第3章 分组密码

3.1 分组密码概述

在许多密码系统中，单钥分组密码是系统安全的一个重要组成部分，用分组密码易于构造伪随机数生成器、流密码、消息认证码（MAC）和哈希函数等，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。实际应用中对于分组密码可能提出多方面的要求，除了安全性外，还有运行速度、存储量（程序的长度、数据分组长度、高速缓存大小）、实现平台（硬件、软件、芯片）、运行模式等限制条件。这些都需要与安全性要求之间进行适当的折中选择。

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 $\boldsymbol{x}=(x_0, x_1, \cdots, x_{n-1})$，各组（长为 n 的矢量）分别在密钥 $\boldsymbol{k}=(k_0, k_1, \cdots, k_{t-1})$ 控制下变换成等长的输出数字序列 $\boldsymbol{y}=(y_0, y_1, \cdots, y_{m-1})$（长为 m 的矢量），其加密函数 $E: V_n \times K \to V_m$，V_n 和 V_m 分别是 n 维和 m 维矢量空间，K 为密钥空间，如图 3-1 所示。它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。

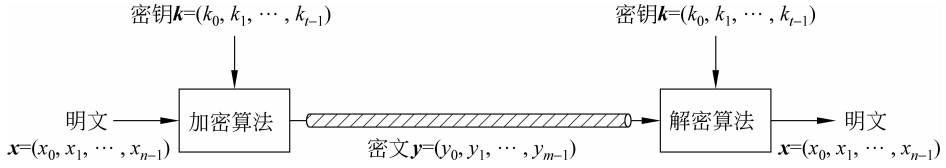


图 3-1 分组密码框图

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。在二元情况下，$\boldsymbol{x}$ 和 $\boldsymbol{y}$ 均为二元数字序列，它们的每个分量 $x_i, y_i \in \mathrm{GF}(2)$。下面主要讨论二元情况。设计的算法应满足下述要求：

（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、FEAL 和 LOKI 等分组密码都采用 $n=64$，在生日攻击下用 2^{32} 组密文成功概率为 1/2，同时要求 $2^{32} \times 64\mathrm{b} = 2^{15}\,\mathrm{MB}$ 存储，故采用穷举攻击是不现实的。

（2）密钥量要足够大（即置换子集中的元素足够多），尽可能消除弱密钥并使所有密钥同等的好，以防止密钥穷举攻击奏效。但密钥又不能过长，以便于密钥的管理。DES 采

用56比特密钥,现在看来太短了,IDEA采用128比特密钥,据估计,在今后30～40年内采用80比特密钥是足够安全的。

(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,没有简单的关系可循,能抗击各种已知的攻击,如差分攻击和线性攻击;有高的非线性阶数,实现复杂的密码变换;使对手破译时除了用穷举法外,无其他捷径可循。

(4) 加密和解密运算简单,易于软件和硬件高速实现。如将分组 n 划分为子段,每段长为8、16或者32。在以软件实现时,应选用简单的运算,使作用于子段上的密码运算易于以标准处理器的基本运算,如加、乘、移位等实现,避免用以软件难以实现的逐比特置换。为了便于硬件实现,加密和解密过程之间的差别应仅在于由秘密密钥所生成的密钥表不同而已。这样,加密和解密就可用同一器件实现。设计的算法采用规则的模块结构,如多轮迭代等,以便于软件和VLSI快速实现。此外,差错传播和数据扩展要尽可能小。

(5) 数据扩展尽可能小。一般无数据扩展,在采用同态置换和随机化加密技术时可引入数据扩展。

(6) 差错传播尽可能小。

要实现上述几点要求并不容易。首先,要在理论上研究有效而可靠的设计方法,而后进行严格的安全性检验,并且要易于实现。

下面介绍设计分组密码时的一些常用方法。

3.1.1 代换

如果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(使解密运算可行),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3-2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个密文比特表示。加密映射和解密映射可由代换表来定义,如表3-1所示。这种定义法是

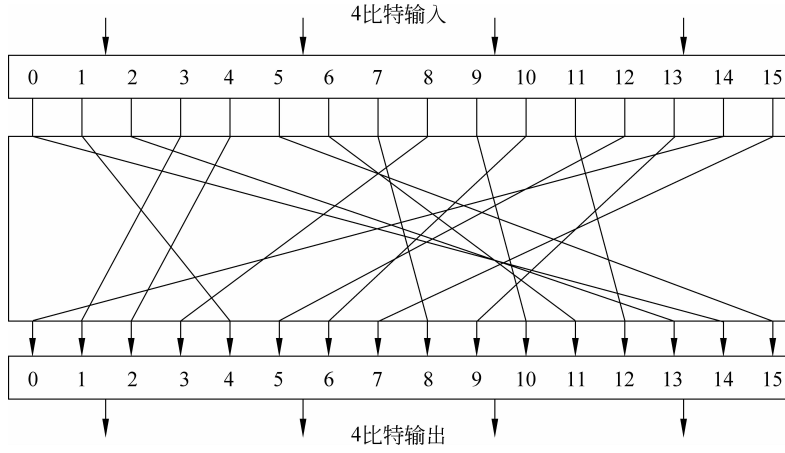


图3-2 代换结构

分组密码最常用的形式，能用于定义明文和密文之间的任何可逆映射。

但这种代换结构在实用中还有一些问题需考虑。如果分组长度太小，如 $n=4$，系统则等价于古典的代换密码，容易通过对明文的统计分析而攻破。这个弱点不是代换结构固有的，只是因为分组长度太小。如果分组长度 n 足够大，而且从明文到密文可有任意可逆的代换，那么明文的统计特性将被隐藏而使以上的攻击不能奏效。

然而，从实现的角度来看，分组长度很大的可逆代换结构是不实际的。仍以表 3-1 为例，该表定义了 $n=4$ 时从明文到密文的一个可逆映射，其中第二列是每个明文分组对应的密文分组的值，可用来定义这个可逆映射。因此从本质上来说，第二列是从所有可能映射中决定某一特定映射的密钥。这个例子中，密钥需要 64 比特。一般地，对 n 比特的代换结构，密钥的大小是 $n\times 2^n$ 比特。如对 64 比特的分组，密钥大小应是 $64\times 2^{64}=2^{70}\approx 10^{21}$ 比特，因此难以处理。实际中常将 n 分成较小的段，如可选 $n=r\cdot n_0$，其中 r 和 n_0 都是正整数，将设计 n 个变量的代换变为设计 r 个较小的子代换，而每个子代换只有 n_0 个输入变量。一般 n_0 都不太大，称每个子代换为代换盒，简称为 S 盒。例如，DES 中将输入为 48 比特、输出为 32 比特的代换用 8 个 S 盒来实现，每个 S 盒的输入端数仅为 6 比特，输出端数仅为 4 比特。

表 3-1　与图 3-2 对应的代换表

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

3.1.2　扩散和混淆

扩散和混淆是由 Shannon 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，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密钥集合。在 Shannon 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。图 3-2 讨论的代换密码就是这样的一个密码系统，然而是不实用的。

扩散是将明文的统计特性散布到密文中去，实现方式是使得密文中每一位由明文中多位产生。例如，对英文消息 $M=m_1m_2m_3\cdots$ 的加密操作，即

$$y_n=\mathrm{chr}\Big(\sum_{i=1}^{k}\mathrm{ord}(m_{n+i})(\mathrm{mod}26)\Big)$$

其中，$\mathrm{ord}(m_i)$ 是求与字母 m_i 对应的序号；$\mathrm{chr}(i)$ 是求与序号 i 对应的字母；密文字母 y_n

是由明文中 k 个连续的字母相加而得。这时明文的统计特性将被散布到密文,因而每一字母在密文中出现的频率比在明文中出现的频率更接近于相等,双字母及多字母出现的频率也更接近于相等。在二元分组密码中,可对数据重复执行某个置换再对这一置换作用于一函数,可获得扩散。

分组密码在将明文分组依靠密钥变换到密文分组时,扩散的目的是使明文和密文之间的统计关系变得尽可能复杂,以使敌手无法得到密钥。混淆是使密文和密钥之间的统计关系变得尽可能复杂,以使敌手无法得到密钥。因此即使敌手能得到密文的一些统计关系,由于密钥和密文之间统计关系复杂化,敌手也无法得到密钥。使用复杂的代换算法可得到预期的混淆效果,而简单的线性代换函数得到的混淆效果不够理想。

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。

3.1.3 Feistel 密码结构

很多分组密码的结构从本质上说都是基于一个称为 Feistel 网络的结构。Feistel 提出利用乘积密码可获得简单的代换密码,乘积密码指顺序地执行两个或多个基本密码系统,使得最后结果的密码强度高于每个基本密码系统产生的结果,Feistel 还提出了实现代换和置换的方法。其思想实际上是 Shannon 提出的利用乘积密码实现混淆和扩散思想的具体应用。

1. Feistel 加密结构

图 3-3 是 Feistel 网络示意图,加密算法的输入是分组长为 $2w$ 的明文和一个密钥 K。将每组明文分成左右两半 L_0 和 R_0,在进行完 n 轮迭代后,左右两半再合并到一起以产生密文分组。其第 i 轮迭代的输入为前轮输出的函数,即

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

其中,K_i 是第 i 轮用的子密钥,由加密密钥 K 得到。一般地,各轮子密钥彼此不同而且与 K 也不同。

Feistel 网络中每轮结构都相同,每轮中右半数据被作用于轮函数 F 后,再与左半数据进行异或运算,这一过程就是上面介绍的代换。每轮的轮函数的结构都相同,但以不同的子密钥 K_i 作为参数。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 Shannon 提出的代换——置换网络(Substitution-Permutation Network,SPN)的特有形式。

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。

(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 比特或更短的密钥是不安全的,通常使用 128 比特长的密钥。

(3) 轮数。单轮结构远不足以保证安全性,多轮结构可提供足够的安全性。典型地,轮数取为 16。

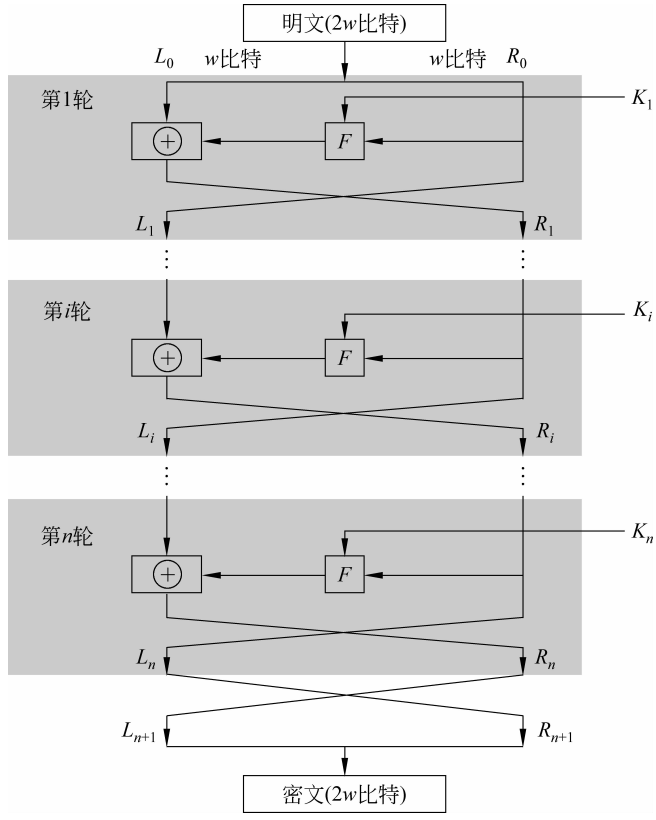


图 3-3　Feistel 网络

(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。轮函数的复杂性越大,密码分析的困难性也越大。

在设计 Feistel 网络时,还有以下两个考虑:

(1) 快速的软件实现。在很多情况下,算法是被镶嵌在应用程序中,因而无法用硬件实现。此时算法的执行速度是考虑的关键。

(2) 算法容易分析。如果算法能被无疑义地解释清楚,就可容易地分析算法抵抗攻击的能力,有助于设计高强度的算法。

2. Feistel 解密结构

Feistel 解密过程本质上和加密过程是一样的,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,即第一轮使用 K_n,第二轮使用 K_{n-1},一直下去,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 3-4 的左边表示 16 轮 Feistel 网络的加密过程,右边表示解密过程,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 $LE_i \parallel RE_i$($\parallel$ 表示链

接),解密过程第 $16-i$ 轮相应的输入是 $RD_i \| LD_i$。

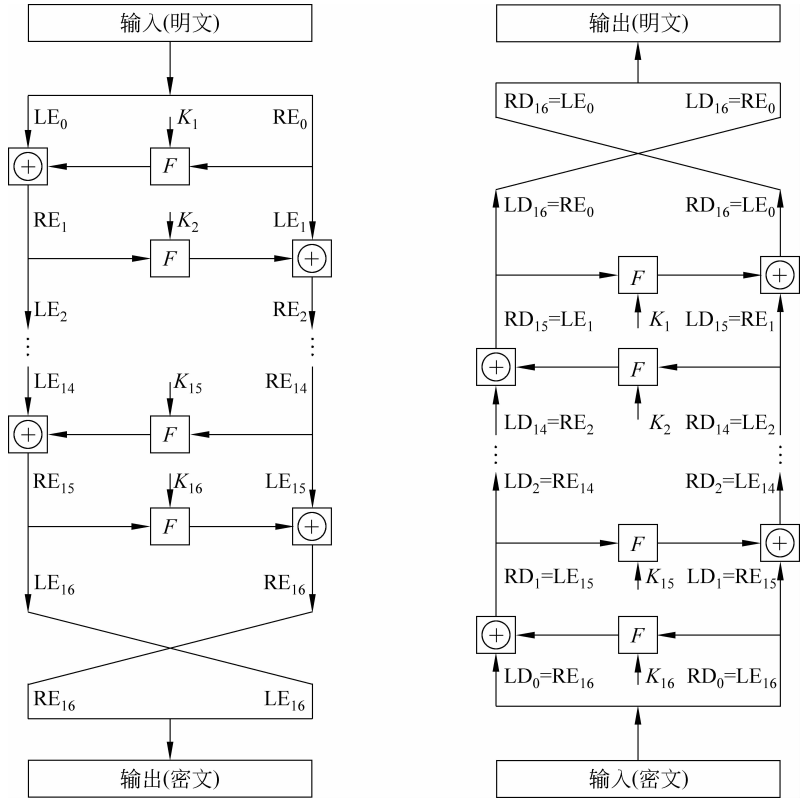


图 3-4　Feistel 加解密过程

加密过程的最后一轮执行完成后,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,即第一轮输入是 $RE_{16} \| LE_{16}$,等于加密过程第 16 轮两半输出交换后的结果。现在显示解密过程第一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,有

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

在解密过程中,有

$$\begin{aligned} LD_1 &= RD_0 = LE_{16} = RE_{15} \\ RD_1 &= LD_0 \oplus F(RD_0, K_{16}) = RE_{16} \oplus F(RE_{15}, K_{16}) \\ &= [LE_{15} \oplus F(RE_{15}, K_{16})] \oplus F(RE_{15}, K_{16}) \\ &= LE_{15} \end{aligned}$$

所以解密过程第一轮的输出为 $LE_{15} \| RE_{15}$,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。一般地,加密过程的第 i 轮有

$$LE_i = RE_{i-1}$$
$$RE_i = LE_{i-1} \oplus F(RE_{i-1}, K_i)$$

因此

$$RE_{i-1} = LE_i$$

$$LE_{i-1} = RE_i \oplus F(RE_{i-1}, K_i) = RE_i \oplus F(LE_i, K_i)$$

以上两式描述了加密过程中第 i 轮的输入与第 i 轮输出的函数关系，由此关系可得图 3-4 右边显示的 LD_i 和 RD_i 的取值关系。

最后可以看到，解密过程最后一轮的输出是 $RE_0 \| LE_0$，左右两半再经一次交换后即得最初的明文。

3.2 数据加密标准

数据加密标准(Data Encryption Standard，DES)是迄今为止世界上最流行和广泛使用的一种分组密码算法，它的分组长度为 64 比特，密钥长度为 56 比特，它是由美国 IBM 公司研制的，是早期的称为 Lucifer 密码的一种发展和修改。DES 在 1975 年 3 月 17 日首次被公布在联邦记录中，在做了大量的公开讨论后于 1977 年 1 月 15 日正式批准并作为美国联邦信息处理标准，即 FIPS-46，同年 7 月 15 日开始生效。规定每隔 5 年由美国国家保密局(National Security Agency，NSA)作出评估，并重新批准它是否继续作为联邦加密标准。最后一次评估是在 1994 年 1 月，且决定 1998 年 12 月以后不再使用 DES。1997 年 DESCHALL 小组经过近 4 个月的努力，通过 Internet 搜索了 3×10^{16} 个密钥，找出了 DES 的密钥，恢复出了明文。1998 年 5 月美国 EFF(Electronics Frontier Foundation)宣布，他们以一台价值 20 万美元的计算机改装成的专用解密机，用 56h 破译了 56 比特密钥的 DES。美国国家标准和技术协会已征集并进行了几轮评估筛选产生了称为 AES (Advanced Encryption Standard)的新加密标准。尽管如此，DES 对于推动密码理论的发展和应用仍起到了重大作用，对于掌握分组密码的基本理论、设计思想和实际应用仍然有着重要的参考价值，下面来描述这一算法。

3.2.1 DES 描述

图 3-5 是 DES 加密算法的框图，其中明文分组长为 64 比特，密钥长为 56 比特。图的左边是明文的处理过程，有 3 个阶段，首先是一个初始置换 IP，用于重排明文分组的 64 比特数据。然后是具有相同功能的 16 轮变换，每轮中都有置换和代换运算，第 16 轮变换的输出分为左右两半，并被交换次序。最后再经过一个逆初始置换 IP^{-1}(为 IP 的逆)从而产生64 比特的密文。除初始置换和逆初始置换外，DES 的结构和图 3-3 所示的 Feistel 密码结构完全相同。

图 3-5 的右边是使用 56 比特密钥的方法。密钥首先通过一个置换函数，然后，对加密过程的每一轮，通过一个左循环移位和一个置换产生一个子密钥。其中每轮的置换都相同，但由于密钥被重复迭代，所以产生的每轮子密钥不相同。

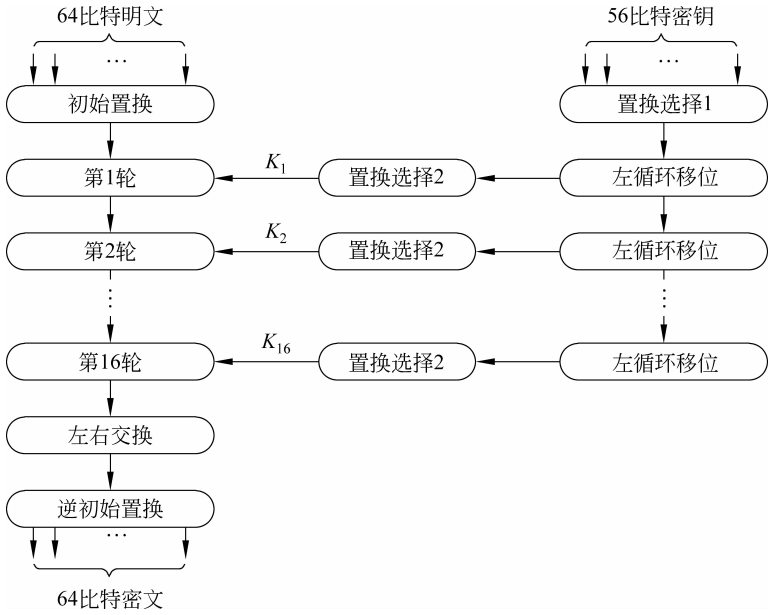


图 3-5　DES 加密算法框图

1. 初始置换

表 3-2(a)和表 3-2(b)分别给出了初始置换和逆初始置换的定义,为了显示这两个置换的确是彼此互逆的,考虑下面 64 比特的输入 M,即

表 3-2　DES 的置换表

(a) 初始置换 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

(b) 逆初始置换 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

(c) 选择扩展运算 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

(d) 置换运算 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

$$
\begin{array}{cccccccc}
M_1 & M_2 & M_3 & M_4 & M_5 & M_6 & M_7 & M_8 \\
M_9 & M_{10} & M_{11} & M_{12} & M_{13} & M_{14} & M_{15} & M_{16} \\
M_{17} & M_{18} & M_{19} & M_{20} & M_{21} & M_{22} & M_{23} & M_{24} \\
M_{25} & M_{26} & M_{27} & M_{28} & M_{29} & M_{30} & M_{31} & M_{32} \\
M_{33} & M_{34} & M_{35} & M_{36} & M_{37} & M_{38} & M_{39} & M_{40} \\
M_{41} & M_{42} & M_{43} & M_{44} & M_{45} & M_{46} & M_{47} & M_{48} \\
M_{49} & M_{50} & M_{51} & M_{52} & M_{53} & M_{54} & M_{55} & M_{56} \\
M_{57} & M_{58} & M_{59} & M_{60} & M_{61} & M_{62} & M_{63} & M_{64}
\end{array}
$$

其中 M_i 是二元数字。由表 3-2(a)得 $X=\mathrm{IP}(M)$为

$$
\begin{array}{cccccccc}
M_{58} & M_{50} & M_{42} & M_{34} & M_{26} & M_{18} & M_{10} & M_2 \\
M_{60} & M_{52} & M_{44} & M_{36} & M_{28} & M_{20} & M_{12} & M_4 \\
M_{62} & M_{54} & M_{46} & M_{38} & M_{30} & M_{22} & M_{14} & M_6 \\
M_{64} & M_{56} & M_{48} & M_{40} & M_{32} & M_{24} & M_{16} & M_8 \\
M_{57} & M_{49} & M_{41} & M_{33} & M_{25} & M_{17} & M_9 & M_1 \\
M_{59} & M_{51} & M_{43} & M_{35} & M_{27} & M_{19} & M_{11} & M_3 \\
M_{61} & M_{53} & M_{45} & M_{37} & M_{29} & M_{21} & M_{13} & M_5 \\
M_{63} & M_{55} & M_{47} & M_{39} & M_{31} & M_{23} & M_{15} & M_7
\end{array}
$$

如果再取逆初始置换 $Y=\mathrm{IP}^{-1}(X)=\mathrm{IP}^{-1}(\mathrm{IP}(M))$，可以看出，$M$ 各位的初始顺序将被恢复。

2. 轮结构

图 3-6 是 DES 加密算法的轮结构，首先看图的左半部分。将 64 比特的轮输入分为

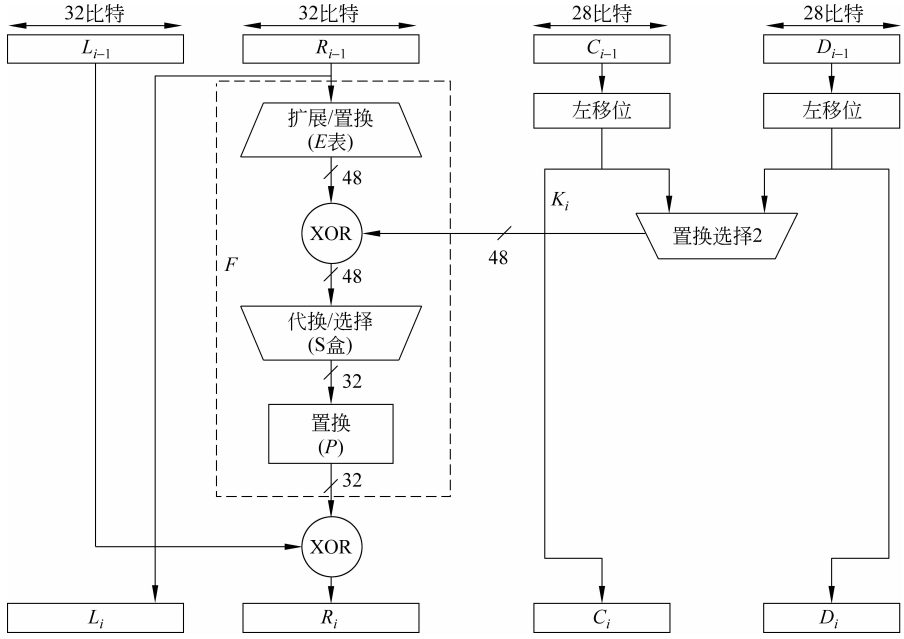


图 3-6　DES 加密算法的轮结构

各为32比特的左、右两半,分别记为L和R。和Feistel网络一样,每轮变换可表示为

$$L_i = R_{i-1}$$

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

其中轮密钥K_i为48比特,函数$F(R,K)$的计算如图3-7所示。输入的右半部分R为32比特,R首先被扩展成48比特,扩展过程由表3-2(c)定义,其中将R的16个比特各重复一次。扩展后的48比特再与子密钥K_i异或,然后再通过一个S盒,产生32比特的输出。该输出再经过一个由表3-2(d)定义的置换,产生的结果即为函数$F(R,K)$的输出。

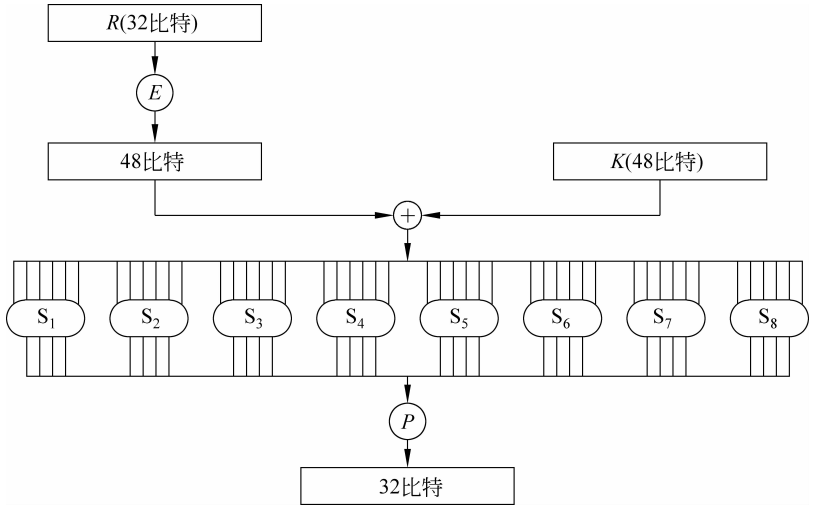


图3-7 函数$F(R,K)$的计算过程

F中的代换由8个S盒组成,每个S盒的输入长为6比特、输出长为4比特,其变换关系由表3-3定义,每个S盒给出了4个代换(由一个表的4行给出)。

表3-3 DES的S盒定义

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

续表

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

对每个盒 S_i，其 6 比特输入中，第 1 个和第 6 个比特形成一个两位二进制数，用来选择 S_i 的 4 个代换中的一个。6 比特输入中，中间 4 位用来选择列。行和列选定后，得到其交叉位置的十进制数，将这个数表示为 4 位二进制数即得这一 S 盒的输出。例如，S_1 的输入为 011001，行选为 01（即第 1 行），列选为 1100（即第 12 列），行列交叉位置的数为 9，其 4 位二进制表示为 1001，所以 S_1 的输出为 1001。

S 盒的每一行定义了一个可逆代换，图 3-2（在 3.1.1 小节）表示 S_1 第 0 行所定义的代换。

3. 密钥的产生

再看图 3-5 和图 3-6，输入算法的 56 比特密钥首先经过一个置换运算，该置换由表 3-4(a)给出，然后将置换后的 56 比特分为各为 28 比特的左、右两半，分别记为 C_0 和 D_0。在第 i 轮分别对 C_{i-1} 和 D_{i-1} 进行左循环移位，所移位数由表 3-4(c)给出。移位后的结果作为求下一轮子密钥的输入，同时也作为置换选择 2 的输入。通过置换选择 2 产生的 48 比特的 K_i，即为本轮的子密钥，作为函数 $F(R_{i-1}, K_i)$ 的输入。其中置换选择 2 由表 3-4(b)定义。

4. 解密

和 Feistel 密码一样，DES 的解密和加密使用同一算法，但子密钥使用的顺序相反。

表 3-4　DES 密钥编排中使用的表

(a) 置换选择 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

(b) 置换选择 2

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

(c) 左循环移位位数

轮数	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

3.2.2　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。

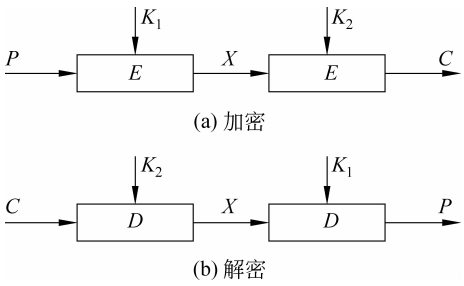


图 3-8　二重 DES

二重 DES 是多重使用 DES 时最简单的形式,如图 3-8 所示。其中明文为 P,两个加密密钥为 K_1 和 K_2,密文为

$$C = E_{K_2}[E_{K_1}[P]]$$

解密时,以相反顺序使用两个密钥,即

$$P = D_{K_1}[D_{K_2}[C]]$$

因此,二重 DES 所用密钥长度为 112 比特,强度极大地增加。然而,如果对任意两个密钥 K_1 和 K_2,能够找出另一密钥 K_3,使得

$$E_{K_2}[E_{K_1}[P]] = E_{K_3}[P]$$

那么,二重 DES 以及多重 DES 都没有意义,因为它们与 56 比特密钥的单重 DES 等价。

但上式对 DES 并不成立。将 DES 加密过程 64 比特分组到 64 比特分组的映射看作一个置换,如果考虑 2^{64} 个所有可能的输入分组,则密钥给定后,DES 的加密将把每个输入分组映射到一个唯一的输出分组;否则,如果有两个输入分组被映射到同一分组,则解密过程就无法实施。对 2^{64} 个输入分组,总映射个数为 $(2^{64})! > (10^{10^{20}})$。

另外,对每个不同的密钥,DES 都定义了一个映射,总映射数为 $2^{56} < 10^{17}$。

因此,可假定用两个不同的密钥两次使用 DES,可得一个新映射,而且这一新映射不出现在单重 DES 定义的映射中。这一假定已于 1992 年被证明。所以使用二重 DES 产生的映射不会等价于单重 DES 加密。但对二重 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：

如果有

$$C = E_{K_2}[E_{K_1}[P]]$$

那么（见图 3-8）

$$X = E_{K_1}[P] = D_{K_2}[C]$$

如果已知一个明文密文对(P,C)，攻击的实施可如下进行：首先，用 2^{56} 个所有可能的 K_1 对 P 加密，将加密结果存入一表并对表按 X 排序，然后用 2^{56} 个所有可能的 K_2 对 C 解密，在上述表中查找与 C 解密结果相匹配的项，如果找到，则记下相应的 K_1 和 K_2。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，用 K_1 和 K_2 对 P' 两次加密，若结果等于 C'，就可确定 K_1 和 K_2 是所要找的密钥。

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}，所以平均来说，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。而再经过另一对明文密文对的检验，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 $1-2^{-16}$。

3.2.3　两个密钥的三重 DES

抵抗中途相遇攻击的一种方法是使用 3 个不同的密钥做 3 次加密，从而可使已知明文攻击的代价增加到 2^{112}。然而，这样又会使密钥长度增加到 $56\times3=168$ 比特，因而过于笨重。一种实用的方法是仅使用两个密钥做 3 次加密，实现方式为加密—解密—加密，简记为 EDE(Encrypt-Decrypt-Encrypt)，见图 3-9，即

$$C = E_{K_1}[D_{K_2}[E_{K_1}[P]]]$$

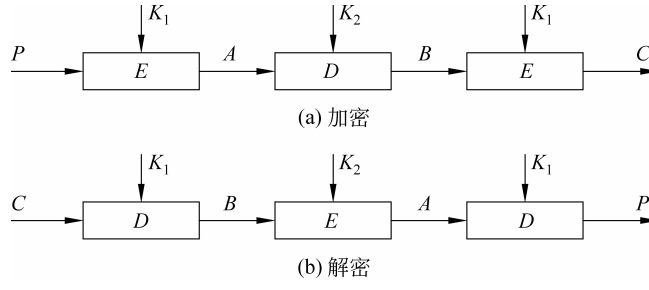


图 3-9　两个密钥的三重 DES

第 2 步解密的目的仅在于使得用户可对一重 DES 加密的数据解密。

此方案已在密钥管理标准 ANSX.917 和 ISO8732 中被采用。

3.2.4　3 个密钥的三重 DES

3 个密钥的三重 DES 密钥长度为 168 比特，加密方式为

$$C = E_{K_3}[D_{K_2}[E_{K_1}[P]]]$$

令 $K_3=K_2$ 或 $K_1=K_2$，则变为一重 DES。

3个密钥的三重DES已在因特网的许多应用(如PGP和S/MIME)中被采用。

3.3 差分密码分析与线性密码分析

3.3.1 差分密码分析

差分密码分析是迄今已知的攻击迭代密码最有效的方法之一,其基本思想是:通过分析明文对的差值对密文对的差值的影响来恢复某些密钥比特。

对分组长度为 n 的 r 轮迭代密码,两个 n 比特串 Y_i 和 Y_i^* 的差分定义为

$$\Delta Y_i = Y_i \otimes Y_i^{*-1}$$

其中,$\otimes$表示 n 比特串集上的一个特定群运算,Y_i^{*-1} 表示 Y_i^* 在此群中的逆元。

由加密对可得差分序列为

$$\Delta Y_0, \Delta Y_1, \cdots, \Delta Y_r$$

其中,Y_0 和 Y_0^* 是明文对,Y_i 和 Y_i^* $(1 \leqslant i \leqslant r)$是第 i 轮的输出,它们同时也是第 $i+1$ 轮的输入。第 i 轮的子密钥记为 K_i,F 是轮函数,且 $Y_i = F(Y_{i-1}, K_i)$。

定义 3-1 r-轮特征(r-round characteristic)Ω 是一个差分序列,即

$$\Omega = \alpha_0, \alpha_1, \cdots, \alpha_r$$

其中,α_0 是明文对 Y_0 和 Y_0^* 的差分,α_i $(1 \leqslant i \leqslant r)$是第 i 轮输出 Y_i 和 Y_i^* 的差分。

定义 3-2 在 r-轮特征 $\Omega = \alpha_0, \alpha_1, \cdots, \alpha_r$ 中,定义

$$p_i^{\Omega} = P(\Delta F(Y) = \alpha_i \mid \Delta Y = \alpha_{i-1})$$

即 p_i^{Ω} 表示在输入差分为 α_{i-1} 的条件下,轮函数 F 的输出差分为 α_i 的概率。

定义 3-3 r-轮特征 $\Omega = \alpha_0, \alpha_1, \cdots, \alpha_r$ 的概率 p^{Ω} 定义为

$$p^{\Omega} = \prod_{i=1}^{r} p_i^{\Omega}$$

对 r-轮迭代密码的差分密码分析可综述为以下步骤:

(1) 找出一个$(r-1)$-轮特征 $\Omega(r-1) = \alpha_0, \alpha_1, \cdots, \alpha_{r-1}$,使得它的概率达到最大或几乎最大。

(2) 均匀随机地选择明文 Y_0 并计算 Y_0^*,使得 Y_0 和 Y_0^* 的差分为 α_0,找出 Y_0 和 Y_0^* 在实际密钥加密下所得的密文 Y_r 和 Y_r^*。若最后一轮的子密钥 K_r(或 K_r 的部分比特)有 2^m 个可能值 K_r^j $(1 \leqslant j \leqslant 2^m)$,设置相应的 2^m 个计数器 Λ_j $(1 \leqslant j \leqslant 2^m)$;用每个 K_r^j 解密密文 Y_r 和 Y_r^*,得到 Y_{r-1} 和 Y_{r-1}^*,如果 Y_{r-1} 和 Y_{r-1}^* 的差分是 α_{r-1},则给相应的计数器 Λ_j 加1。

(3) 重复步骤(2),直到一个或几个计数器的值明显高于其他计数器的值,输出它们所对应的子密钥(或部分比特)。

一种攻击的复杂度可以分为两部分:数据复杂度和处理复杂度。数据复杂度是实施该攻击所需输入的数据量;而处理复杂度是处理这些数据所需的计算量。这两部分主要用来刻画该攻击的复杂度。

差分密码分析的数据复杂度两倍于成对加密所需的选择明文对(Y_0, Y_0^*)的个数。差分密码分析的处理复杂度是从$(\Delta Y_{r-1}, Y_r, Y_r^*)$找出子密钥 K_r(或 K_r 的部分比特)的计算量,它实际上与 r 无关,而且由于轮函数是弱的,所以此计算量在大多数情况下相对较小。因此,差分密码分析的复杂度取决于它的数据复杂度。

3.3.2　线性密码分析

线性密码分析是对迭代密码的一种已知明文攻击,它利用的是密码算法中的"不平衡(有效)的线性逼近"。

设明文分组长度和密文分组长度都为 n 比特,密钥分组长度为 m 比特。记明文分组为 $P[1], P[2], \cdots, P[n]$,密文分组为 $C[1], C[2], \cdots, C[n]$,密钥分组为 $K[1], K[2], \cdots, K[m]$。定义 $A[i, j, \cdots, k] = A[i] \oplus A[j] \oplus \cdots \oplus A[k]$。

线性密码分析的目标就是找出以下形式的有效线性方程,即

$$P[i_1, i_2, \cdots, i_a] \oplus C[j_1, j_2, \cdots, j_b] = K[k_1, k_2, \cdots, k_c]$$

其中,$1 \leqslant a \leqslant n$,$1 \leqslant b \leqslant n$,$1 \leqslant c \leqslant m$。

如果方程成立的概率 $p \neq \frac{1}{2}$,则称该方程是有效的线性逼近。如果 $\left| p - \frac{1}{2} \right|$ 是最大的,则称该方程是最有效的线性逼近。

设 N 表示明文数,T 是使方程左边为 0 的明文数。如果 $T > \frac{N}{2}$,则令

$$K[k_1, k_2, \cdots, k_c] = \begin{cases} 0 & p > \frac{1}{2} \\ 1 & p < \frac{1}{2} \end{cases}$$

如果 $T < \frac{N}{2}$,则令

$$K[k_1, k_2, \cdots, k_c] = \begin{cases} 0 & p < \frac{1}{2} \\ 1 & p > \frac{1}{2} \end{cases}$$

从而可得关于密钥比特的一个线性方程。对不同的明文密文对重复以上过程,可得关于密钥的一组线性方程,从而确定出密钥比特。

研究表明,当 $\left| p - \frac{1}{2} \right|$ 充分小时,攻击成功的概率是

$$\frac{1}{\sqrt{2\pi}} \int_{-2\sqrt{N}\left| p-\frac{1}{2} \right|}^{\infty} \mathrm{e}^{-\frac{x^2}{2}} \mathrm{d}x$$

这一概率只依赖于 $\sqrt{N}\left| p - \frac{1}{2} \right|$,并随着 N 或 $\left| p - \frac{1}{2} \right|$ 的增加而增加。

如何对差分密码分析和线性密码分析进行改进,降低它们的复杂度仍是现在理论研究的热点,目前已推出了很多改进方法,如高阶差分密码分析、截段差分密码分析(truncated differential cryptanalysis)、不可能差分密码分析、多重线性密码分析、非线性

密码分析、划分密码分析和差分—线性密码分析,再如针对密钥编排算法的相关密钥攻击、基于Lagrange插值公式的插值攻击及基于密码器件的能量分析(power analysis)。另外还有错误攻击、时间攻击、Square攻击和Davies攻击等。

3.4 分组密码的运行模式

分组密码在加密时明文分组的长度是固定的,而实用中待加密消息的数据量是不定的,数据格式可能是多种多样的。为了能在各种应用场合使用DES,美国在FIPS PUS 74和81中定义了DES的4种运行模式,如表3-5所示。这些模式也可用于其他分组密码,下面以DES为例来介绍这4种模式。

表3-5 DES的运行模式

模　式	描　述	用　途
电码本(ECB)模式	每个明文组独立地以同一密钥加密	传送短数据(如一个加密密钥)
密码分组链接(CBC)模式	加密算法的输入是当前明文组与前一密文组的异或	传送数据分组;认证
密码反馈(CFB)模式	每次只处理输入的j比特,将上一次的密文用作加密算法的输入以产生伪随机输出,该输出再与当前明文异或以产生当前密文	传送数据流;认证
输出反馈(OFB)模式	与CFB类似,不同之处是本次加密算法的输入为前一次加密算法的输出	有扰信道上(如卫星通信)传送数据流

3.4.1 电码本模式

电码本(Electronic CodeBook,ECB)模式是最简单的运行模式,它一次对一个64比特长的明文分组加密,而且每次的加密密钥都相同,如图3-10所示。当密钥取定时,对明文的每一个分组,都有一个唯一的密文与之对应。因此,可以形象地认为有一个非常大的电码本,对任意一个可能的明文分组,电码本中都有一项对应于它的密文。

如果消息长于64比特,则将其分为长为64比特的分组,最后一个分组如果不够64比特,则需要填充。解密过程也是一次对一个分组解密,而且每次解密都使用同一密钥。图3-10中,明文是由分组长为64比特的分组序列$P_1,P_2,\cdots,P_N$构成,相应的密文分组序列是$C_1,C_2,\cdots,C_N$。

ECB在用于短数据(如加密密钥)时非常理想,因此如果需要安全地传递DES密钥,ECB是最合适的模式。

ECB的最大特性是同一明文分组在消息中重复出现的话,产生的密文分组也相同。

ECB用于长消息时可能不够安全,如果消息有固定结构,密码分析者有可能找出这种关系。例如,如果已知消息总是以某个预定义字段开始,那么分析者就可能得到很多明文密文对。如果消息有重复的元素而重复的周期是64的倍数,那么密码分析者就能够识别这些

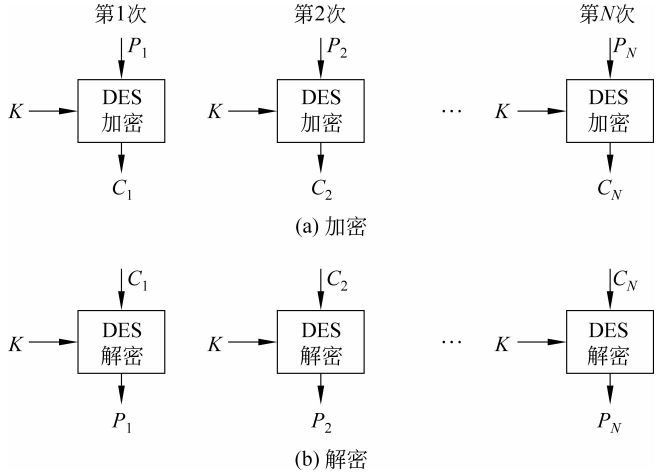


图 3-10　ECB 模式示意图

元素。以上这些特性都有助于密码分析者，有可能为其提供对分组的代换或重排的机会。

3.4.2　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，密码分组连接(Cipher Block Chaining，CBC)模式就可满足这一要求。

图 3-11 是 CBC 模式示意图，它一次对一个明文分组加密，每次加密使用同一密钥，加密算法的输入是当前明文分组和前一次密文分组的异或，因此加密算法的输入不会显

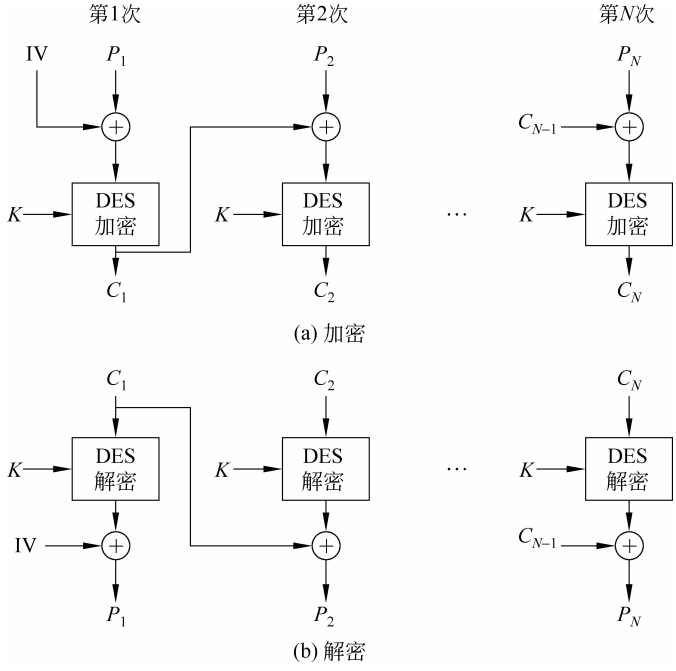


图 3-11　CBC 模式示意图

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。

解密时,每一个密文分组被解密后,再与前一个密文分组异或,即

$$D_K[C_n] \oplus C_{n-1} = D_K[E_K[C_{n-1} \oplus P_n]] \oplus C_{n-1}$$
$$= C_{n-1} \oplus P_n \oplus C_{n-1} = P_n \quad (\text{设 } C_n = E_K[C_{n-1} \oplus P_n])$$

因而产生出明文分组。

在产生第一个密文分组时,需要有一个初始向量 IV 与第一个明文分组异或。解密时,IV 和解密算法对第一个密文分组的输出进行异或以恢复第一个明文分组。

IV 对于收发双方都应是已知的,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 的原因如下:如果敌手能欺骗接收方使用不同的 IV 值,敌手就能够在明文的第一个分组中插入自己选择的比特值,这是因为

$$C_1 = E_K[\mathrm{IV} \oplus P_1]$$
$$P_1 = \mathrm{IV} \oplus D_K[C_1]$$

用 $X(i)$表示 64 比特分组 X 的第 i 个比特,那么 $P_1(i)=\mathrm{IV}(i)\oplus D_K[C_1](i)$,由异或的性质得

$$P_1(i)' = \mathrm{IV}(i)' \oplus D_K[C_1](i)$$

其中撇号表示比特补。上式意味着如果敌手篡改 IV 中的某些比特,则接收方收到的 P_1 中相应的比特也发生变化。

由于 CBC 模式的链接机制,CBC 模式对加密长于 64 比特的消息非常合适。

CBC 模式除能够获得保密性外,还能用于认证。

3.4.3 密码反馈模式

如上所述,DES 是分组长为 64 比特的分组密码,但利用密码反馈(Cipher FeedBack,CFB)模式或 OFB 模式可将 DES 转换为流密码。流密码不需要对消息填充,而且运行是实时的。因此如果传送字母流,可使用流密码对每个字母直接加密并传送。

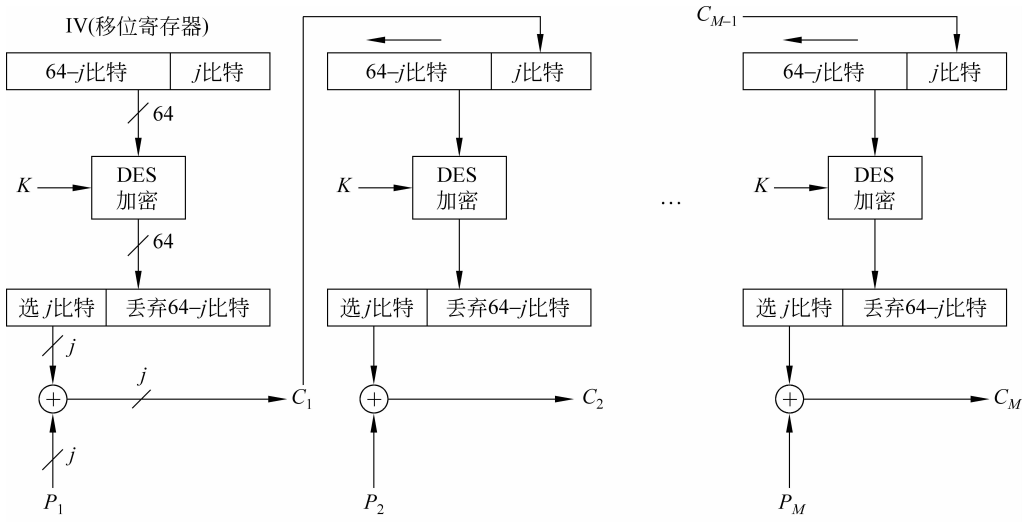
流密码具有密文和明文一样长这一性质,因此,如果需要发送的每个字符长为 8 比特,就应使用 8 比特密钥来加密每个字符。如果密钥长超过 8 比特,则造成浪费。

图 3-12 是 CFB 模式示意图,设传送的每个单元(如一个字符)是 j 比特长,通常取 $j=8$,与 CBC 模式一样,明文单元被链接在一起,使得密文是前面所有明文的函数。

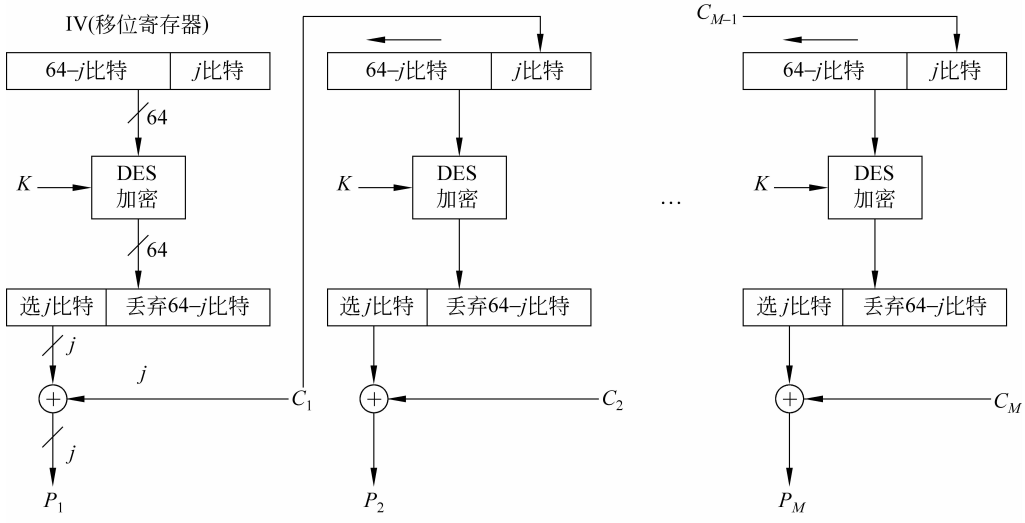
加密时,加密算法的输入是 64 比特移位寄存器,其初值为某个初始向量 IV。加密算法输出的最左(最高有效位)j 比特与明文的第一个单元 P_1 异或,产生出密文的第一个单元 C_1,并传送该单元。然后将移位寄存器的内容左移 j 位并将 C_1 送入移位寄存器最右边(最低有效位)j 位。这一过程继续到明文的所有单元都被加密为止。

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:

设 $S_j(X)$是 X 的 j 个最高有效位,那么 $C_1=P_1\oplus S_j(E(\mathrm{IV}))$,因此



(a) 加密



(b) 解密

图 3-12　CFB 模式示意图

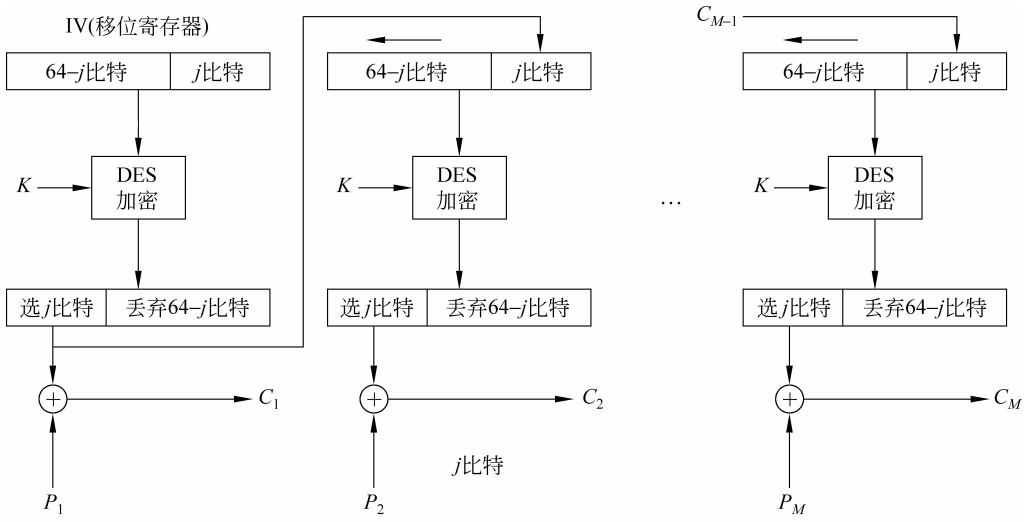
$$P_1 = C_1 \oplus S_j(E(\mathrm{IV}))$$

可证明以后各步也有类似的这种关系。

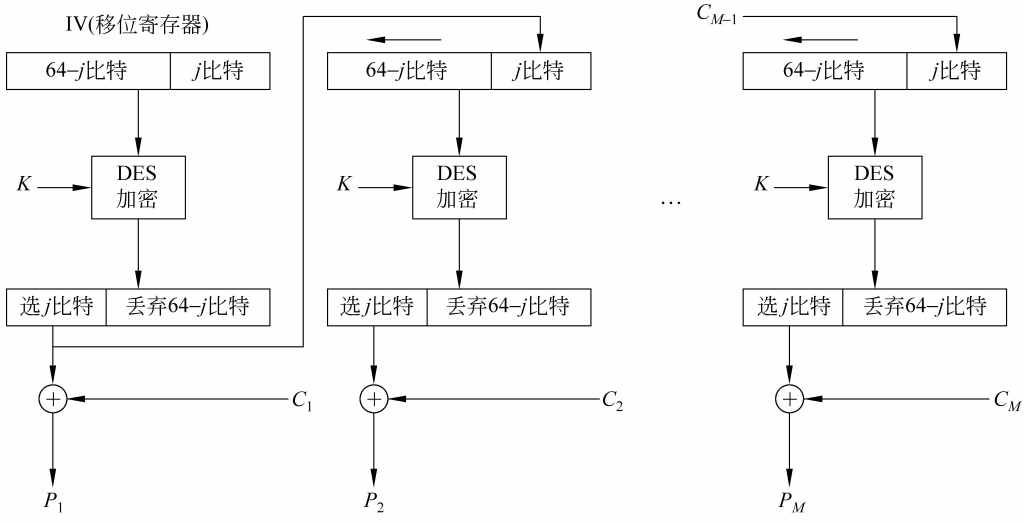
CFB 模式除能获得保密性外，还能用于认证。

3.4.4　输出反馈模式

输出反馈(Output FeedBack，OFB)模式的结构类似于 CFB，见图 3-13。不同之处如下：OFB 模式是将加密算法的输出反馈到移位寄存器，而 CFB 模式中是将密文单元反馈到移位寄存器。



(a) 加密



(b) 解密

图 3-13　OFB 模式示意图

OFB 模式的优点是传输过程中的比特错误不会被传播。例如，C_1 中出现 1 比特错误，在解密结果中只有 P_1 受到影响，以后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。

OFB 的缺点是它比 CFB 模式更易受到对消息流的篡改攻击，比如在密文中取 1 比特的补，那么在恢复的明文中相应位置的比特也为原比特的补。因此使得敌手有可能通过对消息校验部分的篡改和对数据部分的篡改，而以纠错码不能检测的方式篡改密文。