钥的影响尽可能迅速地散布到较多个输出的密文中。产生扩散的最简单方法是通过置换(如重新排列字符)。

(2) 混淆(confusion)。其目的在于使作用于明文的密钥和密文之间的关系复杂化,使明文和密文之间、密文和密钥之间的统计相关特性极小化,从而使统计分析攻击不能奏效。通常的方法是代换。

3. 技术

具体来说,可以综合采用以下两种技术:

(1) 将大的明文分组再分成几个小段,分别完成各个小段的加密置换,最后进行并行操作。这样使得总的分组长度足够大,有利于对密码的实际分析和评测,以保证密码算法的强度。

(2) 采用乘积密码技术。乘积密码就是以某种方式连续执行两个或多个密码变换。例如,设有两个子密码变换 $E1$ 和 $E2$,则先以 $E1$ 对明文进行加密,然后再以 $E2$ 对所得结果进行加密。其中,$E1$ 的密文空间与 $E2$ 的明文空间相同。如果使用得当的话,乘积密码可以有效地掩盖密码变换的弱点,构成比其中任意一个密码变换强度更高的密码系统。

典型的分组密码有 DES、AES。

3.2　数据加密标准

3.2.1　数据加密标准简介

数据加密标准(data encryption standard,DES)是一种对计算机数据进行密码保护的数学算法,它的产生被认为是 20 世纪 70 年代信息加密技术发展史上的里程碑之一。由于 20 世纪 60 年代计算机得到了迅猛的发展,大量的数据资料被集中储存在大型计算机数据库中并在计算机通信网中进行传输,其中有些通信具有高度的机密性,有些数据具有极为重要的价值,因此对计算机通信及计算机数据进行保护的需求日益增长。当时的美国虽然已经制定了数据保密措施,但是人们普遍认为这些保密措施只对业余人员有效,对于职业人员来说,要截获通信信号、绕过报警系统并接通通信设备,取出、复制、删除、插入或更改信息是不成问题的。针对这种情况,有人提出了两种对数据进行保护的方法,一种方法是对数据进行物理保护,即把重要的数据存放到安全的地方,如银行的地下室中;另一种方法是对数据进行密码保护。

由于普遍认为加密算法如果足够复杂的话,密码是一种有效的措施。所以美国国家标准局(NBS)于 1973 年 5 月发出通告,公开征求一种标准算法用于对计算机数据在传输和存储期间实现加密保护的密码算法。1975 年美国国家标准局接受了美国国际商业机器公司

(IBM)推荐的一种密码算法并向全国公布,征求对采用该算法作为美国信息加密标准的意见。

经过两年的激烈争论,美国国家标准局于1977年7月正式采用该算法作为美国数据加密标准。1980年12月美国国家标准协会ANSI正式采用这个算法作为美国的商用加密算法。

DES是一种对称密码体制,所使用的加密和解密密钥是相同的,是一种典型的按分组方式工作的密码。其基本思想是将二进制序列的明文分成每64位一组,用长为64位的密钥对其进行16轮代换和置换加密,最后形成密文。

DES的巧妙之处在于除了密钥输入顺序,其加密和解密的步骤完全相同,这就使得在制作DES芯片时易于做到标准化和通用化。这一点尤其适合现代通信的需要,在DES出现以后,经过许多专家学者的分析论证证明它是一种性能良好的数据加密算法,不仅随机特性好,线性复杂度高,而且易于实现,加上能够标准化和通用化,因此DES在国际得到了广泛的应用。

3.2.2 DES加密解密原理

DES是典型的传统密码体制,它利用传统的代换和置换等加密方法,现介绍算法如下:

加密前,先将明文分成64位的分组,然后将64位二进制码输入到密码器中,密码器对输入的64位码首先进行初始置换,然后在64位主密钥产生的16个子密钥控制下进行16轮乘积变换,接着再进行逆初始置换就得到64位已加密的密文。

算法的主要步骤如图3-1所示。

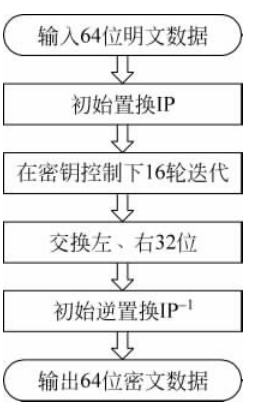

图3-1 DES算法的主要步骤

假定信息空间都是由{0,1}组成的字符串,信息被分成64位的块,密钥是56位。经过DES加密的密文也是64位的块。设m是一个64位的信息块,k为56位的密钥,即

$$m = m_1\, m_2 \cdots m_{64}$$
$$m_i = 1,2,\cdots,64$$
$$k = k_1 k_2 \cdots k_{64}$$
$$k_i = 1,2,\cdots,64$$

其中,k_8,k_{16},k_{24},k_{32},k_{40},k_{48},k_{56},k_{64}是奇偶校验位,真正起作用的密钥仅56位。

下面介绍一个分组的加密过程。

1. 初始置换IP

将64个明文位的位置进行置换,得到一个乱序的64位明文组,然后分成左右两段,每段为32位,以L和R表示,如图3-2所示。

2. 迭代变换

迭代变换是DES算法的核心部分。如图3-3所示,将经过IP置换后的数据分成32位左右两组,在迭代过程中彼此左右交换位置,每次迭代只对右边的32位进行一系列的加密

变换。在此轮迭代即将结束时，把左边的 32 位与右边的 32 位诸位模 2 相加，作为下一轮迭代时右边的段，并将原来右边的未经变换的段直接送到左边的寄存器中作为下一轮迭代时左边的段。在每一轮迭代时，右边段要经过选择扩展运算 E、密钥加密运算、选择压缩运算 S、置换运算 P 和左右混合运算。

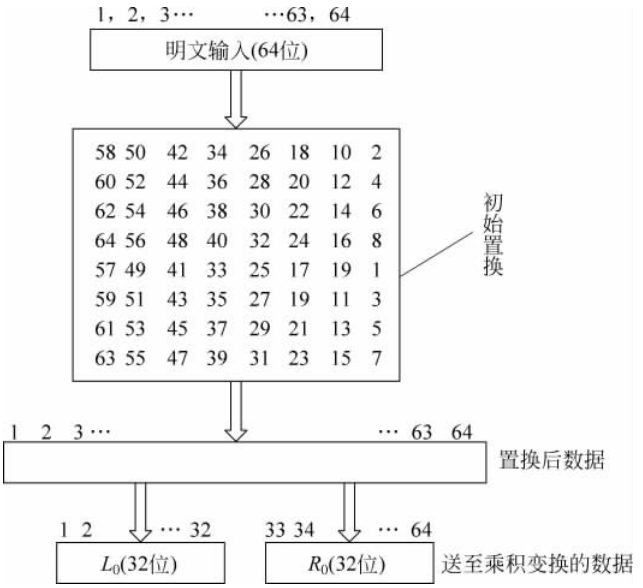

图 3-2　初始置换

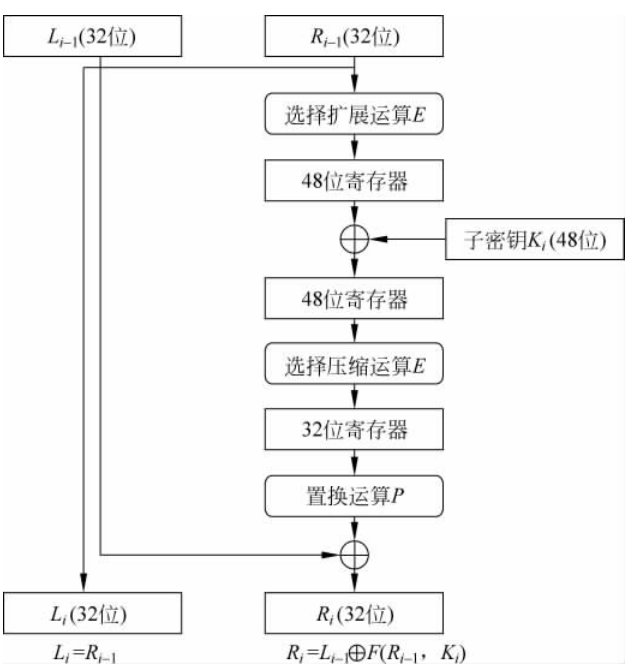

图 3-3　乘积变换

这样的迭代共进行16轮,结束后,再将所得的左、右长度相等的L_{16}和R_{16}进行交换得到64位数据。

1) 选择扩展运算E

将输入的32位R扩展成48位输出,其变换表如图3-4所示。

令s表示E的输入的下标,则E的输出将是对原下标$s=0$或$1(\mathrm{mod}\ 4)$的各位重复一次得到的,即对原第32、1、4、5、8、9、12、13、16、17、20、21、24、25、28、29各位重复一次得到数据扩展。将表中数据按行读出即得到48位输出。

图3-4 选择扩展运算E

2) 密钥加密运算

将子密钥产生器输出的48位子密钥k与选择扩展运算E输出的48位数据按位模2相加(子密钥如何产生请见后文)。

3) 选择压缩运算S

将前面送来的48位数据自左至右分成8组,每组6位。然后并行送入8个S盒,每个S盒为一非线性代换网络,有4个输出。盒S_1～S_8的选择函数关系如表3-1所示,运算S的框图如图3-5所示。

8个S盒是将6位的输入映射为4位的输出。

以S_1盒为例说明它们的功能如下:

若输入为$b_1b_2b_3b_4b_5b_6$其中b_1b_6两位二进制数表达了0～3之间的数。$b_2b_3b_4b_5$为4位二进制数,表达0～15之间的某个数。

在S_1表中的b_1b_6行$b_2b_3b_4b_5$列找到一数m($0\leqslant m\leqslant 15$),若用二进制表示为$m_1m_2m_3m_4$,则$m_1m_2m_3m_4$便是它的4位输出。

例如,输入为001111,$b_1b_6=01=1$,$b_2b_3b_4b_5=0111=7$,即在S_1盒中的第1行第7列求得数1,所以它的4位输出为0001。

又如对于S_2盒输入为101011,则S_2盒中3行5列元素为15,故输出为1111。

表 3-1　DES 的 *S* 盒定义

行＼列	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7	
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8	
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0	S_1
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13	
0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10	
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5	
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15	S_2
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9	
0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8	
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1	
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7	S_3
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12	
0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15	
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9	
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4	S_4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14	
0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9	
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6	
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14	S_5
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3	
0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11	
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8	
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6	S_6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13	
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1	
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6	
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2	S_7
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12	
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7	
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2	
2	1	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8	S_8
3	2	1	14	7	1	10	7	13	15	12	9	0	3	5	6	11	

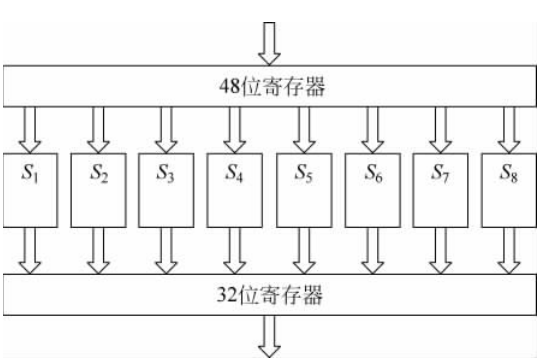

图 3-5　*S* 盒的结构

S 盒是 DES 的核心,也是 DES 算法最敏感的部分,其设计原理至今仍讳莫如深,显得非常神秘。所有的替换都是固定的,但是又没有明显的理由说明为什么要这样,有许多密码学家担心美国国家安全局设计 S 盒时隐藏了某些陷门,使得只有他们才可以破译算法,但研究中并没有找到弱点。

美国国家安全局曾透露了 S 盒的几条设计准则:

(1) 所有的 S 盒都不是它输入的线性仿射函数,换句话说,就是没有一个线性方程能将4个输出比特表示成6个位输入的函数。

(2) 改变 S 盒的1位输入,输出至少改变2位,这意味着 S 盒是经过精心设计的,它最大程度上增大了扩散量。

(3) S 盒的任意一位输出保持不变时0和1个数之差极小,即如果保持一位不变而改变其他五位,那么其输出0和1的个数不应相差太多。

4) 置换运算

置换运算 P 对 S_1 至 S_8 盒输出的32位数据进行坐标变换,如图3-6所示,置换 P 输出的32位数据与左边32位即 R_{i-1} 诸位模2相加所得到的32位作为下一轮迭代用的右边的数字段,并将 R_{i-1} 并行送到左边的寄存器作为下一轮迭代用的左边的数字段。

3. 子密钥产生器

将64位初始密钥经过置换选择 PC-1,循环移位置换、置换选择 PC-2,给出每次迭代加密用的子密钥 k_i,如图3-7所示。

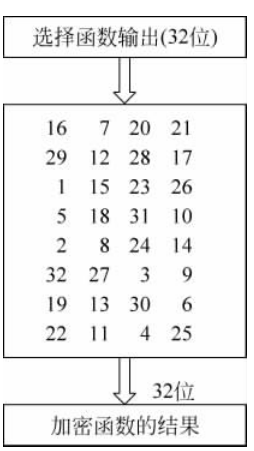

图3-6 置换运算 P

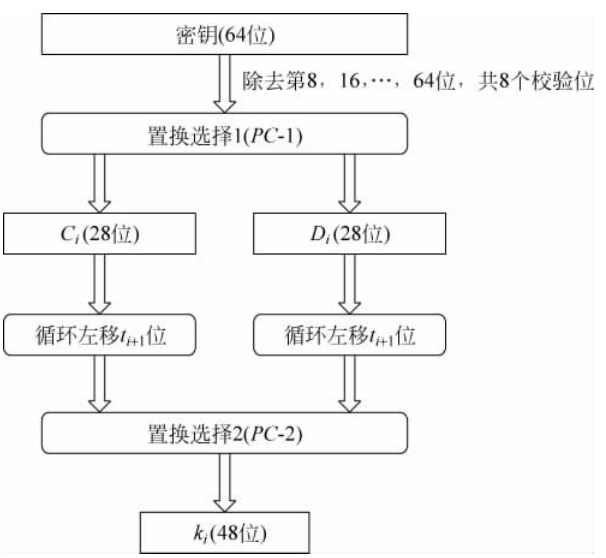

图3-7 子密钥产生器

在64比特初始密钥中有8位为校验位,其位置号为8、16、24、32、48、56和64,其余56位用于子密钥计算,将这56位送入置换选择 PC-1。置换后分为两组,每组为28位,分别送入 C 寄存器和 D 寄存器中,如图3-8所示。

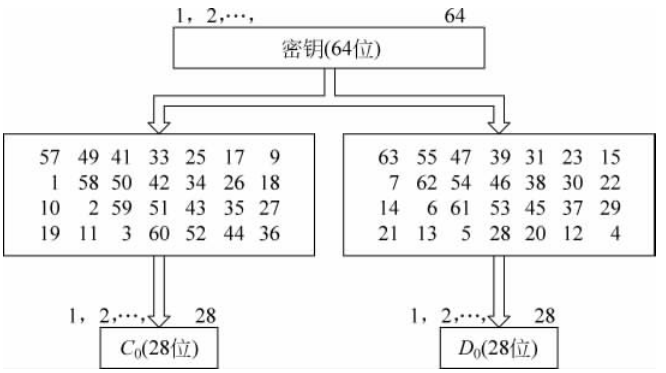

图 3-8　置换选择 *PC*-1

在各次迭代中 C 和 D 寄存器分别将存数进行左循环移位置换，移位次数如表 3-2 所示。每次移位后将 C 和 D 寄存器的存数送给置换选择 *PC*-2。

表 3-2　移位次数表

第 i 次迭代	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
循环左移次数	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

置换选择 *PC*-2 将 C 中第 9、18、22、25 位和 D 中第 7、9、15、26 位删去，并将其余数字置换位置后送出 48 位数字作为第 i 次迭代时所用的子密钥 k_i，如图 3-9 所示。

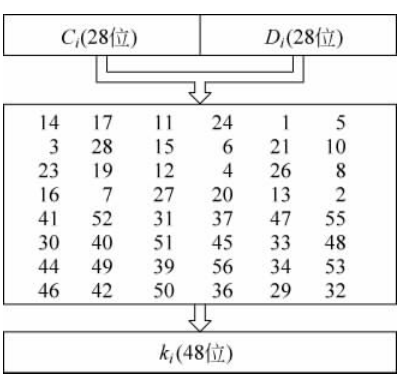

图 3-9　置换选择 *PC*-2

例如，$k=k_1k_2\cdots k_{64}$，则 $C_0=k_{57}k_{49}\cdots k_{44}k_{36}$，$D_0=k_{63}k_{55}\cdots k_{12}k_4$。

下面介绍如何从 C_i、D_i 求 C_{i+1}、D_{i+1}，$i=0,1,2,\cdots,15$。

设 $C_1=c_1c_2\cdots c_{28}$，$D_1=d_1d_2\cdots d_{28}$。首先要做左移运算，左移的位数见表 3-2。例如，设 $C_i=c_1c_2\cdots c_{28}$，$D_i=d_1d_2\cdots d_{28}$，则 $C_2=c_2c_3\cdots c_{28}\ c_1$，$D_2=d_2d_3\cdots d_{28}\ d_1$。

C_3 和 D_3 是由 C_2、D_2 左移 1 位而得到的，C_4 和 D_4 是由 C_3、D_3 左移 2 位而得到的，所以 $C_4=c_5c_6\cdots c_{28}\ c_1\ c_2\ c_3\ c_4$，$D_4=d_5d_6\cdots d_{28}\ d_1\ d_2\ d_3\ d_4$。

因此，$C_i\ D_i=b_1b_2\cdots b_{56}$，则 $k_i=b_{14}\ b_{17}\ b_{11}\ b_{24}\cdots b_{36}\ b_{29}\ b_{32}$。

4. 逆初始置换 IP^{-1}

将16轮迭代后给出的64位组进行置换得到输出的密文组,如图3-10所示,输出为阵中元素按行读的结果。注意IP中的第58位正好是1,也就是说在IP的置换下第58位换为第1位。同样,在IP的置换下应将第1位换回第58位,依此类推,由此可见输入组 m 和 $IP^{-1}(IP(m))$是一样的,IP和 IP^{-1}在密码上的意义不大,它的作用在于打乱原来输入 m 的ASCII码字划分关系。

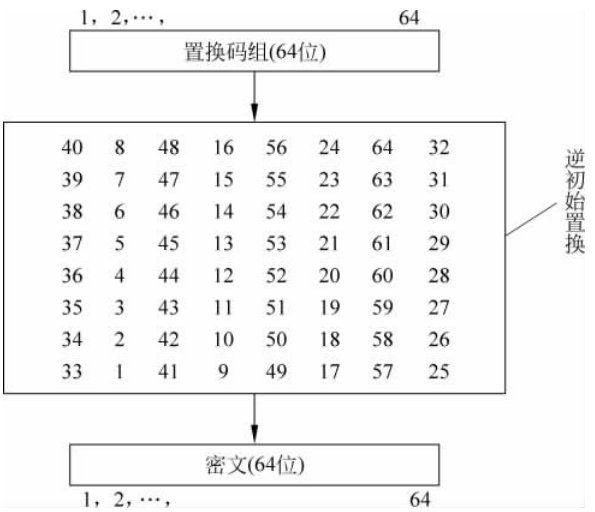

图3-10 逆初始置换 IP^{-1}

逆初始置换后得到的64位数据分组,即为加密后得到的密文。

5. DES的解密

解密算法与加密算法相同,只是子密钥的使用次序相反。把64位密文当作输入,第一次解密迭代使用子密钥 k_{16},第二次解密迭代使用子密钥 k_{15},第16次解密迭代使用子密钥 k_1,最后的输出便是64位的明文。

3.2.3 DES的安全性

DES的出现是密码学史上的一个创举,以前任何设计者对于密码体制及其设计细节都是严加保密的。而DES公开发表,任人研究和分析。无须经过许可就可以制作DES的芯片和以DES为基础的保密设备。DES的安全性完全依赖于所用的密钥。DES在20多年的应用实践中,没有发现严重的安全缺陷,在世界范围内得到了广泛的应用,为确保信息安全做出了不可磨灭的贡献。

然而,出于安全方面的考虑,在DES被采纳为标准之前,曾受到过激烈的批评,其实直到今天都未平息。批评主要集中在两个方面,其一,DES的前身是IBM公司的LUCIFER算法,该算法采用的密钥长度是128位,而DES却只使用了56位的密钥,减少了72位。批

评者担心密钥太短而无法抗击穷举攻击；其二，DES 的内部结构，即 S 盒的设计标准被列入官方机密。所以，用户不能确信 DES 的内部结构是没有弱点的，美国国家安全局（NSA）有可能利用这些弱点在没有密钥的情况下解密。不过近年来的差分分析方面的研究表明，DES 的内部结构是强健的。而且按照 IBM 的参与者所说，原始算法中唯一做过改动的就是 S 盒，这是由 NSA 建议的，它去掉了测试过程中发现的一些算法脆弱性。

自从 DES 问世至今已经有 30 多年了，尽管一开始人们就对它有颇多的担心和争议，这些年来许许多多的人对它进行各种各样的研究攻击。从目前的成果来看，除了穷举攻击之外，就没有更好的方法破译 DES 了。

从应用实践来看，DES 具有良好的“雪崩效应”。所谓雪崩效应，就是明文或密钥的微小改变将对密文产生很大的影响。特别地，明文或密钥的某一位发生变化，会导致密文的很多位发生变化。

DES 显示了很强的雪崩效应，如通过实验可以发现，两条仅有一位不同的明文，使用相同的密钥，仅经过 3 轮迭代，所得两段准密文就有 21 位不同；一条明文，使用两个仅一位不同的密钥加密，经过数轮变换之后，有半数的位都不相同。

当然，DES 也存在着一些有可能被利用的弱点，如：

（1）密钥较短：56 位密钥空间约为 7.2×10^{16}。1997 年 6 月 Rocke Verser 小组通过因特网利用数万台微机历时 4 个月破译了 DES；1998 年 7 月，EFF 用一台 25 万美元的机器，历时 56 小时破译了 DES。

（2）存在弱密钥：有的密钥产生的 16 个子密钥中有重复者。

（3）互补对称性：$C=\mathrm{DES}(M,K)$，则 $C'=\mathrm{DES}(M',K')$。其中，M'，C'，K' 是 M，C，K 的非。

多年来对 DES 进行的大量研究，主要成果集中在以下几个方面。

1. 弱密钥

DES 算法在每次迭代时都有一个子密钥供加密用，如果一个外部密钥所产生的所有子密钥都是一样的，则这个密钥就称为弱密钥（weak key）。

若 k 为弱密钥，则有

$$\mathrm{DES}_k(\mathrm{DES}_k(x)) = x$$

$$\mathrm{DES}_k^{-1}(\mathrm{DES}_k^{-1}(x)) = x$$

即以 k 对 x 加密或解密两次都可以恢复出明文，其加密运算和解密运算没有区别。

如果随机选取密钥，在总数 2^{56} 个密钥中，弱密钥所占比例很小，加以注意就可以避开，对 DES 的安全性影响不大。

2. 密文与明文、密文与密钥的相关性

有人详细研究了 DES 的输入，表明每个密文位都是所有明文位和所有密钥位的复合函数，并且指出达到这一要求所需的迭代次数最少为 5，迭代 8 次以后输出和输入就可认为是不相关的了。

3. 密钥搜索机

对 DES 的安全性的意见中，较为一致的看法是 DES 的密钥短了些，密钥长度是 56 位，

密钥量为 $2^{56}\approx7.2\times10^{16}$ 个,选择长密钥时会使成本提高,运行速度降低,若要对 DES 进行密钥搜索破译,分析者在得到一组明文-密文对的情况下,可对明文进行不同的密钥加密,直到得到的密文与已知的密文-明文对中的相符就可确定所用的密钥了。

1977 年 Differ 和 Hellman 认为利用 100 万个超大规模集成电路块所组成的一台专门用于破译 DES 的并行计算机能在一天中穷举搜索所有 2^{56} 个密钥,每个集成块每微秒检查一个密钥,则每天可以检查 8.64×10^{10} 个密钥。如果用一个集成块检查全部密钥,则几乎要花费 8.33×10^{5} 天,约 2283 年。

但是如果用 100 万个集成块,则在一天时间内可以检查整个密钥空间。这样一台机器在 1977 年需耗资约 2000 万美元。Differ 和 Hellman 据此指出,除了像美国国家安全局那样的机构外,任何人不可能破译 DES。但他们预测到 1990 年制造和破译 DES 专用机的成本将要大幅度下降,那时 DES 将完全是不安全的。

事实证明,他们的预测是有道理的,在过去相当长的一段时间里,人们找不到比穷举搜索更有效的方法攻击 DES。也没有能力对 56 位的密钥进行穷举搜索,因而在过去,DES 是安全的,但目前的事实证明这个历史已成为过去,DES 不能经受住穷举攻击。

3.2.4 多重 DES

DES 在穷举攻击下相对比较脆弱,因此需要用某种算法替代它,有两种解决方法。其一,设计全新的算法;其二,用 DES 进行多次加密,且使用多个密钥,即多重 DES,这种方法能够保证用于 DES 加密的已有软件和硬件继续使用。

1. 二重 DES

二重 DES 的加密与解密过程如图 3-11 所示。给定明文 P 和两个加密密钥 k_1 和 k_2,采用 DES 对 P 进行加密 E,有

$$\text{密文 } C = E_{k2}(E_{k1}(P))$$

对 C 进行解密 D,有

$$\text{明文 } P = D_{k1}(D_{k2}(C))$$

显然,二重 DES 比单一的 DES 要安全,因为这里使用了两个密钥(均为 56 位),设 $k=k_1 \parallel k_2$,则 k 的密钥空间数量为 2^{112}。

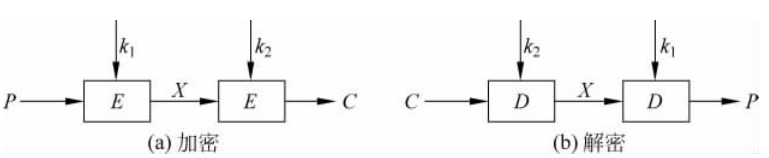

图 3-11 二重 DES

这里要注意,如果采用普通穷举法来攻击二重 DES,其攻击代价并不是 2^{112}。这是因为一个分组为 64 位,明文空间为 2^{64},而密钥空间为 2^{112}。因此对于某个明文 P,可产生密文 C 的密钥平均个数为 $2^{112}/2^{64}=2^{48}$ 个。换句话说,大约有 2^{48} 个不同的密钥,对明文 P 加密后得到的密文相同且都等于 C。这就需要另一个 (P,C) 对,以从这 2^{48} 个不同的密钥中确定真

正的密钥。因此总的计算代价为 $2^{112}+2^{48}$。

但是，如果采用“中间相遇攻击”(meet-in-the-middle attack)攻击二重 DES，则可以大大减少攻击代价。

从图 3-11 可以看出：$X=E_{k1}(P)=D_{k2}(C)$。

若给出一个已知的明-密文对(P,C)，分别用 2^{56}个所有密钥 k_1 对明文 P 进行加密，得到一张密钥对应于密文 X 的表；类似地，对 2^{56}个所有可能的密钥 k_2 对密文 C 进行解密，得到相应的“明文”X，做成一张 X 与 k_2 的对应表。比较两个表中 X 相同的项，就会得到真正使用的密钥对 k_1，k_2。可以看出，计算代价为 $2^{56}+2^{56}=2^{57}$。

2. 三重 DES

对付中间相遇攻击的一个有效的方法是使用三重 DES。

1) 使用两个密钥的三重 DES

Tuchman 建议仅使用两个密钥进行三次加密，并给出了双密钥的 EDE 模式(加密-解密-加密)，如图 3-12 所示。

$$\text{对 } P \text{ 加密：} C = E_{k1}(D_{k2}(E_{k1}(P)))$$

$$\text{对 } C \text{ 解密：} P = D_{k1}(E_{k2}(D_{k1}(C)))$$

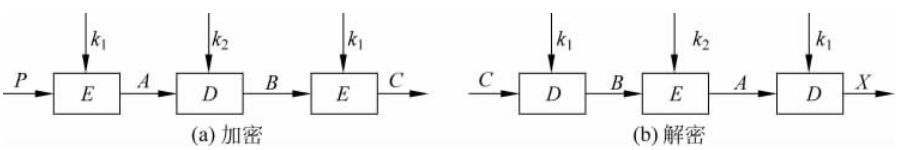

图 3-12　使用两个密钥的三重 DES

其中，E 为加密运算，D 为解密运算。

这种替代 DES 的加密较为流行并且已被采纳用于密钥管理标准(The Key Manager Standards ANSX9.17 和 ISO8732)。到目前为止，还没有人给出对上述三重 DES 的有效攻击方法。对其密钥空间中密钥进行穷举搜索，由于空间太大(为 $2^{112}=5\times10^{33}$)，这实际上是不可行的。

Merkle 和 Hellman 设法创造一个条件，想把中间相遇攻击的方法用于三重 DES，但目前也不太成功。

2) 使用三个密钥的三重 DES

虽然对上述带双密钥的三重 DES 到目前为止还没有好的实际攻击办法，但人们还是放心不下，又建议使用三密钥的三重 DES，此时密钥总长为 168 位。加解密过程如图 3-13 所示。

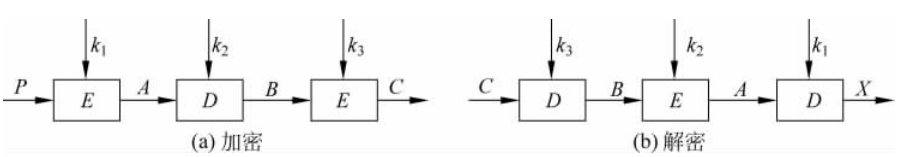

图 3-13　使用三个密钥的三重 DES

加密：$C = E_{k3}(D_{k2}(E_{k1}(P)))$

解密：$P = D_{k1}(E_{k2}(D_{k3}(P)))$

在实际应用中，应根据具体安全需求确定到底是使用两个密钥还是三个密钥的三重DES，因为安全性能越高，则付出的计算代价和密钥管理成本也越高。

3.3 高级加密标准 AES

3.3.1 AES 概述

3DES的根本缺点在于用软件实现该算法的速度比较慢。起初，DES是为20世纪70年代中期的硬件实现设计的，难以用软件有效地实现该算法。在3DES中轮的数量是DES的三倍，所以其速度比DES要慢得多。另一个缺点是DES和3DES的分组长度均为64位，从运算效率和安全性考虑，分组长度应该更大。由于这些原因，DES和3DES不可能长期成为加密算法标准。

1. AES 应满足的需求

1997年1月，美国国家标准局向全世界密码学界发出征集21世纪高级加密标准(advanced encryption standard，AES)算法的公告，要求AES的安全性能不能低于3DES，同时应具有更好的执行性能。除了这些通常的要求之外，NIST特别提出了AES必须是分组长度为128位的对称分组密码，并能支持长度为128位、192位和256位的密钥。对AES候选算法进行评估筛选时主要考虑以下3个方面：

(1) 安全性：指用密码分析方法分析一个算法的代价。评估的重点在于能否防止实际的攻击。AES要求最短密钥长度为128位，故使用目前技术的穷举攻击方式是不可行的。

(2) 成本：NIST希望AES能够广泛地应用于各种实际应用，所以AES必须具有很高的计算效率，以便其能用于各种高速应用。

(3) 算法和执行特征：它包含了各方面需要关注的事项，如算法灵活性、算法适合于多种硬件和软件方式实现、算法的简洁性，以便于分析算法的安全性。

2. AES 算法的产生

1998年4月15日全面征集AES算法的工作结束。1998年8月20日举行了首届AES讨论会，对涉及14个国家的密码学家所提出的候选AES算法进行了评估和测试，初选并公布了15个被选方案，供大家公开讨论。这些算法是

CAST-256	RC6	CRYPTON	DEAL	FROG
DFC	LOKI97	MAGENTA	MARS	HPC
Rijndael	Safer+	Serpent	E2	Twofish

这些算法设计思想新颖，技术水平先进，算法的强度都超过3DES，实现速度快于3DES。

1999年8月9日，美国国家标准局宣布第二轮筛选出的5个候选算法为：

- MARS(IBM，美国)
- RC6 (RSA Lab，美国)
- Rijndael(Joan Daemen 和 Vincent Rijmen，比利时)

- Serpent(Ross Anderson,Eli Biham 和 Lars Knudsen,英国、以色列、挪威)
- Twofish(Bruce Schneier,美国)

2000 年 10 月 2 日,NIST 正式宣布将 Rijndael 作为 AES 标准的加密算法,以下是 NIST 对 Rijndael 算法的最终评估结果。

(1) 一般安全性:没有已知的攻击方法能攻击 Rijndael。它以 S 盒作为非线性组件,表现出了足够的安全性能。

(2) 软件执行:Rijndael 非常利于在包括 8 位和 64 位以及 DSP 在内的各种平台上执行的加密和解密算法。Rijndael 固有的分布执行机制能够充分有效地利用处理器资源,甚至在不能分布执行的模型下仍能达到非常好的软件执行性能。同时,Rijndael 的密钥安装速度非常快。

(3) 受限空间环境:Rijndael 非常适合在受限空间环境中执行加密或解密操作,对 RAM 和 ROM 的要求很低。在这样的环境中,如果既要执行加密操作又要执行解密操作,则其缺陷是它需要更大的 ROM 空间,因为加密和解密的主要步骤是不同的。

(4) 硬件执行:在最后的 5 个候选算法中,Rijndael 在反馈模型下执行的速度最快,在非反馈模型下的执行速度位居第二。但当该算法的密钥长度为 192 位和 256 位时,因执行的轮数增加,其执行速度变慢。当用完全的流水线实现时,该算法需要更多的存储空间,但不影响其执行速度。

(5) 对执行的攻击:Rijndael 所采用的实现方式非常利于防止能量攻击和计时攻击。与其他候选算法相比,Rijndael 算法利用掩码技术使其具有防止这些攻击的能力,并未显著降低该算法的执行性能。同时,它对 RAM 的需求仍在合理的范围内。当使用这些攻击措施时,Rijndael 比其他的候选算法在执行速度上更有优势。

(6) 加密与解密:Rijndael 的加密函数和解密函数不同。尽管在解密算法中密钥安装速度比在加密算法中速度要慢,但 Rijndael 执行加密和解密算法的速度差不多。

(7) 密钥灵活性:Rijndael 支持加密中的快速子密钥计算。Rijndael 要求在加密前用特定密钥产生子密钥,这给 Rijndael 的密钥灵活性稍微增加了一点资源负担。

(8) 其他的多功能性和灵活性:Rijndael 支持分组和密钥长度分别为 128 位、192 位和 256 位的各种组合。原则上,Rijndael 算法结构能通过改变轮数来支持任意长度为 32 的倍数的分组和密钥长度。

(9) 指令级并行执行潜力:Rijndael 对于单个分组加密有很好的并行执行能力。

3.3.2　AES 加密数学基础

AES 是以严谨的数学理论为基础的,下面介绍其数学基础。

1. 群

群是一个代数系统,它由一个非空集合 G 组成,在集合 G 上定义了一个二元运算,其满足:

(1) 封闭性,即对任意的 $a,b\in G$,$a\cdot b\in G$。

(2) 结合律,即对任何的 $a,b,c\in G$,有 $a\cdot b\cdot c=(a\cdot b)\cdot c=a\cdot(b\cdot c)$。

(3) 单位元,即存在一个元素 $1\in G$(称为单位元),对任意元素 $a\in G$ 有 $a\cdot 1=1\cdot a=a$。

(4) 逆元,即对任意 $a\in G$,存在一个元素 $a^{-1}\in G$(称为逆元),使得 $a\cdot a^{-1}=a^{-1}\cdot a=1$。

我们把满足上面性质的代数系统称为群,记做$<G,\cdot>$。

若群$<G,\cdot>$还满足交换律,即对任何$a,b\in G$有:$a\cdot b=b\cdot a$,则称G为交换群(或加法群、阿贝尔群等)。

若集合G中只含有有限多个元素,则我们称$<G,\cdot>$为有限群,此时,把集合G中元素的个数称为有限群G的阶。

群的性质如下:

(1) 群中的单位元是唯一的。

(2) 消去律成立,即对任意的$a,b,c\in G$,如果$ab=ac$,则$b=c$;如果$ba=ca$,则$b=c$。

(3) 群中的每一元素的逆元是唯一的。

2. 有限域

域是一个代数系统,由一个(至少包含两个元素的)非空集合F组成,在集合F上定义有两个二元运算:加法(用符号"+"表示)和乘法(用符号"$\cdot$"表示,有时可以将$a\cdot b$简写成ab),并满足下面条件:

(1) F的元素关于加法"+"成交换群,记其单位元为0(称为域的零元)。

(2) F关于乘法"$\cdot$"成交换群,记其单位元为1(称其为域的单位元)。

(3) 乘法在加法上满足分配律,即对任意的$a,b,c\in F$,有

$$a\cdot(b+c)=ab+ac$$

$$(a+b)\cdot c=ac+bc$$

把满足上面性质的代数系统称为域F,并记为$<F,+,\cdot>$。

若集合F只包含有限个元素,则称这个域F为有限域,也称为伽罗华域或Galois域。有限域中元素的个数称为该有限域的阶。

若有一任意的素数P和正整数$n\in Z^+$,存在P^n阶有限域,这个有限域记为$GF(P^n)$。当$n=1$时,有限域$GF(P)$称为素域。

有限域元素表示方法有很多种,而不同的表示方法可能带来计算上效率的一些差别。

域$<F,+,\cdot>$上x的多项式$a(x)$,定义为$a(x)=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x^1+a_0$(简记为$\sum_{i=0}^{n}a_ix^i$)的表达式,其中$n\in\mathbf{N}$,$a_0,a_1,\cdots,a_n\in F$。若$a_n\neq 0$,称为该多项式的次数,并称$a_n$为首项系数。首项系数为一的多项式称为首1多项式,称0为$-\infty$次多项式。其中0和1分别为F的零元和单位元。F上的x的多项式的全体组成的集合记为$F(x)$。多项式$a(x)$的次数记为$\deg(a(x))$。

域$<F,+,\cdot>$上关于x的多项式的加法"$\oplus$"和乘法"$\otimes$"运算定义如下,设有多项式$a(x)=\sum_{i=0}^{n}a_ix^i$和$b(x)=\sum_{i=0}^{m}b_ix^i$,则有

加法$\oplus$:$a(x)\oplus b(x)=\sum_{i=0}^{M}(a_i+b_i)x^i$

乘法$\otimes$:$a(x)\otimes b(x)=\sum_{i=0}^{n+m}\left(\sum_{j=0}^{i}a_j\cdot b_{i-j}\right)x^i$

其中,$M=\max(m,n)$为n和m中较大者,当$i>n$时,取$a_i=0$;当$i>m$时,取$b_i=0$。

任何有限域都可以用与它同阶的多项式域表示。由于在密码学中,最常用的域一般为素

域(限制 $n=1$)或者阶为 2^n 的有限域。

3. $GF(2^8)$ 域上的多项式表示及运算

根据有限域的知识,一个不可约多项式可以构成一个有限域。在 AES 加密系统中,$GF(2^8)$ 是在不可约多项式 $m(x)=x^8+x^4+x^3+x+1$ 上构造的有限域$<F(x)_{m(x)},+,\cdot>$。

一个字节的 $GF(2^8)$ 元素的二进制展开成的多项式系数为 $b_7b_6b_5b_4b_3b_2b_1b_0$,即

$$b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x^1+b_0x^0$$

例如,$GF(2^8)$上的 37(为十六进制),其二进制为 00110111,对应多项式为

$$x^5+x^4+x^2+x^1+1$$

下面介绍多项式的运算。

1) 加法

加法运算就是以字节为单位进行比特异或运算。如十六进制 37+83=B4,采用二进制表示为 00110111+10000011=10110100;采用多项式表示为

$$(x^5+x^4+x^2+x+1)+(x^7+x+1)=x^7+x^5+x^4+x^2$$

2) 模运算

多项式模运算与实数域上的多项式除法基本相同,但采用以字节为单位的比特异或运算来进行。例如,37 mod (07)=01;采用多项式运算有$(x^5+x^4+x^2+x+1)\bmod(x^2+x+1)=1$,这里采用长除法进行运算,如图 3-14 所示。

$$\begin{array}{r|l} x^2+x+1 & x^3+x \\ \hline & x^5+x^4+x^2+x+1 \\ & x^5+x^4+x^3 \\ \hline & x^3+x^2+x+1 \\ & x^3+x^2+x \\ \hline & 1 \end{array}$$

图 3-14　多项式长除法

3) 乘法运算

例如,57×83=C1,多项式表示为

$$\begin{aligned}&(x^6+x^4+x^2+x+1)(x^7+x+1)\bmod(m(x))\\=&(x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1)\bmod\\&(x^8+x^4+x^3+x+1)=x^7+x^6+1(\text{表示为十六进制为 }C1)\end{aligned}$$

4) x 乘法运算

x 乘以 $b(x)$,则有 $b_7x^8+b_6x^7+b_5x^6+b_4x^5+b_3x^4+b_2x^3+b_1x^2+b_0x$,再将此式模$m(x)$即为所求。这个 x 乘法(x 的十六进制为 02)运算相当于将 $b(x)$表示的字节循环左移一位(如 $b_7=0$),或将其与 11B 比特异或来实现。例如,57×02=AE。

4. $GF(2^8)^4$ 域上的多项式表示及运算

在 AES 加密系统中,$GF(2^8)^4$ 是在不可约多项式 $M(x)=x^4+1$ 上构造的有限域:$<F(x)_{M(x)},\oplus,\otimes>$。在这个有限域上,一个 4 个字节的字(有 32 位)可以看作是 $GF(2^8)^4$ 域上的多项式,每个字对应于一个次数小于 4 的多项式。

两个 $GF(2^8)^4$ 域上的元素相加时,将这两个元素对应多项式系数相加即得到结果,相加时采用比特异或来实现。

两个 $GF(2^8)^4$ 域上的元素相乘时,要将结果对一个特定的多项式取模,以使相乘后的结果还是一个 4 字节的向量。这个特定的多项式为 $M(x)=x^4+1$。例如,若在 $GF(2^8)^4$ 域上有两个多项式为:$a(x)=a_3x^3+a_2x^2+a_1x+a_0$,$b(x)=b_3x^3+b_2x^2+b_1x+b_0$,则有

$$c(x)=a(x)\otimes b(x)=(c_6x^6+c_5x^5+c_4x^4+c_3x^3+c_2x^2+c_1x+c_0)\bmod(M(x))$$

这里,有 $c_6=a_3\cdot b_3$; $c_0=a_0\cdot b_0$; $c_1=a_1\cdot b_0\oplus a_0\cdot b_1$; $c_2=a_2\cdot b_0\oplus a_1\cdot b_1\oplus a_0\cdot b_2$; $c_3=a_3\cdot b_0\oplus a_2\cdot b_1\oplus a_1\cdot b_2\oplus a_0\cdot b_3$; $c_4=a_3\cdot b_1\oplus a_2\cdot b_2\oplus a_1\cdot b_3$; $c_5=a_3\cdot b_2\oplus a_2\cdot b_3$。由于 $x^j \bmod (x^4+1)=x^{j \bmod 4}$,上式化简为 $d(x)=a(x)\otimes b(x)=d_3x^3+d_2x^2+d_1x+d_0$,其中

$$d_0=a_0\cdot b_0\oplus a_3\cdot b_1\oplus a_2\cdot b_2\oplus a_1\cdot b_3;\ d_1=a_1\cdot b_0\oplus a_0\cdot b_1\oplus a_3\cdot b_2\oplus a_2\cdot b_3$$

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

这个变换用矩阵表示为

$$a(x)\otimes b(x)=\begin{bmatrix}d_0\\d_1\\d_2\\d_3\end{bmatrix}=\begin{bmatrix}a_0&a_3&a_2&a_1\\a_1&a_0&a_3&a_2\\a_2&a_1&a_0&a_3\\a_3&a_2&a_1&a_0\end{bmatrix}\begin{bmatrix}b_0\\b_1\\b_2\\b_3\end{bmatrix}$$

在 AES 加密系统中,采用一个具有逆元的固定多项式

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

进行这样的乘法,从而保证了乘法的可逆性。其逆元为

$$a^{-1}(x)=\{0b\}x^3+\{0d\}x^2+\{09\}x+\{0e\}$$

使得 $a(x)\otimes a^{-1}(x)=\{01\}$。

x 乘法运算为

$$x^i\otimes b(x)=b_2x^3+b_1x^2+b_0x+b_3$$

该式相当于将 $b(x)$所表示的字左循环移 i 位。

3.3.3 AES 加密原理

AES 采用分组密码体制,分组长度可以有 3 种选择,即 128、192 和 256 位。如果用 N_b 表示分组长度,单位为 32 位字,那么 3 种分组长度可表示为 $N_b=4$、6、8。

同样,加密密钥也可以有 3 种选择,即 128、192 和 256 位。如果用 N_k 来表示密钥长度,单位为 32 位字,那么 3 种密钥长度可表示为 $N_k=4$、6、8。

与其他分组密码一样,*AES* 也是通过若干轮连续的迭代来对明文进行加密的。根据分组长度和密钥长度的不同,具体迭代的轮数也不一样,对应上面的 3 种分组长度和密钥长度,迭代次数 N_r 见表 3-3。

表 3-3 加密轮数 N_r 的取值

N_r 的取值	$N_b=4$	$N_b=6$	$N_b=8$
$N_k=4$	10	12	14
$N_k=6$	12	12	14
$N_k=8$	14	14	14

在每一轮迭代中,包括 4 步变换,分别是字节代换运算(ByteSub())、行移位(ShiftRows())、列混合(MixColumns())以及轮密钥加变换(AddRoundKey()),其作用就是通过重复简单的非线性变换、混合函数变换,将字节代换运算产生的非线性扩散,达到充分的混合,使加密后的分组信息统计特性分布更均匀,在每轮迭代中引入不同的密钥,这样便以最简单的运算代价得到最好的加密效果,实现加密的有效性。

以一个 128 位的分组采用 128 位的密钥为例,其整个加密过程如图 3-15 所示。

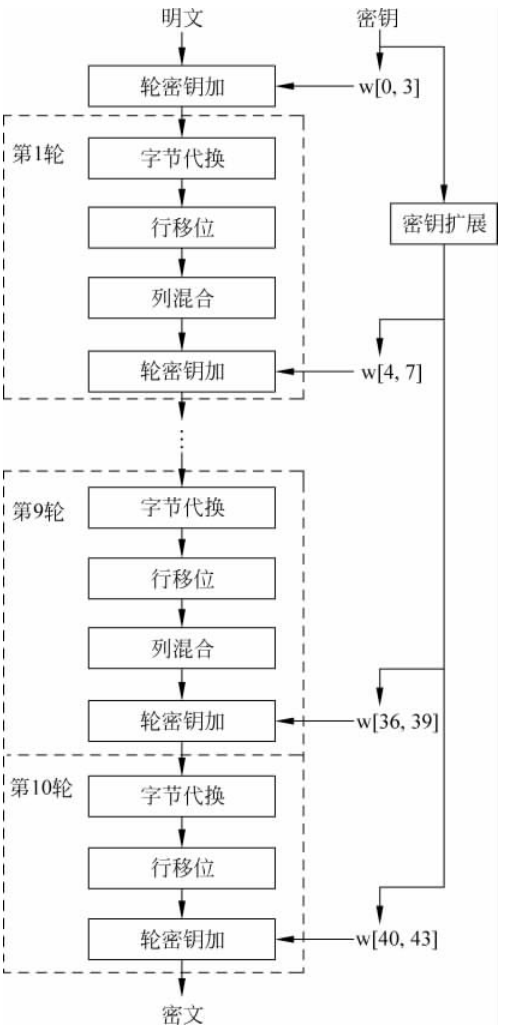

图 3-15　AES 分组加密

其中，最后一轮与其他轮有所不同，没有“列混合”步骤。

下面以一个 128 位的分组采用 128 位的密钥加密为例，分别介绍轮变换中的 4 个步骤。

在进行变换之前，首先将 128 位的明文分组以字节为单位写入一个矩阵中，这个矩阵称为“状态”：

S_0	S_4	S_8	S_{12}
S_1	S_5	S_9	S_{13}
S_2	S_6	S_{10}	S_{14}
S_3	S_7	S_{11}	S_{15}

状态矩阵有 4 行，列数不定(128 位为 4 列，192 位为 6 列，256 位为 8 列)，每个单元格存放一个字节，每一列就是一个 32 位字。以后所有的变换都是基于这个矩阵进行的，到此，准备工作已经完成。

1. 字节代换运算(SubByte())

字节代换运算SubByte()为可逆的非线性字节代换操作,操作元素为状态中的单个字节(即每个格子),对字节的操作遵循一个代换表如表3-4,即S盒。S盒由有限域$GF(2^8)$上的乘法取逆和仿射变换两步构成。

若$b(x)\in GF(2^8)$,其逆为$b^{-1}(x)\in GF(2^8)$,则有$b(x)\cdot b^{-1}(x)=1 \bmod (m(x))$,这里$m(x)=x^8+x^4+x^3+x+1$。如{$f6$}(十六进制)表示为$x^7+x^6+x^5+x^4+x^2+x$,{03}表示为$x+1$,显然有{f6}·{03}=1 mod ($m(x)$)。实际求逆采用设未知数解方程组的方法来实现。

有限域$GF(2^8)$上的仿射变换也对字节进行操作,设输入字节为$\{b_7b_6b_5b_4b_3b_2b_1b_0\}$,经过仿射变换后的输出字节为$\{b'_7b'_6b'_5b'_4b'_3b'_2b'_1b'_0\}$,则用矩阵来表示仿射变换表达式为:

$$\begin{bmatrix}b'_0\\b'_1\\b'_2\\b'_3\\b'_4\\b'_5\\b'_6\\b'_7\end{bmatrix}=\begin{bmatrix}1&0&0&0&1&1&1&1\\1&1&0&0&0&1&1&1\\1&1&1&0&0&0&1&1\\1&1&1&1&0&0&0&1\\1&1&1&1&1&0&0&0\\0&1&1&1&1&1&0&0\\0&0&1&1&1&1&1&0\\0&0&0&1&1&1&1&1\end{bmatrix}\begin{bmatrix}b_0\\b_1\\b_2\\b_3\\b_4\\b_5\\b_6\\b_7\end{bmatrix}+\begin{bmatrix}1\\1\\0\\0\\0\\1\\1\\0\end{bmatrix}$$

例如,求{f6}的SubByte()变换。先求其逆为{03},即为{00000011},将其带入上式可计算$\{b'_7b'_6b'_5b'_4b'_3b'_2b'_1b'_0\}$={01000010},即为{42}。我们同样可以用S盒替换表实现SubByte()变换,先取数{f6}的高低位对应xy值为,x=f,y=6;再查表3-4得到第x行第y列(即f行6列)的数值为{42}。这和分步计算的结果一样,可以用分步变换校对S盒的正确性。

表3-4 AES的S盒替换表-字节xy的代替表(用十六进制表示)

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	ed	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
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(ShiftRows())

行移位变换 ShiftRows()是线性变换,其目的就是使密码信息达到充分的混乱,提高非线性度。行移位变换在状态矩阵 State 的每行间进行,对每行实施左循环移动,移动字节数根据行数和密钥长度来确定,第零行不发生偏移,第一行循环左移 C_1 字节,第二行移 C_2 字节,第三行移 C_3 字节。表 3-5 为移动字节数。

表 3-5　行移位变换字节数

N_k	C_1	C_2	C_3
4	1	2	3
6	1	2	3
8	1	3	4

当 $N_b=4$ 时 $C_1=1, C_2=2, C_3=3$,行位移变换示意如图 3-16。

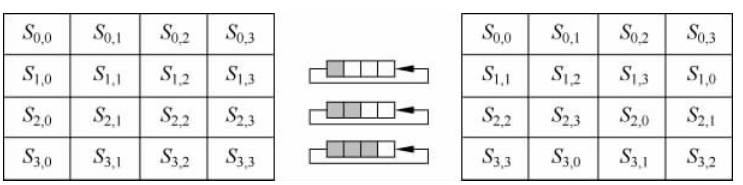

图 3-16　$N_k=4$ 时行位移变换示意

3. 列混合变换(MixColumns())

列混合是对状态(State)列的一种线性变换,状态每列有 4 字节,即一个字($S_{0,i}, S_{1,i}, S_{2,i}, S_{3,i}$),其中 $0 \leqslant i \leqslant 3$,$S_{0,i}, S_{1,i}, S_{2,i}, S_{3,i}$ 都是字节。一个字在有限域 $GF(2^8)^4$ 上可表示成下面的四项多项式。列变换就是从状态中取出一列,表示成多项式的形式 $S_i(x)=S_{3,i}x^3+S_{2,i}x^2+S_{1,i}x+S_{0,i}$,再用它乘以一个固定的多项式 $a(x)=\{03\}x^3+\{01\}x^2+\{01\}x+\{02\}$,然后将所得结果进行取模 $M(x)=x^4+1$ 运算。在 AES 加密算法中,由于 x^4+1 不是有限域 $GF(2^8)^4$ 上的不可约多项式,因此,用 $S(x)$ 乘以任意一个四项式的运算不一定是可逆的,于是,将 $a(x)$设置成一个固定的值,以确保可进行求逆运算,$a(x)$的乘法逆多项式为:$a^{-1}(x)=\{0b\}x^3+\{0d\}x^2+\{09\}x+\{0e\}$。

列混合的算术表达式为 $s'(x)=a(x)\otimes s(x)$。这个表达式用矩阵可表示为:

$$\begin{bmatrix} s'_{0i} \\ s'_{1i} \\ s'_{2i} \\ s'_{3i} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0i} \\ s_{1i} \\ s_{2i} \\ s_{3i} \end{bmatrix}$$

当 $N_b=4$ 时,有

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

4. 轮密钥加变换(AddRoundKey())

从图 3-15 中可以看到,AES 的每一轮转换中都需要用到一个轮密钥。每个轮密钥都是 128 位。当分组长度和密钥长度都为 128 位时,AES 的加密算法共迭代 10 轮,需 11 个轮密钥,即 44 个 32 位字($w_0, w_1, \cdots, w_{43}$)。这些轮密钥是通过对初始密钥的扩展运算得来的。

首先,将 128 位的初始密钥写入密钥矩阵中:

k_0	k_4	k_8	k_{12}
k_1	k_5	k_9	k_{13}
k_2	k_6	k_{10}	k_{14}
k_3	k_7	k_{11}	k_{15}

那么,$w_0 = k_0k_1k_2k_3$,$w_1 = k_4k_5k_6k_7$,$w_2 = k_8k_9k_{10}k_{11}$,$w_3 = k_{12}k_{13}k_{14}k_{15}$。之后的每个轮密钥 w_i 要根据 w_{i-1} 和 w_{i-4} 来计算,如图 3-17 所示。

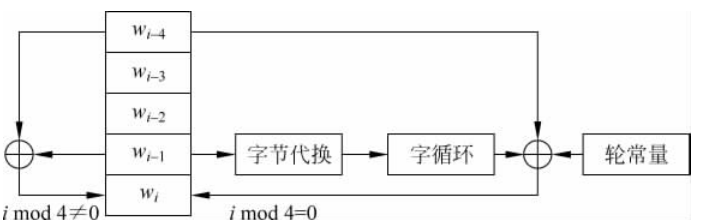

图 3-17 密钥扩展

当 i 不是 4 的倍数时,w_i 为 w_{i-4} 和 w_{i-1} 的异或。

当 i 是 4 的倍数时,则采用更复杂的计算方法:首先对 w_{i-1} 进行字循环,即将其 4 个字节循环左移一个字节。然后对结果进行字节代换,即根据 S 盒对 w_{i-1} 的每个字节进行字节转换。最后再与轮常量 RC 进行异或运算。轮常量也是一个 32 位字,但其右边 3 个字节总为 0。计算每个轮密钥用到的轮常量也不相同,当分组长度和密钥长度都为 128 位时,所用到的 10 个轮常量(最左边的一个字节)如表 3-6 所示:

表 3-6 轮常量

j	1	2	3	4	5	6	7	8	9	10
RC_j	01	02	04	08	10	20	40	80	1B	36

在轮密钥加变换中,轮密钥的各字节与状态中的各对应字节分别异或,实现状态和密钥的混合。

密钥扩展过程的伪 C 代码如下:

```
KeyExpansion ( byte key[4 * Nk]), word w[Nb * (Nr + 1)], Nk)// 用来产生轮密钥 w 数组,w 以字为单位
{                                        // 原始密码 key 数组以字节为单位输入到函数中
    word temp;                           // 定义一个用来存放临时字变量的字
    i = 0;
    for ( i = 0; i < Nk; i++ )           // 原始密码 Key 数组放到前 w[0] - w[Nk - 1],用来密钥扩展
        w[i] = Word (key[4 * i], key[4 * i + 1], key[4 * i + 2], key[4 * i + 3]);
    for ( i = Nk; i < Nb * (Nr + 1); i++ )
```

```
    {                                   //扩展密钥,分 Nk = 4 和 6 或 8 两种情况
        temp = w[i-1];
        if (i % Nk == 0)                // % 表示整数的取模运算
          temp = SubWord ( RotWord ( temp ) ) ^ Rcon[ i / Nk];  // ^ 表示两个数的按位异或
        else if ( Nk == 8 && ( i % Nk == 4) )                   // && 表示两个数的逻辑与运算
            temp = SubWord ( temp );
        w[i] = w[i-Nk] ^ temp;          // 每次都要进行异或运算
    }
}
```

上面的函数 Word 用来将一个字的 4 字节，按由高到低表示成一个字（高位在前）。RotWord 函数将输入的一个字 $a_0a_1a_2a_3$（4 字节），循环左移一个字节后，重新组成一个字 $a_1a_2a_3a_0$ 输出；SubWord 函数对输入的字进行 SubBytes()变换（S 盒替换）后返回变换后的字。字数组 Rcon[i]由下面的方法得到：Rcon[i]＝word (RC[i], {00}, {00}, {00})，其中 RC[i]为 x^{i-1}在 $GF(2^8)$域上所代表的字节数值。如 RC[1]＝01（为 x^0 代表的字节数），RC[i]＝{02}・RC[$i-1$]，这里“・”为在 $GF(2^8)$域上的乘法。若 $i=36$，$N_k=4$，则 $i/N_k=9$，计算 RC[9]为 $x^8 \bmod (m(x))=x^8 \bmod (x^8+x^4+x^3+x+1)=x^4+x^3+x+1$ 所代表的字节{1b}，即有 RC[8]＝{1b}。

存在两个特殊情况：

（1）对于在 N_k 整数倍处的密钥，在异或之前还将对这些字进行相应的变换，具体见上面的实现过程。

（2）密钥长度为 256 位（$N_k=8$）时的密钥扩展方案与密钥长度分别为 128 位（$N_k=4$）及 192 比特（$N_k=6$）时的密钥扩展方案稍有不同。当 $N_k=8$ 时，如果 $i-4$ 是 N_k 的整数倍，则在进行异或之前，需要先对进行 Subword()变换。

加密原理的伪 C 语言代码实现为：

```
Cipher (byte in[4 * Nb], byte out[4 * Nb], word w[Nb * (Nr + 1)] )
{                                  // in、out 为明文分组输入和密文分组输出数组,w 为轮密钥数组
    byte State[4, Nb];                  // 定义一个状态矩阵 4 × Nb
    State = in;                         // 装入明文输入矩阵到状态矩阵
    AddRoundKey (State, w[0] );         // 明文信息和密钥混合,使用 w[0]开始的 Nb 个密钥
    for( int r = 1; r < Nr; r ++ )      // 实现 1 到 Nr - 1 轮加密变换
    {
       SubBytes (State);
       ShiftRows (State);
       MixColumns (State);
       AddRoundKey (State, w[r * Nb]); // 使用从 w[r * Nb]开始的 Nb 个密钥
    }
    SubBytes (State);                   // 从这里开始最后一轮加密变换
    ShiftRows (State);
    AddRoundKey (State, w[Nr * Nb]);    // 结束轮加密变换,使用 w[Nr * Nb]开始的 Nb 个密钥
    Out = State;                        // 将最终的变化结果 State 矩阵,放到密文 out 矩阵中
}                                       // 结束加密变换,得到密文 out 矩阵
```

3.3.4　AES 的解密变换

解密为加密的逆变换，其伪 C 代码如下：

```
InvCipher (byte in[4 * Nb],  byte out[4 * Nb],  word w[Nb * (Nr + 1)] )
{                                   // in、out 为密文和明文分组数组,w 为轮密钥数组
byte State[4,Nb];                   // 定义一个状态矩阵 4 × Nb
State = in;                         // 装入密文输入矩阵到状态矩阵
AddRoundKey (State,w[0] );          // 使用 w[Nr * Nb]开始的 Nb 个密钥来解密 Nr 轮加密变换
InvShiftRows (State);               // 实现 ShiftRows 的逆变换
InvSubBytes (State);                // 实现 SubBytes 的逆变换
for( int r = Nr - 1; r > 0; r -- )  // 实现 Nr - 1 到 1 轮的解密变换
{
     AddRoundKey (State,w[r * Nb]); // 使用从 w[r * Nb]开始的 Nb 个密钥
     InvMixColumns (State);
     InvShiftRows (State);
     InvSubBytes (State);
}
AddRoundKey (State,w[Nr * Nb]);     // 轮密钥混合变换得到明文,使用 w[0]开始的 Nb 个密钥
Out = State;                        // 将最终的变化结果 State 矩阵,放到密文 out 矩阵中
}                                   // 结束加密变换,得到明文 out 矩阵
```

显然,解密变换为加密变换的逆变换。加密和解密密钥扩展方法一样,相应的逆变换如下。

1. 逆字节代换运算(InvSubByte())

逆字节代换运算 InvSubByte()也遵循一个代换表如表 3-7 所示,即逆 S 盒代换表。

表 3-7 AES 的逆 *S* 盒替换表-字节 *xy* 的代替表(用十六进制表示)

x \ *y*	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

这个逆 S 盒也由有限域 $GF(2^8)$ 上的仿射变换和乘法取逆两步构成。$GF(2^8)$ 上的基本运算见上节。有限域 $GF(2^8)$ 上的仿射变换也对字节进行操作，设输入字节为 $\{b_7b_6b_5b_4b_3b_2b_1b_0\}$，经过仿射变换后的输出字节为 $\{b'_7\ b'_6\ b'_5\ b'_4\ b'_3\ b'_2\ b'_1\ b'_0\}$，再对输出字节求其在 $GF(2^8)$ 域上的逆元素。这里仿射变换用矩阵来表示其表达式为

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

2. 逆行移位变换(InvShiftRows())

逆行移位变换 InvShiftRows()是行移位变换 ShiftRows()的逆变换。若加密时行移位变换中 State 矩阵的行位移如下：第零行不发生偏移，第一行循环左移 C_1 字节，第二行移 C_2 字节，第二行移 C_3 字节；则对应逆行位移变换为第零行不发生偏移，第一行循环左移 N_b-C_1 字节，第二行移 N_b-C_2 字节，第二行移 N_b-C_3 字节。

3. 逆列混合变换(InvMixColumns())

逆列混合变换 InvMixColumns()是列混合变换 MixColumns()的逆变换。变换为：$s'(x)=a^{-1}(x)\otimes s(x)$，这里 $a^{-1}(x)=\{0b\}x^3+\{0d\}x^2+\{09\}x+\{0e\}$，$\otimes$ 运算方法见上节，这个表达式用矩阵可表示为

$$\begin{bmatrix} s'_{0i} \\ s'_{1i} \\ s'_{2i} \\ s'_{3i} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0i} \\ s_{1i} \\ s_{2i} \\ s_{3i} \end{bmatrix}$$

当 $N_b=4$ 时，有

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

3.3.5　AES 加密算法性能分析

AES 加密算法和 DES 加密算法有很多的共同点，都要经过一个 S 盒、替换和移位等操作，但是，AES 加密算法最终取代 DES，主要有以下几点原因。

(1) 在 DES 中，密钥长度为 64 位，有效密钥空间仅为 2^{56}，这样小的密钥空间对像穷举

密钥这样的攻击来说显得太小了。相比之下AES有更长的密钥,密钥长度有128位、192位和256位三种情况,明显提高了加密的安全性,同时,对不同机密级别的信息,可采用不同长度的密钥,执行灵活性较高。

(2) 在DES中,还存在一些弱密钥和半弱密钥。DES的16次加密迭代中使用不同的子密钥是确保DES强度的一种重要措施,但实际上存在一些密钥,由它产生的16个子密钥会部分地重合甚至完全一致。这样的密钥为弱密钥或半弱密钥,它们的存在无疑会降低DES的强度。所以,在DES中,对初始密钥的选取存在一定的限制。在AES中,由于密钥扩展函数的特点,所产生的轮密钥随机性很强,对初始密钥的选取也没有特别的限制。

(3) DES加密算法存在互补对称性。若$C=\mathrm{DES}(M,K)$,则有$\overline{C}=\mathrm{DES}(\overline{M},\overline{K})$,其中$\overline{C}$、$\overline{M}$、$\overline{K}$表示$C$、$M$、$K$的非。互补对称性会使DES在选择明文攻击下所需的工作量减半。产生互补对称性的原因在于DES中两次$\oplus$运算。而AES的均衡对称结构既可以提高执行的灵活度,又可防止差分分析方法的攻击。

(4) AES算法的实现更简单。AES算法的迭代次数最多为14次,S盒只有一个,较之DES的16次迭代和8个S盒要简单得多。此外,AES算法有很强的扩散性能,能把非线性部件产生的非线性很快地扩散开,形成的密码有很高的随机性。可有效地防止差分分析和线性分析的攻击。

(5) AES算法在所有的平台上都表现良好,其操作比较容易抵御对物理层实现的某些攻击,能很好地适应现代及将来处理器的发展,有支持并行处理的能力。因此,无论是从安全还是实现的难易度上考虑,AES采用Rijndael作为其加密算法应该是明智的选择。

3.4 序列密码

序列密码也称为流密码,采用对称密码体制,它是密码学的一个重要组成部分。

3.4.1 序列密码的原理

“一次一密”密码在理论上是不可攻破的。序列密码则由“一次一密”密码启发而来。

“一次一密”密码使用的密钥是和明文一样长的随机序列,密钥越长越安全,但长密钥的存储、分配都很困难。序列密码采用密钥生成器,从原始密钥生成一系列密钥流用来加密信息,每个明文可以选用不同的密钥加密。如果序列密码所使用的是真正随机产生的、与消息流长度相同的二进制序列,此时的序列密码就是“一次一密”的密码体制,这种密码的破解难度很大。

序列密码的关键就是产生密钥流的算法,该算法必须能够产生可变长的、随机的、不可预测的密钥流。另外,保持通信双方的精确同步是序列密码实际应用中的关键技术。由于通信双方必须能够产生相同的密钥流,所以这种密钥流不可能是真随机序列,只能是伪随机流。

序列密码的设计核心在于密钥发生器的设计,序列密码的安全强度取决于密钥发生器产生的密钥流的周期、复杂度、随机(伪随机)特性等,安全的密钥流生成器必然会使用非线性变换,本节介绍的例子使用的是线性变换。二进制序列密码体制的诱人之处在于它可以

用硬件实现，线性变换一般采用线性反馈移位寄存器（LFSR）实现，而非线性变换目的是增强密钥的复杂度和随机特性，这些变换一般以线性移位寄存器序列为基序列，经过不规则采样、函数变换等，得到实用安全的密钥流。

一个典型的序列密码每次加密一个字节的明文。当然，序列密码也可以被设计为每次操作一位或一个字节的单元。典型的序列密码的结构如图 3-18 所示。

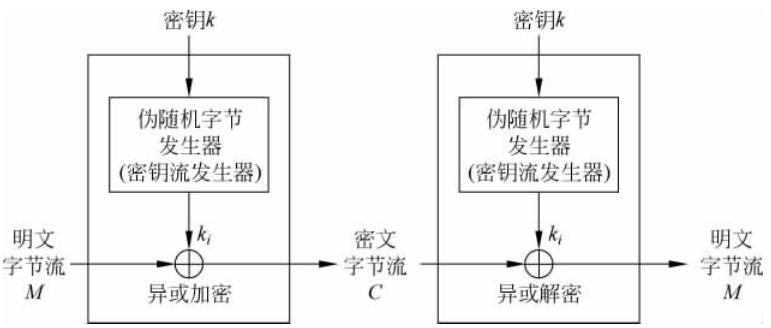

图 3-18　流密码的结构

在该结构中，密钥输入到一个伪随机数发生器，该伪随机数发生器产生一串随机的 8 比特数。这些按先后次序产生的随机数排成序列，即为一个伪随机流（在不知道输入密钥的情况下不可预知的流），也就是用于对每个明文字节进行加密的密钥流。用于产生密钥流的初始密钥也称为“种子”。

每产生一个字节的密钥，这个密钥就与同一时刻的当前明文字节进行异或运算产生密文字节，并依此类推产生密文流。

例如，如果发生器产生的密钥为 01101100，而当前的明文字节为 11001100，则得出的密文字节为

$$\begin{array}{rl} 11001100 & \text{明文} \\ \oplus\ 01101100 & \text{密钥流} \\ \hline 10100000 & \text{密文} \end{array}$$

解密时则需要使用相同的伪随机序列

$$\begin{array}{rl} 10100000 & \text{密文} \\ \oplus\ 01101100 & \text{密钥流} \\ \hline 11001100 & \text{明文} \end{array}$$

可以看出，序列密码与“一次一密”的区别就在于，“一次一密”使用的密钥是真正的随机数流，而序列密码使用的密钥是伪随机数流。

设计序列密码需要考虑以下几个主要因素：

（1）密钥流的周期要长。伪随机数发生器产生的并非完全随机的序列，它是一个产生确定的比特流的函数，该比特流最终将产生重复。重复的周期越长，相当于密钥越长，密码分析也就越困难。

（2）密钥流应尽可能地接近于一个真正的随机数流的特征。例如，1 和 0 的个数应大致相同。密钥流越随机，加密所得的密文也越随机，分析就越困难。

（3）伪随机数发生器的输出取决于输入的密钥的值。

通过设计合适的伪随机发生器,序列密码可以提供和相应密钥长度分组密码相当的安全性。而且相对于分组密码来说,往往速度更快且需要编写的代码更少。序列密码目前的理论已经比较成熟,工程实现也比较容易,加密效率高,在许多重要领域得到应用。

3.4.2 RC4

RC4是RSA三人组中的Ron Rivest为RSA公司在1987年设计的一种序列密码。它是一种可变密钥长度、面向字节操作的序列密码。该算法以随机置换为基础。对该算法的分析显示,该密码的周期大于10^{100}。每输出一字节的结果仅需要8～16条机器操作指令。RC4可能是应用最广泛的序列密码。它被用于SSL/TLS(安全套接字/传输层安全协议)标准,该标准是为网络浏览器和服务器间通信而制定的。它也用于IEEE802.1无线局域网中的WEP协议。RC4起初是用于保护商业机密的。但是在1994年9月,它的算法被发布在互联网上,也就不再有什么商业机密了。

1. RC4算法实现

RC4算法非常简单:用从1～256个字节的可变长度密钥初始化一个256字节的状态矢量$\boldsymbol{S}$,$\boldsymbol{S}$的元素记为$\boldsymbol{S}[0],\boldsymbol{S}[1],\cdots,\boldsymbol{S}[255]$,从始至终置换后的$\boldsymbol{S}$包含从0～255的所有8比特数。对于加密和解密中应用的密钥流的产生,密钥流中的每个密钥k是由$\boldsymbol{S}$中255个元素按一定的方式选出一个元素而生成。每生成一个密钥k,$\boldsymbol{S}$中的元素就被重新置换一次。

1) 初始化

开始时,$\boldsymbol{S}$中元素的值被置为按升序从0～255,即$\boldsymbol{S}[0]=0,\boldsymbol{S}[1]=1,\cdots,\boldsymbol{S}[255]=255$,同时建立一个临时矢量$\boldsymbol{T}$。如果种子密钥$K$的长度为256字节,则将$K$赋给$\boldsymbol{T}$。否则,若$K$的长度为keylen字节,则将$K$的值赋给$\boldsymbol{T}$的前keylen个元素,并循环重复用$K$的值赋给$\boldsymbol{T}$剩下的元素,直到$\boldsymbol{T}$的所有元素都被赋值。其过程如下:

```
/* 初始化 */
For i = 0 to 255 do
      S[i] = i;
      T[i] = K[i mod keylen];              //临时矢量 T(256 字节)
```

然后用$\boldsymbol{T}$产生的$\boldsymbol{S}$的初始置换。从$\boldsymbol{S}[0]$到$\boldsymbol{S}[255]$,对每个$\boldsymbol{S}[i]$,根据$\boldsymbol{T}[i]$确定的方案,将$\boldsymbol{S}[i]$置换为$\boldsymbol{S}$中的另一字节:

```
/* 置换 */
j = 0;
For i = 0 to 255 do
      j = (j + S[i] + T[i]) mod 256;
Swap(S[i],S[j]);                          //S 仍然包含所有值为 0～255 的元素
```

2) 密钥流的生成

矢量$\boldsymbol{S}$一旦完成初始化,种子密钥就不再被使用。密钥流的生成是从$\boldsymbol{S}[0]$～$\boldsymbol{S}[255]$,对每个$\boldsymbol{S}[i]$,根据当前$\boldsymbol{S}$的值,将$\boldsymbol{S}[i]$与$\boldsymbol{S}$中的另一个字节置换。当$\boldsymbol{S}[255]$完成置换后,操

作继续重复，从 $S[0]$ 开始：

```
/* 密钥流的生成 */
i = 0,j = 0;
While(true)
      i = (i + 1) mod 256;
      j = (j  +  S[i]) mod 256;
      swap(S[i],S[j]);
      t = (S[i] + S[j]) mod 256;
      k = S[t];
```

加密中，将 k 的值与下一明文字节异或；解密中，将 k 的值与下一密文字节异或。

2. RC4 的安全性

RC4 算法的优点是简单高效，特别适合软件实现。

其安全性当然也是随着密钥长度的增加而增强。美国政府对 RC4 算法软件的出口做出了明确限制，其密钥长度不能超过 40 位。1995 年有人在 Internet 网上公布了一条用 40 位密钥的 RC4 加密的密文，并提出了破译挑战。法国的一个研究小组通过 Internet 网，用了 120 台计算机和工作站，结果只用了 8 天时间便求出了密钥。在此之后的第二次破译挑战中只用了 31.8 小时便破译成功。

当然，对于密钥足够长（如 128 位）的 RC4，则安全性就非常高了。关于分析 RC4 的攻击方法有许多公开发表的文献，但没有哪种方法对密钥足够长的 RC4 有效。

3.5　其他对称加密算法

在不同的安全系统中，还存在着其他对称加密算法，其中包括：

（1）IDEA。国际数据加密算法（international data encryption algorithm，IDEA）是在由旅居瑞士的华人来学嘉和他的导师 J. L. Massey 共同开发的。IDEA 使用 128 位密钥，明文和密文分组长度为 64 位。已被用在多种商业产品中。

（2）CLIPPER 密码。采用 SKIPJACK 算法，明文和密文分组长度为 64 位，密钥长度为 80 位。

（3）Blowfish。Blowfish 允许使用最长为 448 位的不同长度的密钥，并针对在 32 位处理器上的执行进行了优化。

（4）Twofish。Twofish 使用 128 位分组，可以使用 128、192 或 256 位密钥。

（5）CAST-128。CAST-128 使用 128 位密钥。它在更新版本的 PGP 中使用。

（6）GOST。GOST 是为了回应 DES 而开发的俄罗斯标准，它使用 256 位密钥。

所有这些算法都可能出现在安全性产品中，对于一般性的应用，所有这些算法都将是足够复杂的。

3.6 对称密码的工作模式

加密算法是一种确定性函数,即在使用特定的输入值调用确定性函数的任何时候,它们总是返回相同的结果。

也就是说,当调用加密算法 E 对两个明文 P_1 和 P_2 进行加密时,如果 $P_1 = P_2$ 且 $k_1 = k_2$,那么加密后得到的密文也相等,即

$$E(k_1, P_1) = C_1 = C_2 = E(k_2, P_2)$$

因此,当用同一密钥 k 来加密多个消息时,如果加密后有两个密文完全相同,则可以推断出它们所对应的明文也完全相同。即使密钥 k 只使用一次,即只加密一个消息,任意两个相同的密文分组,都意味着它们所对应的明文分组也相同。

然而,根据香农对安全加密的定义,必须保证攻击者不能根据密文推断出关于明文的任何信息。因此,确定性的加密算法并不能直接用于加密信息,必须在加密算法的基础上设计具有随机性的工作模式,才能满足对信息加密的现实要求。

本节介绍两种最常用的对称密码工作模式。

3.6.1 密文分组链接模式

密文分组链接模式(Cipher Block Chaining Mode)简称为 CBC 模式。在该模式下,当前的明文分组在被代入加密算法之前,先要和上一分组的密文进行异或运算,如图 3-19 所示。

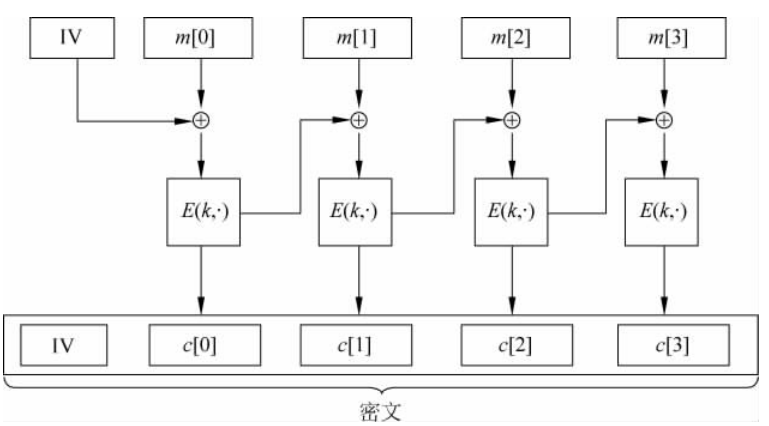

图 3-19 密文分组链接模式的加密过程

每加密一个明文消息 M,首先随机生成一个初始向量 IV。第一个明文分组 $m[0]$ 与 IV 异或后再代入加密算法,之后的每个明文分组都要先和前一个分组的密文进行异或,然后再进行加密,即

$$c[0] = E(k, m[0] \oplus \mathrm{IV}), \qquad c[i] = E(k, m[i] \oplus c[i-1])$$

需要注意的是,IV 和所有密文分组一起构成最后的密文段,发送到接收端。接收端在

没有 IV 的情况下将无法对密文进行解密。密文分组链接模式的解密过程如图 3-20 所示。

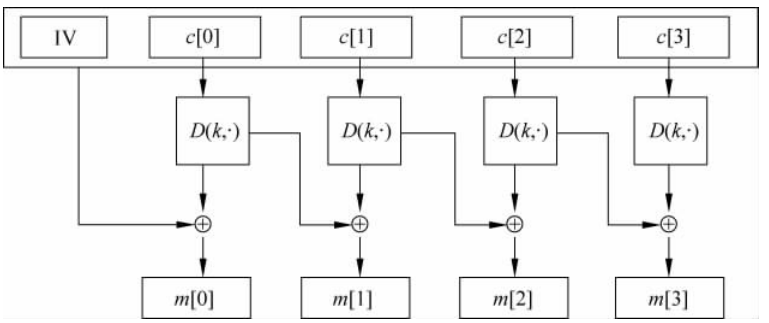

图 3-20　密文分组链接模式的解密过程

在解密时，首先从密文段中提取出 IV，第一个密文分组在解密后与 IV 异或，得到第一个明文分组；其他密文分组则在解密后与前一个分组的解密输出进行异或，得到相应的明文分组，即：

$$m[0] = D(k, c[0]) \oplus \mathrm{IV}, \qquad m[i] = D(k, c[i]) \oplus D(k, c[i-1])$$

可以看出，密文分组链接模式可以实现以下两个随机性：

(1) 由于每次加密一个消息时所用的 IV 在很高的概率上不同，因此用同一密钥加密两个相同的消息，产生的密文不同。

(2) 由于每个密文分组在输入到加密算法之前要与前一个分组的密文进行异或，因此两个相同的明文分组所产生的密文分组不同。

3.6.2　随机计数器模式

随机计数器模式(Randomize Counter Mode)简称为随机 CTR 模式。该模式采用了流密码的设计思想，利用伪随机函数 F 来产生伪随机密钥流，并与明文进行异或计算来生成密文。

随机计数器模式的加密过程如图 3-21 所示。

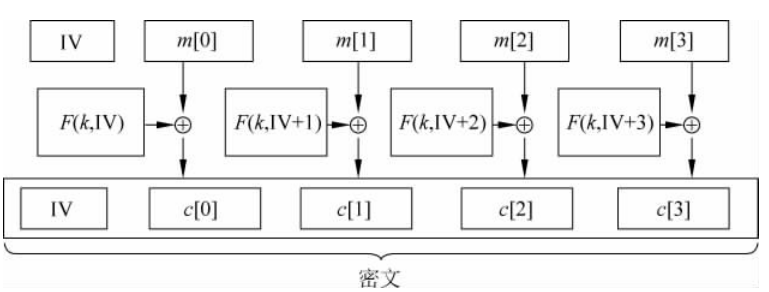

图 3-21　随机计数器模式的加密过程

在随机计数器模式中，每加密一个明文消息 M，首先随机生成一个初始向量 IV。作为一个计数器，每加密一个分组，IV 都要加 1。每个明文分组与 F 函数的相应输出进行异或，从而得到相应的密文分组。计数器可以使得每个明文分组所异或的对象在很高的概率上都

不同。这样,即使两个明文分组完全相同,所得到的密文分组也不相同。

显然,与CBC模式一样,IV也必须和所有密文分组一起发送到接收端。接收端在没有IV的情况下将无法对密文进行解密。

伪随机函数F是一个重要的构成部分。它可以是一个加密函数,但并不要求可逆,因为在解密时只需重新计算F函数,而不需要用到其逆函数。随机计数器模式的解密过程如图3-22所示。

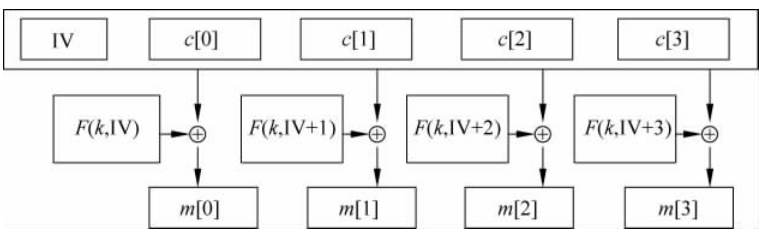

图3-22 随机计数器模式的解密过程

在解密时,首先从密文段中提取出IV,然后作为计数器,代入伪随机函数,输出密钥流并与相应的密文分组异或,从而得到所有明文分组。

显然,随机计数器模式也可以实现前文描述的两个随机性。

相对于CBC模式,随机CTR模式具有一个明显的优点。CBC模式只能进行串行计算,即必须在处理完当前分组后才能处理后续分组。而随机CTR模式则可以并行计算,多个分组的加密或解密可以同时进行,当采用双核或多核计算机时,加解密速度可成倍提高。