第3章　分组密码

3.1　分组密码定义

分组密码是对称密码的典型代表。通俗地说就是数据在密钥的作用下，一组一组、等长地被处理，且通常情况是明文、密文等长。这样做的好处是处理速度快，节约了存储空间，避免了浪费带宽。分组密码也是许多密码组件的基础。例如，它很容易转化为流密码、Hash 函数。分组密码的另一个特点是容易标准化。分组密码由于其固有的特点（高强度、高速率、便于软硬实现）而成为标准化进程的首选体制。DES 是首先成为数据加密标准的分组密码典型代表。作为数据加密标准，DES 算法完全公开，任何个人和团体都可以使用，其信息的安全性取决于各自密钥的安全性，这正是现代分组密码的特征。

分组密码就是将明文消息序列 $m_1, m_2, \cdots, m_k \cdots$ 分成等长的消息组 $(m_1, m_2, \cdots, m_n)$ $(m_{n+1}, m_{n+2}, \cdots, m_{2n}) \cdots$。在密钥控制下，按固定的算法 E_k 一组一组地进行加密，加密后输出密文组 $(y_1, \cdots, y_m)(y_{m+1}, \cdots, y_{2m})$，一般情况下 $m = n$。一个分组长为 n 比特，密钥长为 t 比特的分组密码，数学上可以看作在 2^t 个密钥控制下的 $\mathrm{GF}(2)^n \to \mathrm{GF}(2)^n$ 的置换，而从 $\mathrm{GF}(2)^n \to \mathrm{GF}(2)^n$ 的置换有 $2^n!$ 个不同的方式。故一个极好的 n 比特的分组密码可以接受的密钥长度可达 $\lfloor \log_2(2^n)! \rfloor$ 比特，用来加密的置换只是全体置换所构成集合的一个子集。设计分组密码的问题，关键在于找到一种算法，能在密钥的控制下，从一个足够大且“好”的置换子集中，简单而迅速地选出一个置换。

一般来说，分组密码可以定义为如下一种映射：

$$F_2^n \times F_2^t \to F_2^m$$

记为 $E(X, K)$ 或 $E_k(X), X \in F_2^n, K \in F_2^t$，　F_2^n 称为明文空间，F_2^m 称为密文空间，F_2^t 称为密钥空间。n 为明文分组长度，当 $n > m$ 时，称为有数据压缩的分组密码；当 $n < m$ 时，称为有数据扩展的分组密码，当 $n = m$ 且为一一映射时，$E_k(x)$ 就是 $\mathrm{GF}(2)^n$ 到 $\mathrm{GF}(2)^n$ 的置换。通常的情况是 $n = m$。

分组密码的加密过程如图 3.1 所示。

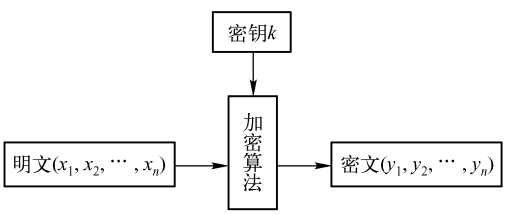

图 3.1　分组密码的加密过程

3.1.1 分组密码的结构

一个安全的分组密码既要难于分析（复杂），又要易于实现（简单）。迭代密码就是为了克服这一对矛盾而产生的一种分组密码。其加密变换（置换）一般采取如下结构：由一个简单的函数 F（易于实现）迭代若干次而形成，如图 3.2 所示。

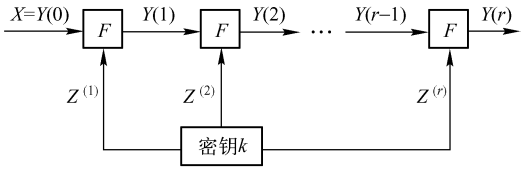

图 3.2 迭代型分组密码的结构

在图 3.2 中，$Y(i-1)$ 是第 i 轮置换的输入，$Y(i)$ 是第 i 轮的输出，$z^{(i)}$ 是第 i 轮的子密钥，k 是种子密钥。每次迭代称为一轮，每轮的输出是输入和该轮子密钥的函数。每轮子密钥由 k 导出。这种密码就是迭代密码，如 DES 就是 16 轮迭代密码。函数 F 称为圈函数或轮函数。一个适当选择的圈函数通过多次迭代可实现必要的混淆和扩散。

如果把一个 GF $(2)^n$ 到 GF $(2)^m$ 的变换看成一个网络，常用的圈函数 F 都是基于代换——置换的网络，即以多次变换的乘积构成，称为置换的变换提供扩散、代换的变换提供混淆，其中代换网络是精心设计起关键作用的，人们常称其为黑盒子。为了增强安全性，n 一般都比较大。在代换的实现中，其难度将随 n 呈指数级增长，而难于处理，不易实现。因此，实际中常将 n 划分成一些较短的段，如将 n 分成长为 n_0 的 r 个段，将设计 n 长变换的“黑盒子”简化为设计 r 个较小的子代换网络，大大降低了实现的难度。这些称为子代换盒，简称 S 盒。如 DES 中有 8 个 S 盒。S 盒的设计是分组密码设计的核心，其遵循的准则是保证整个密码系统安全性的关键所在。以上描述了在一个分组密码设计中为了实现既“复杂”（为了安全），又“简单”（为了实现方便）而采取的典型结构形式。DES 和 AES 体制是这种结构的典型代表。

进一步讲，分组密码又采用两种类型的总体结构：Feistel 网络与 SP 网络，它们的主要区别在于：SP 结构每轮改变整个数据分组，而 Feistel 密码每轮只改变输入分组的一半。AES 和 DES 分别是这两种结构的代表。Feistel 网络（又称 Feistel 结构）可把任何轮函数转化为一个置换，它是由 Horst Feistel 在设计 Lucifer 分组密码时发明的，并因 DES 的使用而流行。“加解密相似”是 Feistel 型密码的实现优点。SP 网络（又称 SP 结构）是 Feistel 网络的一种推广，其结构清晰，S 一般称为混淆层，主要起混淆作用；P 一般称为扩散层，主要起扩散作用。SP 网络与 Feistel 网络相比，可以得到更快速的扩散，不过 SP 网络的加/解密通常不相似。

3.1.2 分组密码的设计原则

分组加密算法其实可以看作一个置换，用来加密的置换只是全体置换所构成集合的一个子集。设计分组密码的问题，关键在于找到一种算法，能在密钥的控制下，从一个足够大且“好”的置换子集中，简单而迅速地选出一个置换。

影响分组密码安全性的因素很多，诸如分组长度 n 和密钥长度 t 等。但有关实用密码的两个一般设计原则是 Shannon 提出的混乱原则和扩散原则。

（1）混乱原则：所设计的密码应使得密钥和明文及密文之间的依赖关系相当复杂，以至于这种依赖性对密码分析者来说是无法利用的。

（2）扩散原则：所设计的密码应使得密钥的每一位数字影响密文的许多位数字，以防止对密钥进行逐段破译，而且明文的每一位数字也应影响密文的许多位数字以便隐蔽明文数字统计特性。

分组密码可以用软件和硬件来实现。硬件实现的优点是可获得高速率，而软件实现的优点是灵活性强、代价低。基于软件和硬件的不同性质，分组密码的设计原则可根据预定的实现方法来考虑。

（1）软件实现的设计原则：使用子块和简单的运算。密码运算在子块上进行，要求子块的长度能自然地适应软件编程，如 8bit、16bit、32bit 等。在软件实现中，按比特置换是难于实现的，因此应尽量避免使用它。子块上所进行的一些密码运算应是一些易于软件实现的运算，最好是用一些标准处理器所具有的一些基本指令，如加法、乘法和移位等。

（2）硬件实现的设计原则：加密和解密可用同样的器件来实现。尽量使用规则结构，因为密码应有一个标准的组件结构，以便其能适应于用超大规模集成电路实现。

前面提到的乘积密码是实现 Shannon 提出的混乱原则和扩散原则的一种有效的方法。DES 就是一种这样的乘积密码。

当然，以上的原则是非常概括的，离构造安全的分组密码还差得很远。下面一些原则也是常常需要考虑的。

（1）简单性原则：包括规范的简单性和分析的简单性。规范的简单性指仅采用了有限个运算，并且这些运算本身很容易解释。简单规范的一个明显优点是便于正确实现，另一个优点是人们在研究密码时，似乎对具有简单规范的密码算法更有兴趣。分析的简单性，好处是便于阐述和理解密码算法以何种方式来抗击已知类型的密码分析。这样，在设计阶段就开始考虑抵抗已知攻击，从而在设计之初就提供了一定程度的密码可信度。规范的简单性并非意味着分析的简单性，提出一个描述简单而已知攻击手段又难以分析的密码算法是相对容易的。

（2）必要条件：设计一个分组密码的最低要求是它必须能抗击所有已知的攻击，特别是差分攻击和线性攻击。因此，密码设计者不仅要熟悉现存的各种攻击方法，而且要预想到一些未知的攻击。

（3）可扩展性：在设计密码时还应充分考虑各种可能的扩展情况，如可变分组或密钥长，这样才能灵活适应多级安全需要。

安全性是分组密码最重要的设计准则，它要求即使攻击者知道分组密码的内部结构，仍不能破译该密码。这也意味着，不存在针对该密码的某种攻击方法，其工作量小于穷密钥搜索。

3.1.3　轮结构分组密码的安全性分析

对于一个 n 轮分组密码，如果存在一个对 $n-k$ 轮简化版本的密码分析攻击，那么称该

密码具有 k 轮绝对安全冗余。因此，安全冗余也意味着即使对现有的密码分析手段进行改进，该密码仍然能够抗击此类攻击。密码分析技术的发展，使得对于具有更多轮的分组密码的破译成为可能。

分组密码提供一种把 N 比特长的明文消息转换为 N 比特长的密文消息的方法，其中，加密操作是由长为 k 比特的秘密密钥串决定的，密钥常常是随机选择的；解密是加密的逆操作，用同样的密钥把 N 比特的密文还原成明文。在考虑安全性时，仍然沿用 Kerchhoff 假设：除了密钥之外，攻击者知道所有有关加/解密的详细过程。因此，直观的要求是，如果不知道密钥的知识，攻击者从密文恢复出明文是不可能的。

对一个固定的 k 比特密钥，如果一个分组长为 N 比特的分组密码只用来加密 $\left\lfloor \frac{k}{N} \right\rfloor$ 个明文，则对每个明文比特来说，每个密钥比特只使用一次，这种情况即为“一次一密制”，明文和密文是统计独立的，分组密码是绝对安全的。然而，在大多数的情况下，一次一密是不实际的，因此，用同样固定的 k 比特密钥加密 T 个明文，其中 $T >> \left\lfloor \frac{k}{N} \right\rfloor$，最大化 T 后仍能达到一个可接受的安全性是现代分组密码追求的目标。

既然一个分组长为 N，密钥长为 k 的分组密码，可以看作对每一个可能的密钥定义了一个 2^N 个元素上的置换，一个简单的带密钥的随机查表（实现一个 N 比特的置换）不用附加任何算法，将实现一个非常强的 N 比特分组密码；不幸的是，该实现将花费太多的存储，这意味着一个巨大的实现复杂性，不能用作任何实际的用途。几乎所有分组密码都用更小的查表（S 盒）合并其他变换（线性变换）模仿这样一个大的随机查表。这种做法实际上是安全性和复杂性的一个折中。

分组密码可看作一个置换的集合，如果可证明某个分组密码渐近等价于伪随机置换的集合，则可认为该密码是安全的。换句话说，我们无法区分输出密文比特和某个随机输出。理论上可以说：依赖于可证明安全的伪随机函数发生器，若不知密钥，在多项式时间内无法与真正的随机置换相区分。伪随机意味着在多项式时间内，加密询问使得没有攻击者可以区分分组密码和一个真实的随机置换，相应于选择明文攻击。超伪随机意味着在多项式时间内，加密、解密询问使得没有攻击者可以区分分组密码和一个真实的随机置换，相应于选择明、密文攻击。由超伪随机性可推出伪随机性。例如，3 – 轮 DES 是一个伪随机置换，4 – 轮 DES 是一个超伪随机置换。

分组密码也可以与单向函数联系起来。给定一个 N 比特明文消息，m，一个 k 比特的密钥，加密函数 E，产生一个 N 比特的密文输出消息 C，其中 $C = E(k, m)$，如果固定 m_0，可定义 $f(k) = E(k, m_0)$。Luby 和 Rackoff 证实如果 E 是伪随机置换的集合，则 f 近似于一个单向函数。然而单向函数概念比安全的分组密码要弱。例如，泄露一个单向函数输入的一半，其仍然是一个单向函数。

3.2 数据加密算法标准（DES）

数据加密算法标准（Data Encryption Standard，DES）是由 IBM 公司在 1970 年发展出的一个加密算法。DES 在 1977 年经过美国国家标准局（NBS）采用为联邦标准之后，已成为

金融界及其他各种产业最广泛应用的对称密钥密码系统。DES 是分组密码的典型代表，也是第一个被公布出来的标准算法。

DES 正式颁布后，世界各国的许多公司都推出了自己实现 DES 的软硬件产品。虽然 DES 的描述相当长，但它能以硬件或软件方式非常有效地实现，需完成的算术运算仍为比特串的异或、扩展函数 E、S 盒，置换 IP 和 P 及 16 个子密钥的计算都能在一个固定时间内通过查表（以软件）或电路中的硬件布线来完成。现在的硬件实现能达到非常快的加密速度。虽然目前 DES 已经被 AES 所取代，但是由于 DES 的基本理论和设计思想仍有重要参考价值，所以下面我们简要地描述 DES 算法。

3.2.1　DES 设计思想

DES 综合应用了置换、代替、移位等多种密码技术，是一种乘积密码。它在算法结构上采用迭代结构，从而使结构紧凑、条理清楚，而且算法为对合运算，便于实现。DES 使用了初始置换 IP 和逆初始置换 IP^{-1}各一次，置换 P16 次，安排使用这 3 个置换的目的是把数据彻底打乱重排。它们在密码意义上作用不大，因为它们与密钥无关，置换关系固定，一旦公开后便无多大密码意义。选择置换 E 一方面把数据打乱重排，另一方面把 32 位输入扩展为 48 位。算法中除了 S 盒是非线性变换外，其余变换均为线性变换，所以保密性的关键是选择 S 盒。DES 的 S 盒经过了精心设计和严格挑选。美国国家安全局曾经确认过下列 3 条“设计准则”。

（1）对任意一个 S 盒而言，没有任何线性方程式等价于此 S 盒的输出/输入关系。也就是说，S 盒是非线性函数。

（2）改变 S 盒的任何一位的输入，则至少有两个以上的输出位会因此而有所改变。换句话说，任一输入位可以影响的输出位越多越好。

（3）当固定某一个位的输入时，我们希望 S 盒的 4 个输出位之间，其“0”和“1”个数之差越小越好。

这个非线性变换的本质是数据压缩，它把 6 位输入压缩为 4 位输出。选择 S 盒函数的输入中任意改变数位，其输出至少变化 2 位。因为算法中使用了 16 次迭代，从而使得即使是改变明文或密钥中的 1 位，密文都会发生约 32 位的变化，大大提高了保密性。DES 的子密钥产生与使用上也很有特色，它确保了原密钥中各位的使用次数基本相等。试验表明，56 位密钥的每位的使用次数在 12 ~ 15 次之间。这也使保密性得到进一步提高。

总体上看，DES 是相当成功的，虽然它有以下一些缺点和不足。

（1）存在一些弱密钥和半弱密钥。在 16 次加密迭代中分别使用不同的子密钥是确保 DES 强度的一种重要措施。但由于子密钥产生过程的设计不当，实际上却存在一些密钥，由它们产生的 16 个子密钥不是互不相同，而是有相重合的，这些密钥称为弱密钥或半弱密钥。使 16 个子密钥全相同的密钥称为弱密钥，使 16 个子密钥中有部分相同的密钥称为半弱密钥。弱密钥的使用会降低 DES 的安全性。若 k 为弱密钥，则下列关系式成立：

$$E_k(E_k(m)) = m \text{ 及 } D_k(D_k(m)) = m$$

但由于弱密钥和半弱密钥的数量与密钥的总数相比仍是微不足道的，所以这并不构成对 DES 的太大威胁，只要在实际应用中避免使用这些密钥即可。

（2）存在互补对称性。在 DES 的明文 m、密文 C 与密钥 k 之间存在互补的特性。此互

补性，简单地说，可以用下列两个式子表示：

$$若\ E_k(m)=C，则\ E_{\overline{k}}(\overline{m})=\overline{C}$$

这个关系是说，如果以密钥 k 对明文 m 加密，得到密文 C；相对地，以密钥 $\overline{k}$ 对明文 $\overline{m}$ 加密，可得到 $\overline{C}$。其中 $\overline{X}$ 表示 X 逐位取补。这个性质使得非法者有机可乘：假设破译者 A 要破解使用者 B 的密钥 k，而且 A 又拥有 B 使用密钥 k 对明文 m 及 $\overline{m}$ 加密的密文 $E_k(m)$ 及 $E_k(\overline{m})$，则 A 可利用 DES 的互补性来找出密钥 k；这比穷举密钥搜索法少花了一半的时间（时间复杂度为 2^{55}）。尽管如此，但在实际上却不太可行。因为两个明文互为补码的概率相当小，所以破译者要想获得 $E_k(m)$ 及 $E_k(\overline{m})$ 也相当困难。因此，这种互补性不能算是 DES 的漏洞。

3.2.2 DES 算法描述

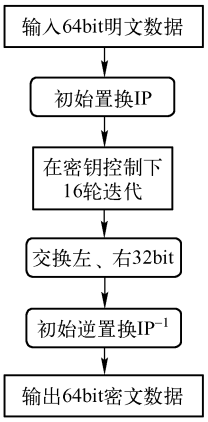

图 3.3 DES 结构框图

DES 是分组长度为 64bit 的分组密码算法，密钥长度也是 64bit，其中每 8bit 有一位奇偶校验位，因此有效密钥长度为 56bit。DES 算法是公开的，其安全性依赖于密钥的保密程度。DES 结构框图如图 3.3 所示。

初始置换 IP 和初始逆置换 IP^{-1}：将 64bit 明文数据用初始置换 IP 置换，得到一个乱序的 64bit 明文分组，然后分成左、右等长的 32bit，分别记为 L_0 和 R_0。进行 16 轮完全类似的迭代运算后，将所得左、右长度相等的两半 L_{16} 和 R_{16} 交换得到 64bit 数据 $R_{16}L_{16}$。最后再用初始逆置换 IP^{-1} 进行置换，产生密文数据组。置换表自左向右、自上而下的 64 个位置对应 64bit 数据组，置换表中的数字表示将 64bit 数据组中该数字所在位置的比特置换为该数字表示的位置的比特。初始置换 IP 和初始逆置换 IP^{-1} 如表 3.1 所示。

表 3.1 初始置换 IP 和初始逆置换 IP^{-1}

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

例如，将 64bit 数据表示为 $M=(m_1,m_2,\cdots,m_{64})$，则在初始置换 IP 的作用下变为 $IP(M)=(m_{58},m_{50},m_{42},m_{34},m_{26},m_{18},m_{10},m_2,\cdots,m_{63},m_{55},m_{47},m_{39},m_{31},m_{23},m_{15},m_7)$。如果将 $IP(M)$ 用初始逆置换 IP^{-1} 作用，将会得到 M。例如，M 中的第 60 个比特 m_{60}，在 $IP(M)$ 中位于第 9 个比特位，$IP(M)$ 在初始逆置换 IP^{-1} 作用下，第 9 个比特移至第 60 个比特位。这说

明 IP 和 IP^{-1}互逆。

迭代变换是 DES 算法的核心部分，如图 3.4 所示。每轮开始时将输入的 64bit 数据分成左、右长度相等的两半，右半部分原封不动地作为本轮输出的 64bit 数据的左半部分，同时对右半部分进行一系列的变换，即用轮函数 F 作用右半部分，然后将所得结果（32bit 数据）与输入数据的左半部分进行逐位异或，最后将所得数据作为本轮输出的 64bit 数据的右半部分。

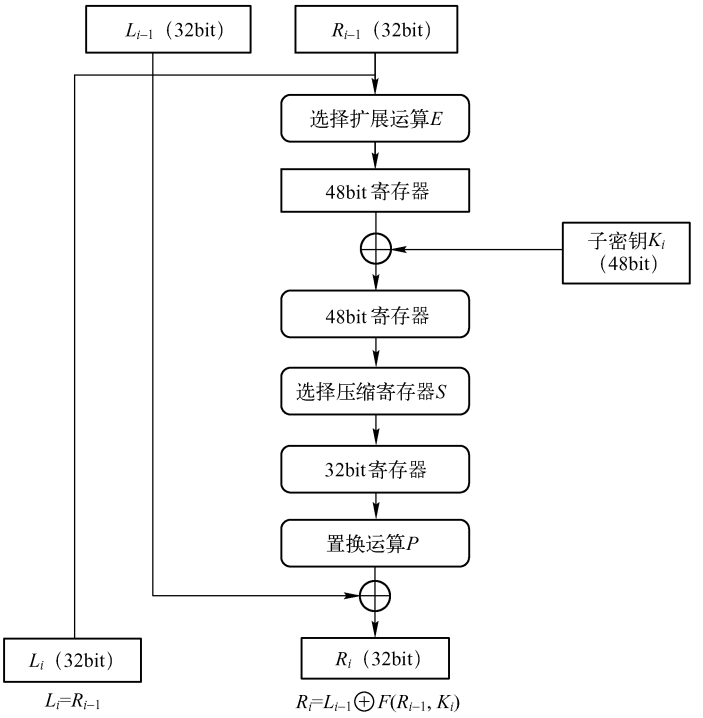

图 3.4　DES 的一轮迭代

f 函数是整个 DES 加密法中最重要的部分，轮函数 F 由选择扩展运算 E、与子密钥的异或运算、选择压缩运算 S 和置换 P 组成。f 函数可记作 $f(A,J)$，其中 A 为 32 位输入，J 为 48 位输入，在第 i 轮 $A=R_{i-1}$，$J=K_i$，K_i 为由初始密钥（也称种子密钥）导出的第 i 轮子密钥。$f(A,J)$输出为 32bit。

$f(A,J)$的计算过程如下。将 A 经过一个选择扩展运算 E 变为 48 位，记为 $E(A)$。计算 $E(A)\oplus J=B$，对 B 施行代换 S，此代换由 8 个代换盒组成，就是前面说过的 S 盒。每个 S 盒有 6 个输入、4 个输出，将 B 依次分为 8 组，每组 6 位，记 $B=B_1B_2B_3B_4B_5B_6B_7B_8$，其中 B_j 作为第 j 个 S 盒 S_j 的输入，S_j 的输出为 C_j，$C=C_1\cdots\cdots C_8$ 就是代换 S 的输出，所以代换 S 是一个 48 位输入，32 位输出的选择压缩运算，将结果 C 再施行一个压缩置换 P，即得 $f(A,J)$。其中在第 i 轮为 $f(R_{i-1},k_i)$。$f(A,J)$可用图 3.5 表示。

选择扩展运算 E：将输入的 32bit 数据扩展为 48bit 的输出数据，扩展变换见表 3.2。

如果将输入的 32 比特数据按 E 中所标位置顺序读出，则可得到 48 比特的输出数据。可以看出 1,4,5,8,9,12,13,16,17,20,21,24,25,28,29,32 这 16 个位置上的数据都被读了两次。

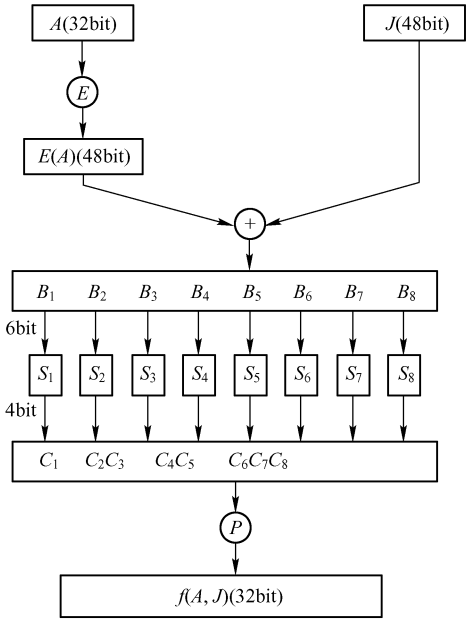

图 3.5 f 函数运算框图

表 3.2 扩展变换 *E*

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

与子密钥的异或运算：将选择扩展运算的 48bit 输出数据与子密钥 K_i（48bit）进行异或运算。

选择压缩运算：将输入的 48bit 数据从左至右分成 8 组，每组 6bit，然后输入 8 个 S 盒，每个 S 盒为一非线性代换，有 4bit 输出，如图 3.6 所示。

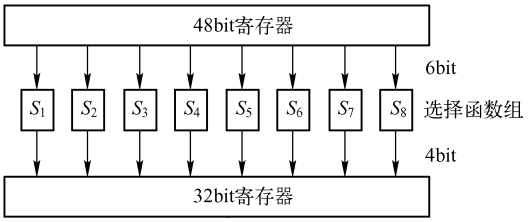

图 3.6 选择压缩运算 S

盒 $S_1 \sim S_8$的选择函数见表 3.3。

表 3.3 选择函数

	行\列	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_1	0	14	4	13	12	15	11	8	3	10	6	1	2	5	9	0	7
	0	15	7	4	14	2	13	1	10	6	12	1	1	9	5	3	8
	2	4	11	4	8	13	6	2	11	15	12	9	7	3	1	0	50
	3	15	12	8	2	4	9	1	7	5	1	13	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

续表

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

对每个盒 S_i，6bit 输入中的第 1bit 和第 6bit 组成的二进制数确定 S_i的行，中间 4 位二进制用来确定 S_i的列。S_i中相应行、列位置的十进制数的 4 位二进制数表示作为输出。

例如，S_2的输入为 101001，则行数和列数的二进制表示分别是 11 和 0100，即第 3 行和第 4 列，S_2的第 3 行、第 4 列的十进制数为 3，用 4 位二进制数表示为 0011，所以 S_2的输出为 0011。置换 P 见表 3.4。

表 3.4　置换 *P*

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

最后需要描述的是用密钥 K 来产生 16 个 48 比特的子密钥 K_i，$1 \leqslant i \leqslant 16$。

子密钥的产生：给定 64bit 的密钥 K，用置换选择 1(PC－1) 作用，去掉了输入的第 8,16,24,32,40,48,56,64 位，这 8bit 是奇偶校验位，并重排实际 56bit 的密钥。将得到的 56bit 数据分成左、右等长的 28bit，分别记为 C_0和 D_0。对 $1 \leqslant i \leqslant 16$，计算 $C_i = LS_i(C_{i-1})$ 和 $D_i = LS_i((D_{i-1})$。将每轮 56bit 数据 C_iD_i用置换选择 2（PC－2）作用，去掉第 9,18,22,25,35,38,43,54 位，同时重排剩下的 48bit，输出作为 K_i，产生子密钥的密钥编排算法如图 3.7 所示。这里的 LS_i表示循环左移 1 位（i = 1,2,9,16 时）或 2 位（其他情况），置换选择 1（PC－1）和置换选择

2（PC－2）见表3.5。

表3.5 置换选择PC－1和置换选择PC－2

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

进行16轮完全类似的迭代运算后，将所得左、右长度相等的两半L_{16}和R_{16}交换，得到64bit数据$R_{16}L_{16}$，用初始逆置换IP^{-1}进行置换，产生密文数据组。置换表自左向右、自上而下的64个位置对应64bit数据组。置换表中的数字表示将64bit数据组中的该数字所在位置的比特转换为该数字表示的位置的比特。

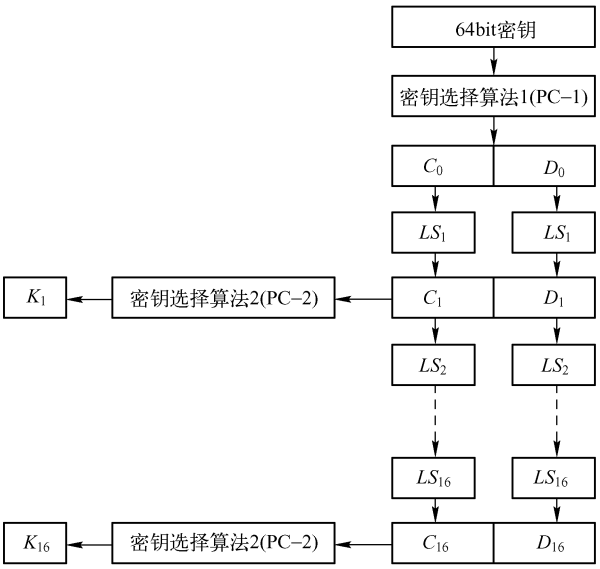

图3.7 密钥编排算法

解密：由于DES算法是在Feistel网络结构的输入和输出阶段分别添加初始置换IP和初始逆置换IP^{-1}而构成的，所以它的解密使用与加密同样的算法，只是子密钥的使用次序相反。

3.2.3 DES的工作模式

在实际应用中，DES根据其加密算法所定义的明文分组的大小（64bit），将数据分割成若干64bit的加密区块，再以加密区块为单位，分别进行加密处理。如果最后剩下不足一个

区块的大小，称之为短块。关于短块的处理方法一般有填充法、序列密码加密法、密文挪用技术。根据数据加密时，每个加密区块间的关联方式来区分，可以分为 4 种加密模式，包括 ECB（Electronic Code Book）、CBC（Cipher Block Chaining）、CFB（Ciphertext FeedBack）及 OFB（Output FeedBack）。

1. 电子密文方式（ECB）

ECB 模式是分组密码的基本工作方式。在 ECB 模式下，每一个加密区块依次独立加密，产生独立的密文区块，每一个加密区块的加密结果均不受其他区块的影响，使用此种方式，可以利用平行处理来加速加/解密运算，且在网络传输时任意一个区块有任何错误发生，也不会影响到其他区块传输的结果，这是此模式的优点。

ECB 模式的缺点是容易暴露明文的数据模式。在计算机系统中，许多数据都具有固有的模式，这主要是由数据结构和数据冗余引起的。如果不采取措施，对于在要加密的文件中出现多次的明文，此部分明文若恰好是加密区块的大小，则可能会产生相同的密文，且密文内容若遭剪贴、替换，也不易被发现。

ECB 加密模式如图 3.8 所示。

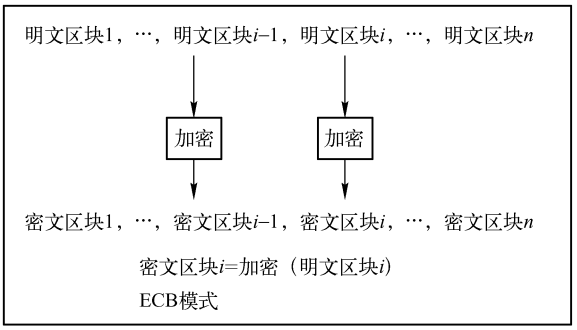

图 3.8　ECB 加密模式

2. 密文分组链接方式（CBC）

第一个加密区块先与初始向量（Initialization Vector，IV）做异或（XOR）运算，再进行加密。其他每个加密区块在加密之前，必须与前一个加密区块的密文做一次异或运算，再进行加密。每一个区块的加密结果均会受到前面所有区块内容的影响，所以即使在明文中出现多次相同的明文，也会产生不同的密文。

再者，密文内容若遭剪贴、替换，或在网络传输过程中发生错误，则其后续的密文将被破坏，无法顺利解密还原，这是这一模式的优点也是缺点。

CBC 加密模式如图 3.9 所示。

其次，必须选择一个初始向量，用以加密第一个区块，且在加密作业时无法利用平行处理来加速加密运算，但其解密运算，因做异或的加密区块结果已存在，则仍可以利用平行处理来加速。

3. 密文反馈方式（CFB）

可以将区块加密算法当作流密码加密器（Stream Cipher）使用，流密码加密器可以按照实际需要，每次加密区块大小可以自定，每一个区块的明文与之前区块加密后的密文做异或后，成为密文。因此，每一个区块的加密结果也受之前所有区块内容的影响，也会使得在明

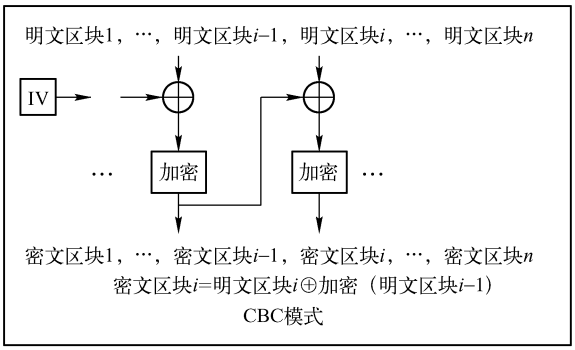

图 3.9　CBC 加密模式

文中出现多次相同的明文均产生不相同的密文。在此模式下，与 CBC 模式一样，为了加密第一个区块，必须选择一个初始向量，且此初始向量必须唯一、每次加密时必须不一样，也难以利用平行处理来加快加密作业。

CFB 加密模式如图 3.10 所示。

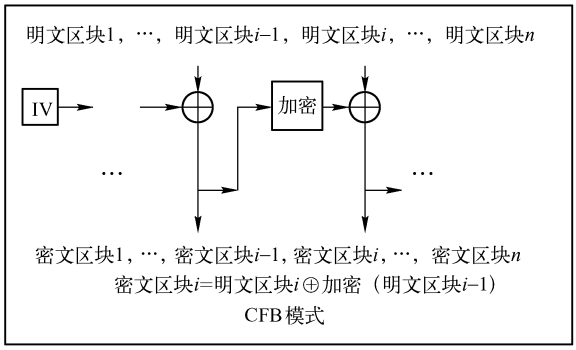

图 3.10　CFB 加密模式

4. 输出反馈模式（OFB）

OFB 与 CFB 大致相同，唯一的差异是每一个区块的明文与之前区块加密后的密文做异或后产生密文。之前区块加密后的密文为独立产生，每一个区块的加密结果不受之前所有区块的内容的影响，如果有区块在传输过程中遗失或发生错误，将不至于无法完全解密，但也会使得在明文中出现多次的明文，均产生相同的密文，也容易遭受剪接攻击。而在此模式下，为了加密第一个区块，必须设置一个初始向量，否则难以利用平行处理来加快加密作业。

容易看出，以上 4 种操作模式有不同的优点和缺点。在 ECB 和 OFB 模式中改变一个明文块将引起相应的密文块的改变，而其他密文块不变。有些情况下这可能是一个好的特性。例如，OFB 模式通常用来加密卫星传输。另一方面，在 CBC 和 CFB 模式中改变一个明文块，那么相应的密文块及其后的所有密文块将会改变，这个特性意味着 CBC 和 CFB 模式适用于鉴别的目的。更明确地说，这些模式能用来产生消息鉴别码，将其附在明文块序列的后面可保护消息的完整性。

OFB 加密模式如图 3.11 所示。

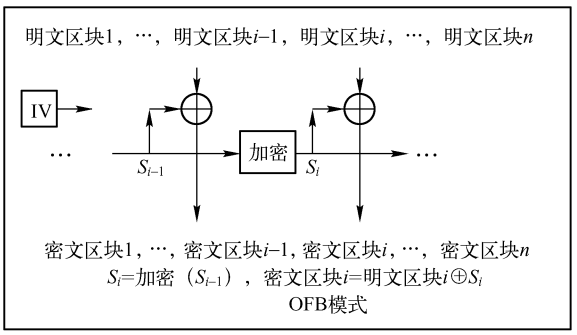

图 3.11　OFB 加密模式

3.2.4　DES 的软/硬件实现

DES 算法明文以 64bit 为分组，密钥长度为 64bit，有效密钥长度为 56bit，其中加密密钥有 8bit 是奇偶校验，DES 算法的加密和解密用的是同一算法，它首先把需要加密的明文划分为每 64bit 的二进制的数据块，用 56bit 有效密钥对 64bit 二进制数据块进行加密，每次加密可对 64bit 的明文输入进行 16 轮的替换和移位后，输出完全不同的 64bit 密文数据。具体流程图如图 3.12 所示。

1. 二进制与字符的转换

当输入的明文 8 个字符进入初始置换之前需要将明文转换为二进制形式再进行 IP 置换。采用移位的方法来进行二进制与字符之间的转换。

```
static void CharToBit(const char input[],int output[],int bits)//把 CHAR 转换为 INT
{
    int temp,i,j;
    for(i=0;i<8;i++)
    {
        temp=(int)input[i];
        for(j=i*8+7;j>=i*8;j--)
        {

            output[j]=temp%2;
            temp=temp/2;

        }
    }}
static void BitToChar(const int intput[],charoutput[],int bits)//把 INT 转换为 CHAR
{
    int i,j;
    for(j=0;j<8;j++)
    {
        for(i=0;i<8;i++)
```

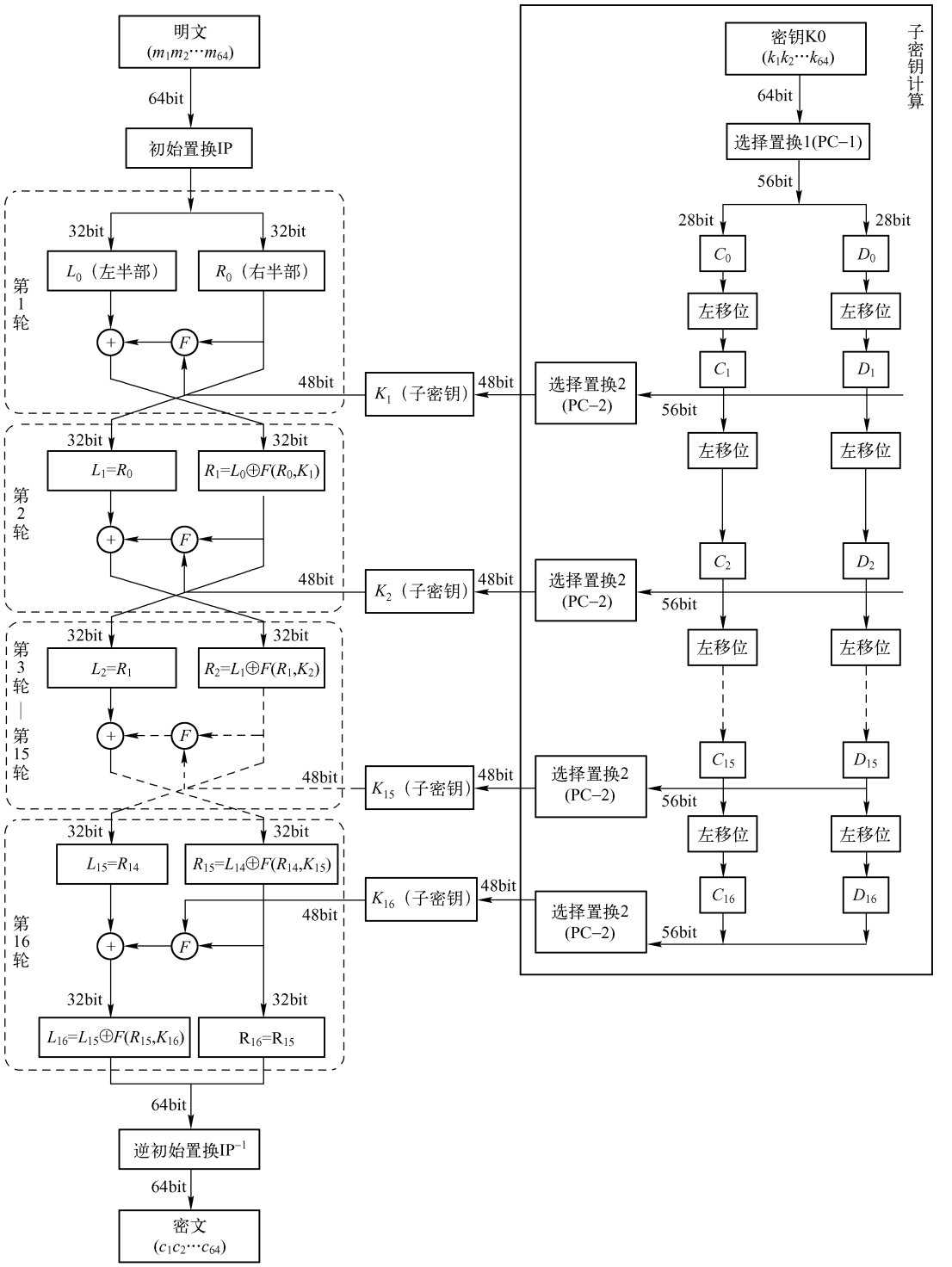

图 3.12　DES 算法总流程

```
        {
            output[j] = output[j] * 2 + intput[i + 8 * j];
        }
    }
}
```

2. 初始置换

第一阶段：初始置换 IP 对于给定的明文 x，通过初始置换 IP（在第一轮迭代运算之前），需要加密的 64 位明文首选通过初始置换 IP 的作用，对输入分组实施置换（即把 58 位置换为第 1 位，…，原来输入明文的第 7 位将作为置换结果的最后一位）获得，并将 x_0 分为两部分，前面 32 位记为 L_0，后面 32 位记为 R_0。

```
const static int IP_Table[64] = {                                    //IP 置换矩阵
        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 };
static void IP(const int input[64],int output[64],int table[64])
    {
        int i;
        for(i = 0;i < 64;i ++)
        {
            output[i] = input[table[i] - 1];//减 1 操作不可少!!
        }
    }
```

3. 轮结构（*F* 函数）

经过 DES 算法第一阶段的初始置换得到 64bit 的块，分为两部分，具体过程如下。下一轮左半部分 32bit L_i 等于上一轮右半部分 32bit R_{i-1}；而下一轮右半部分的 32bit R_i 的计算则先由上一轮右半部分 R_{i-1} 和轮密钥 K_i 输入到 F 函数中进行变换，变换的结果再与上一轮左半部分 L_i 进行异或运算，得到 R_i。经过一轮完整变换之后，左半部分与右半部分共同作为下一轮的输入，总共 16 轮运算。具体流程如图 3.13 所示。

F 函数是整个 DES 加密算法中最重要的部分。参数 In 是 32bit 的输入，参数 K_i 是由初始密钥导出的第 i 轮子密钥，F 函数输出的 32bit 存在于 In 数组中。次函数的实现过程为，首先经过一个扩展运算，然后进行 S 盒替换，再进行 P 盒置换。其中，扩展运算和 P 盒置换的主要作用是增加算法的扩展效果。

```
static void F_func( int input[32],int output[32],int subkey[48])//完成 DES 算法轮变换
{

    int len = 48;
    int temp[48] = {0};
    int temp_1[32] = {0};
    E(input,temp,E_Table);
```

```
    Xor(temp,subkey,len);
    S(temp,temp_1,S_Box);
    P(temp_1,output,P_Table);
}
```

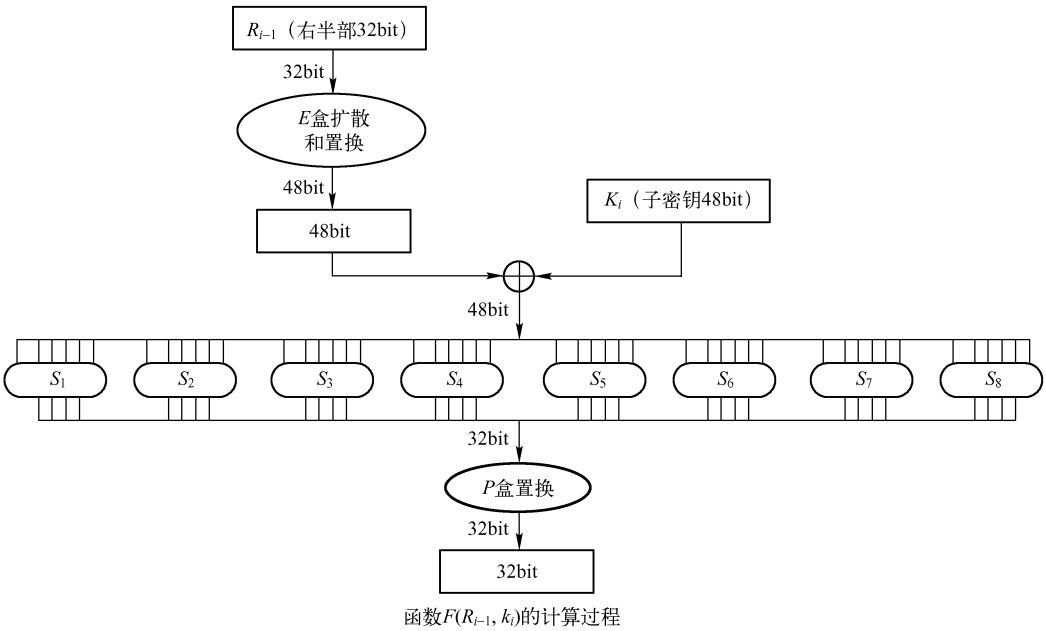

图 3.13　轮结构流程图

（1）*E* 盒置换

f 以前一轮迭代的结果 R_{i-1} 作为输入，首先根据一个固定的扩展函数 *E*（也称为 *E* 盒）“扩展”长度为 48 位的比特串，其中有 16bit 出现了两次。

具体过程：把输入 32bit 按照 8 行 4 列方式依次排列，形成 4 ×8 矩阵，然后 *E* 盒扩展之后输出 6 ×8 矩阵。

```
static void E(const int input[32],int output[48],int table[48])//E 扩展
{
    int i;
    for(i=0;i<48;i++)
    {
        output[i]=input[table[i]-1];
    }

}
```

（2）异或运算

函数 *F* 将扩展置换得到的 48 位输出与子密钥 K_i（48 位）进行异或运算（按位模 2 加）。

若子密钥48 比特位为：001111 011000 111111 001101 001101 110011 111101 001000，则操作如下。

输入：100000 000001 011111 111110 100000 001101 010000 000110

密钥：001111 011000 111111 001101 001101 110011 111101 001000

输出：101111 011001 100000 110011 101101 111110 101101 001110

```
static void Xor(int *INA,int *INB,int len)//异或操作
{
    int i;
    for(i=0;i<len;i++)
    {
        *(INA+i)=*(INA+i)^*(INB+i);
    }
}
```

（3）*S* 盒变换

以6bit 为单位将异或得到的结果分为 8 个小分组，并将它们送入替代函数（即 *S* 盒）中进行替代运算。

DES 算法中共有 8 个 *S* 盒。每一个分组将对应于一个 *S* 盒进行替代运算。具体替代方式：将 *S* 盒的6 位输入定义为 $a_1a_2a_3a_4a_5a_6$，将 a_1a_6 组成一个 2 位二进制数，对应着表中的行号，将 $a_2a_3a_4a_5$ 组成一个 4 位二进制数，对应表中的列号，交叉点的数据就是该 *S* 盒的输出。

S 盒是函数 *f* 的核心所在，同时，也是 DES 算法的关键步骤。实际上除了 *S* 盒以外，DES 的其他运算都是线性的，而 *S* 盒是非线性的，它决定了 DES 算法的安全性。48 位的比特串（分为8 个6 位分组）在经过8 个 *S* 盒进行代替运算后，得到8 个4 位的分组，它们重新组合在一起形成一个 32 位的比特串。这个比特串将进行下一步运算：*P* 盒置换。

示例：*S* 盒输入的 48 比特位为 101111 011001 100000 110011 101101 111110 101101 001110，则分成8 组，每组6 位，则对应的 *S* 盒的行和列号见表3. 6。

表 3. 6　*S* 盒的行和列号

S 盒	1	2	3	4	5	6	……
输入	101111	011001	100000	110011	101101	111110	……
行	3	1	2	3	3	2	……
列	7	12	0	9	5	15	……
输出	0111	0110	0011	0100	0010	0110	……

经过 *S* 盒作用，输出 32 比特位：0111 0110 0011 0100 0010 0110 1010 0001。

```
static void S(const int input[48],int output[32],inttable[8][4][16])//S 盒压缩
{
    int i=0;
    int j=0;
    int INT[8];
```

```
    for( ;i<48;i=i+6)
    {

    INT[j]=table[j][2*(input[i])+(input[i+5])][2*2*2*(input[i+1])+2*2*(input
[i+2])+2*(input[i+3])+(input[i+4])];
        j++;}
    for(i=0;i<8;i++)
    {
        j=4*(i+1)-1;
        while(INT[i])
        {
            output[j]=INT[i]%2;
            INT[i]=INT[i]/2;
        }

    }
}
```

(4) P 盒置换

是将 S 盒输出的 32 位比特串根据固定的置换 P (也称为 P 盒) 置换到相应的位置，它也称为直接置换。

```
const static int P_Table[32]={                                    //P 盒
        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};
```

从 S 盒出来后经置换 P 重排。

输入：0111 0110 0011 0100 0010 0110 1010 0001

输出：0100 0100 0010 0000 1001 1110 1001 1111

```
static void P(const int input[32],int output[32],int table[32])     //P 置换
{
    int i;
    for(i=0;i<32;i++)
    {
        output[i]=input[table[i]-1];
    }


}
```

4. 逆初始置换

逆 (初始) 置换是初始置换 IP 的逆置换，记为 IP^{-1}。在对 16 轮迭代的结果 ($R_{16}L_{16}$)，再使用逆 (初始) 置换 IP^{-1}后，得到的结果即可作为 DES 加密的密文 Y 输出，即 $Y=IP^{-1}(R_{16},L_{16})$。