第 3 章 分组密码体制

3.1 概　　述

设明文消息编码表示后的二元数字序列为 $M=\{x_0,x_1,\cdots,x_k,\cdots\}$。分组密码首先将 M 划分成长为 m 的组块 $\boldsymbol{B}_i=(x_{i\times m},x_{i\times m+1},\cdots,x_{i\times m+m-1})$(长为 m 的向量)的集合,然后各组分别在密钥 $\boldsymbol{K}$ 的控制下变换成长度为 n 的组块 $\boldsymbol{C}_i=(y_{i\times n},y_{i\times n+1},\cdots,y_{i\times n+n-1})$(长为 n 的向量)。若 $n>m$,则为有数据扩展的分组密码;若 $n<m$,则为有数据压缩的分组密码;若 $n=m$,则为无数据扩展和压缩的分组密码,通常研究的是最后这种情况。

通常记 $E_K(\cdot)$为密钥为 $\boldsymbol{K}$ 时的加密函数,称 $E_K(\cdot)$的逆函数为密钥为 $\boldsymbol{K}$ 时的解密函数,记为 $D_K(\cdot)$。分组密码的数学模型如图 3.1 所示。

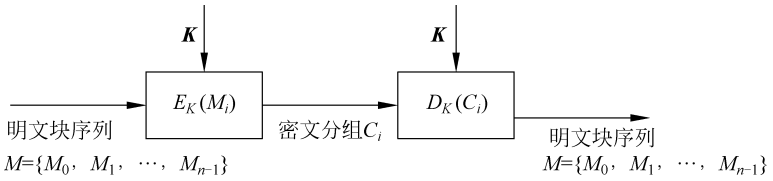


图 3.1　分组密码的数学模型

分组密码具有速度快、易于标准化和便于软硬件实现等特点,通常是信息与网络安全中实现数据加密和认证的核心体制,在计算机通信和信息系统安全领域有着最广泛的应用。

3.2 分组密码的设计原则与评估

3.2.1 分组密码的设计原则

分组密码的设计就是找到一种算法,能在密钥控制下从一个足够大且足够“好”的置换子集合中简单而迅速地选出一个置换,用来对当前输入的明文进行加密变换。一个好的分组密码应该既难破译又容易实现。加密函数 $E_k(\cdot)$和解密函数 $D_k(\cdot)$都必须是容易计算的,但是要从方程 $y=E_k(x)$或 $x=D_k(y)$中解出密钥 k 应该是一个困难问题。分组密码的设计原则可以分为安全性原则和实现原则。

1. 针对安全性的一般设计原则

分组长度和密钥长度:当明文分组长度为 n 位时,至多需要 2^n 个明文—密文对就可以破解该密码。同理,当密钥长度为 n 位时,对一个截获的密文,至多需要试验 2^n 个密钥就可以破解所使用的密钥。因此从安全性角度来考虑,明文分组长度和密钥长度应尽可能大。

扰乱原则：又称混淆原则，是指人们所设计的密码应使得密钥和明文、密文之间的依赖关系足够复杂，以至于这种依赖性对密码分析者来说是无法利用的。

扩散原则：人们所设计的密码应使得密钥的每一位影响密文的许多位，以防止对密钥进行逐段破译；而且明文的每一位也应影响密文的许多位，以便隐藏明文的统计特性。

另外还有一个重要的设计原理：密码体制必须能抵抗现有的所有攻击方法。

2. 针对实现的设计原则

分组密码可以用软件或硬件来实现。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。

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64、128和256位等(想想看为什么)。在软件实现中，按位置换是难于实现的，因此应尽量避免使用。子块所进行的密码运算应该是一些易于软件实现的，最好是标准处理器所具有的基本指令，如加法、乘法和移位等。

硬件实现的设计原则：加密和解密的相似性，即加密和解密过程的区别应该仅仅在于密钥的使用方式不同，以便同样的器件既可用来加密，又可用来解密；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于用超大规模集成电路实现。

3.2.2　分组密码的评估

对分组密码的评估主要有3个方面：安全性；性能；算法和实现特性。

安全性是评估中的最重要因素，包括下述要点：算法抗密码分析的强度，可靠的数学基础，算法输出的随机性，与其他候选算法比较时的相对安全性。

算法性能主要包括在各种平台上的计算效率和对存储空间的需求。计算效率主要是指算法在用软硬件实现时的执行速度。

算法和实现特性主要包括：灵活性、硬件和软件适应性、算法的简单性等。算法的灵活性是指可以满足更多应用的需求，例如：(1)密钥和分组长度可以进行调整；(2)在许多不同类型的环境中能够安全、有效地实现；(3)可以为序列密码、HASH函数等的实现提供帮助；(4)算法必须能够用软件和硬件两种方法实现。另外，算法设计应相对简单也是一个评估因素。

3.3　分组密码的基本设计结构

分组密码算法各不相同，但有些重要的结构在很多分组密码算法中都会有所应用。当今绝大多数的分组密码都是乘积密码。所谓乘积密码，就是以某种方式连续使用两个或多个密码，以使得所得到的最后结果或乘积比使用其中任意一个密码都更强。乘积密码通常伴随一系列置换与代换操作，常见的乘积密码是迭代密码，即对同一个密码进行迭代使用。下面首先介绍分组密码设计所使用的四种常用结构，然后再介绍基本结构的细化。

3.3.1　Feistel结构

Feistel结构是Horst Feistel[①]于1973年在设计Lucifer分组密码时发明的，随着分组

① Horst Feistel，德国密码学家，曾在IBM公司从事密码设计工作，是最早研究分组密码设计和理论的非政府研究人员之一。

密码算法 DES(Data Encryption Standard)的使用而流行。许多分组密码算法采用了 Feistel 结构,以下会对其中部分算法进行介绍。

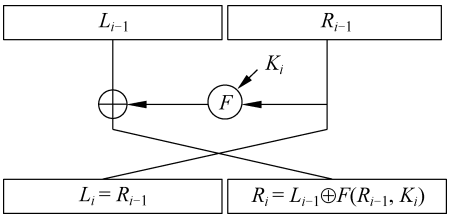


图 3.2 Feistel 结构的第 i 轮

Feistel 结构是典型的迭代密码,图 3.2 是其一轮的结构示意图。对一个分组长度为 $2n$ 位的 r-轮 Feistel 型密码,其加密过程如下:

(1) 给定明文 P,记 $P=L_0R_0$,这里 L_0 是 P 的最左边 n 位,R_0 是 P 的最右边 n 位。

(2) 进行 r 轮完全相同的运算,如第 i($1 \leqslant i \leqslant r$)轮的运算如下:

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

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

这里的 F 是轮函数,$K_1, K_2, \cdots, K_r$ 是由种子密钥生成的子密钥。

(3) 输出密文 $C=R_rL_r$。

Feistel 结构保证了加密过程的可逆性,因为:

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

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

再注意到,Feistel 结构在加密的最后一轮略去"左右交换",可以看出 Feistel 结构的解密与加密是完全一样的,除了所使用的子密钥的顺序正好相反。

"加解密相似"是 Feistel 型密码在实现上的优点。但另一方面,Feistel 型密码的扩散要慢一些。例如,算法至少需要两轮才能改变输入的每一位。这导致 Feistel 型密码的轮数较多,一般大于或等于 16。

3.3.2 SPN 结构

SPN(Substitution-Permutation Network)结构也是一种特殊的迭代密码,著名分组密码算法 AES(Advanced Encryption Standard)就是基于此结构。SPN 结构分为两层:第一层为 S 层,也称为替换层,主要起扰乱的作用;第二层为 P 层,也称为置换层,主要起扩散的作用。

在这种密码的每一轮中,首先输入被作用于一个由密钥控制的可逆函数 S,然后再被作用于一个置换(或一个可逆的线性变换)P。SPN 结构如图 3.3 所示,其中 X_i 表示第 i 轮的输出,也就是第 $i+1$ 轮的输入。

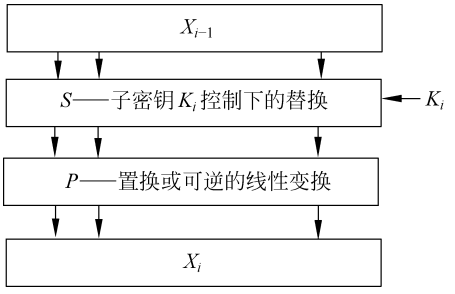


图 3.3 SPN 型密码的一轮加密过程

SPN 结构和 Feistel 结构相比,可以得到更快的扩散,但是 SPN 密码的加解密通常不相似。

3.3.3 Feistel 结构的变种

在基本 Feistel 结构的基础上,又发展出多种衍生结构,如三类广义 Feistel 结构、收缩

Feistel 结构、扩张 Feistel 结构等[2][11]。以下介绍三类 Feistel 结构变种。

Type-1 型广义 Feistel 结构继承了 Feistel 结构加解密相似的特点。该结构提出后受到了广泛的关注，CAST-256 算法就采用了 Type-1 型广义 Feistel 结构。该结构的表达式为

$$g_i(X_{i,1},X_{i,2},\cdots,X_{i,d})=\begin{cases}F_i(X_{i,1}\oplus K_i)\oplus X_{i,2},X_{i,3},\cdots,X_{i,d},X_{i,1}, & \text{若 } 1\leqslant i<r;\\ X_{i,1},F_i(X_{i,1}\oplus K_i)\oplus X_{i,2},X_{i,3},\cdots,X_{i,d}, & \text{若 } i=r\end{cases}$$

以四路 Type-1 结构为例，其结构如图 3.4 所示。

Type-2 型广义 Feistel 结构同样继承了 Feistel 结构加解密相似的优点。CIEFIA 算法就采用了 Type-2 型广义 Feistel 结构。同样，Type-2 型广义 Feistel 结构由图 3.5 通过 r 轮迭代得到，最后一轮取消拉线变换。

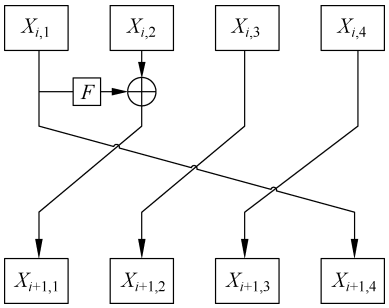


图 3.4　四路 Type-1 结构

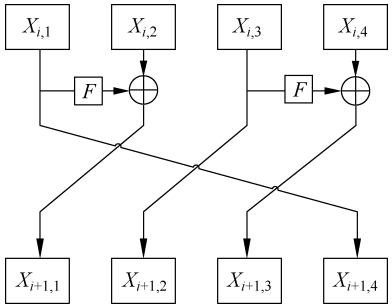


图 3.5　四路 Type-2 结构

设第 i 圈从左向右数的第 j 个轮函数为 $F_{i,j}(1\leqslant j\leqslant d/2)$，轮子密钥为 $K_{ij}\in\{0,1\}^{n/d}$，第 i 圈的 d 个输入分别为 $X_{i,1},X_{i,2},\cdots,X_{i,d}\in\{0,1\}^{n/d}$。每一圈的函数表达式为

$$g_i(X_{i,1},X_{i,2},\cdots,X_{i,d})=\begin{cases}(F_{i,1}(X_{i,1}\oplus K_{i,1})\oplus X_{i,2},X_{i,3},F_{i,2}(X_{i,3}\oplus K_{i,2})\oplus X_{i,4},\cdots,\\ \quad F_{i,\frac{d}{2}}(X_{i,d-1}\oplus K_{i,\frac{d}{2}})\oplus X_{i,d},X_{i,1}), & \text{若 } 1\leqslant i<r;\\ (X_{i,1},F_{i,1}(X_{i,1}\oplus K_{i,1})\oplus X_{i,2},X_{i,3},F_{i,2}(X_{i,3}\oplus K_{i,2})\oplus X_{i,4},\cdots,\\ \quad F_{i,\frac{d}{2}}(X_{i,d-1}\oplus K_{i,\frac{d}{2}})\oplus X_{i,d}), & \text{若 } i=r\end{cases}$$

Type-3 型广义 Feistel 结构与 Type-1 型和 Type-2 型广义 Feistel 结构一样，其与 Type-1 型和 Type-2 型最大的区别为每圈的圈函数中含有的非线性轮函数数目。Type-1 型广义 Feistel 结构每圈有一个非线性轮函数，Type-2 型广义 Feistel 结构每圈有 $d/2$ 个非线性轮函数，Type-3 型广义 Feistel 结构每圈有 $d-1$ 个非线性轮函数。Type-3 型广义 Feistel 结构如图 3.6 所示。

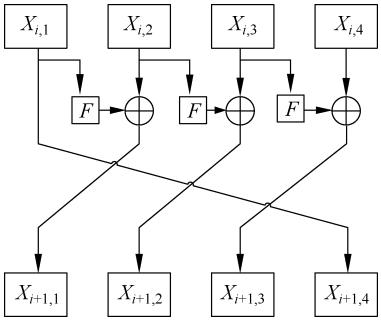


图 3.6　四路 Type-3 结构

设第 i 圈从左向右数的第 j 个轮函数为 $F_{i,j}(1\leqslant j\leqslant d/2)$，轮子密钥为 $K_{i,1},K_{i,2},\cdots,K_{i,d-1}\in\{0,1\}^{n/d}$，第 i 圈的 d 个输入分别为 $X_{i,1},X_{i,2},\cdots,X_{i,d}\in\{0,1\}^{n/d}$。第 i 圈的函数表达式如下：

$$g_i(X_{i,1},X_{i,2},\cdots,X_{i,d})=\begin{cases}(F_{i,1}(X_{i,1}\oplus K_{i,1})\oplus X_{i,2},F_{i,2}(X_{i,2}\oplus K_{i,2})\oplus X_{i,3},\cdots,\\ \quad F_{i,d-1}(X_{i,d-1}\oplus K_{i,d-1})\oplus X_{i,d-1},X_{i,1}), & \text{若 } 1\leqslant i<r;\\ (X_{i,1},F_{i,1}(X_{i,1}\oplus K_{i,1})\oplus X_{i,2},F_{i,2}(X_{i,2}\oplus K_{i,2})\oplus X_{i,3},\cdots,\\ \quad F_{i,d-1}(X_{i,d-1}\oplus K_{i,d-1})\oplus X_{i,d-1}), & \text{若 } i=r\end{cases}$$

3.3.4 Lai-Massey 结构

Lai-Massey 结构由 Serge Vaudenay① 提出,该结构由模加、减运算和一个正型置换 σ 构建而成。实际上,Lai-Massey 结构出自 Lai② 和 James Lee Massey③ 在 1990 年共同提出的 IDEA 分组密码算法[13],在当时被称为 PES(Proposed Encryption Standard,推荐加密标准)。2000 年,Vaudenay 提取出 IDEA 中的结构来构建一般化的密码模型[3],并且用原设计者的名字为之命名。Lai-Massey 结构虽然不如 Feistel 结构著名,但其优点为实现代价低,运算速度快。Lai-Massey 结构的运算有群中加和减,这里仅仅对⊕运算的情况进行介绍。Lai-Massey 结构如图 3.7 所示。

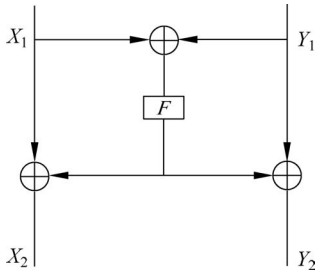


图 3.7 Lai-Massey 结构

令 $x\in\{0,1\}^{2n}$,F 为轮函数,定义基本 Lai-Massey 结构 $\mathrm{LM}_p(x,y)\in P_{2n}$ 如下:

$$\mathrm{LM}_p(x,y)=(x\oplus F(\Delta,K),y\oplus F(\Delta,K)),$$

其中 $\Delta=x\oplus y$

因为 $x_0\oplus y_0=x_1\oplus y_1$,任意轮的 L-M 结构都不具有伪随机性。这个问题可以通过对左侧的输入加入一个 σ 函数,对 x 做一个简单变换来解决。例如,$\sigma(x_\mathrm{L},x_\mathrm{R})=(x_\mathrm{R},x_\mathrm{L}\oplus x_\mathrm{R})$并且 Lai-Massey 结构中的模运算也统一为异或运算时,有

$$\mathrm{LM}_p(x,y)=((\sigma(x_\mathrm{R}\oplus F_\mathrm{R}(\Delta,K))\,\|\,(x_\mathrm{L}\oplus x_\mathrm{R}\oplus F_\mathrm{R}(\Delta,K)\oplus F_\mathrm{L}(\Delta,K)),y\oplus F(\Delta,K))$$

Lai-Massey 结构是一个典型的密码结构,被广泛应用于算法领域,其中包括分组密码算法 IDEA 和 FOX[3]等。由 Vaudenay 等人对 Lai-Massey 结构的分析结果可知,加入了上述变换的四轮 Lai-Massey 结构在传统的安全性假设下可以实现强伪随机性,而三轮 Lai-Massey 结构只能具有伪随机性[12]。

3.3.5 基本设计结构的细化

实际上,在设计现代分组密码算法时,很多时候会基于多种结构,这些结构会进一步考虑其中涉及的子模块函数所使用的结构,从而得到具体细化的整体结构。其中,通常把 Feistel 结构和 SPN 结构结合起来,就可以得到一些细化的分组密码算法的整体结构[4]:

(1) Feistel-Feistel 结构:算法整体结构是 Feistel 结构,轮函数 F 也采用 Feistel 结构。特别地,三轮 Feistel 结构具有伪随机性。

(2) Feistel-SPN 结构:算法整体结构是 Feistel 结构,轮函数 F 采用 SPN 结构。

---

① Serge Vaudenay,法国密码学家,于 2006 年当选国际密码学研究协会理事。

② 来学嘉,上海交通大学教授,中国密码学家。

③ James Lee Massey,信息理论家和密码学家,苏黎世联邦理工学院数字技术名誉教授。

(3) SPN-Feistel 结构：算法的整体结构是 SPN 结构，轮函数 F 中嵌套 Feistel 结构。

除此之外，还有其他一些设计结构，如 SPN-SPS 结构（其中 SPS 是指 Substitution-Permutation-Substitution 函数）、将广义 Feistel 结构和 SPN 结构结合的 GFN-SPN 结构等。

3.4　数据加密标准 DES

1972 年，美国国家标准局（NBS）制定了一个保护计算机和通信开发计划，准备开发一个标准的密码算法，来规范密码技术应用的混乱局面。要求这个算法能够被测试和验证，不同设备间可以互操作，且实现成本低。

1973 年 5 月 15 日，NBS 在《联邦记事》上发布了征集保密传输存储系统中计算机数据密码算法的请求及相关的技术、经济和兼容性要求如下：

(1) 算法具有较高的安全性，安全性仅依赖于密钥，不依赖于算法；

(2) 算法完全确定，且易于理解；

(3) 算法对任何用户没有区别，适合于各种用途；

(4) 实现算法的电子器件必须很经济；

(5) 算法必须能够验证和出口。

但是，由于公众此前对密码知识的缺乏，所以，虽反响强烈，但提交的方案均不理想。

1974 年 8 月 27 日，NBS 再次发布征集公告，第一次 IBM 公司提交了一个良好的候选算法。该算法是 Luciffer 密码算法的改进，由 Feistel 于 1971 年设计。1975 年 3 月 17 日，NBS 在《联邦记事》上公布了这一算法的细节，随后发表通知，征求公众的评论。

尽管受到了强烈的批评和责难，1976 年 11 月 23 日，这一算法还是被采纳为联邦标准，并授权在非密级的政府通信中使用。该标准的正式文本 FIPS PUB46（DES）在 1977 年 1 月 15 日公布，其他文件如 FIPS PUB81（DES 工作方式）、FIPS PUB74（DES 实现和使用指南）、FIPS PUB112（DES 口令加密）、FIPS PUB113（DES 计算机数据监别）等联邦信息处理标准也相继于 20 世纪 80 年代初公布。1981 年，美国国家标准研究所（ANST）批准 DES 作为私营部门的数据加密标准，称为 DEA，相关文件如 ANSI X3.92（DEA）、ANSI X3.106（DEA 工作方式）、ANSI X3.105（网络加密标准）等相继公布。

3.4.1　算法描述

DES 是一种明文分组为 64 位、有效密钥为 56 位、输出密文为 64 位的，具有 16 轮迭代的分组对称密码算法。DES 由初始置换、16 轮迭代和初始逆置换组成。

1. DES 的基本运算

1) 初始置换 IP 和初始逆置换 IP^{-1}

表 3.1 和表 3.2 分别给出了初始置换 IP 和初始逆置换 IP^{-1}。从表中容易看出这两个置换是互补的。

表 3.1　初始置换 IP

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4

续表

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

表 3.2 初始逆置换 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

2) E-扩展运算

E-扩展运算是扩位运算,将32位扩展为48位,用方阵形式可以容易地看出其扩展位,其中中间四列为原始输入数据,如表3.3所示。

表 3.3 E-扩展运算

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

3) S-盒运算

S-盒运算由8个S-盒函数构成,如下:

$$S(x_1x_2\cdots x_{48})=S_1(x_1\cdots x_6)\,\|\,S_2(x_7\cdots x_{12})\,\|\cdots\|\,S_8(x_{43}\cdots x_{48})$$

其中,每一个S-盒都是6位的输入、4位的输出。$S_i(h_1\cdots h_6)$的值对应表3.4的s_i中$(h_1h_6)_2$行和$(h_2h_3h_4h_5)_2$列上的值,其中$(h_1h_6)_2$和$(h_2h_3h_4h_5)_2$为二进制表示,即,$S_i(h_1\cdots h_6)=s_i((h_1h_6)_2,(h_2h_3h_4h_5)_2)$。

表 3.4 S-盒运算

		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

续表

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

4）P-置换

P-置换是对 8 个 S-盒的输出进行变换，可以扩散单个 S-盒的效果，如表 3.5 所示。

表 3.5　P-置换

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

2. DES 结构

DES 是一个 16 轮的 Feistel 型密码，它的分组长度为 64 位，密钥长度为 56 位。在进行 16 轮加密之前，先对明文做一个固定的初始置换 IP，$\mathrm{IP}(M)=L_0R_0$。在 16 轮加密之后，对比特串 $L_{16}R_{16}$ 换位为 $R_{16}L_{16}$ 后做逆置换 IP^{-1} 来给出密文 C，如图 3.8 所示。DES 的一轮加密如图 3.9 所示。

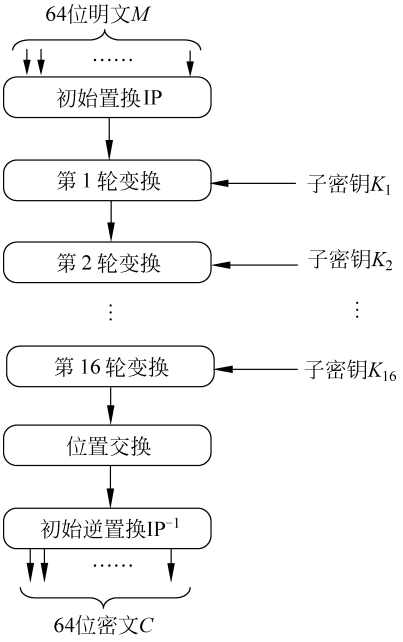


图 3.8 DES的总体结构

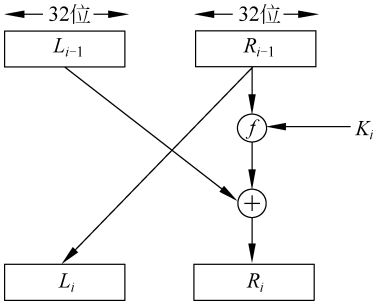


图 3.9 DES的轮函数

其中 f 函数为 $f(R,K_i)=P(S(K_i \oplus E(R)))$,如图 3.10 所示。

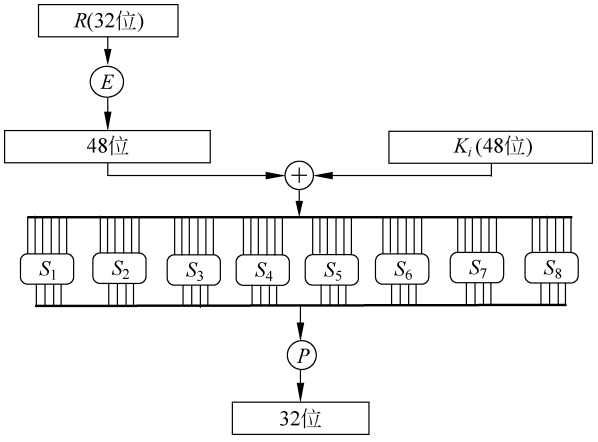


图 3.10 DES的 f 函数

3.4.2 密钥扩展算法

DES的每一轮都使用不同的、从算法密钥 K 导出的48位子密钥 K_i(也称轮密钥)。密钥是一个64位的分组,但是其中8位用于奇偶校验,所以密钥的有效位只有56位。子密钥的生成过程如下:

(1) 给定一个64位的密钥 K,删掉8个校验位并利用一个固定的置换PC-1(见表3.6)来置换剩下的56位,记 PC-1(K)$=C_0D_0$,这里 C_0 是 PC-1(K)的前28位,D_0 是后28位。

(2) 对每一个 i($1 \leqslant i \leqslant 16$),计算

$$C_i = \mathrm{LS}_i(C_{i-1})$$

$$D_i = \mathrm{LS}_i(D_{i-1})$$

$$K_i = \text{PC-2}(C_i D_i)$$

其中 LS_i 表示一个或两个位置的左循环移位，LS_i 的具体取值见表 3.7；PC-2 是另一个固定置换，如表 3.8 所示。密钥方案的计算过程如图 3.11 所示。

表 3.6　置换选择 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

表 3.7　LS_i 的取值

轮数(i)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
LS_i	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

表 3.8　置换选择 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

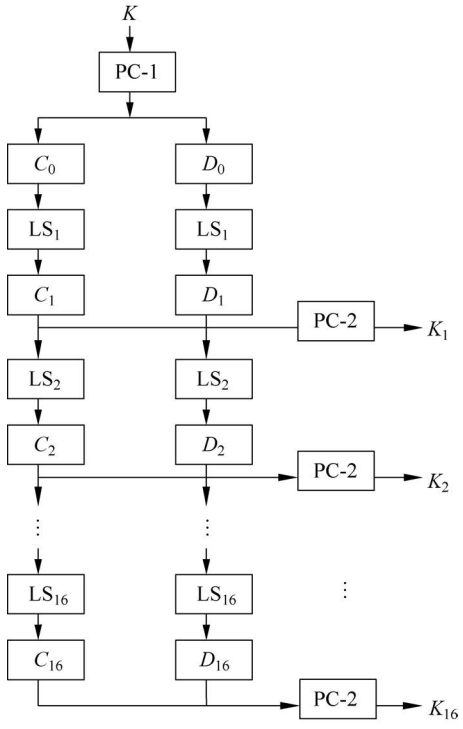


图 3.11　子密钥生成算法

3.4.3 三重 DES 算法

DES 运用了置换、替代、代数等多种密码技术,算法结构紧凑,条理清楚,而且加密与解密算法类似,便于工程实现。

然而,从 DES 诞生之日起,人们就对它的安全持怀疑态度,并展开了激烈的争论,主要表现在以下几点。

1. S-盒的设计准则

这个问题涉及美国国家安全局(NSA),他们修改了 IBM 公司的 S-盒。虽然修改后的 S-盒满足 IBM 公司关于 S-盒的设计原则,但是,人们还是怀疑 NSA 在 S-盒中嵌入了陷门,使得 NSA 可以借助于这个陷门及 56 位的短密钥解密 DES。

在 1990 年以前,S-盒的设计准则一直没有公布,直到差分密码分析方法发表后才公布。公布的 S-盒设计准则如下。

P1:S-盒的每一行是整数 0,1,…,15 的一个置换;

P2:没有一个 S-盒是它输入的线性或仿射函数;

P3:改变 S-盒的一个输入位至少会引起两位输出的改变;

P4:如果固定输入的最左边和最右边的 2 位,变换中间 4 位,则每个可能的 4 位输出只能得到一次;

P5:如果两个输入仅中间 2 位不同,则它们的输出至少有 2 位不同;

P6:对任何一个 S-盒,如果两个输入的前两位不同,而最后两位相同,则两个输出一定不同;

P7:对任何一个 S-盒,如果固定一个输入位保持不变而使其他 5 位输入变化,观察一个固定输出位的值,使这个输出位为 D 的输入的数目与使这个输出位为 I 的输入的数目总数接近相等;

P8:在有相同且非零的差分的 32 对输入中,至多有 8 对具有相同的输出差分。

但是,S-盒的设计准则还没有完全公开,我们仍然不知道 S-盒的构造中是否使用了进一步的设计准则,这也使得人们的猜测无法消除。

2. 56 位有效密钥太短

对 DES 的批评中,最中肯的是关于 DES 密钥长度的争议。实际上,在 IBM 公司最初向 NSA 提交的方案当中,密钥的长度为 112 位,但在 DES 成为一个标准时,密钥被削减至 56 位。

56 位的密钥确实太短,下面两个事件很有说服力。

(1) 分布式穷举攻击:1997 年 1 月 28 日,美国数据安全公司 RSA 在 Internet 上开展了一项名为“秘密密钥挑战”的竞赛,悬赏一万美元来破解一段 DES 密文,得到了许多网络用户的积极响应。美国科罗拉多州的程序员 R. Verser 设计了一个可以通过互联网分段运行的密钥搜索程序,组织了一个称为 DESCHALL 的搜索行动,成千上万志愿者加入该计划中。第 96 天,即竞赛发布后的第 140 天,1997 年 6 月 17 日晚上,美国盐湖城的 M. Sanders 成功找到了密钥并解密出明文。

(2) 专用搜索机:1998 年 5 月,美国电子边境基金会宣布已将一台价值 20 万美元的计算机改装成专用设备,仅用 56 小时就破译了 DES。这更加直接地说明了 56 位密钥太短,彻

底宣布了 DES 的终结。

3. 弱密钥和半弱密钥

如果对于初始密钥 k，生成的 16 个子密钥都相同，则称 k 是弱密钥。显然，$\mathrm{DES}_k(\mathrm{DES}_k(x))=x$。在 DES 中有 4 个弱密钥。如果有一对密钥 k_1、k_2，使得：$\mathrm{DES}_{k_2}(\mathrm{DES}_{k_1}(x))=\mathrm{DES}_{k_1}(\mathrm{DES}_{k_2}(x))=x$，则称 k_1、k_2 是(互逆的)半弱密钥。在 DES 中有 12 个半弱密钥。

虽然 DES 存在弱密钥，但这并不是较大的安全问题。弱密钥与半弱密钥的数量与密钥总量相比是微不足道的。如果随机地选择密钥，那么选中这些弱密钥或半弱密钥的概率可以忽略不计。而且，也可以在密钥产生时进行检查，确保不使用弱密钥或半弱密钥作为 DES 的密钥。

4. 代数结构存在互补对称性

由 DES 中函数 f 的结构可知：

$$f(R_{i-1},K_i)=f(\overline{R_{i-1}},\overline{K_i})$$

其中 $\overline{K}$ 表示 K 的按位取反。再考虑 DES 的结构，有：

$$\mathrm{DES}_{\overline{K}}(\overline{M})=\overline{\mathrm{DES}_K(M)}$$

这种特性称为互补对称性。

由于互补对称性，攻击者在进行穷举攻击时仅需试验所有 2^{56} 个密钥的一半。

为了提高 DES 算法的安全性，人们提出了 DES 的许多改进方案。其中，称为三重 DES 的多重加密算法是 DES 的一个重要的改进算法。多重加密使用同一算法，在多重密钥的作用下，多次加密同一个明文分组。在多重密钥彼此不同且独立的情况下，多重加密对提高密码的安全性有所帮助。

三重 DES 最主要的一般结构如图 3.12 所示。

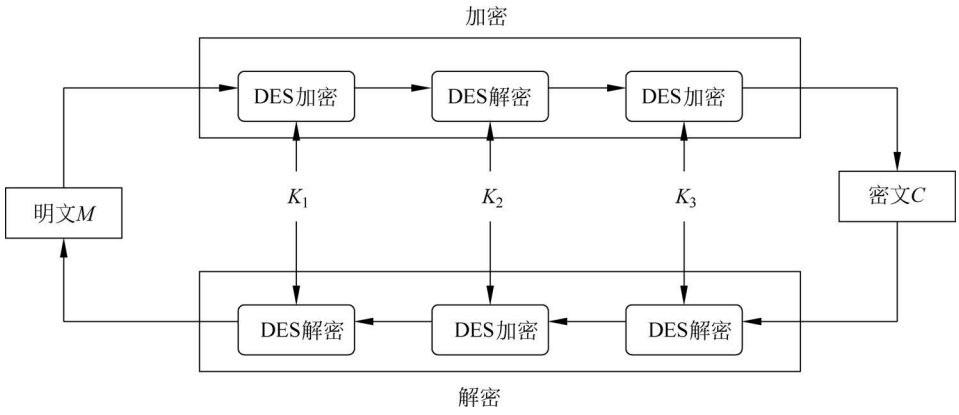


图 3.12　三重 DES 最主要的一般结构

即三重 DES 的加密过程为

$$C=E_{K_3}(D_{K_2}(E_{K_1}(M)))$$

解密过程为

$$M=D_{K_1}(E_{K_2}(D_{K_3}(C)))$$

再根据 K_1 是否等于 K_3，可以把三重 DES 分为两个密钥和三个密钥两种类型。能不

能使用二重DES来代替两个密钥的三重DES呢？答案是否定的，因为存在一种简单的攻击方式使得二重DES的有效密钥长度远远小于112，具体原因请读者自己思考。

三重DES的主要优点是密钥长度增加到112或168位，可以有效抵抗DES面临的穷举搜索攻击。其主要缺点是：(1)由于三重加密，处理速度相对较慢；(2)分组长度仍为64位，就效率和安全性而言，与密钥的增长不相匹配，分组长度应更长。

3.5 高级加密标准AES

在DES的安全性难以保证的情况下，1997年1月2日，美国国家标准与技术研究所(NIST)开始了征集DES替代者的工作。该替代者称为高级加密标准，即AES。制定AES的主要目标是确保信息可实现100年的安全性，即加密后的密文在100年内不会被破解，因此其安全性与运算效率需在三重DES之上。1997年9月12日，NIST发布了征集算法的正式公告，要求AES具有128位的分组长度，并支持128、192和256位的密钥长度，而且要求AES能在全世界范围内免费使用。

在截至1998年6月15日提交的21个算法里，有15个满足所有的必备条件并被接纳为AES的候选算法。NIST在1998年8月20日的“第一次AES候选大会”上公布了15个AES的候选算法。1999年3月“第二次AES候选大会”举行，1999年8月，5个候选算法入围了决赛：MARS、RC6、Rijndael、Serpent和Twofish。

2000年4月举行了“第三次AES候选大会”。2000年10月2日，Rijndael被选择为AES。2001年2月28日，NIST宣布关于AES的联邦信息处理标准的草案可供公众讨论。2001年11月26日，AES被采纳为标准，并在2001年12月4日的联邦记录中作为FIPS197公布。

AES的征集过程以其公开性和国际性而闻名。第二次候选算法大会和官方请求公众评审，为候选算法的意见反馈、公众讨论与分析提供了足够的机会，而且这一过程为置身其中的每一个人所称道。15个AES候选算法的作者代表着不同国家：澳大利亚、比利时、加拿大、哥斯达黎加、法国、德国、以色列、日本、韩国、挪威、英国及美国，这正表明了AES的国际性。最终被选为AES的Rijndael算法就是由两位比利时研究者J. Daemen和V. Rijmen提出的。

AES的候选算法根据以下三条主要原则进行评判：

(1) 安全性；

(2) 代价；

(3) 算法与实现特性。

其中，算法的“安全性”无疑是最重要的，因为一个算法如果被发现不安全就不会再被考虑。“代价”指的是各种实现的计算效率(速度和存储需求)，包括软件实现、硬件实现和智能卡实现。“算法与实现特性”包括算法的灵活性、简捷性及其他因素。最后，五个入围最终决赛的算法都被认为是安全的。Rijndael之所以当选是由于它集安全性、高性能、高效率、可实现性及灵活性于一体，被认为优于其他四个算法。

3.5.1　AES算法的数学基础

1. 有限域 GF(2^8)

AES把1字节看成在有限域 GF(2^8)上的一个元素，并采用多项式表示域中的元素。对于1字节 b，其二进制表示为 $b_7b_6b_5b_4b_3b_2b_1b_0$，可以把 b 看成系数在{0,1}中的多项式：

$$b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x+b_0$$

例如，十六进制数{57}(二进制表示为"01010111")该字节对应于如下多项式：

$$x^6+x^4+x^2+x+1$$

下面给出一些例子来说明 GF(2^8)中的运算。

(1) 加法。

在多项式表示中，两个元素的和是一个多项式，其系数是两个元素的系数模2加(即异或)。

例如，{57}和{83}的和为{57}+{83}={D4}，或者采用其多项式记法如下：

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

显然，该加法与简单的以字节为单位的按位异或是一致的。

(2) 乘法。

在多项式表示的域中，有限域 GF(2^8)中的乘法就是两个多项式模一个特定的、次数为8的不可约多项式的乘积。对于AES，这个不可约多项式称为 $m(x)$，且由下式给出：

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

或由十六进制的{11B}表示。

例如，{57}×{83}={C1}，因为

$$\begin{aligned}&(x^6+x^4+x^2+x+1)(x^7+x+1)\\=&x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1\\&(x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1)\bmod m(x)\\=&(x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1)\bmod(x^8+x^4+x^3+x+1)\\=&x^7+x^6+1\end{aligned}$$

(3) x 乘法。

如果用多项式 x 乘以 $b(x)$，有

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

将上面的结果进行模 $m(x)$ 化简就得到 $x\cdot b(x)$。如果 $b_7=0$，则这一化简是恒等运算，即为 $b_6x^7+b_5x^6+b_4x^5+b_3x^4+b_2x^3+b_1x^2+b_0x$；如果 $b_7=1$，则必须减掉 $m(x)$。由此得出 x 乘法可以用字节内左移操作和紧接着的一个与{1B}的有条件的按位异或操作来实现，故 x 乘法的实现效率非常高，该运算记为 xtime(b)。

可以利用 x 乘法来加快 GF(2^8)中任意两个元素的乘法。

【例 3.1】 计算{57}×{13}。

解： 因为

$$\{57\}\times\{13\}=\{57\}\times(\{01\}+\{02\}+\{10\})=\{57\}\times\{01\}+\{57\}\times\{02\}+\{57\}\times\{10\}$$

$$\{57\}\times\{02\}=\text{xtime}(\{57\})=\{AE\}$$

$$\{57\}\times\{10\}=\text{xtime}(\text{xtime}(\text{xtime}(\text{xtime}(\{57\}))))$$
$$=\text{xtime}(\text{xtime}(\text{xtime}(\{AE\})))$$
$$=\text{xtime}(\text{xtime}(\{47\}))$$
$$=\text{xtime}(\{8E\})$$
$$=\{07\}$$

所以

$$\{57\}\times\{13\}=\{57\}\times\{01\}+\{57\}\times\{02\}+\{57\}\times\{10\}$$
$$=\{57\}+\{AE\}+\{07\}$$
$$=\{FE\}$$

2. GF(2^8)域上的多项式

一个字(4字节)可以看作是GF(2^8)域上的多项式,每个字对应一个次数小于4的多项式。

多项式对应的系数简单相加可以实现多项式的相加。由于GF(2^8)中的加法为位异或,因此,两个字的加法是一个简单的按位异或。

乘法则比较复杂。假设有GF(2^8)上的两个多项式:$a(x)=a_3x^3+a_2x^2+a_1x+a_0$和$b(x)=b_3x^3+b_2x^2+b_1x+b_0$,它们的乘积$c(x)=a(x)b(x)$如下:

$$c(x)=c_6x^6+c_5x^5+c_4x^4+c_3x^3+c_2x^2+c_1x+c_0$$

其中

$$c_0=a_0b_0$$
$$c_1=a_1b_0\oplus a_0b_1$$
$$c_2=a_2b_0\oplus a_1b_1\oplus a_0b_2$$
$$c_3=a_3b_0\oplus a_1b_2\oplus a_2b_1\oplus a_0b_3$$
$$c_4=a_3b_1\oplus a_2b_2\oplus a_1b_3$$
$$c_5=a_3b_2\oplus a_2b_3$$
$$c_6=a_3b_3$$

显然,$c(x)$不能再被表示为4字节。通过模一个4次多项式$c(x)$进行化简,这样,乘法的结果可以化简为一个次数小于4的多项式。

在AES中,使用的模数多项式为$M(x)=x^4+1$。由于

$$x^j \bmod (x^4+1)=x^{j \bmod 4}$$

记$a(x)b(x) \bmod (x^4+1)$的结果为$d(x)=a(x)\otimes b(x)$,设$d(x)=d_3x^3+d_2x^2+d_1x+d_0$,则有

$$d_0=a_0b_0\oplus a_3b_1\oplus a_2b_2\oplus a_1b_3$$
$$d_1=a_1b_0\oplus a_0b_1\oplus a_3b_2\oplus a_2b_3$$
$$d_2=a_2b_0\oplus a_1b_1\oplus a_0b_2\oplus a_3b_3$$
$$d_3=a_3b_0\oplus a_2b_1\oplus a_1b_2\oplus a_0b_3$$

其矩阵表示为

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

注意：模 $M(x)=x^4+1$ 不是 GF(2^8)上的不可约多项式，因为 $M(x)=x^4+1=(x+1)(x^3+x^2+x+1)$。因此，与一个固定多项式相乘的乘法将不一定是可逆的。

在 AES 中，这里选择了一个具有逆元的固定多项式进行这样的乘法，从而保证了乘法的可逆性。

3.5.2　算法的总体描述

AES 的分组长度为 128 位，有三种可选的密钥长度，即 128 位、192 位和 256 位。AES 是一个迭代型密码，轮数 N_r 依赖于密钥长度。如果密钥长度为 128 位，则 $N_r=10$；如果密钥长度为 192 位，则 $N_r=12$；如果密钥长度为 256 位，则 $N_r=14$。AES 的密钥长度与加解密轮数之间的对照如表 3.9 所示。

表 3.9　AES 的密钥长度与加解密轮数的对照表

算法名称	密钥长度	加解密轮数(N_r)
AES-128	128	10
AES-192	192	12
AES-256	256	14

AES 中的操作都是以字节为基础的，所有用到的变量都由适当数量的字节组成。中间变量 State 用如下 4×4 字节矩阵表示：

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

下面给出 AES 加密过程的总体描述：

(1) 给定一个明文 M，将 State 初始化为 M，并将轮密钥与 State 异或(称为 AddRoundKey)；

(2) 对前 N_r-1 轮中的每一轮，用 S-盒进行一次替换操作(称为 SubBytes)，对替换的结果 State 做行移位操作(称为 ShiftRows)，再对 State 做列混合变换(MixColumns，也称为列混淆变换)，然后进行 AddRoundKey 操作；

(3) 在最后一轮中依次进行 SubBytes、ShiftRows 和 AddRoundKey 操作；

(4) 将 State 定义为密文 C。

AES 的加密过程可用如下伪代码描述：

```
AESCipher(byte in[16], byte out[16], word w[4 * (Nr + 1)])
//in[]、out[]和 w[]分别表示 AES 加密的输入、输出和子密钥
{
    byte state [4,4]; //中间变量
```

```
    state = in;   //用以列为顺序的输入来初始化中间变量,即 state[r, c] = in[r + 4c], 0 <= r、
c < 4
    AddRoundKey (state, w[0, 3]);

    //前 Nr - 1 轮加密
    for(int round = 1; round < Nr - 1; round++)
    {
        SubBytes (state);
        ShiftRows (state);
        MixColumns (state);
        AddRoundKey (state, w[4 * round, 4 * round + 3]);
    }

        //最后一轮加密
        SubBytes (state);
        ShiftRows (state);
        AddRoundKey (state, w[4 * Nr, 4 * Nr + 3]);

        out = state;
}
```

AES的解密算法与加密算法有较大的不同,它的伪代码描述如下:

```
AESDecipher(byte in[16], byte out[16], word w[4 * ( Nr + 1)])
{
    byte state[4, 4];

    state = in;
    AddRoundKey(state, w[4 * Nr, 4 * Nr + 3]);

    //前 Nr - 1 轮解密
    for(int round = Nr - 1; round > 0; round -- )
    {
        InvShiftRows(state);
        InvSubBytes(state);
        AddRoundKey(state, w[4 * round, 4 * round + 3]);
        InvMixColumns(state);
    }
    //最后一轮解密
    InvShiftRows(state);
    InvSubBytes(state);
    AddRoundKey(state, w[0, 3]);
}
```

容易看出,AES的解密过程使用了四种逆变换,即InvSubBytes(·)、InShiftRows(·)、InvMixColumns(·)和AddRoundKey(·)(AddRoundKey的逆变换是它自身),以相反的顺序对由密文映射得到的状态矩阵进行变换。另外,AES的解密过程使用的子密钥与加密过程相同,但使用的顺序相反。下面介绍算法中的基本变换。

3.5.3 算法的基本变换

1. 字节替换变换(SubBytes)

字节替换操作使用一个S-盒对State的每字节都进行独立的替换。表3.10给出了AES的S-盒。

表 3.10　AES 的 S-盒

		Y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
X	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
	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

与 DES 的 S-盒相比，AES 的 S-盒能进行代数上的定义，而且不是像 DES 的 S-盒那样比较明显的“随机”代换。

S-盒按如下方式构造：

(1) 行 x、列 y 的字节值初始化为十六进制的{xy}；

(2) 把 S-盒中的每字节映射为在有限域 GF(2^8)中的逆，{00}不变；

(3) 把 S-盒中的每字节转换为二进制表示($b_7,b_6,b_5,b_4,b_3,b_2,b_1,b_0$)，然后进行如下的仿射变换：

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

【例 3.2】 这里用一个小例子来说明 S-盒的构造算法。以十六进制的{53}开始，二进制表示为 01010011，它表示有限域 GF(2^8)中的元素为

$$x^6+x^4+x+1$$

它在有限域 GF(2^8)中的乘法逆元素为

$$x^7+x^6+x^3+x$$

因此，在二进制下有

$$(b_7,b_6,b_5,b_4,b_3,b_2,b_1,b_0)=(1,1,0,0,1,0,1,0)$$

下面计算

$$\begin{aligned} b'_0&=(b_0+b_4+b_5+b_6+b_7+1) \bmod 2\\ &=(0+0+0+1+1+1) \bmod 2\\ &=1 \end{aligned}$$

同理,可计算其他位,结果为

$$(b'_0,b'_1,b'_2,b'_3,b'_4,b'_5,b'_6,b'_7)=(1,1,1,0,1,1,0,1)$$

以十六进制表示就是{ED},故 S 盒中的第 5 行、第 3 列中的元素为{ED},即 SubBytes({53})={ED}。

2. 行移位变换(ShiftRows)

State 的第一行保持不变,第二行循环左移 1 字节,第三行循环左移 2 字节,第四行移行循环左移 3 字节,如图 3.13 所示。

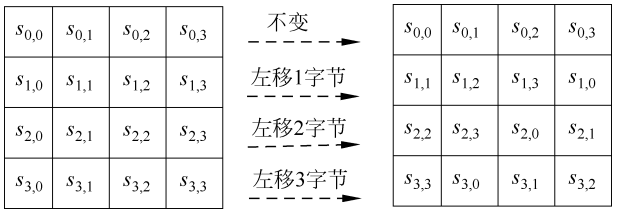


图 3.13 行移位变换

3. 列混合变换(MixColumns)

列混合变换对 State 中的每列进行独立的操作,它把每列都看成 GF(2^8)中的一个四项多项式 $s(x)$,再与 GF(2^8)上的固定多项式 $a(x)=\{03\}x^3+\{01\}x^2+\{01\}x+\{02\}$进行模 x^4+1 的乘法运算。如对第 c 列($0\leqslant c\leqslant 3$),其对应的 GF(2^8)中的多项式为 $s_c(x)=s_{0,c}+s_{1,c}x+s_{2,c}x^2+s_{3,c}x^3$,则列混合变换后为 $s'_c(x)=s_c(x)\otimes a(x)$。其矩阵乘法表示如下:

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

其中 $0\leqslant c\leqslant 3$。

解密变换中涉及的三种基本变换如下。

4. InvSubBytes 变换

InvSubBytes 变换是字节替换变换(SubBytes)的逆变换,即先用到了仿射变换的逆变换,再计算 GF(2^8)中的乘法逆。逆 S-盒对状态矩阵中的每字节进行逆变换。InvSubBytes 变换可以通过如表 3.11 所示的逆 S-盒实现。

表 3.11 AES 解密过程中的逆 S-盒

	Y																
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
X	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

续表

		Y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
X	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	af	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

5. InvShiftRows 变换

InvShiftRows 变换是行移位变换(ShiftRows)的逆变换,即对状态矩阵的各行按相反的方向进行循环移位操作。因此,状态矩阵各行的移位情况如下:第一行保持不变;第二行循环右移 1 字节;第三行循环右移 2 字节;第四行循环右移 3 字节。

6. InvMixColumns 变换

InvMixColumns 变换是列混合变换(MixColumns)的逆变换。InvMixColumns 同样逐列处理状态矩阵,它把每一列都当作系数 GF(2^8)有限域上的四项多项式。

与 MixColumns 变换对应,InvMixColumns 变换把列多项式与多项式 $a(x)$ 相对于模多项式 x^4+1 的逆 $a^{-1}(x)$ 相乘,即

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

设列多项式为 $s_c(x)=s_{0,c}+s_{1,c}x+s_{2,c}x^2+s_{3,c}x^3$,$0\leqslant c\leqslant 3$,InvMixColumns 变换可改写为如下的矩阵形式:

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

3.5.4　密钥扩展算法

轮密钥是由密钥经过一个扩展算法产生的,其长度由加解密轮数决定,具体地说,轮密钥有(分组长度)×(N_r+1)位。例如,AES-128 的加解密轮数为 10,则轮密钥共有 128×(10+1)=1408 位。

密钥扩展算法以一个字(即 4 字节)为基本单位,其伪代码描述如下:

```
KeyExpansion(byte key[4 * Nk], word w[4 * (Nr + 1)], Nk)
```

```
// key[]表示初始密钥,w[]表示扩展后的密钥,Nk 为密钥长度(以字为单位)
{
    word temp;
    //扩展密钥的前 Nk 个字是初始密钥
    for (i = 0;i < Nk;i++)
        //把四个单字节按照从高位到低位的顺序表示为一个字
        w[i] = (key[4 * i], key[4 * i + 1], key[4 * i + 2], key[4 * i + 3]);
    for(i = Nk; i < 4 * ( Nr + 1); i++)
    {
        temp = w[i - 1];
        if (i % Nk == 0)
            temp = SubWord(RotWord(temp)) ^ Rcon[i/ Nk];
        else if(Nk == 8 && (i % Nk == 4))
            temp = SubWord(temp);
        w[i] = w[i - Nk] ^ temp;
    }
}
```

密钥扩展算法中包括两个函数 RotWord 和 SubWord。RotWord(B_0,B_1,B_2,B_3)对输入的4字节进行循环左移操作,即

$$\text{RotWord}(B_0,B_1,B_2,B_3)=(B_1,B_2,B_3,B_0)$$

SubWord(B_0,B_1,B_2,B_3)对输入的4字节分别使用S-盒的替换操作SubBytes。Rcon[i]=(RC[i],'00','00','00'),RC[i]的所有可能值(用十六进制表示)如表3.12所示。实际上,RC[i]的值为有限域GF(2^8)中的多项式 x^{i-1} 的十六进制表示。

表3.12 RC[·]

i	1	2	3	4	5	6	7	8	9	10	11
RC[i]	01	02	04	08	10	20	40	80	1B	36	6c

可以看到,扩展密钥的最前面 N_k 个字是直接由输入的密钥填充的,而后面的每个字 $w[i]$则主要由前面的字 $w[i-1]$与 N_k 个位置之前的字 $w[i-N_k]$进行异或运算得到。

3.6 国密分组密码算法SM4

3.6.1 算法的总体描述

SM4分组密码算法[5]是我国无线局域网标准WAPI中所使用的密码标准,该算法采用非平衡Feistel结构(一种基本Feistel结构的变种),分组长度为128位,密钥长度为128位。加密算法与密钥扩展算法均采用非线性迭代结构。加密运算和解密运算的算法结构相同,解密运算的轮密钥使用顺序与加密运算相反。

SM4分组密码算法的加密密钥长度为128位,表示为 MK=(MK_0,MK_1,MK_2,MK_3),其中 MK_i(i=0,1,2,3)为32位。轮密钥表示为(rk_0,rk_1,…,rk_{31}),其中 rk_i(i=0,1,…,31)为32位。轮密钥由加密密钥生成。FK=(FK_0,FK_1,FK_2,FK_3)为系统参数,CK=(CK_0,CK_1,…,CK_{31})为固定参数,用于密钥扩展算法,其中 FK_i(i=0,1,…,3),CK_i(i=0,1,…,31)均为32位。

SM4加密算法由32次迭代运算和1次反序变换 R 组成。设明文输入为(X_0,X_1,X_2,

$X_3)\in(Z_2^{32})^4$，密文输出为$(Y_0,Y_1,Y_2,Y_3)\in(Z_2^{32})^4$，轮密钥为 $\mathrm{rk}_i\in Z_2^{32}$，$i=0,1,\cdots,31$。加密算法的运算过程如下：

(1) 首先执行 32 次迭代运算：

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

其中 F 为轮函数。

(2) 对最后一轮数据进行反序变换如下：

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

SM4 的加密和解密算法的变换结构相同，不同的只有轮密钥的使用顺序，解密时反序使用轮密钥。

轮函数 F 的细节如下。

设输入为$(X_0,X_1,X_2,X_3)\in(Z_2^{32})^4$，轮密钥为 $\mathrm{rk}\in Z_2^{32}$，则轮函数 F 为

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

其中，$T:Z_2^{32}\to Z_2^{32}$ 是一个可逆变换，由非线性变换 τ 和线性变换 L 复合而成，即 $T(\cdot)=L(\tau(\cdot))$。

非线性变换 τ 由 4 个并行的 S-盒(见表 3.13)构成，设输入为 $\boldsymbol{A}=(a_0,a_1,a_2,a_3)\in(Z_2^8)^4$，非线性变换 τ 的输出为 $\boldsymbol{B}=(b_0,b_1,b_2,b_3)\in(Z_2^8)^4$，即

$$(b_0,b_1,b_2,b_3)=\tau(\boldsymbol{A})=(\mathrm{Sbox}(a_0),\mathrm{Sbox}(a_1),\mathrm{Sbox}(a_2),\mathrm{Sbox}(a_3))$$

表 3.13　SM4 加密过程的 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	A	44	13	26	49	86	06	99
2	9C	42	50	F4	91	EF	98	7A	33	54	0B	43	E	CF	AC	62
3	E4	B3	1C	A9	C9	08	E8	95	80	DF	94	F	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	E	BF	8A	D2	40	C7	38	B5	A3	F7	F2	CE	F9	61	15	A1
9	E0	AE	5D	A4	9B	34	1A	55	A	93	32	30	F5	8C	B1	E3
A	1D	F6	E2	2E	82	66	CA	60	C0	29	23	AB	0	53	4E	6F
B	D5	DB	37	45	DE	FD	8E	2F	03	FF	6A	72	6	6C	5B	51
C	8D	1B	A	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	D	CB	39	48

设 S-盒的输入为 EF，则经 S-盒运算的输出结果为第 E 行、第 F 列的值，即 Sbox(EF)=0x84。

L 是线性变换，非线性变换 τ 的输出是线性变换 L 的输入。设输入为 $B\in Z_2^{32}$，则：

$$C=L(B)=B(B\lll 2)\oplus(B\lll 10)\oplus(B\lll 18)\oplus(B\lll 24)$$

其中，$\lll i$ 表示 32 位循环左移 i 位；Z_2^n 表示长度为 n 的二进制序列集合。

3.6.2 密钥扩展算法

本算法的轮密钥由加密密钥通过密钥扩展算法生成。设加密密钥为 MK,MK=$(MK_0,MK_1,MK_2,MK_3)\in(Z_2^{32})^4$。

轮密钥生成方法如下:

$$K_0=MK_0\oplus FK_0$$
$$K_1=MK_1\oplus FK_1$$
$$K_2=MK_2\oplus FK_2$$
$$K_3=MK3\oplus FK_3$$
$$rk_i=K_{i+4}\oplus T'(K_{i+1}\oplus K_{i+2}\oplus K_{i+3}\oplus CK_i),\quad i=0,1,2,\cdots,31$$

式中:T'是将合成置换 T 的线性变换 L 替换为 $L'(B)=B\oplus(B<<<13)\oplus(B<<<23)$。

系统参数 FK 的取值如下:

$$FK_0=(A2B1BAC6)$$
$$FK_1=(56AA3350)$$
$$FK_2=(677D9197)$$
$$FK_3=(B27022DC)$$

固定参数 CK 的取值方法如下:

设 $ck_{i,j}$ 为 CK_i 的第 j 字节($i=0,1,\cdots,31$;$j=0,1,2,3$),即 $CK_i=(ck_{i,0},ck_{i,1},ck_{i,2},ck_{i,3})\in(Z_2^8)^4$,则 $ck_{i,j}=(4i+j)\times7(\mathrm{mod}\ 256)$。

固定参数 $CK_i(i=0,1,2,\cdots,31)$的具体取值如下:

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

解密密钥同加密密钥,解密使用的轮密钥由解密密钥生成,生成方法同加密过程的轮密钥生成方法。

3.7 轻量级分组密码

本节将介绍轻量级分组密码算法的相关内容。所谓轻量级分组密码算法是指相对于传统的分组密码算法(如 DES 和 AES)来说算法结构简单,在硬件实现和运行功耗上有明显优势的一类分组密码算法。轻量级分组密码算法更适合于资源受限的应用场景(如物联网相关的应用场景)。下面将具体介绍两种著名的轻量级分组密码算法 PRESENT[6] 和 CLEFIA[7],这两个算法均已入选国际标准化组织(ISO)的轻量级分组密码标准。

3.7.1　PRESENT 密码算法

在 2007 年的密码硬件和嵌入式系统国际会议上，丹麦密码学家 Bogdanov、Kundsen 等公布了 PRESENT 算法[6]，该算法属于超轻量级加密算法。所谓分组密码算法的量级，主要通过其硬件实现时的复杂程度来衡量。PRESENT 算法与其他一些轻量级分组密码算法相比，有着最简单的硬件实现，因此它被称为超轻量级密码算法。PRESENT 算法为 SPN 结构，分组长度为 64 位，密钥长度有 80 位和 128 位两种，两种均为 31 轮。硬件实现 80 位密钥的 PRESENT 算法仅需要约 1570 个与非门。该算法在功耗方面也有很优秀的表现：在物联网的低耗节点中，80 位的密钥能够提供足够的安全性，至于 128 位的密钥可以用在安全性要求更高的场景。PRESENT 面向硬件设计，因此硬件性能良好，但软件性能一般，甚至较差。

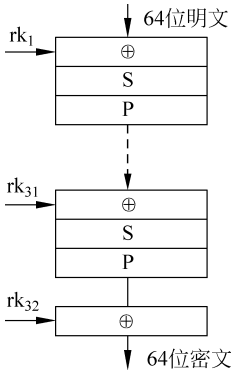


图 3.14　PRESENT 加密算法的结构

PRESENT 加密算法的结构如图 3.14 所示。算法先将轮函数迭代 31 次，轮函数包括轮密钥加(addRoundKey)、非线性置换 S 层(sBoxLayer)、线性置换 P 层(pLayer)，再经过一次轮密钥加运算后得到密文。

设 64 位的明文 $P=p_{63}p_{62}p_{61}p_{60}\cdots p_0$，其中 p_0 表示最低位，p_{63} 表示最高位。轮函数各步操作如下：

① 轮密钥加。完成 64 位中间状态与 64 位轮密钥的异或运算。

② 设 S 层(见表 3.14)输入 $W=w_{63}w_{62}w_{61}\cdots w_0$，输出 $V=v_{63}v_{62}v_{61}\cdots v_0$，则有：

$$v_{63}v_{62}v_{61}v_{60}=S(v_{63}v_{62}v_{61}v_{60})$$
$$v_{59}v_{58}v_{57}v_{56}=S(v_{59}v_{58}v_{57}v_{56})$$
$$\cdots$$
$$v_3v_2v_1v_0=S(v_3v_2v_1v_0)$$

③ P 层将输入的每一位映射到输出的另一位，取值如表 3.15 所示。

表 3.14　PRESENT 算法的 S 层

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

表 3.15　P 层真值表

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(i)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(i)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(i)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P(i)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

解密时先将解密轮函数迭代 31 次，再经过一次轮密钥加，得到明文。与加密不同的方面如下：

① 解密轮函数依次是：轮密钥加、线性置换逆P层、非线性置换逆S层；

② 轮密钥加实现中间状态与轮密钥的异或，轮密钥的使用顺序与加密时相反，依次是rk_{32}，rk_{31}，…，rk_2，最后一步轮密钥加使用rk_1；

③ 逆P层的表达式为

$$P^{-1}(i)=\begin{cases}(4i)\%63, & i\neq 63\\ 63, & i=63\end{cases}$$

④ 逆S层依次将连续4位经过逆S层，逆S层的取值如表3.16所示。

表3.16 PRESENT运算的逆S层

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S^{-1}(x)$	5	E	2	8	C	1	2	D	B	4	6	3	0	7	9	A

密钥扩展算法

设80位的种子密钥为$K=k_{79}k_{78}k_{77}\cdots k_0$，最左边的64位作为第一轮的轮密钥，即$rk_1=k_{79}k_{78}k_{77}\cdots k_{16}$，接着循环执行如下步骤31次，每次将最左边64位作为轮密钥。

① 循环左移61位，即

$$k_{79}k_{78}k_{77}\cdots k_0 \leftarrow k_{18}k_{17}k_{16}\cdots k_{19}$$

② 将最左边4位经过S层，即

$$k_{79}k_{78}k_{77}k_{76} \leftarrow S(k_{79}k_{78}k_{77}k_{76})$$

③ 将轮常量i和$k_{19}k_{18}k_{17}k_{16}k_{15}$异或，即

$$k_{19}k_{18}k_{17}k_{16}k_{15} \leftarrow i \oplus k_{19}k_{18}k_{17}k_{16}k_{15}$$

轮常量依次是1、2、3、…、31。

3.7.2 CLEFIA密码算法

CLEFIA[7]是由SONY公司研制开发的一种分组加密算法，该算法被用来保护SONY公司的音乐和图像等数字内容，以及进行“高级”版权的保护与认证。CLEFIA算法在2007年的快速软件加密国际会议上公布。SONY公司称CLEFIA算法可以抵抗“现有的密码破解手段”，他们希望这种新技术可以应用到软件和AV硬件中。

CLEFIA算法由数据处理部分和密钥扩展部分组成。它的基本结构是一种广义的Feistel结构，由四条输入分支组成。每一轮中有两个F函数，每个F函数使用两个不同的S-盒以及两个不同的扩散矩阵。密钥扩展部分与数据处理部分共享Feistel结构，从而使得CLEFIA只需要较小的硬件和软件规模。

CLEFIA算法支持128位的分组长度，密钥长度可选128/192/256位。密钥长度是128位、192位和256位时，加密轮数分别是18轮、22轮和26轮。下面以128位密钥为例对CLEFIA算法进行详细介绍，如图3.15所示。

设$P=(P_0,P_1,P_2,P_3)$和$C\in\{0,1\}^{128}$分别为128位的明文和密文，其中$P_i,C_i\in\{0,1\}^{32}(0\leqslant i<4)$为32位的分支；$K_i\in\{0,1\}^{32}(0\leqslant i<2r)$为轮密钥，$WK_0$，$WK_1$，$WK_2$，$WK_3\in\{0,1\}^{32}$为白化密钥。$r$轮CLEFIA算法加密过程如下。

① 初始白化层$T=(T_0,T_1,T_2,T_3)=(P_0,P_1\oplus WK_0,P_2,P_3\oplus WK_1)$为第一轮的输入。

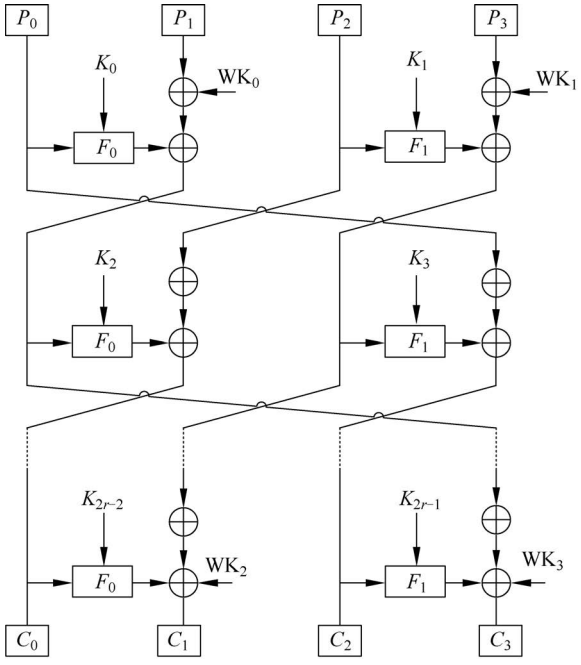


图 3.15　CLEFIA 加密算法的结构

② r 轮轮变换。设 $T^{(i-1)}=(T_0^{(i-1)},T_1^{(i-1)},T_2^{(i-1)},T_3^{(i-1)})$ 为第 i 轮的输入，则第 i 轮的输出为

$$T^{(i)}=(T_0^{(i)},T_1^{(i)},T_2^{(i)},T_3^{(i)})$$
$$=(F_0(T_0^{(i-1)},K_{2i-2})\oplus T_1^{(i-1)},T_2^{(i-1)},F_1(T_2^{(i-1)},K_{2i-1})\oplus T_3^{(i-1)},T_0^{(i-1)})$$

③ 末尾白化层。密文 $C=(C_0,C_1,C_2,C_3)=(T_3^{(r)},T_0^{(r)}\oplus WK_2,T_1^{(r)},T_2^{(r)}\oplus WK_3)$。

上述 F_0 和 F_1 是两个非线性可逆函数，F_0 定义如下。

① 轮密钥加：计算 $A=T_0^{(i-1)}\oplus K_{2i-2}$；

② 非线性变换 $A=(A_0,A_1,A_2,A_3)$，其中 $A_i\in\{0,1\}^8$ $(0\leqslant i\leqslant 3)$，计算 $B_0=S_0(A_0),B_1=S_1(A_1),B_2=S_0(A_2),B_3=S_1(A_3)$；

③ 线性变换：计算 $D=M_0(B_0,B_1,B_2,B_3)^{\mathrm{T}}$。

其中，S_0 和 S_1 是两个非线性 S-盒，M_0 为一个 4×4 的矩阵。非线性函数 F_1 的定义与 F_0 类似，只需将其中的 S_0 和 S_1 的位置互换，将 M_0 变为 M_1 即可。M_0 和 M_1 的十六进制表示如下：

$$\boldsymbol{M}_0=\begin{pmatrix}0\text{x}01 & 0\text{x}02 & 0\text{x}04 & 0\text{x}06\\ 0\text{x}02 & 0\text{x}01 & 0\text{x}06 & 0\text{x}04\\ 0\text{x}04 & 0\text{x}06 & 0\text{x}01 & 0\text{x}02\\ 0\text{x}06 & 0\text{x}04 & 0\text{x}02 & 0\text{x}01\end{pmatrix}$$

$$\boldsymbol{M}_1=\begin{pmatrix}0\text{x}01 & 0\text{x}08 & 0\text{x}02 & 0\text{x}0\text{a}\\ 0\text{x}08 & 0\text{x}01 & 0\text{x}0\text{a} & 0\text{x}02\\ 0\text{x}02 & 0\text{x}0\text{a} & 0\text{x}01 & 0\text{x}08\\ 0\text{x}0\text{a} & 0\text{x}02 & 0\text{x}08 & 0\text{x}01\end{pmatrix}$$

矩阵与向量的乘法定义在有限域 GF(2^8)上,其本原多项式为 $z^8+z^4+z^3+z^2+1$。容易验证矩阵 $\boldsymbol{M}_0$ 和 $\boldsymbol{M}_1$ 都是自逆的,即 $\boldsymbol{M}_0=\boldsymbol{M}_0^{-1}$,$\boldsymbol{M}_1=\boldsymbol{M}_1^{-1}$。

CLEFIA 算法的解密过程与加密过程类似,如图 3.16 所示。

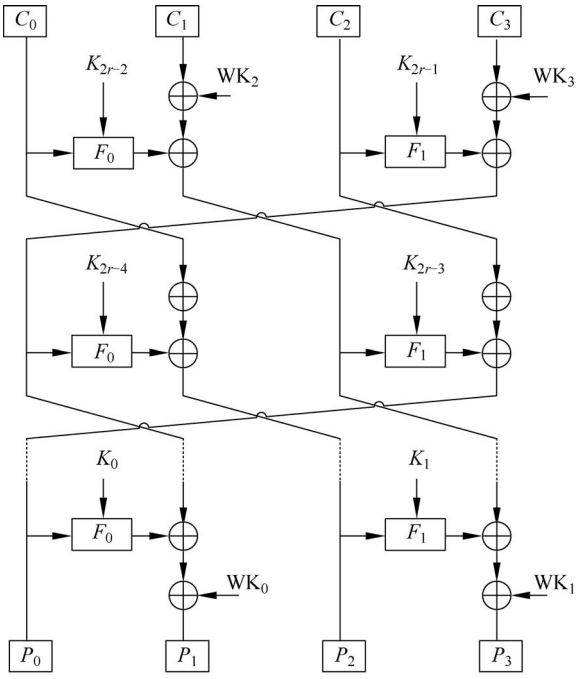


图 3.16 CLEFIA 解密算法的结构

密钥扩展算法

CLEFIA 的密钥扩展算法可分为两部分:首先由种子密钥 SK 生成 L,然后由 SK 和 L 扩展得到子密钥 WK 与 K,其中生成 L 的结构与其密钥长度有关。因此,首先定义两种整体结构 $GFN_{4,r}$ 及 $GFN_{8,r}$($GFN_{d,r}$ 表示含有 d 个 32 位的分支和 r 轮迭代):当密钥长度为 128 位时,密钥扩展算法的结构为 $GFN_{4,r}$,与图 3.15 中的结构相同;当密钥长度为 192/256 位时,密钥扩展算法的结构为 $GFN_{8,r}$,如图 3.17 所示。

当密钥长度为 128 位时的密钥扩展算法步骤如下:

① $L \leftarrow GFN_{4,12}(CON_0^{128}, CON_1^{128}, \cdots, CON_{23}^{128}, SK)$;

② $WK_0 | WK_1 | WK_3 | WK_4 \leftarrow SK$;

③ For $i=0$ to $i=8$

{

$T=L \oplus (CON_{24+4i}^{128} | CON_{24+4i+1}^{128} | CON_{24+4i+2}^{128} | CON_{24+4i+3}^{128})$;

$L \leftarrow \sum(L)$;

当 i 是奇数时,$T \leftarrow T \oplus SK$;

$K_{4i} | K_{4i+1} | K_{4i+2} | K_{4i+3} \leftarrow T$;

}

算法在步骤①中用到了 24 个 32 位常数 CON_i^{128}($0 \leqslant i < 24$);然后由 L 和 SK 生成子密钥 K_i($0 \leqslant i < 36$)和 WK_i($0 \leqslant i < 4$),其中用到了 36 个 32 位常数 CON_i^{128}($24 \leqslant i < 60$)。

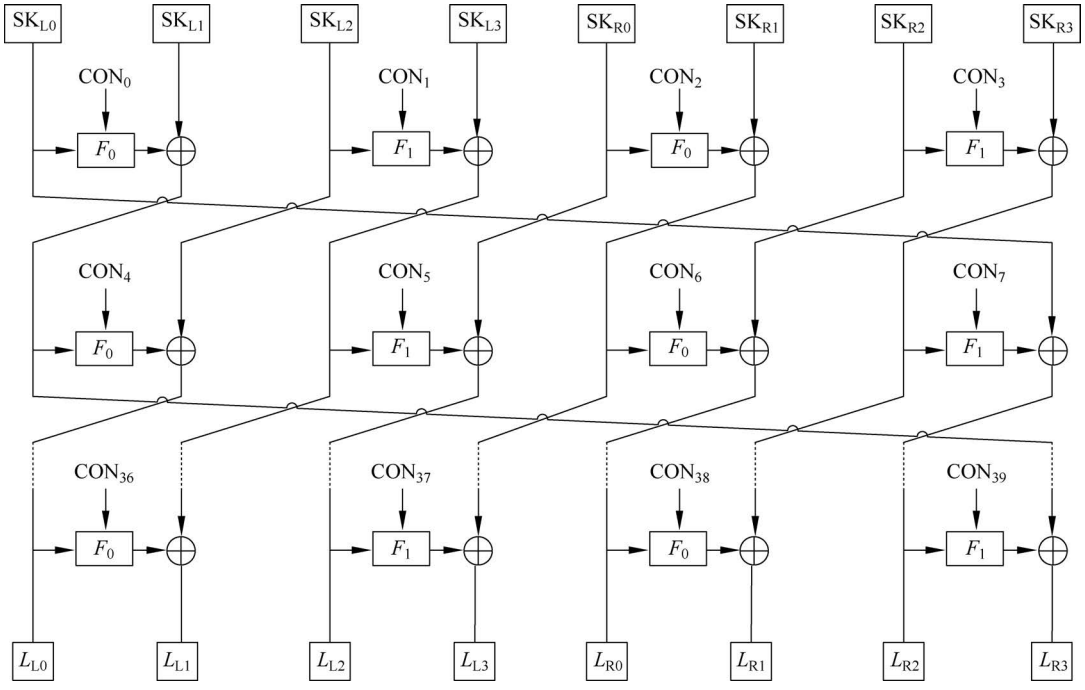


图 3.17　$GFN_{8,10}$ 结构

当密钥长度为 192 位或者 256 位两种情况下，密钥扩展算法是比较相似的，不同之处仅在于所用的常数数量不同：192 位的情况下使用了 76 个 32 位常数 CON_i^{192}，256 位密钥的情况下使用了 92 个 32 位常数 CON_i^{256}。用 k 表示密钥长度，下面是密钥长度为 192 和 256 位的密钥扩展算法的步骤：

① 当 $k=192$ 时，$SK_L \leftarrow SK_0 | SK_1 | SK_2 | SK_3$，$SK_R \leftarrow SK_4 | SK_5 | \overline{SK_0} | \overline{SK_1}$；

当 $k=256$ 时，$SK_L \leftarrow SK_0 | SK_1 | SK_2 | SK_3$，$SK_R \leftarrow SK_4 | SK_5 | SK_6 | SK_7$；

② 令 $SK_L = SK_{L0} | SK_{L1} | SK_{L2} | SK_{L3}$，$SK_R \leftarrow SK_{R0} | SK_{R1} | SK_{R2} | SK_{R3}$，

$L_L | L_R \leftarrow GFN_{8,10}(CON_0^{(k)}, CON_1^{(k)}, \cdots, CON_{39}^{(k)}, SK_{L0}, \cdots, SK_{L3}, SK_{R0}, \cdots, SK_{R3})$；

③ $WK_0 | WK_1 | WK_3 | WK_4 \leftarrow SK_L \oplus SK_R$；

④ For $i=0$ to $i=10$（当 $k=192$）或 $i=12$（当 $k=256$）

{

当 i mod 4＝0 或 1：

$T = L_L \oplus (CON_{40+4i}^{(k)} | CON_{40+4i+1}^{(k)} | CON_{40+4i+2}^{(k)} | CON_{40+4i+3}^{(k)})$；

$L_L \leftarrow \sum(L_L)$；

当 i 是奇数时，$T \leftarrow T \oplus SK_R$

当 i mod 4＝2 或 3：

$T = L_R \oplus (CON_{40+4i}^{(k)} | CON_{40+4i+1}^{(k)} | CON_{40+4i+2}^{(k)} | CON_{40+4i+3}^{(k)})$；

$L_R \leftarrow \sum(L_R)$；

当 i 是奇数时，$T \leftarrow T \oplus SK$；

$K_{4i} | K_{4i+1} | K_{4i+2} | K_{4i+3} \leftarrow T$；

}

关于密钥扩展算法中涉及的常数 $CON_i^{(k)}$,在文献[7]中有详细的介绍,此处不再赘述。

3.8 分组密码的工作模式

本节将介绍分组密码的工作模式。由于前述章节介绍的分组密码算法一次只能处理单块数据,因此,引入分组密码的工作模式可以用于处理多个数据块。这里需要注意的是,只有使用了"合适的"分组密码工作模式,才能基于具体的分组密码算法设计出能实现数据机密性的对称加密方案。下面将具体介绍五种典型的分组密码工作模式,即电子密码本模式、密码分组链接模式、密码反馈模式、输出反馈模式和计数器模式。

3.8.1 电子密码本模式(ECB)

电子密码本模式(Electronic Codebook Mode,ECB)是最简单的模式,它直接利用加密算法分别对每个明文分组使用相同密钥进行加密。设明文分组序列为 $M=M_1M_2\cdots M_n$,相应的密文分组序列为 $C=C_1C_2\cdots C_n$,则

$$C_i=E_k(M_i),\quad i=1,2,\cdots,n$$

对于长度大于 b 位的报文,整个加密前需要把这个报文分成 b 位的分组,如果必要的话对最后一个分组进行填充。使用"密码本"这个术语的原因是对每个 b 位的明文分组都有一个唯一的密文,因此可以想象一个巨大的密码本,其中对每一个可能的 b 位明文项,都有一个密文项与之对应。

在 ECB 模式下,加密算法的输入是明文,算法的输出将直接作为密文进行输出,不进行任何形式的反馈,每个明文分组的处理是相互独立的,这种方式也是分组密码工作的标准模式。

ECB 模式的主要缺点是由于没有任何形式的反馈,相同的明文加密后将产生相同的密文,这样在处理具有固定数据结构的明文数据时容易暴露明文数据的固有格式。例如某些格式化的报文,除了作业号、通信地址、发报时间、密级等数据位置固定外,明文数据也非常有规律,这些常用的数据通常加密后在密文的同一个位置出现,密码分析者就可能得到许多明文-密文对,从而为分析、破译密码提供线索。

同时,由于 ECB 模式下各个报文分组之间是相互独立的,这样,如果攻击者有能力对报文进行分组的插入与删除,就可以进行主动攻击,带来安全风险。另外,ECB 模式无法纠正传输中的同步差错,如果在传输中增加或丢失一位或多位数据,将导致密文分组的对齐错误,这样,整个密文序列都将不能正确地解密。

3.8.2 密码分组链接模式(CBC)

在密码分组链接模式(Cipher Block Chaining Mode,CBC 模式)中,加密算法的输入是当前的明文分组 M_i 和上一次产生的密文分组 C_{i-1} 的异或,其输出为密文分组 C_i,即

$$C_0=\mathrm{IV}$$
$$C_i=E_k(M_i\oplus C_{i-1}),\quad 1\leqslant i\leqslant n$$

相应的解密规则为

$$C_0=\mathrm{IV}$$
$$M_i=D_k(C_i)\oplus C_{i-1},\quad 1\leqslant i\leqslant n$$

CBC 模式的工作示意图如图 3.18 所示。

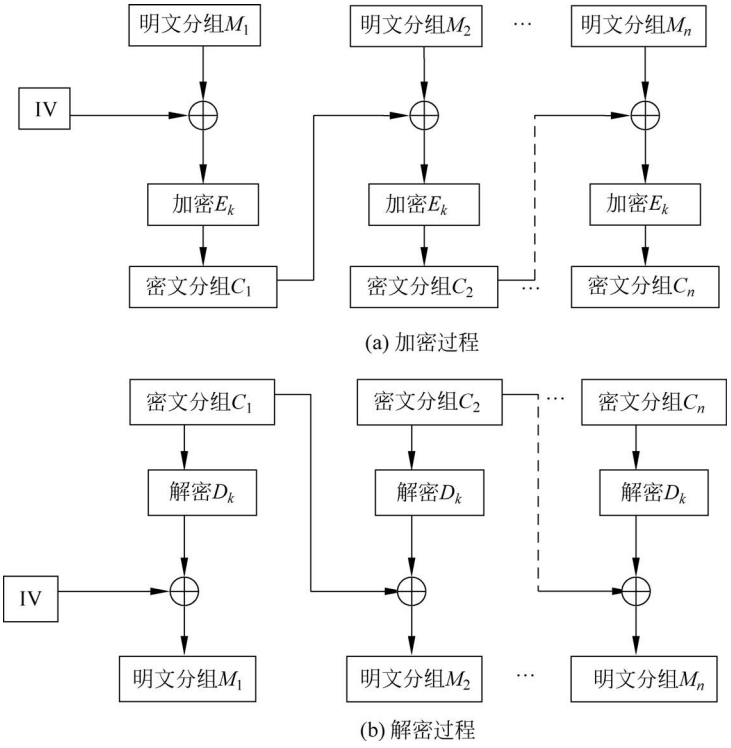


图 3.18　CBC 模式的工作示意图

IV 被用来和解密算法的输出进行异或，以产生第一个明文分组。因此，IV 必须被发送方和接收方双方所知，通常作为密文的一部分。IV 一般无须保密，但需随消息进行更换。

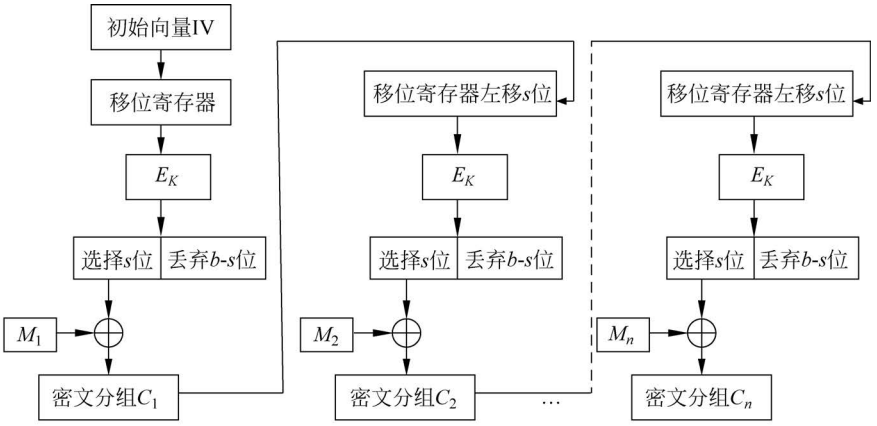
CBC 模式有效地改进了 ECB 模式的缺点，它能够隐蔽明文的数据模式，相同的明文对应的密文一般是不同的。同时，在一定程度上，它能够有效地抵抗密文传输过程中对数据的篡改，如分组的重放、插入和删除等。

CBC 模式由于引进了反馈，当信道噪声等干扰导致密文传输错误时，密文中的一位错误将影响当前分组及下一个分组的解密，但其他分组不受影响，人们把这种错误传播形式称为“有限传播”。请考虑，如果接收方不知道 IV，对解密有什么影响？

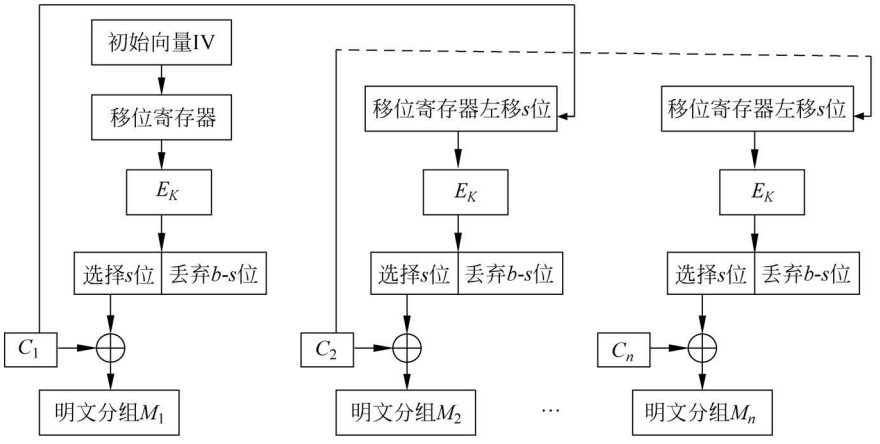
3.8.3　密码反馈模式(CFB)

为了实现更大的灵活性，密码反馈模式(Cipher Feedback Mode，CFB 模式)需要引入一个整数参数 s，$1 \leqslant s \leqslant b$。需要注意的是，明文不是按 b 进行分组，而是按 s 分组，且明文的长度必须是 s 的倍数。故 CFB 模式能够通过对参数 s 的选择来对长度小于 b 的数据直接进行加密，而 CBC 模式却做不到。

在 s 位 CFB 模式，加密函数的输入是一个 b 位的移位寄存器，这个移位寄存器被初始化为一个初始向量 IV。加密函数处理结果的最高(最左边)s 位与明文的第一个分组 M_1 进行异或运算，产生密文分组 C_1。同时，移位寄存器的值向左移 s 位，且用 C_1 替换寄存器的最低(最右边)s 位。这个过程如此重复，直到完成加密。CFB 模式的加解密过程如图 3.19 所示。



(a) 加密过程



(b) 解密过程

图 3.19 CFB 模式的加解密过程

令函数 $\mathrm{LSB}_i(\cdot)$表示输入的前 i 个最低有效位,函数 $\mathrm{MSB}_i(\cdot)$表示输入的前 i 个最高有效位。例如,$\mathrm{LSB}_3(111011011)=011$,和 $\mathrm{MSB}_4(111011011)=1110$。则 CFB 模式的加解密规则如下:

加密:

$$
\begin{aligned}
I_1 &= \mathrm{IV} \\
I_j &= \mathrm{LSB}_{b-s}(I_{j-1}) \| C_{j-1}, \quad 2 \leqslant j \leqslant n \\
C_j &= M_j \oplus \mathrm{MSB}_s(E_K(I_j)), \quad 1 \leqslant j \leqslant n
\end{aligned}
$$

解密:

$$
\begin{aligned}
I_1 &= \mathrm{IV} \\
I_j &= \mathrm{LSB}_{b-s}(I_{j-1}) \| C_{j-1}, \quad 2 \leqslant j \leqslant n \\
M_j &= C_j \oplus \mathrm{MSB}_s(E_K(I_j)), \quad 1 \leqslant j \leqslant n
\end{aligned}
$$

CFB 模式由于采用密文反馈,若某个密文分组在传输中出现一位或多位的错误,将会引起当前分组和后续部分分组的解密错误。因为只有当错误的密文位从寄存器中移出后,解密才会恢复正常,因此一个密文分组出错会影响后面最多 $\left\lceil \frac{b}{s} \right\rceil$ 个分组的解密($\lceil x \rceil$表示

大于或等于 x 的最小整数)。

3.8.4　输出反馈模式(OFB)

在CFB模式中,一位密文的传输错误会影响至少 $b+1$ 位的明文,这种错误传播的影响对于有些应用来讲太大。对于这些应用来说,可以采用第四种工作模式,即输出反馈模式(Output Feedback Mode,OFB模式)。

在OFB模式中,先产生一个密钥流,然后将其与明文相异或,其加解密过程如图3.20所示。因此,OFB模式实际上就是一个同步流密码,通过反复加密一个初始向量IV来得到密钥流。这种方法有时也叫作"内部反馈",因为反馈机制独立于明文和密文而存在。

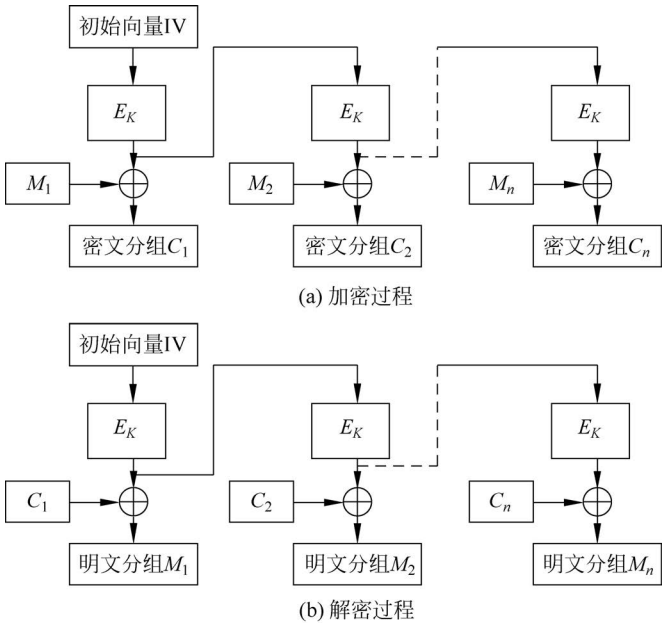


图3.20　OFB模式的加解密过程

定义 $Z_0=\mathrm{IV}$,然后用下述公式计算密钥流 $Z_0Z_1\cdots Z_n$:

$$Z_i=E_K(Z_{i-1}),\quad 1\leqslant i\leqslant n$$

最后用如下简单运算来完成加解密:

加密:$C_i=M_i\oplus Z_i(1\leqslant i\leqslant n)$

解密:$M_i=C_i\oplus Z_i(1\leqslant i\leqslant n)$

OFB模式要求在密钥相同的情况下,每次加密必须使用不同的IV,否则消息的机密性就得不到保证。

OFB模式具有普通序列密码的优缺点,例如可加密任意长度的数据(即不需要进行分组填充),不会传播错误,适于加密冗余度较大的数据、语音和图像数据,但对密文的篡改难以检测。

3.8.5　计数器模式(CTR)

CBC和CFB等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。例如,若用CBC模式来加密数据库,则为了访问一个特定的加密字段,就需要解密在它之前的所有数据,这是代价很高的操作。为此又出现了另一种操作模式,即计数器模式(Counter Mode,CTR模式)。

CTR模式不是用加密算法的输出填充寄存器,而是将一个计数器输入寄存器中,其加解密过程如图3.21所示。每一个分组完成加密后,计数器都要增加某个常数,典型常数值是1。其加解密公式如下:

加密:$C_i=M_i\oplus E_K(\mathrm{CTR}+i)(1\leqslant i\leqslant n)$

解密:$M_i=C_i\oplus E_K(\mathrm{CTR}+i)(1\leqslant i\leqslant n)$

其中,CTR表示计数器的初始值。

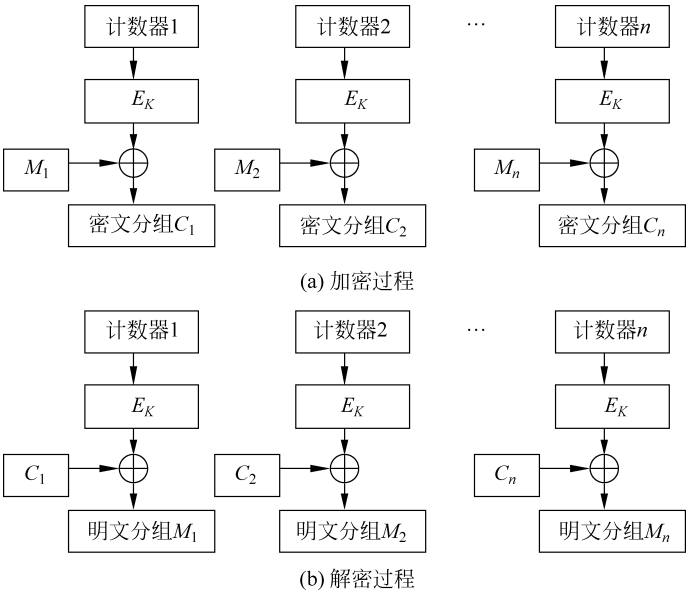


图3.21 CTR模式的加解密过程

CTR模式的主要优点如下。

(1) 随机访问特性:可以随机地对任意一个密文分组进行解密,对该密文分组的处理与其他密文分组无关。

(2) 高效率:能并行处理,即能对多个分组的加解密同时进行处理,而不必等前面分组处理完再开始;而且,CTR模式最主要和最耗时的处理并不依赖于明文或密文,因此可以提前进行预处理,这也可以极大提高处理效率。因此CTR模式很适合对实时性和速度要求很高的场景。

此外,CTR模式还可以处理任意长度的数据,而且加解密过程仅涉及加密运算,不涉及解密运算,因此不用实现解密算法。

3.9 分组密码的基本密码分析方法

本节介绍主流且通用的对分组密码算法的三种分析方法,这些方法的攻击复杂度依赖于所分析密码算法的数学结构,其中差分分析和线性分析方法被认为是现代密码学发展至今极具意义的密码分析方法。

3.9.1 差分密码分析

差分密码分析[4]是 Eli Biham① 和 Adi Shamir② 在 1990 年提出的一种分析方法，最初的差分密码分析是针对 DES 提出的，后来的发展表明差分密码分析几乎适用于所有的分组密码。差分密码分析是迄今为止攻击分组密码算法最有效的分析方法之一。差分分析是一种选择明文攻击方法，它的基本思想是：首先固定明文的差分，经若干轮后，研究明文差分对密文差分的影响，以此来恢复部分或全部的密钥位。对分组长度为 n 的 r 轮迭代密码算法，将两个 n 位串 X_i 和 X_i^* 的差分定义为

$$\Delta X_i = X_i \otimes X_i^*$$

其中：X_0 和 X_0^* 是明文对，X_i 和 X_i^* 是第 i 轮的输出，同时也是第 $i+1$ 轮的输入。若记第 i 轮的子密钥为 K_i，轮函数为 F，则 $X_i = F(X_{i-1}, K_i)$。利用加密算法获得密文对，寻找一个高概率的区分器；选择具有特定输入差分值 a_0 的明文对 (X_0, X_0^*)，该明文对在最后一轮的输入差分 ΔX_{r-1} 以很高的概率为取特定值 a_{r-1}；最后，恢复最后一轮的子密钥。为了叙述方便，定义以下概念。

定义 3.1 r 轮特征 Φ 是一个差分序列：

$$a_0, a_1, \cdots, a_r$$

其中 a_0 表示明文对 X_0 和 X_0^* 的差分，$a_i (1 \leqslant i \leqslant r)$ 表示第 i 轮 X_i 和 X_i^* 的差分。r 轮特征 Φ 的概率是指在明文对 X_0 和 X_0^* 的差分为 a_0，且明文 X_0 和子密钥 $K_1, K_2, \cdots, K_r$ 相互独立且均匀随机的条件下，第 $i (1 \leqslant i \leqslant r)$ 轮输出 X_i 和 X_i^* 的差分为 a_i 的概率。

定义 3.2 如果明文对 X_0 和 X_0^* 的差分为 a_0，X_i 和 X_i^* 的差分为 a_i，$1 \leqslant i \leqslant r$，则称明文对 X_0 和 X_0^* 为特征 $\Phi = a_0, a_1, \cdots, a_r$ 的一个正确对；否则，称为特征的错误对。

用 $p_i^{\Phi} = P(\Delta F(X) = a_i \mid \Delta X = a_{i-1})$ 表示在输入差分为 a_{i-1} 的条件下，轮函数 F 的输出差分为 a_i 的概率。通常 r 轮特征 $\Phi = a_0, a_1, \cdots, a_r$ 的概率用 $\prod_{i=1}^{r} p_i^{\Phi}$ 来近似替代。

对 r 轮迭代密码算法的差分密码分析的基本过程可总结为如下步骤：

① 找出一个概率最大的 $(r-1)$ 轮特征 $\Phi(r-1) = a_0, a_1, \cdots, a_{r-1}$；

② 均匀、随机地选择明文 X_0，计算出 X_0^*。对 2^m 个最后一轮的子密钥 K_r（或 K_r 的部分）的可能值 K_r^j，设置相应的 2^m 个计数器，用每个可能值 K_r^j 解密 X_r 和 X_r^*，得到 X_{r-1} 和 X_{r-1}^*，如果 X_{r-1} 和 X_{r-1}^* 的差分是 a_{r-1}，则相应的计数器加 1；

③ 反复执行②，若出现一个或几个明显高于其他计数器的值，则输出它们所对应的密钥或部分。

以上只是差分分析的基本原理，在实际应用中，有些步骤可能难以实现。例如，可能由于 K_r 的选取空间过大，攻击者只能转而选择推测 K_r 的部分。在 1992 年的国际密码学会议上，Biham 和 Shamir 改进了之前的攻击方法，对 16 轮的 DES 来说，差分攻击的攻击复杂度为 2^{47} 个明文密文对。

---

① Eli Biham(1960—)，以色列密码学家，目前是以色列 Technion 理工学院计算机科学系的教授。

② Adi Shamir(1952—)，RSA 算法的发明者之一，差分分析的发明者之一，Eli Biham 的博士生导师。

3.9.2 线性密码分析

线性密码分析[14]最早由松井充①于1993年提出,它起初也是针对DES的一种分析手段。线性密码分析属于已知明文攻击方法,攻击者能获取明文密文对,通过寻找明、密文间的不平衡的线性逼近表达式,从而恢复密钥。在介绍线性分析前,首先介绍线性分析中用到的堆积引理。

堆积引理 设 $X_i(1\leqslant i\leqslant n)$ 是独立的随机变量 $P(X_i=0)=p_i$,$P(X_i=1)=1-p_i$,则

$$P(X_1\oplus X_2\oplus\cdots\oplus X_i=0)=1/2+2^{n-1}\cdot\prod_{i=1}^{n}(p_i-1/2)$$

线性分析的第一步是寻找以下形式的不平衡的线性表达式:

$$M_{[i_1,i_2,\cdots,i_a]}\oplus C_{[j_1,j_2,\cdots,j_b]}=K_{[k_1,k_2,\cdots,k_c]}$$

其中用 $i_1,i_2,\cdots,i_a,j_1,j_2,\cdots,j_b,k_1,k_2,\cdots,k_c$ 表示固定的数据位位置。对随机的明文 M 和相应的密文 C,上述等式成立的概率 $p\neq1/2$,将 $|p-1/2|$ 称为逼近优势,以此来刻画该等式的有效性。为了寻找非平衡的线性表达式,首先计算单个轮函数的输入、输出的线性逼近及其成立的概率,分组密码算法的线性逼近的概率与每一轮的线性逼近的概率有关,由堆积引理来计算明文密文对和密钥的线性逼近的概率。

对 r 轮迭代密码算法的线性密码分析的基本过程可以总结为以下步骤:

① 首先找出一个$(r-1)$轮线性逼近式(α,β),其中 α 表示输入掩码,β 表示输出掩码,且其偏差$|\varepsilon(\alpha,\beta)|$较大;

② 利用区分器的输出,攻击者确定恢复的第 r 轮子密钥 K_r(或 K_r 的部分),设攻击的密钥量为 h,对每个可能的候选密钥 gk_i,$0\leqslant i\leqslant2^h-1$,设置相应的 2^h 个计数器;

③ 均匀、随机地选取明文 X,在同一个未知密钥 k(即在未知的轮密钥 $k_1,k_2,\cdots,k_r$)下加密,获得相应的密文 Z;

④ 对每个密文 Z,用第 r 轮中每个猜测密钥的轮密钥 gk_i 对其解密,得到 Y_{r-1},计算 $\alpha X\oplus\beta Y_{r-1}$ 是否为0,若是,则给相应的计算器 λ_i 加1;

⑤ 将 2^h 个计数器中 $\left|\frac{\lambda_i}{m}-\frac{1}{2}\right|$ 最大值所对应的密钥 gk_i 作为攻击获得的正确密钥值。

差分密码分析和线性密码分析是现代密码学极有意义的成果之一。在这些分析手段的基础上,之后高阶差分密码分析、飞来去器攻击、不可能差分密码分析,以及两者相结合的差分-线性分析等相继被提出,这里不再赘述,感兴趣的读者可以在文献[4]中找到这些分析手段的相关原理和具体步骤。

3.9.3 中间相遇攻击

1977年,Whitfield Diffie② 等人提出中间相遇攻击方法[8],其分析的对象依旧是DES算法。中间相遇攻击大致可分为两部分:中间相遇阶段通过部分加解密匹配过滤一部分错

① 松井充(1961—),日本密码学家,现为三菱电机公司高级研究员。

② Whitfield Diffie(1944—),美国著名密码学家,公钥加密的先驱之一。

误密钥，减小密钥空间；密钥检测阶段穷尽搜索剩余密钥，利用新的明文密文对确定唯一正确的密钥。中间相遇攻击的核心思想是将一个密码算法分为两部分：$C=E_{(k_1,k_2)}(P)=E''_{k_2}(E'_{k_1}(P))$，假设攻击者拥有明文密文对$(P_i,C_i)$，$1\leqslant i\leqslant n$，中间相遇攻击的步骤如下：

① 首先，固定明文P_i的某字节x之外的全部字节，单独取遍字节x的所有可能值。计算$C_i^*=E'_{k_1}(P_i)$。记录C_i^*在某个位置s的值y，且将x和y的关系表示为$y=f(x)$，函数f可以由算法E'和k_1决定。

② 对所有可能的k_2，计算出$C_i^{**}=E''^{-1}_{k_2}(C_i^*)$，检查位置$s$的值$y'=y$是否成立，若不成立，则淘汰相应的$k_2$。

③ 对所有的i，重复上述两步，最终获得正确的密钥。

以使用两个不同密钥的双重DES算法EE_2为例，如下：

$$EE_2(m)=Enc_{k_2}(Enc_{k_1}(m))$$

每个密钥的长度为κ，攻击者首先选择信息m，之后计算出在所有可能的k_1下的$Enc_{k_1}(m)$，并放入表中；在得到密文c后，使用所有的k_2值对c解密，将得到的结果放入另一个表中。通过寻找两个表中相同的值，可以得到密钥(k_1,k_2)的可能值。注意，例子中并不是对部分字节固定，而是计算了密钥的所有可能值。这是因为实际应用中需要对内存和时间代价权衡考虑，分析证明，对于双重DES，中间相遇攻击大致需要$2^{56+\alpha+3}$次加/解密操作，以及$2^{56-\alpha}$条用于保存中间信息的内存，α为整数。

基础的中间相遇攻击需要对密钥空间进行独立子集合划分，通常要求加密结构和密钥扩展算法足够简单且扩散速度慢，这种要求在一定程度上限制了中间相遇攻击的应用范围。此后出现的多种改进的中间相遇攻击放宽了对密钥集合划分或筛选条件的限制，例如2008年提出的针对8轮AES-256的中间相遇攻击[9]、2014年提出的对AES的安全性分析[10]等都取得了较好的效果。

思考题与习题

3-1　为了保证分组密码算法的安全强度，对分组密码算法的要求有哪些？

3-2　简述分组密码的设计原则。

3-3　什么是SPN网络？

3-4　什么是Feistel密码结构？主要有什么特点？

3-5　什么是分组密码的操作模式？分组密码有哪些主要的操作模式？其工作原理是什么？各有何特点？

3-6　简述DES的加密思想和f函数特性。

3-7　请证明DES结构的互补对称性，即$DES_{\overline{K}}(\overline{M})=\overline{DES_K(M)}$。

3-8　请用DES的8个S-盒对48位串70a990f5fc36进行压缩置换，用十六进制写出每个S-盒的输出。

3-9　除三重DES外，还有什么方式可以提高DES的安全性？

3-10　设DES算法的8个S-盒都为$S1$，且$R_0=$FFFFFFFF，$K_1=$555555555555(均为十六进制表示)，求$f(R_0,K_1)$。

3-11 对 DES 和 AES 进行比较,说明两者的特点和优缺点。

3-12 AES 的基本变换有哪些?

3-13 设 m = b5c9179eb1cc1199b9c51b92b5c8159d,请给出 AES 中的字节替换变换 SubBytes(m)的输出(用十六进制表示)。

3-14 AES 的加密算法和解密算法之间有何不同?

3-15 在 CBC 模式中,若传输中一个密文字符的中位发生了错误,这个错误将传播多远?

3-16 使用 C 或者 C++语言编程实现 SM4 在 CBC 模式和 OFB 模式下的加解密过程。要求指定明文文件、密钥文件、初始化向量文件的位置和名称,加密完成后密文文件的位置和名称。(加密时先分别从指定的明文文件、密钥文件和初始化向量文件中读取有关信息,然后分别按 CBC 和 OFB 模式进行加密,最后将密文(用十六进制表示)写入指定的密文文件。解密过程与此类似。)

3-17 SM4 是基于非平衡 Feistel 结构的国密分组密码算法,PRESENT 和 CLEFIA 是两种可面向硬件环境的轻量级分组密码算法,试从算法结构上分析和比较这三种分组密码算法。

3-18 差分密码分析和线性密码分析是针对分组密码算法的两种密码分析方法,请简述这两种方法的基本思想。