第 3 章　对称密码技术

本章导读

- 本章主要介绍对称密码技术及其相关的内容，包括一些加密算法、密钥的产生和密钥的分配等。
- 对称密码是一种加密密钥和解密密钥相同的密码体制。
- 对称密码分为分组密码和流密码。分组密码每次操作(如加密和解密)是针对一个分组而言。流密码则每次加密(或者解密)一位或者一个字节。
- 对密码的攻击方法有基于密码算法性质的密码分析和穷举搜索攻击。
- 从发展阶段来看，对称密码主要分为两种：20 世纪 70 年代以前的对称密码(主要指计算机出现以前)和 20 世纪 70 年代以后的对称密码。
- 我们称 20 世纪 70 年代以前的对称密码为古典加密技术，主要使用代换或者置换技巧。20 世纪 70 年代以后的对称密码则同时使用代换和置换技巧。
- 古典加密技术分为两类：一类是单字母代换密码，它将明文的一个字符用相应的一个密文字符代替。另一类是多字母代换密码，它是对多于一个字母进行代换。单字母代换密码又分为单表代换密码和多表代换密码。
- DES 是第一个加密标准，它与古典加密技术不一样，DES 同时使用了代换和置换两种技巧。用 56 位密钥加密 64 位明文。
- AES 是用来取代 DES 的高级加密标准，其结构与 DES 不同，它是用 128、192 或者 256 位密钥加密 128 位的分组。
- SMS4 是我国官方公布的第一个商用密码算法，它是一种分组对称密码算法，用 128 位密钥加密 128 位的分组。
- RC6 是 RSA 公司提交给 NIST 的一个候选高级加密标准算法，其效率非常高。
- RC4 是被广泛使用的一种同步流密码。
- 在密码学中的很多场合下都要使用随机数，安全的随机数应该满足随机性和不可预测性。
- 密钥分配为通信的双方发送会话密钥。

密码技术是信息系统最重要的安全机制。密码技术主要分为对称密码技术(也称单钥或者传统密码技术)和非对称密码技术(也称双钥或者公钥密码技术)。在对称密码技术中，加密密钥和解密密钥相同，或者一个密钥可以从另一个密钥导出。而非对称密码技术则使用两个密钥，加密密钥和解密密钥不相同。对称密码技术主要使用两种技巧：代换和置换。代换是将明文中的每个元素映射成另一个元素。置换是将明文中的元素重新排列。在 20 世纪 70 年代以前的加密技术都是对称加密技术，并且在这些加密技术中只使用了代换或者置换技巧。这个时期的加密技术也称为古典加密技术。在 20 世纪 70 年代以后出现的对称加密技术则同时使用了代换和置换两种技巧。这两个阶段的加密技术还有一个典型区别：古典加密技术一般将加密算法保密，而现代的对称加密技术则公开加密算法，加密算法的安

全性只取决于密钥,不依赖于算法。非对称密码技术则产生于20世纪70年代。

3.1 基本概念

密码学(Cryptology)是以研究秘密通信为目的,即对所要传送的信息采取一种秘密保护,以防止第三者对信息进行窃取的一门学科。密码学作为数学的一个分支,包括密码编码学(Cryptography)和密码分析学(Cryptanalysis)两部分。密码编码学是研究加密原理与方法,使消息保密的技术和科学,它的目的是掩盖消息内容。密码分析学则是研究破解密文的原理与方法。密码分析者(Cryptanalyst)是从事密码分析的专业人员。

采用加密的方法伪装消息,使得未授权者不可理解被伪装的消息。被伪装的原始消息(Message)称为明文(Plaintext)。将明文转换为密文的过程称为加密(Encryption),加了密的消息称为密文(Ciphertext),而把密文转变为明文的过程称为解密(Decryption)。加密解密过程如图3.1所示。将明文转换为密文的算法称为密码(Cipher)。一个加密系统采用的基本工作方式叫做密码体制(Cryptosystem)。实际上在密码学中的"系统或体制(System)"、"方案(Scheme)"和"算法(Algorithm)"等术语本质上是一回事,在本书中我们也将使用这些术语。加密和解密算法通常是在一组密钥(Key)控制下进行的,分别称为加密密钥和解密密钥。如果加密密钥和解密密钥相同,则密码系统为对称密码系统。

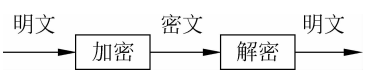

图3.1 加密解密过程

3.2 对称密码模型

对称密码也称传统密码,它的特点是发送方和接收方共享一个密钥。对称密码分为两类:分组密码(Block Ciphers)和流密码(Stream Ciphers)。分组密码也称为块密码,它是将信息分成一块(组),每次操作(如加密和解密)是针对一组而言。流密码也称序列密码,它每次加密(或者解密)一位或者一个字节。

一个对称密码系统(也称密码体制)由5个部分组成。用数学符号描述为$S=\{M,C,K,E,D\}$,如图3.2所示。

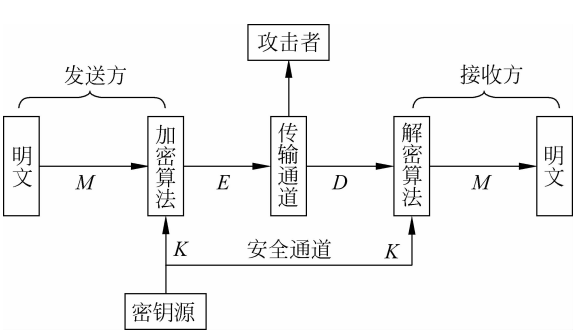

图3.2 对称密码系统模型

(1) 明文空间 M,表示全体明文的集合。

(2) 密文空间 C,表示全体密文的集合。

(3) 密钥空间 K,表示全体密钥的集合,包括加密密钥和解密密钥。

(4) 加密算法 E,表示由明文到密文的变换。

(5) 解密算法 D,表示由密文到明文的变换。

在发送方,对于明文空间的每个明文,加密算法在密钥的作用下生成对应的密文。接收方将接收的密文,用解密算法在解密密钥的控制下变换成明文。我们可以看到加密算法有两个输入,一个是明文;另一个是密钥。加密算法的输出是密文。解密算法本质上是加密算法的逆运行,解密算法的输入是密文和密钥,输出是明文。

对明文 M 用密钥 K,使用加密算法 E 进行加密,常常表示为 $E_k(M)$,同样用密钥 K 使用解密算法 D 对密文 C 进行解密,表示为 $D_k(C)$。在对称加密体制中,解密密钥相同,有:

$$C = E_k(M)$$

$$M = D_k(C) = D_k(E_k(M))$$

从对称密码模型可以看到,发送方和接收方主要进行加密和解密运算,我们希望这个运算越容易越好,对于攻击者而言,我们希望他们破译密文的计算越难越好。因此一个好的密码体制至少要满足下面几个条件。

(1) 已知明文 M 和加密密钥 K 时,易于计算 $C=E_k(M)$。

(2) 加密算法必须足够强大,使破译者不能仅根据密文破译消息,即在不知道解密密钥 K 时,由密文 C 计算出明文 M 是不可行的。

(3) 由于对称密码系统双方使用相同的密钥,因此还必须保证能够安全地产生密钥,并且能够以安全的形式将密钥分发给双方。

(4) 对称密码系统的安全只依赖于密钥的保密,不依赖于加密和解密算法的保密。

3.3　密 码 攻 击

分析一个密码系统是否安全,一般是在假定攻击者知道所使用的密码系统情况下进行分析的。一般情况下,密码分析者可以得到密文,知道明文的统计特性、加密体制、密钥空间及其统计特性,但不知道加密截获的密文所用的特定密钥。这个假设称为 Kerckhoff 假设。分析一个密码系统的安全性一般是建立在这个假设的基础上。当然,如果攻击者不知道所使用的密码体制,那么破译是更难的。但是,不应当把密码系统的安全性建立在攻击者不知道所使用的密码体制这个前提之下。因此,在设计一个密码系统时,其目的应当是在 Kerckhoff 假设下达到一定的安全程度。

攻击对称密码体制有两种方法:密码分析和穷举攻击(Brute Force Search)。密码分析是依赖加密算法的性质和明文的一般特征等,试图破译密文得到明文或试图获得密钥的过程。穷举攻击则是试遍所有可能的密钥对所获密文进行解密,直至得到正确的明文;或者用一个确定的密钥对所有可能的明文进行加密,直到得到与所获得的密文一致。

3.3.1　穷举攻击

穷举攻击是最基本的,也是比较有效的一种攻击方法。从理论上讲,可以尝试所有的密

钥。因此只要有足够的资源,任何密码体制都可以用穷举攻击将其攻破。幸运的是,攻击者不可能有无穷的可用的资源。

穷举攻击的代价与密钥大小成正比。穷举攻击所花费的时间等于尝试次数乘以一次解密(加密)所需的时间。显然可以通过增大密钥位数或加大解密(加密)算法的复杂性来对抗穷举攻击。当密钥位数增大时,尝试的次数必然增大。当解密(加密)算法的复杂性增大时,完成一次解密(加密)所需的时间增大。从而使穷举攻击在实际上不能实现。表3.1是穷尽密钥空间所需的时间。从表3.1中我们可以发现,当密钥长度达到128位以上时,以目前的资源来说,穷举攻击将不会成功。

表3.1 穷尽密钥空间所需的时间

密钥长度(位)	密 钥 数 目	每微秒尝试1次 所需时间	每微秒尝试10^6次 所需时间
32	$2^{32}=4.3\times10^9$	2^{31}微秒=35.8分	2.15毫秒
56	$2^{56}=7.2\times10^{16}$	2^{55}微秒=1142年	10.01小时
128	$2^{128}=3.4\times10^{38}$	2^{127}微秒=5.4×10^{24}年	5.4×10^{18}年
168	$2^{168}=3.7\times10^{50}$	2^{167}微秒=5.9×10^{36}年	5.9×10^{30}年
26个字母排列	$26!=4\times10^{26}$	2×10^{26}微秒=6.4×10^{12}年	6.4×10^{6}年

3.3.2 密码攻击类型

密码分析是基于Kerckhoff假设的。密码分析者所使用的策略取决于加密方案的性质以及可供密码分析者使用的信息,正是基于密码分析者所知的信息量,可把对密码的攻击分为以下几种类型。

- 唯密文攻击(Ciphertext-Only Attack)。密码分析者有一些消息的密文,这些消息都用同一算法加密。密码分析者的任务是恢复尽可能多的明文,或者是最好能推算出加密消息的密钥,以便采用相同的密钥解出其他被加密的消息。这种情况下,密码分析者知道的东西只有两样:加密算法和待破译的密文。
- 已知明文攻击(Known-Plaintext Attack)。密码分析者除知道加密算法和待破译的密文外,而且也知道有一些明文和同一个密钥加密的这些明文所对应的密文,即知道一定数量的明文和对应的密文。
- 选择明文攻击(Chosen-Plaintext Attack)。密码分析者知道加密算法和待破译的密文,并且可以得到所需要的任何明文所对应的密文,这些明文和待破译的密文是用同一密钥加密得来的,即知道选择的明文和对应的密文。如在公钥密码体制中,攻击者可以利用公钥加密他任意选择的明文。
- 选择密文攻击(Chosen-Ciphertext Attack)。密码分析者知道加密算法和待破译的密文,密码分析者能选择不同的被加密的密文,并可得到对应的解密的明文,即知道选择的密文和对应的明文。解密这些密文所使用的密钥与解密待破解的密文的密钥是一样的。这种攻击主要用于公钥密码算法。
- 选择文本攻击(Chosen Text Attack)。选择文本攻击是选择明文攻击和选择密文攻击的结合。密码分析者知道加密算法和待破译的密文,并且知道任意选择的明文和它对应的密文,这些明文和待破译的密文是用同一密钥加密得来的,以及有目的地

选择的密文和它对应的明文，解密这些密文所使用的密钥与解密待破解的密文的密钥是一样的。

在以上任何一种情况下，攻击者的目标都是为了确定正在使用的密钥。显然，上述 5 种攻击类型的强度按序递增，如果一个密码系统能够抵抗选择明文攻击，那么它也能抵抗唯密文攻击和已知明文攻击。一般来说，一个密码体制是安全的，通常是指在受到前三种攻击下系统的安全性，即攻击者一般容易具备前 3 种攻击条件。在这几种攻击类型中，唯密文攻击难度最大，因为攻击者可利用的信息最少。在此情况下，一种可能的攻击方法是对所有可能的密钥尝试的强行攻击法，即穷举攻击。如果密钥量非常大，则该方法是不现实的。因此，攻击者通常运用各种统计方法对密文本身进行分析。如果攻击者知道的信息越多，就越容易破解密文。在多数情况下，密码分析者能够获得除密文以外的更多信息，如能够获得一段或者多段明文以及对应的密文，或者可能知道某种明文模式将出现在某个消息中，此时可以进行已知明文攻击，攻击者可以从转换明文的方法来推导密钥。

对密码设计者而言，被设计的加密算法一般要能经受得住已知明文的攻击。如果无论攻击者有多少密文，由一个加密算法产生的这些密文中包含的信息都不足以唯一决定对应的明文，也无论用什么技术方法进行攻击都不能被攻破，这种加密算法则是绝对安全(Unconditional Security)。绝对安全指不论攻击者具有多少计算能力都无法破解密文。除一次一密(One-Time Pad)外，没有绝对安全的加密算法。因此，加密算法的使用者应该挑选满足下列标准中的一个或两个的算法。

(1) 破译该密码的成本超过被加密信息的价值。

(2) 破译该密码的时间超过该信息有价值的生命周期。

如果满足上述的两个准则，一个加密算法就可认为是在计算上安全(Computational Security)的。计算上安全是指在计算能力有限的情况下(如计算所需时间比宇宙生存时间还长)，无法破解此密文。目前的加密算法一般在计算上是安全的。

3.3.3　密码分析方法

当密钥长度增加到一定的大小时，穷举攻击变得不切实际。因此用密码分析的方法攻击密码越来越引起人们的重视，目前比较流行的密码分析方法是线性密码分析和差分密码分析。这两种方法主要是针对现代密码的攻击。

线性分析是一种已知明文攻击，最早由 Matsui 在 1993 年提出。线性分析是一种统计攻击，它以求线性近似为基础。通过寻找现代密码算法变换的线性近似来攻击。如用这种方法在只需要知道 2^{43} 个已知明文的情况下就可以找到 DES 的密钥。

差分密码分析在许多方面与线性密码分析相似，它与线性密码分析的主要区别在于差分密码分析包含了将两个输入的异或与其相对应的两个输出的异或相比较。差分密码分析也是一个选择明文攻击。差分密码分析被公认为近年来密码分析的最大成就。差分密码分析出现于 20 世纪 70 年代，但在 1990 年才被公开发布。它的基本思想是：通过分析明文对的差值与密文对的差值的影响来恢复某些密钥位。差分分析可用来攻击任何一个拥有固定迭代轮函数结构的密码算法。

3.4 古典加密技术

古典加密技术主要使用代换或者置换技术。代换(Substitution)是将明文字母替换成其他字母、数字或者符号。置换(Permutation)则保持明文的所有字母不变,只是打乱明文字母的位置和次序。这些古典代换加密技术分为两类,一类是单字母代换密码(Monogram Substitution Cipher),它将明文的一个字符用相应的一个密文字符代替。另一类是多字母代换密码(Polygram Substitution Cipher),它是对多于一个字母进行代换。在单字母代换密码中又分为单表代换密码(Monoalphabetic Substitution Cipher)和多表代换密码(Polyalphabetic Substitution Cipher)。单表代换密码只使用一个密文字母表,并且用密文字母表中的一个字母来代替一个明文字母表中的一个字母。多表代换密码是将明文消息中出现的同一个字母,在加密时不完全被同一个固定的字母代换,而是根据其出现的位置次序,用不同的字母代换。

3.4.1 单表代换密码

单表代换密码只使用一个密文字母表,并且用密文字母表中的一个字母来代替一个明文字母表中的一个字母。设 M 和 C 分别表示为含 n 个字母的明文字母表和密文字母表。

$$M = \{m_0, m_1, \cdots, m_{n-1}\}$$

$$C = \{c_0, c_1, \cdots, c_{n-1}\}$$

如果 f 为一种代换方法,那么密文为 $C=E_k(m)=c_0c_1\cdots c_{n-1}=f(m_0)f(m_1)\cdots f(m_{n-1})$。

单表代换密码常见的方法有加法密码、乘法密码和仿射密码。在本章的例子中,我们将用小写字母表示明文,用大写字母表示密文。明文和密文空间都假设为26个字母,即属于 Z_{26},当然很容易推广到 n 个字母的情况。

1. 加法密码

对每个 $c, m \in Z_n$,加法密码的加密算法和解密算法是:

$$C = E_k(m) = (m + k) \bmod n$$

$$M = D_k(c) = (c - k) \bmod n$$

k 是满足 $0<k<n$ 的正整数。若 n 是26个字母,加密方法是用明文字母后面第 k 个字母代替明文字母。因此,代换密码中的加密和解密可以看做是字母表上的一个字母的置换。Caesar密码是典型的加法密码。

Caesar密码是已知最早的单表代换密码,采用加法加密的方法,由Julius Caesar发明,最早用在军事上。将字母表中的每个字母,用它后面的第3个字母代替,如下所示。

- 明文:meet me after the toga party
- 密文:PHHW PH DIWHU WKH WRJD SDUWB

代换方式的定义,如下:

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

可让每个字母等价一个数字，如下：

a	b	c	d	e	f	g	h	i	j	k	l	m	n
0	1	2	3	4	5	6	7	8	9	10	11	12	13

o	p	q	r	s	t	u	v	w	x	y	z
14	15	16	17	18	19	20	21	22	23	24	25

对每个明文字母 m，用密文字母 c 代换，那么 Caesar 密码算法如下所示。

- 加密：$C=E(m)=(m+3) \bmod 26$
- 解密：$M=D(c)=(c-3) \bmod 26$

移位可以是任意的，如果用 $k(1\leqslant k\leqslant 25)$表示移位数，则通用的 Caesar 密码算法表示如下。

- 加密：$C=E_k(m)=(m+k) \bmod 26$
- 解密：$M=D_k(c)=(c-k) \bmod 26$

对 Caesar 密码安全性的分析如下所示。

前面已经介绍过，对密码的分析是基于 Kerckhoff 假设的。因此假设攻击者知道使用 Caesar 密码加密。如果攻击者只知道密文，即唯密文攻击，只要穷举测试所有可能字母移位的距离，最多尝试 25 次。实际上攻击者为了加快穷举速度，只要对密文中一个单词进行猜想解密，就可以加快判断密钥的正确性。如果攻击者知道一个字符以及它对应的密文，即已知明文攻击，那么攻击者很快就会通过明文字符和对应的密文字符之间的距离推算出密钥。这个例子说明一个密码体制安全至少要能够抵抗穷举密钥搜索攻击，普通的做法是将密钥空间变得足够大。但是，很大的密钥空间并不是保证密码体制安全的充分条件，下面的例子可以说明这一点。

我们对 Caesar 密码进行改进，假设密文是 26 个字母的任意代换，密钥是明文字母到密文字母的一个字母表，密钥长度是 26 字长。

例如字母代换表如下：

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
D	K	V	Q	F	I	B	J	W	P	E	S	C	X	H	T	M	Y	A	U	O	L	R	G	Z	N

若加密的明文为 ifwewishtoreplaceletters，那么对应的密文为 WIRFRWAJUHYFTSDVFSFUUFYA。

上面的字母代换表由通信双方事先设计好，一个更实际的构造字母代换表的方法是使用一个密码句子。如密钥句子为 the message was transmitted an hour ago，按照密钥句子中的字母依次填入字母表(重复的字母只用一次)，未用的字母按自然顺序排列。这样可以构造如下的字母代换表。

原字母表如下：

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

代换字母表如下：

T	H	E	M	S	A	G	W	R	N	I	D	O	U	B	C	F	J	K	L	P	Q	V	X	Y	Z

若明文为 please confirm receipt,使用上面的代换字母表,则密文为 CDSTKSEBUARJ OJSESRCL。

使用上面的方法代换,总共有 $26!=4\times10^{26}$ 种密钥,从表 3.1 可以看到穷举搜索这么多的密钥很困难。但这并不表示该密码不容易破解。破解这类密码的突破点是由于语言本身的特点是充满冗余的,每个字母使用的频率不相等。由于上面加密后的密文实际上是明文字母的一个排列,因此单表代换密码没有改变字母相对出现的频率,明文字母的统计特性在密文中能够反映出来,即保持明文的统计特性不变。通过统计密文字母的出现频率,可以确定明文字母和密文字母之间的对应关系。英文字母中单字母出现的频率如图 3.3 所示。

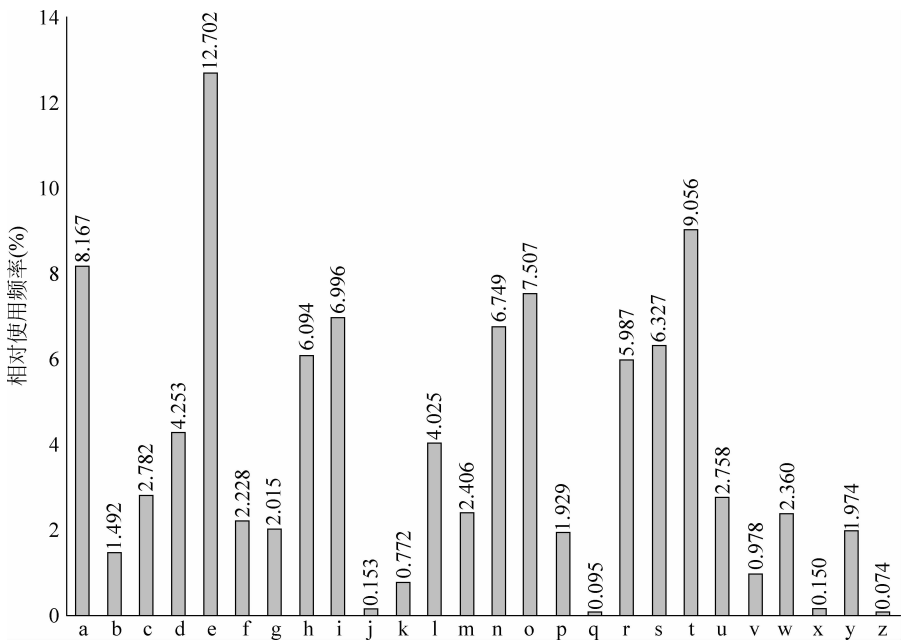

图 3.3 英文字母中单字母出现的频率

图 3.3 中的 26 个字母按照出现频率的大小可以分为下面 5 类。

(1) e:出现的频率大约为 12.7%。

(2) t、a、o、i、n、s、h、r:出现的频率大约为 6%~9%。

(3) d 和 l:出现的频率约为 4%。

(4) c、u、m、w、f、g、y、p、b:出现的频率大约为 1.5%~2.8%。

(5) v、k、j、x、q、z:出现的频率小于 1%。

双字母和三字母组合都有现成的统计数据,常见的双字母组合和三字母组合统计表能够帮助破解密文。

出现频率最高的 30 个双字母(按照频率从高到低排列)如下:

th he in er an re ed on es st
en at to nt ha nd ou ea ng as
or ti is et it ar te se hi of

出现频率最高的 20 个三字母(按照频率从高到低排列)如下:

the ing and her ere ent tha nth was eth
for dth hat she ion int his sth ers ver

例 3.1　已知下面的密文是由单表代换产生的。

UZQSOVUOHXMOPVGPOZPEVSGZWSZOPFPESXUDBMETSXAIZVUEPHZHMDZSHZOWSFPAPPDTSVPQUZWYMXUZUHSXEPYEPOPDZSZUFPOMBZWPFUPZHMDJUDTMOHMQ

试破译该密文。

首先统计密文中字母出现的频率，然后与英文字母出现的频率进行比较。密文中字母的相对频率统计如表 3.2 所示。

表 3.2　密文中字母的相对频率统计

字母	次数	频率(%)	字母	次数	频率(%)	字母	次数	频率(%)	字母	次数	频率(%)
A	2	1.67	H	7	5.83	O	9	7.50	V	5	4.17
B	2	1.67	I	1	0.83	P	16	13.33	W	4	3.33
C	0	0.00	J	1	0.83	Q	3	2.50	X	5	4.17
D	6	5.00	K	0	0.00	R	0	0.00	Y	2	1.67
E	6	5.00	L	0	0.00	S	10	8.33	Z	14	11.67
F	4	3.33	M	8	6.67	T	3	2.55	V	5	4.17
G	2	1.67	N	0	0.00	U	10	8.33	W	4	3.33

将统计结果与图 3.3 进行比较，可以猜测密文中 P 与 Z 可能是 e 和 t，密文中的 S、U、O、M 出现频率比较高，可能与明文字母中出现频率相对较高的 a、o、i、n、s、h、r 这些字母对应。密文中出现频率很低的几个字母 C、K、L、N、R、I、J 可能与明文字母中出现频率较低的字母 v、k、j、x、q、z 对应。就这样边试边改，最后得到如下明文。

it was disclosed yesterday that several informal but direct contacts have been made with political representatives of the viet cong in moscow

在尝试过程中，如果同时使用双字母和三字母的统计规律，那么更容易破译密文。如上面的密文中出现最多的双字母是 ZW，它可能对应明文双字母出现频率较大的 th，那么 ZWP 就可能是 the，这样就更容易试出明文。

2. 乘法密码

对每个 $c, m \in Z_n$，乘法密码的加密和解密算法是：

$$C = E_k(m) = (mk) \bmod n$$

$$M = D_k(c) = (ck^{-1}) \bmod n$$

其中 k 和 n 互素，即 $\gcd(k,n)=1$，否则不存在模逆元，不能正确解密。显然乘法密码的密码空间大小是 $\varphi(n)$，$\varphi(n)$是欧拉函数。可以看到乘法密码的密钥空间很小，当 n 为 26 字母，则与 26 互素的数是 1、3、5、7、9、11、15、17、19、21、23、25，即 $\varphi(n)=12$ 因此乘法密码的密钥空间为 12。

乘法密码也称采样密码，因为密文字母表是将明文字母按照下标每隔 k 位取出一个字母排列而成。

例 3.2　英文字母，选取密码为 9，使用乘法密码的加密算法，那么明文字母和密文字母的代换表构造如表 3.3 所示。

表 3.3 明文字母和密文字母的代换表

原字母	a	b	c	d	e	f	g	h	i	j	k	l	m
原字母的值	0	1	2	3	4	5	6	7	8	9	10	11	12
代换字母的值	0	9	18	1	10	19	2	11	20	3	12	21	4
代换字母	A	J	S	B	K	T	C	L	U	D	M	V	E
原字母	n	o	p	q	r	s	t	u	v	w	x	y	z
原字母的值	13	14	15	16	17	18	19	20	21	22	23	24	25
代换字母的值	13	22	5	14	23	6	15	24	7	16	25	8	17
代换字母	N	W	F	O	X	G	P	Y	H	Q	Z	I	R

若明文为 a man liberal in his views,那么密文为 AENVUJKXUNLUGHUKQG。

3. 仿射密码

将加法密码和乘法密码结合就构成了仿射密码,仿射密码的加密和解密算法是:

$$C = E_k(m) = (k_1 m + k_2) \bmod n$$

$$M = D_k(c) = k_1^{-1}(c - k_2) \bmod n$$

仿射密码具有可逆性的条件是 $\gcd(k,n)=1$。当 $k_1=0$ 时,仿射密码变为加法密码,当 $k_2=0$ 时,仿射密码变为乘法密码。

仿射密码中的密钥空间的大小为 $n\varphi(n)$,当 n 为英文字母的个数,即 26,$\varphi(n)=12$,因此仿射密码的密钥空间为 $12\times26=312$。

例 3.3 设密钥 $K=(7,3)$,用仿射密码加密明文 hot。

3 个字母对应的数值是 7、14 和 19。分别加密如下:

$$(7\times7+3) \bmod 26 = 52 \bmod 26 = 0$$

$$(7\times14+3) \bmod 26 = 101 \bmod 26 = 23$$

$$(7\times19+3) \bmod 26 = 136 \bmod 26 = 6$$

3 个密文数值为 0、23 和 6,对应的密文是 AXG。

例 3.4 假设获得仿射密码加密的密文是:

FMXVEDKAPHRERBNDKRXRSREFMORUD5DXYV5HVUPEDKAPRKDLYEVLRHHHRH

试破译该密码。

同样可以统计密文中各字母出现的频率,然后与英文字母出现频率比较,在尝试过程中同时要考虑仿射密码的条件。

各个字母出现的频率统计如表 3.4 所示。

表 3.4 例 3.4 中各字母出现的频率

字母	频率(次数)	字母	频率(次数)	字母	频率(次数)	字母	频率(次数)
A	2	H	5	O	1	V	4
B	1	I	0	P	2	W	0
C	0	J	0	Q	0	X	2
D	7	K	5	R	8	Y	1
E	5	L	2	S	3	Z	0
F	4	M	2	T	0		
G	0	N	1	U	2		

这里虽然只有 57 个字母，但它足以分析仿射密码，最大频率的密文字母是 R(8 次)，D(7 次)，E、H、K(各 5 次)和 S、F、V(各 4 次)。首先，我们可以猜想 R 是 e 的加密，而 D 是 t 的加密，因为 e 和 t 是两个出现频率最高的字母。e 和 t 对应的数值是 4 和 19，R 和 D 对应的数值是 17 和 3。对于仿射密码，有 $c=(k_1m+k_2)$。

所以我们有如下的关于两个未知数线性方程组：

$$17=4k_1+k_2$$

$$13=19k_1+k_2$$

这个方程组有唯一解 $k_1=6, k_2=19$，但这不是一个合法的密钥，因为 gcd(6,26)=2，不等于 1。

我们再猜测 R 是 e 的加密，而 E 是 t 的加密，继续使用上述的方法，得到 $k_1=13$，这也是一个不合法的密钥。再试一种可能性：R 是 e 的加密，H 是 t 的加密，则有 $k_1=8$，这也是不合法的。继续进行，我们猜测 R 是 e 的加密，K 是 t 的加密，这样可得 $k_1=3, k_2=5$，首先它至少是一个合法的密钥，下一步工作就是检验密钥 $K=(3,5)$ 的正确性。如果我们能得到有意义的英文字母串，则可证实该密钥是有效的。对密文进行解密有：

algorithms are quite general definitions of arithmetic processes

3.4.2　多表代换密码

单表代换密码是将明文的一个字母唯一地代换为一个字母。加密后的密文具有明文的特征，通过统计密文中字母出现的频率能够比较方便地破解密文。要提高密码的强度，应该让明文结构在密文中尽量少出现。多表代换密码和多字母代换密码能够减少这种密文字母和明文字母之间的对应关系。本节将介绍多表代换密码，第 3.4.3 节将介绍多字母代换密码。

多表代换密码是对每个明文字母信息采用不同的单表代换，也就是用一系列(两个以上)代换表依次对明文消息的字母进行代换的加密方法。

如果明文字母序列为 $m=m_1m_2\cdots$，令 $f=f_1, f_2, \cdots$ 为代换序列，则对应的密文字母序列为：

$$C = E_k(m) = f_1(m_1)f_2(m_2)\cdots$$

若代换系列为非周期无限序列，则相应的密码为非周期多表代换密码。这类密码对每个明文字母都采用了不同的代换表或密钥进行加密，称作是一次一密密码(One-Time Pad Cipher)。这是一种在理论上唯一不可破的密码，一次一密对于明文的特征可实现完全隐蔽，但由于需要的密钥量和明文消息长度相同而难以广泛使用。

在实际中，经常采用周期多表代换密码，它通常只使用有限的代换表，代换表被重复使用以完成对消息的加密。此时代换表系列为：

$$f = f_1, f_2, \cdots, f_d, f_1, f_2, \cdots, f_d, \cdots$$

在对明文字母序列为 $m=m_1m_2\cdots$ 进行加密时，相应的密文字母系列为：

$$C = E_k(m) = f_1(m_1)f_2(m_2)\cdots f_d(m_d)f_1(m_{d+1})f_2(m_{d+2})\cdots f_d(m_{2d})\cdots$$

当 $d=1$ 时，多表代换密码变为单表代换密码。

下面介绍一种比较有名的多表代换密码——维吉尼亚密码。

1. 维吉尼亚密码

维吉尼亚(Vigenère)密码是一种周期多表代换密码,由1858年法国密码学家维吉尼亚提出。它的形式化描述如下。

密钥 $K=(k_1,k_2,\cdots,k_d)$,d 为代换周期长度,将明文 $M=(m_1,m_2,\cdots,m_t)$分为长度为 d 的分段。

加密函数为:

$$\begin{aligned}C=E_k(m)=&((m_1+k_1)\bmod n,(m_2+k_2)\bmod n,\cdots,(m_d+k_d)\bmod n,\\&(m_{d+1}+k_1)\bmod n,(m_{d+2}+k_2)\bmod n,\cdots,(m_{2d}+k_d)\bmod n,\\&\vdots\\&(m_{t-d+1}+k_1)\bmod n,(m_{t-d+2}+k_2)\bmod n,\cdots,(m_t+k_d)\bmod n)\end{aligned}$$

如果密文为 $C=(c_1,c_2,\cdots,c_n)$,则解密函数为:

$$\begin{aligned}M=D_k(c)=&((c_1-k_1)\bmod n,(c_2-k_2)\bmod n,\cdots,(c_d-k_d)\bmod n,\\&(c_{d+1}-k_1)\bmod n,(c_{d+2}-k_2)\bmod n,\cdots,(c_{2d}-k_d)\bmod n,\\&\vdots\\&(c_{t-d+1}-k_1)\bmod n,(c_{t-d+2}-k_2)\bmod n,\cdots,(c_t-k_d)\bmod n)\end{aligned}$$

假设明文和密文都是26个英文字母,维吉尼亚密码常常使用英文单词作为密钥字,密钥则是密钥字的重复。比如密钥字是 computer,用它加密明文 sender and recipient share a common key。那么密钥将如下所示。

- 明文:senderandrecipientshareacommonkey
- 密钥:computercomputercomputercomputerc

维吉尼亚密码的加密过程简述如下。

(1) 写下明文,表示为数字形式。

(2) 在明文之上重复写下密钥字,也表示为数字形式。

(3) 加密相对应的明文:给定一个密钥字母 k 和一个明文字母 m,那么密文字母则是 $(m+k)\bmod 26$ 计算结果所对应的字母。

例 3.5 设密钥字是 cipher,明文串是 this cryptosystem is not secure,求密文。

在明文下面重复写密钥字,组成密钥。

- 明文 M:thiscryptosystemisnotsecure
- 密钥 K:ciphercipherciphercipherciphercip

将明文和密钥转化为数字:

明文 $M=(19,7,8,18,2,17,24,15,19,14,18,24,18,19,4,12,8,18,13,14,19,18,4,2,20,17,4)$

密钥 $K=(2,8,15,7,4,17,2,8,15,7,4,17,2,8,15,7,4,17,2,8,15,7,4,17,2,8,15)$

对每个明文数字和对应的密钥数字,使用 $c_i=(m_i+k_i)\bmod 26$ 加密。得到密文数字为 $C=(21,15,23,25,6,8,0,23,8,21,22,15,21,1,19,19,12,9,15,22,8,25,8,19,22,25,19)$

于是密文为:

VPXZGIAXIVWPUBTTMJPWIZITWZT

可以看出，维吉尼亚密码是将每个明文字母映射为几个密文字母，如果密钥字的长度是 m，则明文中的一个字母能够映射成这 m 个可能的字母中的一个。因此密文中字母出现的频率被隐蔽了，它的安全性明显比单表代换密码提高了。维吉尼亚密码的密钥空间比较大，对于长度是 m 的密钥字，密钥空间为 26^m，当 $m=5$，密钥空间所含密钥的数量大于 1.1×10^7。

2. 一次一密

一次一密是非周期多表代换密码，它使用与明文一样长且无重复的随机密钥来加密明文，并且该密钥使用一次后就不再使用。在实际使用时，通信双方事先协商一个足够长的密钥序列，要求密钥序列中的每一项都是按均匀分布随机地从一个字符表中选取的。双方各自秘密保存密钥序列。每次通信时，发送方用自己保存的密钥序列中的密钥，按次序对要发送的消息进行加密。消息加密完成后，把密钥序列中刚使用过的这一段销毁。接收方每次收到密文消息后，使用同样的密钥序列解密。解密完成后，立即把密钥序列中刚使用过的这一段销毁。由于密钥是随机的，用它加密明文后的密文也是随机的，密文中没有任何明文的特征，因此这个加密方案是绝对完全的。

例如，明文是 cryptosystem，选取的密钥序列是 djfstlngwjpw。

明文转换数字为：

$$M=(2,17,24,15,19,14,18,24,18,19,4,12)$$

密钥转换数字为：

$$K=(3,9,5,18,19,11,13,6,22,9,15,22)$$

那么可以得到密文数字为：

$$C=(5,0,3,7,12,25,5,4,14,2,19,8)$$

则密文为 fadhmzfeocti。

当明文中的字符是位时，密钥序列中的一项也是一位。加密时常采用位异或。

例如，设明文消息是 0010101，密钥是 10101100，那么加密过程如下：

$$\text{密文}=\text{明文}\oplus\text{密钥}=0010101\oplus10101100=10000101$$

解密时将密文与密钥异或：

$$\text{明文}=\text{密文}\oplus\text{密钥}=10000101\oplus10101100=0010101$$

一次一密不可破解的原因是，对于一段密文，与密文相同长度的字母串都可能是明文，密文不能提供明文和密钥的任何信息。对任何与密文一样长的明文，也存在一个密钥用于产生这个明文。也就是说你用穷举搜索所有可能的密钥，就会找到大量可读的明文，也就不可能确定哪一个是真正需要的明文，可见一次一密是绝对安全的。

由于一次一密的安全性是取决于密钥的随机性，因此首先需要解决产生随机的密钥序列的问题，但产生大规模随机密钥是一件很困难的事情，目前还没有很好的办法来解决这个问题。另外，密钥分配也是一个难点，由于密钥不允许重复使用，因此存在大量的密钥分配问题。由于这些困难，在实际中人们很少使用一次一密，一次一密主要是用于高度机密的低带宽信道。

3.4.3　多字母代换密码

前面介绍的密码都是以单字母作为代换对象，如果每次对多个字母进行代换就是多字

母代换密码。多字母代换的优点是容易隐藏字母的自然出现频率,有利于对抗统计分析。下面介绍常见的多字母代换密码。

1. Playfair 密码

Playfair 密码是将明文中的双字母音节作为一个单元,并将其转换成密文的双字母音节(即一次代换两个字母)。Playfair 算法是基于一个由密钥组成的 5×5 阶矩阵。假设密钥是 monarchy,构建矩阵的方法是将密钥(去掉重复的字母)从左到右、从上到下填入矩阵中,再将剩余的字母按照字母表的顺序依次填入。在该矩阵中,字母 i 和 j 暂且被看作一个字母。这样可以构成如下的密钥矩阵。

M	O	N	A	R
C	H	Y	B	D
E	F	G	I/J	K
L	P	Q	S	T
U	V	W	X	Z

Playfair 按照下面的原则加密与解密。

每次以两个字母为一个单位进行操作。

(1) 如果这两个字母一样,则在中间插入一个字母 x(事先约定的一个字母),如 balloon 变成 ba lx lo on。

(2) 如果明文长度不是 2 的倍数,则在最后填入一个实现约定的字母 x。如 table 变为 ta bl ex。

(3) 如果两个字母在矩阵的同一行,则用它右边的字母来代替它(最后一个字母的右边是第 1 个字母),如 ar 被加密变为 RM。

(4) 如果两个字母在同一列,则用它下面的字母来代替它(最底下的字母的下一个是该列第 1 个字母),如 mu 被加密变为 CM。

(5) 其他的字母都用它同一行,另一个字母的同一列相交的字母代替,如 hs 加密变为 BP,ea 变为 IM 或者 JM(由加密者自行决定)。

例 3.6 假设密钥是 cipher,使用 Playfair 算法加密 playfair cipher was actually invented by wheatston。

由密钥词 cipher 可构建如下的密钥矩阵。

C	I	P	H	E
R	A	B	D	F
G	K	L	M	N
O	Q	S	T	U
V	W	X	Y	Z

将明文按照两个字母分组为:

pl ay fa ir ci ph er wa sa ct ua lx ly in ve nt ed by wh ea ts to nx

则密文为:

BS DW RB CA IP HE CF IK QB HO QF SP MX EK ZC MU HF DX YI IF UT UQ LZ

Playfair 密码的安全性比单字母代换密码提高了许多,双字母共有 26×26=676 组合,

因此频率统计分析表中需要 676 条统计数据(而单表代换密码只有 26 条),另外 Playfair 密码比单字母代换更好地隐藏了明文中单字母的结构。一个字母可能代换为不同的字母,如在使用密钥词是 monarchy 构成的矩阵中,我们观察字母 a 与不同的字母组合加密后的变化,如 as →BX、ar→RM、af→OI 等,字母 a 可以变换为 B,以及 a 所在行的所有字母(除 a 自己外)。如果密钥词变化,那么 a 在理论上可以代换为除自己外的任何一个字母,因此 Playfair 密码能够较好地隐藏明文的特征。Playfair 密码在过去一段时间曾经被广泛使用,在第一次世界大战中被英军作为最好的密码系统使用,在第二次世界大战中也曾经被美军和盟军大量使用。当然现在看来,该密码的安全性是很低的,它还有明文的部分特征,只要给定几百个字母的密文情况下,该加密方法就可以被破解。

2. Hill 密码

该密码是 1929 年由数学家 Lester Hill 发明的一种多字母代换密码。加密算法将 m 个明文字母替换成 m 个密文字母(Hill_m 密码表示 m 个明文字母为一组)。这种代换由 m 个线性方程决定。

如果 $m=3$,则该密码系统可以表示为:

$$C_1 = (k_{11}m_1 + k_{12}m_2 + k_{13}m_3) \bmod 26$$
$$C_2 = (k_{21}m_1 + k_{22}m_2 + k_{23}m_3) \bmod 26$$
$$C_3 = (k_{31}m_1 + k_{32}m_2 + k_{33}m_3) \bmod 26$$

用向量或者矩阵表示为:

$$\begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix} \bmod 26 \quad \text{或者 } \mathbf{C} = \mathbf{KM} \bmod 26$$

其中 **C** 和 **M** 是长度为 3 的列向量,分别代表密文和明文,**K** 是一个 3×3 的矩阵,代表加密密钥。运算按照模 26 执行。

一个 Hill_m 密码加密过程可以简单描述如下。

(1) 将明文字母以 m 个字母为单位进行分组,若最后一组没有 m 个字母,则补足没有任何实际意义的哑字母(双方可以事先约定这些字母),并用数字表示这些字母。

(2) 选择一个 m 阶可逆方阵 **K**,称为 Hill_m 密码的加密矩阵。

(3) 对每 m 个字母为一组的明文字母,用它对应的值构成一个 m 维向量。

(4) 计算密文的值 $C=km \bmod 26$,然后反查字母表的值,得到对应的 m 个密文字母。

(5) 同样的方式计算其他明文分组的密文。

例 3.7　用 Hill_3 对明文 pay more money 加密,加密密钥为:

$$\mathbf{k} = \begin{bmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{bmatrix}$$

将明文 3 个字母分为一组,pay mor emo ney,前面 3 个字母的值用向量表示为:

$$\mathbf{pay} = \begin{bmatrix} 15 \\ 0 \\ 24 \end{bmatrix}$$

该 3 个字母的加密过程为:

$$\mathbf{K}\begin{bmatrix}15\\0\\24\end{bmatrix}=\begin{bmatrix}375\\819\\486\end{bmatrix}\bmod 26=\begin{bmatrix}11\\13\\18\end{bmatrix}=\text{LSN}$$

以此类推,可得整个明文对应的密文是 LSN HDL EWN TRW。

解密时使用逆矩阵:

$$\mathbf{K}^{-1}\begin{bmatrix}4&9&15\\15&17&6\\24&0&17\end{bmatrix}$$

对于密文 LSN,即 $\begin{bmatrix}11\\13\\18\end{bmatrix}$,解密过程为:

$$\mathbf{K}^{-1}\begin{bmatrix}11\\13\\18\end{bmatrix}\bmod 26=\begin{bmatrix}15\\0\\24\end{bmatrix}=\text{pay}$$

Hill 密码能够很好地隐藏单字母出现的频率,可以抵抗统计频率分析,但面对已知明文攻击就很容易被破译。

3.4.4 置换密码

代换密码是将明文字母用不同的密文字母代替。置换密码(Permutation Cipher)则保持明文的所有字母不变,只是打乱明文字母的位置和次序,所以有时也称换位密码(Transposition Cipher),它的实现方法有很多。下面介绍一种列置换加密方法。

假如用密钥 network,加密明文 permutation cipher hide the message by rearranging the letter order。

将明文按照密钥的长度一行一行地写成一个矩阵,然后按照密钥字母对应的数值从小到大,按照列读出即为密文。

密钥:	**n**	**e**	**t**	**w**	**o**	**r**	**k**
明文:	p	e	r	m	u	t	a
	t	i	o	n	c	i	p
	h	e	r	h	i	d	e
	t	h	e	m	e	s	s
	a	g	e	b	y	r	e
	a	r	r	a	n	g	i
	n	g	t	h	e	l	e
	t	t	e	r	o	r	d
	e	r					

在密钥 network 中,字母对应的数字从小到大排列是 eknortw,按照这个顺序读出上面矩阵的列即是密文:

EIEHGRGTRAPESEIEDPTHTAANTEUCIEYNEOTIDSRGLRROREERTE MNHMBAHR

置换密码比较简单,经不起已知明文的攻击。但是置换密码与代换密码相结合,可以得到效果很好的密码。

3.5　数据加密标准

1949 年 Shannon 的论文《保密系统的通信理论》,标志着密码学作为一门独立的学科的形成。从此,信息论成为密码学的重要的理论基础之一。Shannon 建议采用扩散(Diffusion)、混淆(Confusion)和乘积迭代的方法设计密码。所谓扩散就是将每一位明文和密钥的影响扩散到尽可能多的密文数字中。这样使得密钥和明文以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无法利用。产生扩散的最简单的方法是置换。混淆用于掩盖明文和密文之间的关系。使得密钥的每一个位影响密文的许多位,以防止对密钥进行逐段破译,并且明文的每一个位也应影响密文的许多位,以便隐蔽明文的统计特性。用代换方法可以实现混淆。混淆就是使密文和密钥之间的关系复杂化。密文和密钥之间的关系越复杂,则密文和明文之间、密文和密钥之间的统计相关性就越小,从而使统计分析不能奏效。设计一个复杂的密码一般比较困难,而设计一个简单的密码相对比较容易,因此利用乘积迭代的方法对简单密码进行组合迭代,可以得到理想的扩散和混淆,从而得到安全的密码。近代各种成功的分组密码(如 DES、AES 等),都在一定程度上采用和体现了 Shannon 的这些设计思想。

为了适应社会对计算机数据安全保密越来越高的需求,美国国家标准局(NBS),即现在的美国国家标准和技术研究所(NIST)于 1973 年 5 月向社会公开征集标准加密算法,并公布了它的设计要求。

(1) 算法必须提供高度的安全性。

(2) 算法必须有详细的说明,并易于理解。

(3) 算法的安全性取决于密钥,不依赖于算法。

(4) 算法适用于所有用户。

(5) 算法适用于不同应用场合。

(6) 算法必须高效、经济。

(7) 算法必须能被证实有效。

1974 年 8 月 27 日,NBS 开始第二次征集,IBM 提交了算法 LUCIFER,该算法由 Feistel 领导的团队研究开发,采用 64 位分组以及 128 位密钥。IBM 用改版的 Lucifer 算法参加竞争,最后获胜,成为数据加密标准(Data Encryption Standard,DES)。1976 年 11 月 23 日,采纳为美国联邦标准,批准用于非军事场合的各种政府机构。1977 年 1 月 15 日,数据加密标准,即 FIPS PUB 46 被正式发布。DES 是分组密码的典型代表,也是第一个被公布出来的加密标准算法。现代大多数对称分组密码也是基于 Feistel 密码结构的。

3.5.1　DES 加密过程

DES 同时使用了代换和置换两种技巧。它用 56 位密钥加密 64 位明文,最后输出 64 位密文。整个过程由两大部分组成:一个是加密过程;另一个是子密钥产生过程。图 3.4 是 DES 加密算法简图。

可以将图 3.4 左半边的处理过程分以下 3 个部分。

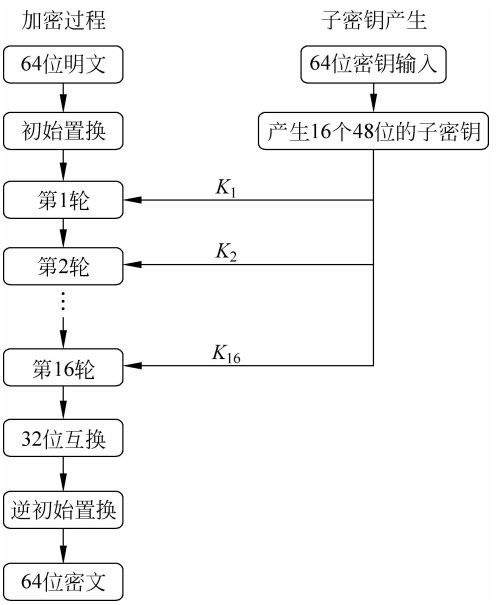

图 3.4 DES 加密算法简图

(1) 64 位明文经过初始置换被重新排列,然后分左右两半,每半各 32 位。

(2) 左右两半经过 16 轮置换和代换迭代,即 16 次实施相同的变换,然后再左右两半互换。

(3) 互换后的左右两半合并,再经过逆初始置换输出 64 位密文。

图 3.4 右半部则由 56 位密钥产生 16 个 48 位子密钥,分别供左半边的 16 轮迭代加密使用。

1. 初始置换

初始置换(Initial Permutation,IP)是数据加密的第 1 步,将 64 位的明文按照图 3.5 置换。置换表中的数字表示输入位在输出中的位置。

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

图 3.5 初始置换

置换后将数据 M 分成两部分:左半部分 L_0 和右半部分 R_0 各 32 位。划分方法原则是偶数位移到左半部,奇数位移到右半部,即:

$$\begin{aligned} L_0 = & M_{58} \quad M_{50} \quad M_{42} \quad M_{34} \quad M_{26} \quad M_{18} \quad M_{10} \quad M_2 \\ & M_{60} \quad M_{52} \quad M_{44} \quad M_{36} \quad M_{28} \quad M_{20} \quad M_{12} \quad M_4 \\ & M_{62} \quad M_{54} \quad M_{46} \quad M_{38} \quad M_{30} \quad M_{22} \quad M_{14} \quad M_6 \\ & M_{64} \quad M_{56} \quad M_{48} \quad M_{40} \quad M_{32} \quad M_{24} \quad M_{16} \quad M_8 \end{aligned}$$

$$R_0 = M_{57}\quad M_{49}\quad M_{41}\quad M_{33}\quad M_{25}\quad M_{17}\quad M_9\quad M_1$$
$$M_{59}\quad M_{51}\quad M_{43}\quad M_{35}\quad M_{27}\quad M_{19}\quad M_{11}\quad M_3$$
$$M_{61}\quad M_{53}\quad M_{45}\quad M_{37}\quad M_{29}\quad M_{21}\quad M_{13}\quad M_5$$
$$M_{63}\quad M_{55}\quad M_{47}\quad M_{39}\quad M_{31}\quad M_{23}\quad M_{15}\quad M_7$$

2. DES 每轮结构

DES 每轮的结构如图 3.6 所示。上一轮的右边 R_{i-1} 直接变换为下一轮的左边 L_i，上一轮的左边 L_{i-1} 与加密函数 F 异或后作为下一轮的右边 R_i。加密函数 F 则是上一轮右边 R_{i-1} 和子密钥 K_i 的函数。即：

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

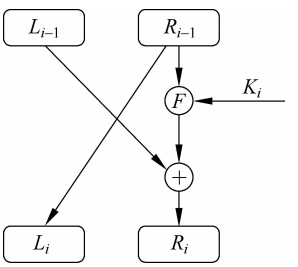

图 3.6　DES 每一轮结构

加密函数 F 本质上是 R_{i-1} 和子密钥 K_i 的异或，如图 3.7 所示。但由于它们的位数不一样，不能直接运算。从上式可以看出加密函数 F 是 32 位的，而 R_{i-1} 是 32 位的，子密钥 K_i 是 48 位的，因此 R_{i-1} 和 K_i 不能直接异或。DES 这样处理该问题：先用扩展置换 E（见图 3.8）将 R_{i-1} 扩展为 48 位，与 48 位子密钥异或，输出 48 位，再使用 8 个 S 盒压缩成 32 位，然后经置换函数 P（见图 3.9）输出 32 位的加密函数 F。

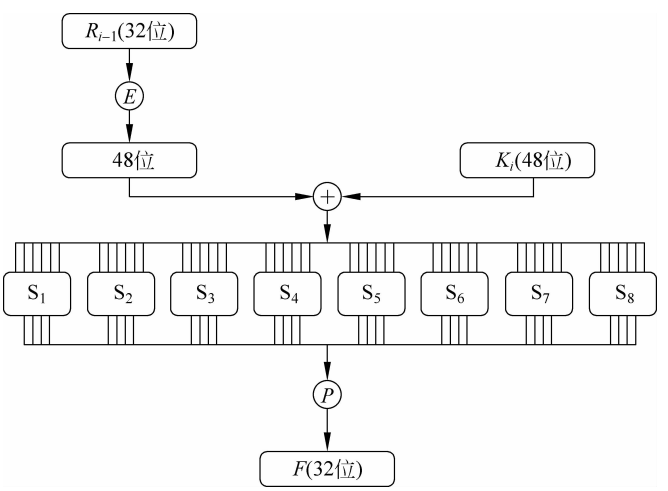

图 3.7　加密函数 *F* 的计算过程

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

图 3.8　扩展置换 *E*

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

图 3.9　置换函数 *P*

在加密函数计算过程中使用了8个S盒,S盒是DES保密性的关键所在,它是一种非线性变换,也是DES中唯一的非线性运算。S盒有6位输入,4位输出。48位数据经过8个S盒后输出32位数据。

每个S盒都由4行(表示为0、1、2、3)和16列(0、1、…、15)组成,如图3.10所示。每行都是全部的16个长为4比特串的一个全排列,每个比特串用它对应的二进制整数表示。如1001用9表示。48位的输入被分成8个6位的分组,每个分组进入一个S盒进行代换操作,然后映射为4位输出。对每个S盒,将6位输入的第一位和最后一位组成一个二进制数,用于选择S盒中的一行。用中间的4位选择S盒16列中的某一列,行列交叉处的十进制数转换为二进制数可得到4位输出。

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

图3.10 DES的S盒

例如对于S_1盒而言,如果输入为011001,则行是01(十进制1,即S盒的第2行),列1100(12,即S盒的第13列),该处的值是9,转换为二进制数为1001,即为该S盒的输出。

3.5.2　DES 子密钥产生

DES 加密过程共迭代 16 轮，每轮用一个不同的 48 位子密钥。这些子密钥由算法的 56 位密钥产生。DES 算法的输入密钥长度是 64 位，但只用了其中的 56 位，如图 3.11 所示。图中无阴影部分（也就是每行的第 8 位）将被忽略，主要用于奇偶校验，也可随意设置。

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

图 3.11　DES 的输入密码

子密钥的产生过程如图 3.12 所示。

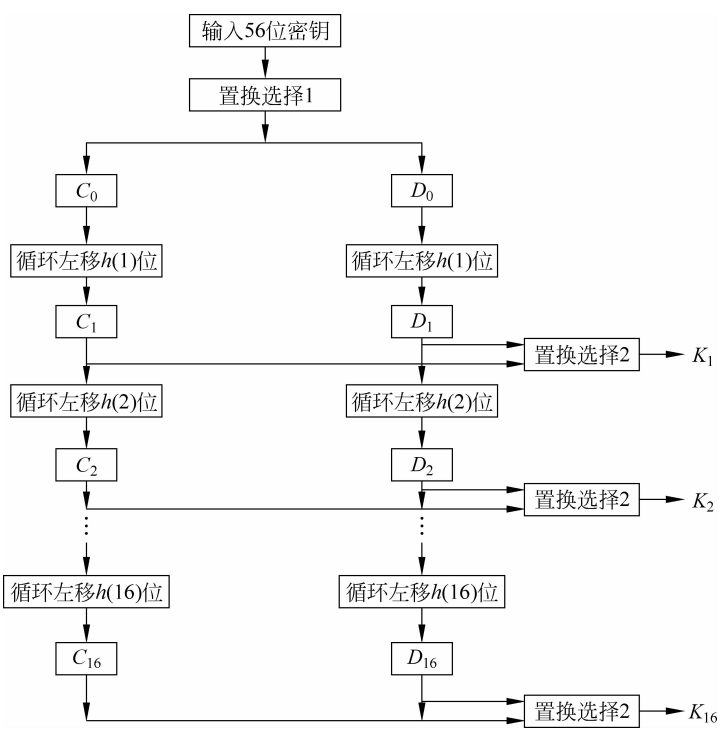

图 3.12　子密钥产生过程

56 位密钥首先经过置换选择 1（见图 3.13）将其位置打乱重排，并将前 28 位作为 C_0（见图 3.13 中的上面部分），后 28 位 D_0（见图 3.13 中的下面部分）。

接下来经过 16 轮，产生 16 个子密钥。每一轮迭代中，C_{i-1} 和 D_{i-1} 循环左移一位或者两位，如图 3.14 所示。C_{i-1} 和 D_{i-1} 循环左移后变为 C_i 和 D_i，将 C_i 和 D_i 合在一起的 56 位，经过

置换选择 2(见图 3.15),从中挑出 48 位作为这一轮的子密钥,这个子密钥作为前面介绍的加密函数的一个输入。再将 C_i 和 D_i 循环左移后,使用置换选择 2 产生下一轮的子密钥,如此继续,产生所有 16 个子密钥。

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

图 3.13 置换选择 1

迭代轮数	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
移位次数	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

图 3.14 每轮左移次数的规定

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 3.15 置换选择 2

例 3.8 用 DES 加密,明文 $M=(0123456789ABCDEF)_{16}=(00000001\ 00100011\ 01000101\ 01100111\ 10001001\ 10101011\ 11001101\ 11101111)_2$,密钥 $K=(133457799BBCDFF1)_{16}=(00010011\ 00110100\ 01010111\ 01111001\ 10011011\ 10111100\ 11011111\ 11110001)_2$。

加密过程如下所示。

(1) 初始置换。

将明文 M 经过初始置换后分为左右两半。

$L_0=11001100\ 00000000\ 11001100\ 11111111$

$R_0=11110000\ 10101010\ 11110000\ 10101010$

(2) 第 1 轮迭代运算。

① 先确定子密钥 K_1,将密钥 K 经置换选择 1 得:

$C_0=11110000\ 11001100\ 10101010\ 1111$

$D_0=01010101\ 01100110\ 01111000\ 1111$

左移 1 位后经过置换选择 2 输出 48 位 K_1。

$K_1=00011011\ 00000010\ 11101111\ 11111100\ 01110000\ 01110010$

② 计算加密函数 F。

用扩展置换 E 将 R_0 扩展为 48 位,再和 K_1 异或。

$E(R_0)\oplus K_1=01100001\ 00010111\ 10111010\ 10000110\ 01100101\ 00100111$

经 8 个 S 盒输出 32 位。

$S_1(011000)=0101, S_2(010001)=1100, S_3(011110)=1000, S_4(111010)=0010$

$S_5(100001)=1011, S_6(100110)=0101, S_7(010100)=1001, S_8(100111)=0111$

经置换函数 P 输出加密函数 F 如下。

$F=$00100011 01001010 10101001 10111011

③ 由 L_0 和 R_0 计算出 L_1 和 R_1。

$L_1=R_0=$11110000 10101010 11110000 10101010

$R_1=L_0 \oplus F(R_0, K_1)=$11101111 01001010 01100101 01000100

因此经过第 1 轮，得到：

$[R_1, L_1]=(\text{EF4A6544F0AAF0AA})_{16}$

进行类似的运算，经过 16 轮后得到的结果是：

$[R_{16}, L_{16}]=(\text{0A4CD99543423234})_{16}$

(3) 逆初始置换。

将第 16 轮输出合并为一个 64 位比特串，经过逆初始置换后得到 64 位密文：

10000101 11101000 00010011 01010100

00001111 00001010 10110100 00000101

3.5.3　DES 解密

DES 解密过程与加密过程在本质上是一致的，加密和解密使用同一个算法，使用相同的步骤和相同的密钥。主要不同点是将密文作为算法的输入，但是逆序使用子密钥 k_i，即第 1 轮使用子密钥 k_{16}，第 2 轮使用子密钥 k_{15}，最后一轮使用子密钥 k_1。

3.5.4　DES 的强度

从发布时起，DES 就备受争议，很多研究者怀疑它所提供的安全性。争论的焦点主要集中在密钥的长度、迭代次数以及 S 盒的设计等方面。

DES 的安全性依赖于 S 盒。由于 DES 里的所有计算，除去 S 盒，全是线性的。可见 S 盒对密码体制的安全性是非常重要的。但是自从 DES 公布以来，S 盒设计详细标准至今没有公开。因此就有人怀疑 S 盒里隐藏了陷门(Trapdoors)。然而到目前为止也没有任何证据证明 DES 里存在陷门。事实上，后来表明 S 盒是被设计成能够防止差分密码分析的。

DES 将 Lucifer 算法作为标准，Lucifer 算法的密钥长度为 128 位，但 DES 将密钥长度改为 56 位。56 位密钥共有 $2^{56}=7.2\times10^{16}$ 个可能值，这不能抵抗穷尽密钥搜索攻击。例如在 1997 年，美国科罗拉多州的程序员 Verser 在 Internet 上数万名志愿者的协作下用 96 天的时间找到了密钥长度为 40 位和 48 位的 DES 密钥。1998 年电子边境基金会(EFF)使用一台价值 25 万美元的计算机在 56 小时之内破译了 56 位的 DES。1999 年，电子边境基金会(EFF)通过 Internet 上的十万台计算机合作，仅用 22 小时 15 分钟就破译了 56 位的 DES。因此需要寻找一个算法替代 DES。

另外，DES 存在弱密钥。如果一个密钥所产生的所有子密钥都是一样的，则这个外部密钥就称为弱密钥。DES 算法的子密钥是通过对一个 64 位的外部密钥进行置换得到的。外部密钥输入到 DES 后，经密钥置换后分成两半，每一半各自独立移位。如果每一半的所

有位都是0或者1,那么在算法的任意一轮所有的子密钥都是相同的。当主密钥是全0全1,或者一半是全0、一半是全1时,就会发生这种情况。因此,DES存在弱密钥。

3.5.5 三重DES

DES由于安全问题,美国政府于1998年12月宣布DES不再作为联邦加密标准。新的美国联邦加密标准是高级加密标准(ASE)。在新的加密标准实施之前,为了不浪费已有的DES算法投资,NIST在1999年发布了一个新版本的DES标准(FIPS PUB46-3),该标准指出DES仅能用于遗留的系统,同时将三重DES(3DES)取代DES成为新的标准。3DES明显存在几个优点。首先它的密钥长度是168位,足以抵抗穷举攻击。其次,3DES的底层加密算法与DES的加密算法相同,该加密算法比任何其他加密算法受到分析的时间要长得多,也没有发现有比穷举攻击更有效的密码分析攻击方法。

最简单的多重DES加密是用DES加密两次,每次用不同的密钥,这就是双重DES。但双重DES不安全,没有被人们使用。图3.16是双重DES的加密和解密。对于一个明文M和两个加密密钥K_1和K_2,加密过程为$C=E_{K_2}[E_{K_1}[M]]$,解密过程为$M=D_{K_1}[D_{K_2}[C]]$。密钥总长度为112位,似乎密码强度增加了一倍,但由于双重DES存在中间相遇攻击,使它的强度跟一个56位DES强度差不多。

从图3.16中可以观察到$C=E_{K_2}[E_{K_1}[M]]$,则$X=E_{K_1}[M]=D_{K_2}[C]$。若已知(M,C),攻击方法如下:先用2^{56}个可能的K_1加密M,得到2^{56}个可能的值,将这些值从小到大存入一个表中;再对2^{56}个可能的K_2解密C,每次做完解密,将所得的值与表中的值比较,如果产生匹配,则它们对应的密钥可能是K_1和K_2。用一个新的明文密文对检测所得两个密钥,如果两密钥产生正确的密文,则它们是正确的密钥。

为防止中间相遇攻击,可以采用3次加密方式,如图3.17所示。这是使用两个密钥的三重DES,采用加密-解密-加密(E-D-E)方案。加密为$C=E_{K_1}[D_{K_2}[E_{K_1}[M]]]$,解密为$M=D_{K_1}[E_{K_2}[D_{K_1}[C]]]$。要注意的是,加密与解密在安全性上来说是等价的。这种加密方案的攻击代价是2^{112}。

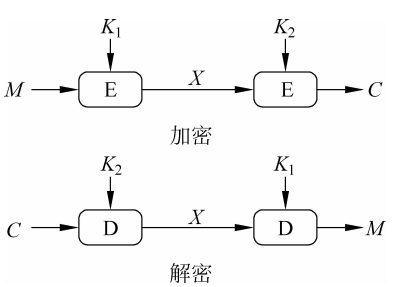

图3.16 双重DES

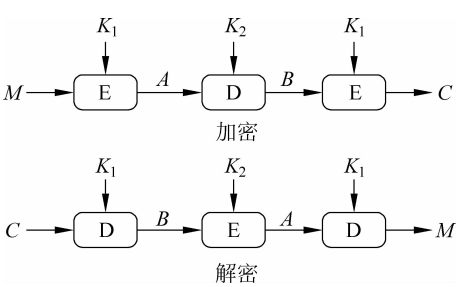

图3.17 三重DES

目前还没有针对两个密钥的三重DES的实际攻击方法,但是感觉它不太可靠,如果采用三个密钥的三重DES则比较让人放心。三个密钥的三重DES的密钥长度是168位,采用加密-解密-加密(E-D-E)方案。其加密过程为$C=E_{K_3}[D_{K_2}[E_{K_1}[M]]]$,解密过程为$M=D_{K_1}[E_{K_2}[D_{K_3}[C]]]$。目前这种加密方式已经被一些网络应用采用,如本书后面章节要讨论的PGP和S/MIME就采用了这种方案。

3.6　高级加密标准

由于 DES 存在安全问题，而三重 DES 算法运行速度比较慢，三重 DES 迭代的轮数是 DES 的 3 倍，因此速度比 DES 慢很多。三重 DES 的分组长度为 64 位，就效率和安全性而言，分组长度应该更长。这就注定三重 DES 不能成为长期使用的加密标准。为此，美国国家标准技术研究所（NIST）在 1997 年公开征集新的高级加密标准（Advanced Encryption Standards，AES），要求 AES 比 3DES 快而且至少和 3DES 一样安全，并特别提出高级加密标准的分组长度为 128 位的对称分组密码，密钥长度支持 128 位、192 位和 256 位。

1997 年 9 月给出的选择高级加密标准的评估准则如下。

（1）安全性：由于 AES 最短的密钥长度是 128 位，所以使用目前的技术，穷举攻击是没有任何可能的。因此 AES 应重点考虑是否能抵抗各种密码分析方法的攻击。

（2）代价：指计算效率方面。NIST 期望 AES 能够广泛应用于各种实际应用，因此要求 AES 必须具有很高的计算效率。

（3）算法和执行特征：指算法的灵活性、简洁性以及硬件与软件平台的适应性等方面。1998 年 6 月 NIST 共收到 21 个提交的算法，在同年的 8 月首先选出 15 个候选算法。1999 年 NIST 从 15 个 AES 候选算法中遴选出 5 个候选算法，它们是：MARS（由 IBM 公司研究部门的一个庞大团队发布，对它的评价是算法复杂、速度快、安全性高）、RC6（由 RSA 实验室发布，对它的评价是极简单、速度极快、安全性低）、Rijndael（由 Joan Daemen 和 Vincent Rijmen 两位比利时密码专家发布，对它的评价是算法简洁、速度快、安全性好）、Serpent（由 Ross Anderson、Eli Biham 和 Lars Knudsen 发布，对它的评价是算法简洁、速度慢、安全性极高）和 Twofish（由 Counterpane 公司的一个庞大的团队发布，对它的评价是算法复杂、速度极快、安全性高）。从全方位考虑，Rijndael（读成 Rain Doll）汇聚了安全、性能、效率、易用和灵活等优点，使它成为了 AES 最合适的选择。在 2000 年 10 月 Rijndael 算法被选为高级加密标准，并于 2001 年 11 月发布为联邦信息处理标准（Federal Information Processing Standard，FIPS），用于美国政府组织保护敏感信息的一种特殊的加密算法，即 FIPS PUB 197 标准。

3.6.1　AES 的基本运算

第 2 章已经介绍了 AES 的主要数学基础，为了更好地理解 AES 的加密过程，这部分将介绍与 AES 相关的一些规定以及运算方法。AES 算法中有些是以字节为单位进行运算的，也有的是以 4 个字节（即一个字）为单位的。它将一个字节看做是在有限域 $GF(2^8)$ 上的一个元素，将一个字看成是系数取自 $GF(2^8)$，并且次数小于 4 的多项式。

1. AES 中的字节运算

在 AES 中，一个字节是用有限域 $GF(2^8)$ 上的元素表示。有限域上的元素有多种表示方法，AES 主要采用多项式表示。

有限域 $GF(2^8)$ 上的加法定义为二进制多项式的加法，其系数是模 2 相加。此处加法是

异或运算(记为⊕),1⊕1=0,1⊕0=1,0⊕0=0。因此,多项式减法与多项式加法的规则相同。对于两个字节$\{a_7a_6a_5a_4a_3a_2a_1a_0\}$和$\{b_7b_6b_5b_4b_3b_2b_1b_0\}$,其和为$\{c_7c_6c_5c_4c_3c_2c_1c_0\}$,$c_i=a_i\oplus b_i$(即$c_7=a_7\oplus b_7$,$c_6=a_6\oplus b_6$,…,$c_0=a_0\oplus b_0$)。

例如,下述表达式彼此相等。

$(x^6+x^4+x^2+x+1)+(x^7+x+1)=x^7+x^6+x^4+x^2$ (多项式记法)

$\{01010111\}\oplus\{10000011\}=\{11010100\}$ (二进制记法)

$\{57\}\oplus\{83\}=\{d4\}$ (十六进制记法)

在有限域GF(2^8)上的乘法(记为·)定义为多项式的乘积模一个次数为8的不可约多项式:

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

用十六进制表示该多项式为{01}{1B}。

例如,{57}·{83}={c1},因为:

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

而$(x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1)\bmod(x^8+x^4+x^3+x+1)=x^7+x^6+1$

模$m(x)$确保了所得结果是次数小于8的二进制多项式,因此可以用一个字节表示。

如果$a(x)\cdot b(x)\bmod m(x)=1$,则称$b(x)$为$a(x)$的逆元。

在AES中的倍乘函数$x$time()是用多项式x乘一个二进制多项式后再模$m(x)$。

用多项式x乘以$b(x)$将得到:

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

将上述结果模$m(x)$即可得到$x\cdot b(x)$的结果。如果$b_7=0$,则该结果已经是模运算后的形式。如果$b_7=1$,则模运算需要异或多项式$m(x)$完成。由此,乘x(即{00000010}或十六进制{02})可通过字节内左移一位,紧接着的一个与{1B}按位异或来实现。将该操作记为$b=x$time(a)。通过将中间结果相加,可以用$x$time()实现任意常数的乘法。

例如,{57}·{13}={fe} 因为:

$$\begin{aligned}\{57\}\cdot\{02\}&=x\text{time}(\{57\})=\{ae\}\\\{57\}\cdot\{04\}&=x\text{time}(\{ae\})=\{47\}\\\{57\}\cdot\{08\}&=x\text{time}(\{47\})=\{8e\}\\\{57\}\cdot\{10\}&=x\text{time}(\{8e\})=\{07\}\end{aligned}$$

因此:

$$\begin{aligned}\{57\}\cdot\{13\}&=\{57\}\cdot(\{01\}\oplus\{02\}\oplus\{10\})\\&=\{57\}\oplus\{ae\}\oplus\{07\}\\&=\{fe\}\end{aligned}$$

2. AES中的字运算

AES中的32位字表示为系数在有限域GF(2^8)上的次数小于4的多项式。考虑含有4个项且系数为有限域元素的多项式,即:

$$a(x)=a_3x^3+a_2x^2+a_1x+a_0$$

它可以表示为如下形式[a_0,a_1,a_2,a_3]的字(Word)。注意这里的多项式与前面有限域元素定义中使用的多项式操作不同,即使这两类多项式均使用相同的变量 x。这里的系数本身就是有限域元素,即字节(Bytes)而不是位(Bits)。另外,该 4 项多项式的乘法使用了一个不同于前面的模多项式,将在下面定义。

为说明加法和乘法运算,令

$$b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$$

为另一个 4 项多项式。加法是对 x 相应次数项的有限域系数进行相加运算。该加法对应于相应字节间的异或运算。

因此:

$$a(x) + b(x) = (a_3 \oplus b_3)x^3 + (a_2 \oplus b_2)x^2 + (a_1 \oplus b_1)x + (a_0 \oplus b_0)$$

乘法要用两步完成。第 1 步,对多项式相乘的结果 $c(x)=a(x) \cdot b(x)$进行代数扩展:

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

其中:

$$c_0 = a_0 \cdot b_0$$
$$c_1 = a_1 \cdot b_0 \oplus a_0 \cdot b_1$$
$$c_2 = a_2 \cdot b_0 \oplus a_1 \cdot b_1 \oplus a_0 \cdot b_2$$
$$c_3 = a_3 \cdot b_0 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \oplus a_0 \cdot b_3$$
$$c_4 = a_3 \cdot b_1 \oplus a_2 \cdot b_2 \oplus a_1 \cdot b_3$$
$$c_5 = a_3 \cdot b_2 \oplus a_2 \cdot b_3$$
$$c_6 = a_3 \cdot b_3$$

所得结果 $c(x)$并没有表示为一个 4 字节的字。因此,乘法的第 2 步是模一个 4 次多项式来化简 $c(x)$,使得结果化简为一个次数小于 4 的多项式。在 AES 算法中,这一模多项式取为 x^4+1,由于:

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

则 $a(x)$和 $b(x)$取模的乘积记为 $a(x) \otimes b(x)$,表示为下述的 4 项多项式 $d(x)$,即:

$$d(x) = d_3x^3 + d_2x^2 + d_1x + d_0$$

其中:

$$d_0 = a_0 \cdot b_0 \oplus a_3 \cdot b_1 \oplus a_2 \cdot b_2 \oplus a_1 \cdot b_3$$
$$d_1 = a_1 \cdot b_0 \oplus a_0 \cdot b_1 \oplus a_3 \cdot b_2 \oplus a_2 \cdot b_3$$
$$d_2 = a_2 \cdot b_0 \oplus a_1 \cdot b_1 \oplus a_0 \cdot b_2 \oplus a_3 \cdot b_3$$
$$d_3 = a_3 \cdot b_0 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \oplus a_0 \cdot b_3$$

当 $a(x)$是一个固定多项式时,等式中定义的运算可以写成矩阵形式,如:

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

由于 x^4+1 不是 GF(2^8)上的不可约多项式,因此被一个固定的 4 次多项式相乘的乘法不一定可逆。然而,在 AES 算法中选择了一个固定的有逆元的 4 项多项式的乘法,它按照乘法运算有逆元。

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$
$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$$

在AES算法的密钥扩展部分中还要用到的另一个多项式(RotWord()函数)为 $a_0=a_1=a_2=\{00\}$,$a_3=\{01\}$,即多项式 x^3。该多项式的效果是将输入字中的字节循环移位来得到输出字,即$[b_0,b_1,b_2,b_3]$将变换为$[b_1,b_2,b_3,b_0]$。

3.6.2 AES加密

在Rijndael算法中,分组长度和密钥长度可以分别是128位、192位和256位。而在AES中,分组长度只能是128位。AES算法中基本的运算单位是字节,即视为一个整体的8比特序列。如果分组长度和密钥长度为128位,假如字节数组将表示为如下形式。

$$a_0,a_1,a_2,\cdots,a_{15}$$

其字节排列方式将如图3.18所示。

如果密钥长度(或在Rijndael中的明文分组)为192位、256位,组成的两个矩阵如图3.19和图3.20所示。它们的特点是行数都是4,列数不同。

a_0	a_4	a_8	a_{12}	a_{16}	a_{20}
a_1	a_5	a_9	a_{13}	a_{17}	a_{21}
a_2	a_6	a_{10}	a_{14}	a_{18}	a_{22}
a_3	a_7	a_{11}	a_{15}	a_{19}	a_{23}

图3.19 192位(24个字节)的矩阵排列

a_0	a_4	a_8	a_{12}	a_{16}	a_{20}	a_{24}	a_{28}
a_1	a_5	a_9	a_{13}	a_{17}	a_{21}	a_{25}	a_{29}
a_2	a_6	a_{10}	a_{14}	a_{18}	a_{22}	a_{26}	a_{30}
a_3	a_7	a_{11}	a_{15}	a_{19}	a_{23}	a_{27}	a_{31}

图3.20 256位(32个字节)的矩阵排列

这些矩阵有4行,分组的列数记为 Nb,Nb=分组长度(位)÷32(位)。显然 Nb 可以取的值为4、6和8,分别对应的分组长度为128位、192位和256位。类似地密钥的列数记为 Nk,Nk=密钥长度(位)÷32(位)。Nk 可以取的值为4、6和8,对应的密钥长度分别为128位、192位和256位。

密码运算的中间结果都是以上面的形式表示,称之为状态(State)数组。AES将这些中间结果复制到状态(State)数组中。算法的运行过程是将需要加密的分组从一个状态转换为另一个状态,最后该数组被复制到输出矩阵中。如对于128位分组,假设加密和解密的初始阶段将输入字节数组 $in_0,in_1,\cdots,in_{15}$ 复制到如图3.21所示的状态(State)矩阵中。加密或解密的运算都在该状态矩阵上进行,最后的结果将被复制到输出字节数组 out_0,$out_1,\cdots,out_{15}$。

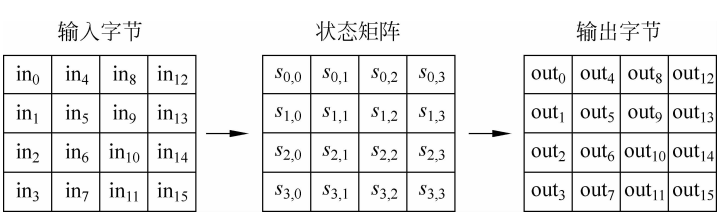

图3.21 状态矩阵、输入和输出

状态矩阵中每一列的4个字节可以看做一个32比特字,用行号 r 作为每一个字中4个字节的索引。因此状态可以看做32比特字(列),$w_0\cdots w_3$ 的一维数组,用列号 c 表示该数组的索引。在图3.21中,该状态可以看做4个字组成的数组,如下所示:

$$w_0 = s_{0,0}s_{1,0}s_{2,0}s_{3,0} \quad w_2 = s_{0,2}s_{1,2}s_{2,2}s_{3,2}$$

$$w_1 = s_{0,1}s_{1,1}s_{2,1}s_{3,1} \quad w_3 = s_{0,3}s_{1,3}s_{2,3}s_{3,3}$$

在加密和解密的初始阶段，输入数组 in 按照下述规则复制到状态矩阵中。

$$s[r,c]=\text{in}[r+4c] \quad 0\leqslant r<4 \text{ 且 } 0\leqslant c<Nb$$

在加密和解密的结束阶段，状态矩阵将按照下述规则被复制到输出数组 out 中。

$$\text{out}[r+4c]=s[r,c] \quad 0\leqslant r<4 \text{ 且 } 0\leqslant c<Nb$$

AES 密码是一种迭代式密码结构，但不是 Feistel 密码结构。Rijndael 算法迭代的轮数与分组长度和密钥长度相关。对于 AES 算法，算法的轮数依赖于密钥长度。将轮数表示为 Nr，当 $Nk=4$ 时 $Nr=10$；当 $Nk=6$ 时 $Nr=12$；当 $Nk=8$ 时 $Nr=14$。表 3.5 是 Rijndael 算法不同分组长度和密钥长度对应的迭代轮数列。

表 3.5　Rijndael 算法迭代轮数

	分组长度为 128 位	分组长度为 192 位	分组长度为 256 位
密钥长度为 128 位	10 轮	12 轮	14 轮
密钥长度为 192 位	12 轮	12 轮	14 轮
密钥长度为 256 位	14 轮	14 轮	14 轮

当分组长度和密钥长度均为 128 位时，AES 共迭代 10 轮，需要 11 个子密钥。其加密过程如图 3.22 所示。前面 9 轮完全相同，每轮包括 4 阶段，分别是字节代换(Byte Substitution)、行移位(Shift Rows)、列混淆(Mix Columns)和轮密钥加(Add Round Key)，最后一轮只有三个阶段，缺少列混淆。

在加密时，将输入复制到状态矩阵中。经过初始轮子密钥加后，通过执行 10 轮来变换状态矩阵，最后状态将被复制到输出。图 3.23 是 AES 加密算法的伪代码表示，每一个变换如 SubBytes()、ShiftRows()、MixColumns()和 AddRoundKey()都作用在状态(State)上，数组 $w[]$中包含了密钥编排得到的密钥，这些将后面部分说明。除了最后一轮，所有的轮变换均相同。最后一轮不包括 MixColumns()变换。

下面是 AES 中出现的一些参数、符号和函数。

AddRoundKey()是加密和解密中使用的变换，它将一个轮密钥异或到状态上。轮密钥的长度等于状态的大小(即对于 $Nb=4$，轮密钥长度等于 128 比特/16 字节)。

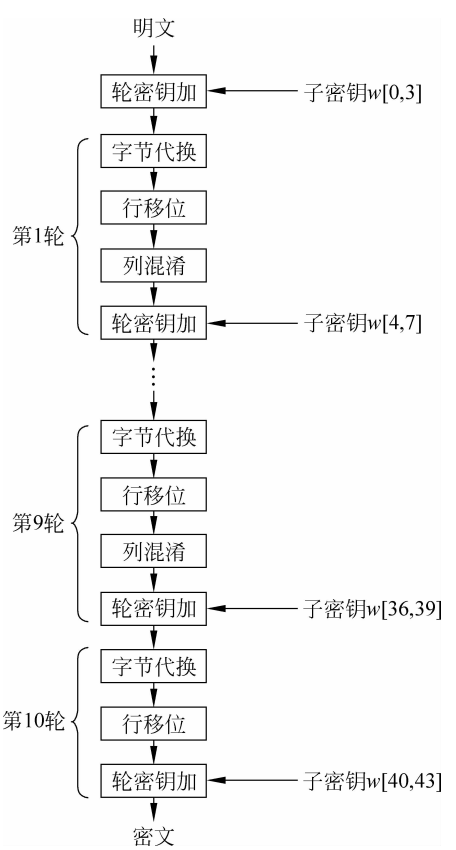

图 3.22　AES 加密过程

InvMixColumns()　解密中使用的变换，是 MixColumns()的逆变换。

InvShiftRows()	解密中使用的变换,是 ShiftRows()的逆变换。
InvSubBytes()	解密中使用的变换,是 SubBytes()的逆变换。
K	密码所使用的秘密密钥。
MixColumns()	加密中使用的变换,以状态的每一列作为输入,混合每一列的数据(彼此独立的)得到新的列。
Nb	状态包含的列(32 比特字)的个数。对于 AES,$Nb=4$。
Nk	密钥包含的 32 比特字的个数。对于该标准,$Nk=4,6$ 或 8。
Nr	轮数,是 Nk 和 Nb(固定的)的函数。对于该标准,$Nr=10,12$ 或 14(参见第 6.3 节)。
Rcon[]	密钥扩展算法中用到的轮常数。
RotWord()	密钥扩展算法中使用的函数,对 4 字节字进行循环移位。
ShiftRows()	加密中使用的变换,将状态的最后 3 行循环移动不同的位移量。
SubBytes()	加密中使用的变换,利用一个非线性字节替代表(S 盒),独立地对状态的每个字节进行操作。
SubWord()	密钥扩展算法中使用的函数,它以 4 字节字作为输入,对于 4 字节中的每一字节分别应用 S 盒,得到一个输出字。
XOR	异或运算。
$\oplus$	异或运算。
$\otimes$	两个多项式(每一个的度(Degree)均小于 4)相乘再模 x^4+1。
•	有限域上的乘法。

```
Cipher(byte in[4 * Nb],byte out[4 * Nb],word w[Nb * (Nr+1)])
begin
   byte state[4,Nb]
   state=in
   AddRoundKey(state,w[0,Nb-1])
   for round=1 step 1 to Nr-1
      SubBytes(state)
      ShiftRows(state)
      MixColumns(state)
      AddRoundKey(state,w[round * Nb,(round+1) * Nb-1])
   end for
   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state,w[Nr * Nb,(Nr+1) * Nb-1])
   out=state
end
```

图 3.23 AES 加密算法伪代码

3.6.3 字节代换

字节代换(SubBytes())是非线性的,它独立地将状态中的每个字节利用代换表(S 盒)进行运算。S 盒被设计成能够抵挡所有已知的攻击。该 S 盒(见图 3.24)是由 16×16 个字节组成的矩阵,包含了 8 位值所能表达的 256 种可能的变换。State 中的每个字节按照如下

的方式映射为一个新的字节：将该字节的高 4 位作为行值，低 4 位作为列值，然后取出 S 盒中对应行列交叉处的元素作为输出。例如，十六进制{95}对应的 S 盒的行值是 9，列值是 5，S 盒中此处的值是{2A}。因此{95}被映射为{2A}。

		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

图 3.24　S 盒

S 盒按照如下的方式构造。

(1) 逐行按照上升排列的方式初始化 S 盒。第一行是{00}，{01}，…，{0F}；第二行是{10}，{11}，…，{1F}等。因此在 x 行 y 列的字节值是{xy}。

(2) 把 S 盒中的每个字节映射为它在有限域 GF(2^8)上的乘法逆；其中，元素{00}映射到它自身{00}。

(3) 把 S 盒中的每个字节表示为(b_7，b_6，b_5，b_4，b_3，b_2，b_1，b_0)8 位，对 S 盒中的每个字节中的每个位做如下变换：

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

其中，对于 $0 \leqslant i < 8$，b_i 是字节的第 i 位，c_i 是值为{63}或{01100011}的字节 c 的第 i 位。在此处和其他地方，在变量的右上角作标记(如 b')表示该变量将用右侧的值更新。AES 按照如下的方式用矩阵描述 S 盒的变换。

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

还是用{95}作为输入,可以计算出{95}在GF(2^8)上的逆为{8A},用二进制表示为10001010,代入上述变换有:

$$\begin{bmatrix}1&0&0&0&1&1&1&1\\1&1&0&0&0&1&1&1\\1&1&1&0&0&0&1&1\\1&1&1&1&0&0&0&1\\1&1&1&1&1&0&0&0\\0&1&1&1&1&1&0&0\\0&0&1&1&1&1&1&0\\0&0&0&1&1&1&1&1\end{bmatrix}\begin{bmatrix}0\\1\\0\\1\\0\\0\\0\\1\end{bmatrix}\oplus\begin{bmatrix}1\\1\\0\\0\\0\\1\\1\\0\end{bmatrix}=\begin{bmatrix}1\\0\\0\\1\\0\\0\\1\\0\end{bmatrix}\oplus\begin{bmatrix}1\\1\\0\\0\\0\\1\\1\\0\end{bmatrix}=\begin{bmatrix}0\\1\\0\\1\\0\\1\\0\\0\end{bmatrix}$$

结果为00101010,用十六进制表示为{2A},与前面查表所得结果一样。

3.6.4 行移位

行移位(ShiftRows())是一个简单的置换,在行移位变换中,对State的各行进行循环左移位,State的第一行保持不变,第二行循环左移一个字节,第三行循环左移两个字节,第四行循环左移三个字节,如图3.25所示。

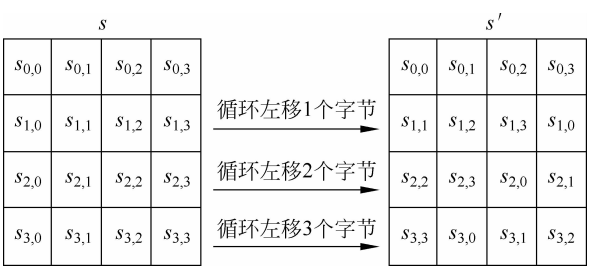

图3.25 对State的各行移位

对Rijndael而言,分组长度是128位、196位或256位。移位的字节数与分组的大小有关系。后三行的左移量C_1、C_2、C_3与分组长度Nb(矩阵的列数)的关系如表3.6所示。

表3.6 移位值

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

3.6.5 列混淆

列混淆(MixColumns())变换在State上按照每一列(即对一个字)进行运算,并将每一列看作4次多项式,即将State的列看作GF(2^8)上的多项式且被一个固定的多项式$a(x)$模x^4+1乘,$a(x)$为:

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

这可以写成矩阵乘法。令$s'(x)=a(x)\otimes s(x)$:

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

经过该乘法计算后，一列中的 4 个字节将由下述结果取代：

$$s'_{0,c} = (\{02\} \cdot s_{0,c}) \oplus (\{03\} \cdot s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$
$$s'_{1,c} = s_{0,c} \oplus (\{02\} \cdot s_{1,c}) \oplus (\{03\} \cdot s_{2,c}) \oplus s_{3,c}$$
$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \cdot s_{2,c}) \oplus (\{03\} \cdot s_{3,c})$$
$$s'_{3,c} = (\{03\} \cdot s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \cdot s_{3,c})$$

图 3.26 为列混淆变换示意图。

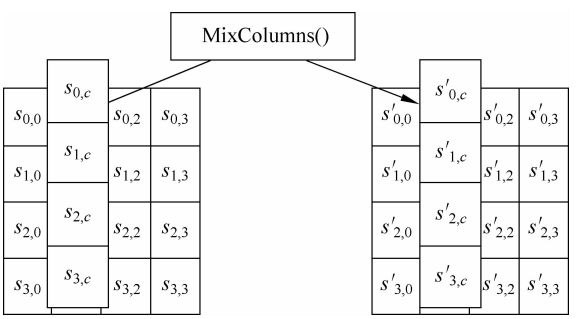

图 3.26　列混淆变换示意图

例 3.9　假设 State 矩阵的第一列分别是 $s_{0,0} = \{87\}$，$s_{1,0} = \{6E\}$，$s_{2,0} = \{46\}$，$s_{3,0} = \{A6\}$。经过列混淆变换后，$s_{0,0} = \{87\}$ 映射为 $s'_{0,0} = \{47\}$，试计算验证这一结果。

第 1 列第 1 个字节的代换方程为：

$$(\{02\} \cdot \{87\}) \oplus (\{03\} \cdot \{6E\}) \oplus \{46\} \oplus \{A6\} = \{47\}$$

下面验证上面等式成立。用多项式表示为：

$$\{02\} = x$$
$$\{87\} = x^7 + x^2 + x + 1$$

那么：

$$x \cdot (x^7 + x^2 + x + 1) = x^8 + x^3 + x^2 + x$$

再模一个次数为 8 的不可约多项式：

$$m(x) = x^8 + x^4 + x^3 + x + 1$$
$$(x^8 + x^3 + x^2 + x) \bmod (x^8 + x^4 + x^3 + x + 1) = x^4 + x^2 + 1$$

写成二进制形式为 00010101。

同样可以计算出 $\{03\} \cdot \{6E\} = 10110010$，$\{46\} = 01000110$，$\{A6\} = 10100110$。因此 $(\{02\} \cdot \{87\}) \oplus (\{03\} \cdot \{6E\}) \oplus \{46\} \oplus \{A6\}$ 计算结果为：

$$\begin{array}{rl} & 0001\ 0101 \\ & 1011\ 0010 \\ & 0100\ 0110 \\ \oplus & 1010\ 0110 \\ \hline & 0100\ 0111 \quad = \{47\} \end{array}$$

3.6.6 轮密钥加

在轮密钥加(AddRoundKey())变换中,128位的State按位与128位子密钥异或。可以将这种操作看成是State一列中的4个字节与轮密钥的一个字进行异或;也可以看成是两者之间的字节异或。

3.6.7 AES的密钥扩展

如果分组长度和密钥长度都是128位,AES的加密算法共迭代10轮,需要11个子密钥。AES的密钥扩展的目的是将输入的128位密钥扩展成11个128位的子密钥。AES的密钥扩展算法是以字为一个基本单位,而不是以位为操作对象。一个字等于4个字节,共32位,刚好是密钥矩阵的一列。因此4个字(128位)密钥需要扩展成11个子密钥,共44个字。若 Nr 表示加密算法轮数,那么密钥扩展总共生成 $Nb(Nr+1)$ 个字,并需要一个 Nb 个字组成的初始集合(即原始密钥),为每一轮产生 Nb 个字的子密钥。扩展后的密钥编排结果由一个4字节字的线性数组组成,记为 $[w_i]$,其中 $0 \leqslant i < Nb(Nr+1)$。图3.27是AES的密钥扩展算法的伪码表示。

```
KeyExpansion(byte key[4 * Nk],work w[Nb * (Nr+1)],Nk)
begin
    word temp
    i=0
    while(i<Nk)
        w[i]=word(key[4 * i],key[4 * i+1],key[4 * i+2],key[4 * i+3])
        i=i+1
    end while
    i=Nk
    while(i<Nb * (Nr+1))
        temp=w[i-1]
        if(i mod Nk=0)
          temp=SubWord(RotWord(temp))xor Rcon[i/Nk]
        else if(Nk>6 and i mod Nk=4)
          temp=SubWord(temp)
        end if
        w[i]=w[i-Nk]xor temp
        i=i+1
    end while
end
```

图3.27 AES密钥扩展伪代码

SubWord()函数将接受一个4字节输入字,对每一个字节应用S盒得到输出字。函数RotWord()接受字 $[a_0, a_1, a_2, a_3]$ 作为输入,执行循环移位后返回字 $[a_1, a_2, a_3, a_0]$。轮常数字数组 $Rcon[i]$,包含了由 $[x^{i-1}, \{00\}, \{00\}, \{00\}]$ 给定的值(注意这里的 i 从1开始,而不是0),x^{i-1} 是有限域 $GF(2^8)$ 上 x(x 记为{02})的指数幂。

如图3.27所示,第一个子密钥的 Nk 个字由密码的原始密钥直接填充。接下来的每个字 $w[i]$,等于其前一个字 $w[i-1]$ 与 Nk 个位置之前的字 $w[i-Nk]$ 的异或。对于 Nk 的整数倍位置的字,在异或之前,要对 $w[i-1]$ 进行一次变换,该变换先进行一次字的字节左循

环移位(RotWord())，然后再做一次字节替代变换(SubWord())，即对字中的 4 个字节应用查表。再异或一个轮常数。

需要注意 256 比特密钥($Nk=8$)的密钥扩展程序与 128 比特和 192 比特密钥的扩展程序稍有不同。如果 $Nk=8$ 且 $i-4$ 是 Nk 的整数倍，异或之前对 $w[i-1]$要做一次字节替代(SubWord())变换。

下面以 $Nk=4$(即密钥为 128 位)为例，先概括 AES 的密钥扩展过程，随后将给出密钥扩展的例子。

密钥扩展过程如下。

(1) 将输入密钥直接复制到扩展密钥数组的前 4 个字中，得到 $w[0]$、$w[1]$、$w[2]$、$w[3]$。

(2) 然后每次用 4 个字填充扩展密钥数组 w 的余下部分，$w[i]$值依赖于 $w[i-1]$和 $w[i-4]$，$i\geqslant 4$。

(3) 当数组 w 下标不是 4 的倍数时，$w[i]$值为 $w[i-1]$和 $w[i-4]$ 的异或。

(4) 当数组 w 下标为 4 的倍数时，按照下面方法计算：

① 将 w_{i-1}的一个字的 4 个字节循环左移一个字节，即将输入字$[b_0,b_1,b_2,b_3]$变为$[b_1,b_2,b_3,b_0]$。

② 用 S 盒对输入字的每个字节进行字节代换。

③ 将 w_{i-4}异或步骤①和步骤②的结果，再与轮常数 $R\mathrm{con}[i]$相异或。

轮常数 $R\mathrm{con}[i]$是一个字，这个字的最右边三个字节总是 0。因此与轮常数的一个字异或，其结果是与该字最左边的那个字节相异或。每轮的轮常数均不同，其定义为 $R\mathrm{con}[i]=(\mathrm{RC}[i],\{00\},\{00\},\{00\})$，其中 $\mathrm{RC}[1]=\{01\}$，$\mathrm{RC}[i]=\{02\}\cdot(\mathrm{RC}[i-1])$，用多项式表示为 $\mathrm{RC}[i]=x\cdot(\mathrm{RC}[i-1])=x^{i-1}$，$i\geqslant 2$。

字节用十六进制表示，同时理解为 $\mathrm{GF}(2^8)$上的元素。x^{i-1}为 $\mathrm{GF}(2^8)$中的多项式 x 的 $i-1$ 次方所对应的字节。考虑 x 对应的字节为$\{02\}$，轮常数也可以写为：

$$R\mathrm{con}[i]=((02)^{i-1},\{00\},\{00\},\{00\})$$

前 10 个轮常数 $\mathrm{RC}[i]$的值如表 3.7 所示，对应的 $R\mathrm{con}[i]$ 如表 3.8 所示。

表 3.7　RC[*i*]

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

表 3.8　*R*con[*i*]

i	1	2	3	4	5
***R*con[*i*]**	01000000	02000000	04000000	08000000	10000000
i	6	7	8	9	10
***R*con[*i*]**	2000000	40000000	80000000	1B000000	36000000

例 3.10　AES 的加密密钥为 2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F3C，$Nk=4$，写出扩展后前三个子密钥。

直接得到第一个子密钥的 4 个字为 $w[0]=$2B7E1516，$w[1]=$28AED2A6，$w[2]=$ABF71588，$w[3]=$09CF43C。第二个和第三个子密钥扩展如表 3.9 所示。

表 3.9 密钥扩展例子

i	$w[i-1]$	RotWord()后	SubWord()后	$Rcon[i/Nk]$	与 $Rcon$ 异或后	$w[i-Nk]$	$w[i]=w[i-1]\oplus w[i-Nk]$
4	09CF4F3C	CF4F3C09	8A84EB01	01000000	8B84EB01	2B7E1516	A0FAFE17
5	A0FAFE17					28AED2A6	88542CB1
6	88542CB1					ABF71588	23A33939
7	23A33939					09CF4F3C	2A6C7605
8	2A6C7605	6C76052A	50386BE5	02000000	52386BE5	A0FAFE17	F2C29512
9	F2C29512					88542CB1	7A96B943
10	7A96B943					23A33939	5935807A
11	5935807A					2A6C7605	7359F67F

3.6.8 AES解密算法

AES解密算法是AES加密算法的逆变换,其结构类似于加密算法。解密算法和加密算法轮结构的顺序不同。但是加密和解密算法中的密钥编排形式相同。在加密过程中,其轮结构是字节代换、行移位、列混淆和轮密钥加。在解密过程中,其轮结构是逆向行移位、逆向字节代换、轮密钥加和逆向列混淆。图3.28描述了解密算法的伪代码。

```
InvCipher(byte in[4 * Nb],byte out[4 * Nb],word w[Nb * (Nr+1)])
begin
    byte state[4,Nb]

    state=in

    AddRoundKey(state,w[Nr * Nb,(Nr+1) * Nb-1])

    for round=Nr-1 step -1 downto 1
        InvShiftRows(state)
        InvSubBytes(state)
        AddRoundKey(state,w[round * Nb,(round+1) * Nb-1])
        InvMixColumns(state)
    end for

    InvShiftRows(state)
    InvSubBytes(state)
    AddRoundKey(state,w[0,Nb-1])

    out=state
end
```

图 3.28 AES解密算法的伪代码

1. 逆向行移位

逆向行移位(InvShiftRows())是行移位(ShiftRows())的逆变换。对State的各行按照一定量进行循环移位。当$Nb=4$或者6时,第0行不移位;第1行循环右移1位;第2行循环右移2位;第3行循环右移3位。

2. 逆向字节代换

逆向字节代换(InvSubBytes())是字节代换(SubBytes())的逆变换,对State的每个字节应用逆S盒进行代换。逆字节替代变换中使用的逆S盒如图3.29所示。

		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
	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	IF	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
	D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
	E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
	F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

图 3.29　逆 S 盒

3. 轮密钥加

轮密钥加变换，其逆变换就是它本身，因为其中只应用了异或运算。

4. 逆向列混淆

逆向列混淆(InvMixColumns())是列混淆(MixColumns())的逆变换。逆向列混淆在 State 上对每一列进行运算，将 State 的列看做 GF(2^8)上的多项式且被一个固定的多项式 $a^{-1}(x)$模 x^4+1 乘，$a^{-1}(x)$为：

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

可以写成矩阵乘法。令 $s'(x)=a^{-1}(x)\otimes s(x)$：

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

经过该乘法计算后，一列中的 4 个字节将由下述结果取代：

$$s'_{0,c} = (\{0e\} \cdot s_{0,c}) \oplus (\{0b\} \cdot s_{1,c}) \oplus (\{0d\} \cdot s_{2,c}) \oplus (\{09\} \cdot s_{3,c})$$
$$s'_{1,c} = (\{09\} \cdot s_{0,c}) \oplus (\{0e\} \cdot s_{1,c}) \oplus (\{0b\} \cdot s_{2,c}) \oplus (\{0d\} \cdot s_{3,c})$$
$$s'_{2,c} = (\{0d\} \cdot s_{0,c}) \oplus (\{09\} \cdot s_{1,c}) \oplus (\{0e\} \cdot s_{2,c}) \oplus (\{0b\} \cdot s_{3,c})$$
$$s'_{3,c} = (\{0b\} \cdot s_{0,c}) \oplus (\{0d\} \cdot s_{1,c}) \oplus (\{09\} \cdot s_{2,c}) \oplus (\{0e\} \cdot s_{3,c})$$

3.6.9　等价的解密变换

在第 3.6.8 节描述的解密算法中，其各个变换的操作顺序与加密算法不同，但是加密和解密算法中的密钥编排形式相同。其缺点是对同时要求加密和解密的应用而言，需要两个不同的软件或者固件模块。因此，需要构造一个等价的解密算法，解密时各个变换的操作顺序与加密(由逆向变换取代原来的变换)相同。为了达到这个要求，需要对密钥扩展进行改进。

通过两处改进可以使解密算法结构和加密算法结构一致。前面已经提到,加密算法和解密算法的轮结构顺序不同,如果将解密轮中的前两个变换阶段交换,后两个变换阶段也交换就能保证解密算法结构和加密算法结构相同。

1. 交换逆向行移位和逆向字节代换

字节代换和行移位的顺序不影响结果。先进行行字节代换,再进行行移位等价于先进行行移位,再进行字节代换。对于逆向行移位和逆向字节代换同样成立,因此可以将这两个操作交换。

2. 交换轮密钥加和逆向列混淆

轮密钥加和逆向列混淆并不改变 State 中的字节顺序。如果将密钥看成是字的序列,那么轮密钥加和逆向列混淆每次都是对 State 的一列进行操作。列混淆(MixColumns())和逆向列混淆(InvMixColumns())运算是关于列输入的线性变换,那么:

$$\text{InvMixColumns}(\text{State} \oplus \text{Round Key}) = \text{InvMixColumns}(\text{State}) \oplus \text{InvMixColumns}(\text{Round Key})$$

这表明密钥加和逆向列混淆可以互换,只要将解密中密钥编排得到的密钥列(字)应用 InvMixColumns()变换进行修改即可。

等价的解密算法如图 3.30 所示。算法将 InvSubBytes()变换和 InvShiftRows()变换的顺序反转过来,同时当利用 InvMixColumns()变换时,修改了 1～$Nr-1$ 轮的解密密钥编排结果后,并将"轮循环"中使用的 AddRoundKey()变换和 InvMixColumns()变换的顺序反转。解密密钥编排结果的第一个和最后一个 *Nb* 字将不使用该方式进行修改。字数组 dw[ ]中包含了修改过的解密密钥编排结果。图 3.30 中也显示了我们对密钥扩展程序的修改。

```
EqInvCipher(byte in[4 * Nb],byte out[4 * Nb],word dw[Nb * (Nr+1)])
begin
    byte state[4,Nb]

    state=in

    AddRoundKey(state,dw[Nr * Nb,(Nr+1) * Nb-1])

    for round=Nr-1 step -1 downto 1
        InvSubBytes(state)
        InvShiftRows(state)
        InvMixColumns(state)
        AddRoundKey(state,dw[round * Nb,(round+1) * Nb-1])
    end for

    InvSubBytes(state)
    InvShiftRows(state)
    AddRoundKey(state,dw[0,Nb-1])

    out=state
end
下面的伪代码加在密钥扩展算法后面:
    for i=0 step 1 to (Nr+1) * Nb-1
        dw[i]=w[i]
    end for

    for round=1 step 1 to Nr-1
        InvMixColumns(dw[round * Nb,(round+1) * Nb-1])
    end for
```

图 3.30 AES 等价解密算法

3.6.10　AES 的安全性

AES 的设计的各个方面都使它具有能够抵抗所有已知攻击的能力。AES 的轮函数设计为基于宽轨迹策略(Wide Trail Strategy),这种设计策略是针对差分密码分析和线性密码分析的。主要包括两个设计准则:其一是选择差分均匀性比较小和非线性度比较高的 S 盒;其二是适当选择线性变换,使得固定轮数中的活动 S 盒的个数尽可能多。如果差分特征(或线性逼近)中某一轮的活动 S 盒的个数比较少,那么下一轮中的活动 S 盒的个数就必须要多一些。宽轨迹策略的最大优点是可以估计算法的最大差分特征概率和最大线性逼近概率,由此可以评估算法抵抗差分密码分析和线性密码分析的能力。另外 AES 的密钥长度也足以抵抗穷举密钥攻击。并且 AES 算法对密钥的选择没有任何限制,还没有发现弱密钥和半弱密的存在。

3.7　中国商用密码算法——SMS4

SMS4 算法是中国国家商用密码管理办公室于 2006 年 1 月公布的用于无线局域网产品的分组对称密码算法,是国内官方公布的第一个商用密码算法,具有较好的抗破解能力。SMS4 是分组对称密码算法,分组长度和密钥长度为 128 比特。SMS4 算法的 S 盒设计已经达到欧美分组密码标准算法 S 盒的设计水准,具有较好平衡性和非线性,但其算法的整体安全特性还有待进一步研究。SMS4 算法的提出,无论是对无线局域网产业还是对商用密码研究都有非常重要的意义,极大地推进了对密码算法的研究及其开发本土化的进程。

3.7.1　SMS4 加密

SMS4 算法是一个分组算法,分组长度为 128 位,密钥长度也为 128 位。SMS4 算法中数据处理单位有字和字节,字为一个整体的 32 比特序列,字节为 8 比特序列。加密算法与密钥扩展算法都采用 32 轮非线性迭代结构,一次迭代运算为一轮变换。因此,128 位明文的字数组表示为 X_0,X_1,X_2,X_3;密文为 Y_0,Y_1,Y_2,Y_3。加密过程如图 3.31 所示,128 位明文分为 4 个 32 位的字,经过 32 轮加密变换,每一轮的加密变换为:首先循环左移,最后一个字经过轮函数 F 得到。32 轮的加密变换结束后,将 4 个字反序变化后,得到 128 位密文。

1. 基本运算

SMS4 算法中,采用了两种基本运算:“⊕”表示 32 比特异或运算,“$<<<i$”表示 32 比特循环左移 i 位。

2. SMS4 每轮结构

SMS4 每轮的结构如图 3.32 所示。上一轮的数据($X_{i-1},X_i,X_{i+1},X_{i+2}$),$i=1,2,\cdots,32$。循环左移 32 位,然后利用加密函数(又称轮函数)F,得到 $X_{i+3}=F(X_{i-1},X_i,X_{i+1},X_{i+2},rk_{i-1})$。因此,此轮输出($X_i,X_{i+1},X_{i+2},X_{i+3}$)作为下一轮迭代的输入。如此迭代 32 轮后,得到输出数据为($X_{32},X_{33},X_{34},X_{35}$)。再做反序变换 R,得到最终密文:$(Y_0,Y_1,Y_2,Y_3)=R(X_{32},X_{33},X_{34},X_{35})=(X_{35},X_{34},X_{33},X_{32})$。

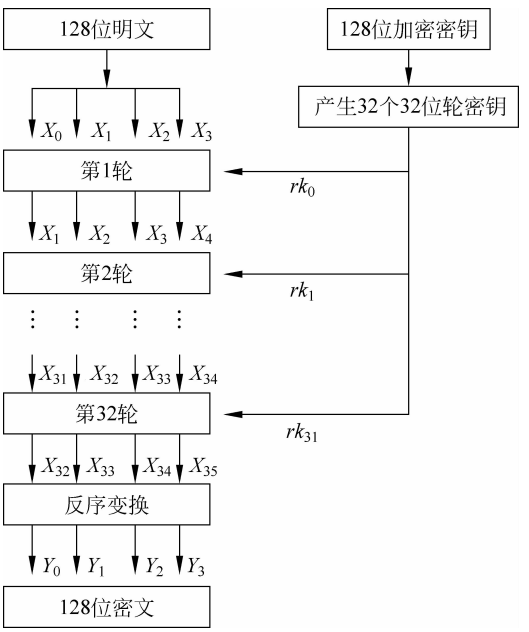

图 3.31 SMS4 加密算法

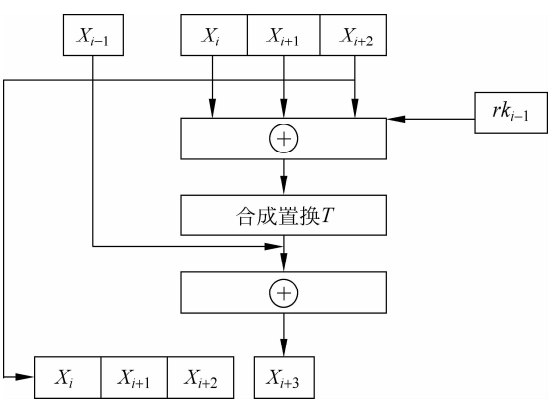

图 3.32 SMS4 加密每轮结构

3. 加密函数 *F*

加密函数 F 为 $F(X_{i-1},X_i,X_{i+1},X_{i+2},rk_{i-1})=X_{i-1}\oplus T(X_i\oplus X_{i+1}\oplus X_{i+2}\oplus rk_{i-1})$,其中合成置换 T 是一个可逆变换,由非线性变换 τ 和线性变换 L 构成。加密函数 F 的计算过程为:输入数据的 X_{i-1} 与合成置换 T 的输出做异或运算即可。合成置换 T 的过程如图 3.33 所示。

τ 是由 4 个相同的八进八出 S 盒并置而成,每一轮 $X_i\oplus X_{i+1}\oplus X_{i+2}\oplus rk_{i-1}$ 进行计算后,假设得到结果记为 32 位的 $A=(a_0,a_1,a_2,a_3)$,a_0、a_1、a_2、a_3 每个 8 位,分为 4 组进入 S 盒,输出为 $B=(b_0,b_1,b_2,b_3)=\tau(A)=(S(a_0),S(a_1),S(a_2),S(a_3))$。

S 盒为固定的 8 比特输入 8 比特输出的置换,记为 S(.),如图 3.34 所示。S 盒中的数据都是通过十六进制表示的,它的置换规则是:以输入的前半字节为行号,后半字节为列

号，行列交叉点处的数据即为输出。

举例：若输入“ef”，则经 S 盒后的值为表中第 e 行和第 f 列的值，S(ef)＝84。

非线性变换 τ 的输出是线性变换 L 的输入。设输入为 B，单位为字，输出为 C，单位为字。则 $C=L(B)=B\oplus(B<<<2)\oplus(B<<<10)\oplus(B<<<18)\oplus(B<<<24)$。

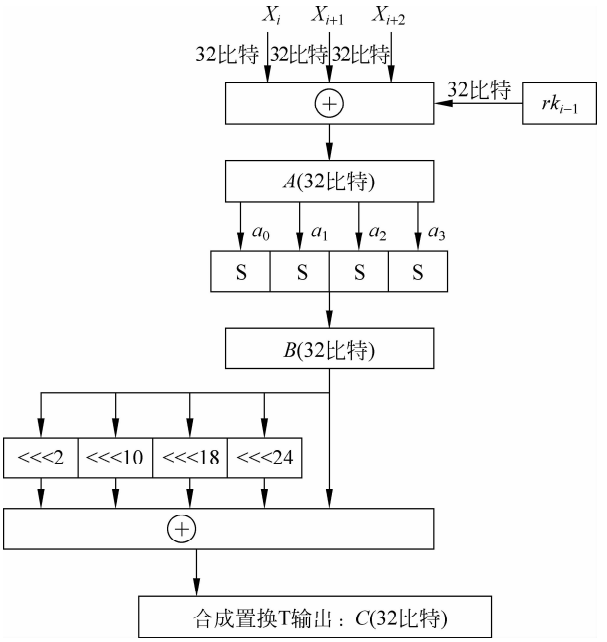

图 3.33　合成置换 T

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

图 3.34　SMS4 的 S 盒

3.7.2　密钥扩展算法

SMS4 加密算法的轮密钥由加密密钥通过密钥扩展算法生成。加密密钥由 4 个字，128

位组成,表示为 $MK=(MK_0,MK_1,MK_2,MK_3)$,其中 $MK_i(i=0,1,2,3)$为字。轮密钥表示为$(rk_0,rk_1,\cdots,rk_{31})$,其中 $rk_i(i=0,1,2,\cdots,31)$为字。

$FK=(FK_0,FK_1,FK_2,FK_3)$为系统参数,$CK=(CK_0,CK_1,\cdots,CK_{31})$为固定参数,用于密钥扩展算法,其中 $FK_i(i=0,1,2,3)$、$CK_i(i=0,1,\cdots,31)$为字。

令 $K_i(i=0,1,\cdots,35)$为字,轮密钥生成方法如图 3.35 所示。

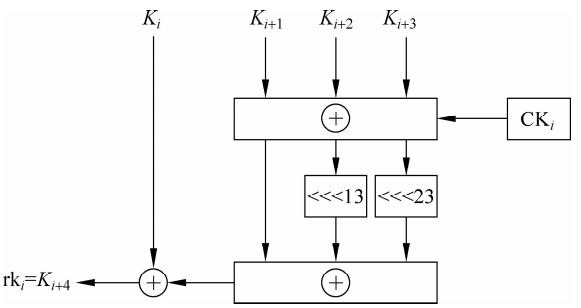

图 3.35 SMS4 轮密钥生成

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

然后,对 $i=0,1,2,\cdots,31$:$rk_i=K_{i+4}=K_i\oplus T'(K_{i+1}\oplus K_{i+2}\oplus K_{i+3}\oplus CK_i)$

说明:

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

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

(2) 系统参数 FK 的取值,采用十六进制表示为:

$$FK_0=(A3B1BAC6),\quad FK_1=(56AA3350)$$
$$FK_2=(677D9197),\quad FK_3=(B27022DC)$$

(3) 固定参数 CK 的取值方法为:

设 $ck_{i,j}$为 CK_i 的第 j 字节$(i=0,1,\cdots,31;\ j=0,1,2,3)$,即 $CK_i=(ck_{i,0},ck_{i,1},ck_{i,2},ck_{i,3})$,则 $ck_{i,j}=(4i+j)\times 7(\mathrm{mod}\ 256)$。32 个固定参数 CK_i,其十六进制表示为:

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

例 3.11 按照 SMS4 加密算法,对一组明文(01 23 45 67 89 ab def fe dc ba 98 76 54 32 10)用密钥加密一次。加密密钥:01 23 45 67 89 ab def fe dc ba 98 76 54 32 10。其中,数据采用十六进制表示。求每一轮的轮密钥、每轮最后 32 位的输出状态以及最终的密文。

解:(1) 第一轮轮密钥 rk_0的计算过程如下。

① 首先,将加密密钥 $MK=(MK_0,MK_1,MK_2,MK_3)$按字分为 4 组,分别为:

$$MK_0=(01234567),\quad MK_1=(89abcdef),\quad MK_2=(fedcba98),\quad MK_3=(76543210)$$

已知 $FK_0=(A3B1BAC6)$，$FK_1=(56AA3350)$，$FK_2=(677D9197)$，$FK_3=(B27022DC)$

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

求出(K_0,K_1,K_2,K_3)各个的值：

$$MK_0 = 0000\ 0001\ 0010\ 0011\ 0100\ 0101\ 0110\ 0111$$

$$FK_0 = 1010\ 0011\ 1011\ 0001\ 1011\ 1010\ 1100\ 0110$$

两者做异或运算后，得：

$$K_0=1010\ 0010\ 1001\ 0010\ 1111\ 1111\ 1010\ 0001$$

同理计算，可以分别得：

$$K_1 = 1101\ 1111\ 0000\ 0001\ 1111\ 1110\ 1011\ 1111$$

$$K_2 = 1001\ 1001\ 1010\ 0001\ 0010\ 1011\ 0000\ 1111$$

$$K_3 = 1100\ 0100\ 0010\ 0100\ 0001\ 0000\ 1100\ 1100$$

② 然后，求出 T'变换的输出。

固定参数 CK_0 的十六进制表示为 00070e15。则通过异或运算 $A=(a_0,a_1,a_2,a_3)=K_1\oplus K_2\oplus K_3\oplus CK_0=(1000\ 0010\ 1000\ 0011\ 1100\ 1011\ 0110\ 1001)$，十六进制表示为 82 83 cb 69。

再将输出结果 A 分为 4 组进入 S 盒，通过查表，输出 B 的十六进制为 8a d2 41 22。

利用公式 $L'(B)=B\oplus(B<<<13)\oplus(B<<<23)$，得：

$$L'(B) = 0101\ 0011\ 1011\ 0011\ 0111\ 1001\ 0101\ 1000$$

③ 最后，利用公式 $rk_i=K_{i+4}=K_i\oplus T'(K_{i+1}\oplus K_{i+2}\oplus K_{i+3}\oplus CK_i)$，得到：

$rk_0=K_4=K_0\oplus T'(B)=1111\ 0001\ 0010\ 0001\ 1000\ 0110\ 1111\ 1001$，十六进制表示为 f1 21 86 f9。

(2) 根据第 1 轮轮密钥 rk_0 的计算结果，将 128 位的明文同样分组，利用加密函数 $F(X_{i-1},X_i,X_{i+1},X_{i+2},rk_{i-1})=X_{i-1}\oplus T(X_i\oplus X_{i+1}\oplus X_{i+2}\oplus rk_{i-1})$，得到第 1 轮 X_4 的输出状态为 27 fa d3 45。

(3) 按照第(1)步和第(2)步计算，能够求出每轮的轮密钥和后 32 位的输出状态。

rk_0=f12186f9	X_4=27fad345，	rk_1=41662b61	X_5=a18b4cb2
rk_2=5a6ab19a	X_6=11c1e22a，	rk_3=7ba92077	X_7=cc13e2ee
rk_4=367360f4	X_8=f87c5bd5，	rk_5=776a0c61	X_9=33220757
rk_6=b6bb89b3	X_{10}=77f4c297，	rk_7=24763151	X_{11}=7a96f2eb
rk_8=a520307c	X_{12}=27dac07f，	rk_9=b7584dbd	X_{13}=42dd0f19
rk_{10}=c30753ed	X_{14}=b8a5da02，	rk_{11}=7ee55b57	X_{15}=907127fa
rk_{12}=6988608c	X_{16}=8b952b83，	rk_{13}=30d895b7	X_{17}=d42b7c59
rk_{14}=44ba14af	X_{18}=2ffc5831，	rk_{15}=104495a1	X_{19}=f69e6888
rk_{16}=d120b428	X_{20}=af2432c4，	rk_{17}=73b55fa3	X_{21}=ed1ec85e
rk_{18}=cc874966	X_{22}=55a3ba22，	rk_{19}=92244439	X_{23}=124b18aa
rk_{20}=e89e641f	X_{24}=6ae7725f，	rk_{21}=98ca015a	X_{25}=f4cba1f9
rk_{22}=c7159060	X_{26}=1dcdfa10，	rk_{23}=99e1fd2e	X_{27}=2ff60603
rk_{24}=b79bd80c	X_{28}=eff24fdc，	rk_{25}=1d2115b0	X_{29}=6fe46b75
rk_{26}=0e228aeb	X_{30}=893450ad，	rk_{27}=f1780c81	X_{31}=7b938f4c
rk_{28}=428d3654	X_{32}=536e4246，	rk_{29}=62293496	X_{33}=86b3e94f
rk_{30}=01cf72e5	X_{34}=d206965e，	rk_{31}=9124a012	X_{35}=681edf34

(4) 按照$(Y_0,Y_1,Y_2,Y_3)=R(X_{32},X_{33},X_{34},X_{35})=(X_{35},X_{34},X_{33},X_{32})$,得到的密文结果为 68 1e df 34 d2 06 96 5e 86 b3 e9 4f 53 6e 42 46。

3.7.3 SMS4 解密

SMS4 的解密变换与加密变换结构相同,不同的仅是轮密钥的使用顺序。

- 加密时轮密钥的使用顺序为:$(rk_0,rk_1,\cdots,rk_{31})$
- 解密时轮密钥的使用顺序为:$(rk_{31},rk_{30},\cdots,rk_0)$

3.7.4 SMS4 的安全性

SMS4 算法是由中国国家专业机构设计,自 2006 年公布之后,SMS4 分组密码引起了国内外学术界和产业界的极大关注,先后有学者研究了 SMS4 对差分故障攻击、积分攻击、不可能差分密码分析、代数攻击、矩阵攻击、差分密码分析、线性密码分析等分析方法的安全性。至今为止,从专业机构对 SMS4 进行的密码分析来看,SMS4 算法还是安全的,虽然有学者提出 21 轮 SMS4 在受到差分密码分析和差分能量分析的攻击时将面临威胁,但尚需经过实践检验。

3.8 RC6

RC6 是 RSA 公司提交给 NIST 的一个候选高级加密标准算法,它是在 RC5 的基础上设计的。RC6 继承了 RC5 的优点。为了使它符合高级加密标准,RC6 在 RC5 基础上将分组长度扩展成 128 位,用 4 个 32 位区块代替 RC5 的两个 32 位区块。RC6 是参数可变的分组密码算法,3 个可变的参数是:分组大小、密钥大小和加密轮数。RC6 常常写为 RC6-w/r/b,其中 w 是字的大小,以位为单位,r 为加密轮数,允许值是 0,…,255,b 为密钥长度,单位是字节,$0\leqslant b\leqslant 255$。例如 RC6-32/16/10 表示字长为 32 位,迭代的轮数是 16,密钥长度为 10 字节。由于高级加密标准的要求是 $w=32$,$r=20$,因此满足 AES 的 RC6 算法是 RC6-32/20/16,也就是 4 个 32 位字(128 位),迭代的轮数为 20,密钥长度 16 个字节。RC6 定义了 6 种运算基本操作,以 2 为底的对数表示为 $\lg w$。

$a+b$:模 2^w 整数加。

$a-b$:模 2^w 整数减。

$a\oplus b$:w 位的字按位异或。

$a\times b$:模 2^w 整数乘。

$a\lll b$:循环左移 w 位的字 a,移动位数由 b 的低位 $\lg w$ 位决定。

$a\ggg b$:循环右移 w 位的字 a。

3.8.1 RC6 的加密和解密

RC6 和 RC5 在加解密方面是不一样的,RC6 用这 4 个 w 位寄存器 A、B、C、D 来存放输入的明文和输出的密文。明文和密文的第一个字节放在 A 的最低字节(即第一个字节),明

文和密文的最后一个字节放在 D 的最高字节(即最后一个字节)。RC6 的加密过程如图 3.36 所示,其中 $f(x)=x\times(2x+1)$。加密算法和解密算法分别如图 3.37 和图 3.38 所示。

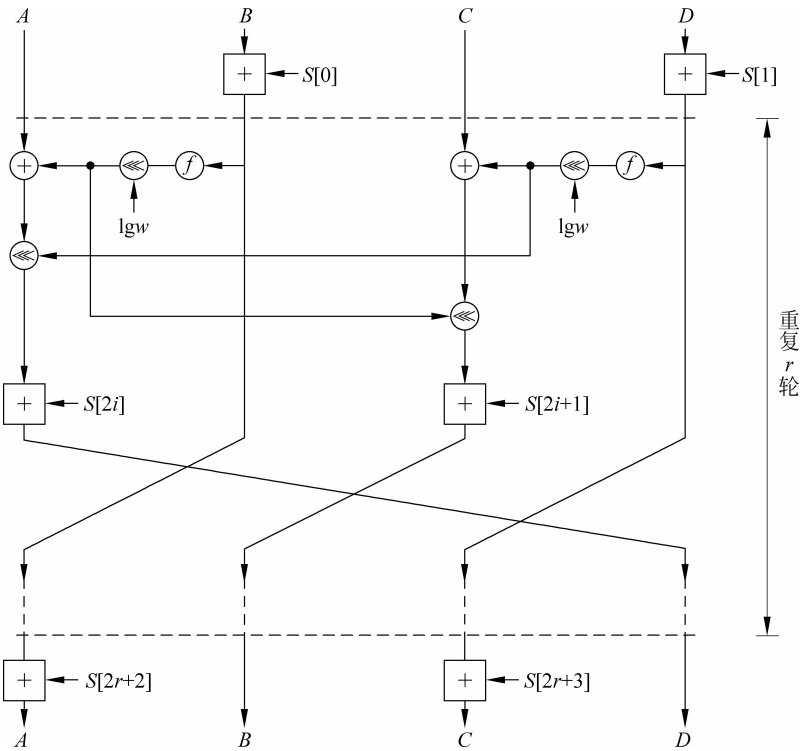

图 3.36　RC6 加密过程

RC6-$w/r/b$ 加密
输入：明文存入 4 个 w 位寄存器 A、B、C、D
　　轮数 r
　　w 位轮密钥 $S[0,1,\cdots,2r+3]$
输出：密文存入 4 个 w 位寄存器 A、B、C、D
过程：$B=B+S[0]$
　　$D=D+S[1]$
　　for $i=1$ to r do
　　{
　　$t=(B\times(2B+1))\lll\lg w$
　　$u=(D\times(2D+1))\lll\lg w$
　　$A=((A\oplus t)\lll u)+S[2i]$
　　$C=((C\oplus u)\lll t)+S[2i+1]$
　　$(A,B,C,D)=(B,C,D,A)$
　　}
　　$A=A+S[2r+2]$
　　$C=C+S[2r+3]$

图 3.37　RC6 加密算法

3.8.2　密钥扩展

密钥扩展算法是从密钥 K 中导出 $2r+4$ 个字长的密钥,储存在数组 $S[0,\cdots,2r+3]$中用

于加密和解密。在这其中用到了两个常量 Pw 和 Qw,Pw 和 Qw 大小是一个字长,定义如下所示。

$$Pw=\mathrm{Odd}((e-2)2^w)$$

$$Qw=\mathrm{Odd}((\varphi-2)2^w)$$

其中,$e=2.718\ 281\ 828\ 4\cdots$(自然对数),$\varphi=1.618\ 033\ 988\ 74\cdots$(黄金分割),$\mathrm{Odd}(x)$是离 x 最近的奇数。

```
RC6-w/r/b 解密
输入: 密文存入 4 个 w位寄存器 A、B、C、D
      轮数 r
      w 位轮密钥 S[0,1,…,2r+3]
输出: 明文存入 4 个 w 位寄存器 A、B、C、D
过程: C=C-S[2r+3]
      A=A-S[2r+2]
      for i=r downto 1 do
         {
            (A,B,C,D)=(D,A,B,C)
            u=(D×(2D+1))<<<lgw
            t=(B×(2B+1))<<<lgw
            C=((C-S[2i+1])>>>t)⊕u
            A=((A-S[2i])>>>u)⊕t
         }
      D=D-S[1]
      B=B-S[0]
```

图 3.38 RC6 解密算法

密钥扩展时,首先将密钥 $K[0,\cdots,b-1]$放入 c 个 w 位字的另一个数组 $L[0,\cdots,c-1]$中,其中,c 为 b/u 的整数部分,$u=w/8$,即 L 数组上的元素大小为 uw 位。将 u 个连续字节的密钥顺序放入 L 中,先放入 L 中的低字节,再放入其高字节。如果 L 未填满,用 0 填充。当 $b=0$,$c=0$ 时,$c=1$,$L[0]=0$。

其次利用 Pw 和 Qw 将数组 S 初始化为一个固定的伪随机的数组,最后将用户密钥扩展到数组 S 中,密钥扩展算法如图 3.39 所示。

```
RC6-w/r/b 密钥扩展
输入: 用户密钥字节预放入数组 L[0,…,c-1]
      轮数 r
输出: w 位的轮密钥 S[0,…,2r+3]
过程:
   S[0]=Pw
   for i=1 to 2r+3 do
      S[i]=S[i+1]+Qw
   A=B=i=j=0
   v=3×max{c,2r+4}
   for s=1 to v do
   {
      A=S[i]=(S[i]+A+B)<<<3
      B=L[j]=(L[j]+A+B)<<<(A+B)
      i=(i+1) mod (2r+4)
      j=(j+1) mod c
```

图 3.39 RC6 密钥扩展算法

3.8.3　RC6 的安全性和灵活性

RC6 是由 RC5 发展而来的，加入了二次函数 $f(x)=x\times(2x+1)$，这个函数提高了函数密码扩散速度。用二次函数变换的寄存器 B 和 D 的值来修改寄存器 A 和 C 的值，增加了密码的非线性。因此 RC6 有很好的抗差分攻击和线性攻击的能力。另外 RC6 的加密和解密的时间都与数据无关，可以有效地避免计时攻击。同时，也没有 RC6 存在类似 DES 中的弱密钥。

与其他加密算法不同的是，RC6 算法在加密过程中不需要查找表，加之算法中的乘法运算也可以用平方代替，所以该算法对内存的要求很低。这使得 RC6 特别适合在单片机上实现。

3.9　流　密　码

3.9.1　流密码基本原理

一次一密密码是绝对安全的密码，如果能以某种方式仿效一次一密密码，将可以得到安全性很高的密码。长期以来，人们试图以流密码方式仿效一次一密密码，从而促进了流密码的研究和发展。目前，序列密码的理论已经比较成熟，而且流密码实现简单、加密速度快、密文传输中的错误不会在明文中产生扩散，使得流密码成为许多重要领域应用的主流密码体制。

流密码也称为序列密码，它是对明文以一位或者一个字节为单位进行操作。为了使加密算法更安全，一般选取尽可能长的密钥，但是长密钥的存储和分配都很困难。于是流密码采用一个短的种子密钥来控制密钥流发生器创建出长的密钥序列，供加解密使用，而短的种子密钥的存储、分配都较容易。图 3.40 是流密码的加密过程。种子密钥 k 输入到密钥流发生器，产生一系列密码流，通过与同一时刻的一个字节或者一位明文流进行异或操作产生密文流。解密时只要将密文流与密钥流进行异或操作产生明文流。例如，如果密钥流发生器产生的密钥流一个字节为 10011001，明文流一个字节为 01001010，那么密钥流与明文流异或可以产生密钥流 11010011。同样，将密文流与密钥流异或就能得到明文流。

在流密码中，如果密钥流的产生完全独立于明文流或密文流，则称该流密码为同步流密码(Synchronous Stream Cipher)，如图 3.41 所示。如果密钥流的产生与明文或者密文相关，则称这类流密码为自同步流密码(Self-Synchronous Stream Cipher)，如图 3.42 所示。

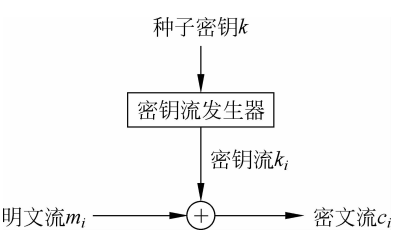

图 3.40　流密码加密过程

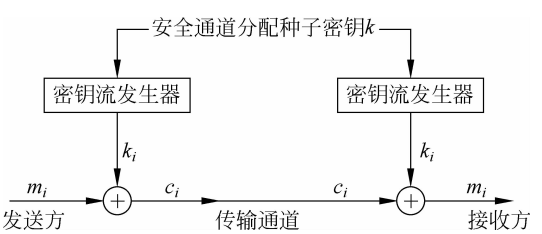

图 3.41　同步流密码

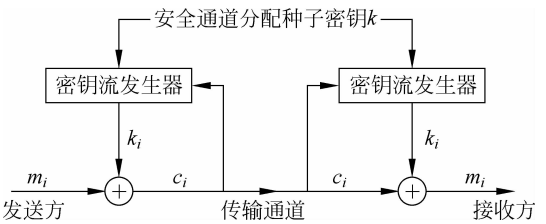

图 3.42 自同步流密码

对于同步流密码,只要通信双方的密钥流产生器具有相同的种子密钥和相同的初始状态,就能产生相同的密钥流。在保密通信过程中,通信的双方必须保持精确的同步,收方才能正确解密,如果失步收方将不能正确解密。例如,如果通信中丢失或增加了一个密文字符,则接收方将会一直收到错误的信息,直到重新同步为止,这是同步流密码的一个主要缺点。但是同步流密码对失步的敏感性,使我们能够容易检测插入、删除、重播等主动攻击。由于同步流密码各操作位之间相互独立,因此应用这种方式进行加解密时无错误传播,当操作过程中产生一位错误时只影响一位,不影响后续位,这是同步流密码的一个优点。

对于自同步流密码,每一个密钥位是由前面 n 个密文位参与运算推导出来的,其中 n 为定值。因此,如果在传输过程中丢失或更改了一个位,则这一错误就要向前传播 n 个位。因此,自同步流密码有错误传播现象。不过,在收到 n 个正确的密文位以后,密码自身会实现重新同步。在自同步流密码系统中,密文流参与了密钥流的生成,这使得对密钥流的分析非常复杂,从而导致对自同步流密码进行系统的理论分析非常困难。

3.9.2 密钥流产生器

流密码的安全强度完全取决于它所产生的密钥流的特性,如果密钥流是无限长且为无周期的随机序列,那么流密码属于"一次一密"的密码体制,但遗憾的是满足这样条件的随机序列在现实中无法生成。在实际应用当中的密钥流都是由有限存储和有限复杂逻辑的电路产生的字符序列,由于密钥流生成器只具有有限状态,那么它产生的序列具有周期性,不是真正的随机序列。现实设计中只能追求密钥流的周期尽可能长,随机性尽可能好,近似于真正的随机序列。一个好的密钥流须要满足下面几个条件。

(1) 加密序列的周期要长。密钥流生成器产生的比特流最终会出现重复。重复的周期越长,密码分析的难度越大。

(2) 密钥流应该尽可能地接近一个真正的随机数流的特征。如 1 和 0 的个数应近似相等。如果密钥流为字节流,则所有的 256 种可能的字节的值出现频率应近似相等。

(3) 为了防止穷举攻击,种子密钥值也应该有足够的长度,至少要保证它的长度不小于 128 位。

生成一个具有良好特性的密钥流序列的常见方法有:线性反馈移位寄存器(Linear Feedback Shift Register,LFSR)、非线性移位寄存器(NLFSR)、有限自动机、线性同余、混沌密码序列等方法。这些方法都是通过一个种子(有限长)密码产生具有足够长周期的、随机性良好的序列。只要生成方法和种子都相同,就会产生完全相同的密钥流。目前密钥流生成器大多是基于移位寄存器。因为移位寄存器结构简单,易于实现且运行速度快。本节主要介绍线性移位寄存器。

密钥流产生器一般由线性移位寄存器(LFSR)和一个非线性组合函数两部分构成。其中线性移位寄存器部分称为驱动部分,另一部分称为非线性组合部分,如图 3.43 所示。其工作原理是将驱动部分,即线性移位寄存器在 j 时刻的状态变量 x 作为一组值输入非线性组合部分的 f,将 $f(x)$作为当前时刻的密钥 k_j。驱动部分负责提供非线性组合部分使用的周期大、统计性能好的序列,而非线性组合部分以各时刻移位寄存器的状态组合出密钥序列。

移位寄存器是流密码产生器的主要部分,图 3.44 是一个域 GF(2)上的反馈移位寄存器,图中标有 $a_1,a_2,\cdots,a_{n-1},a_n$ 的小方框表示二值(0,1)存储单元,可以是一个双稳触发器,信号流从左向右。这 n 个二值存储单元称为该反馈移位寄存器的级。在任意一个时刻,这些级的内容构成该反馈移位寄存器的状态。每一个状态对应于 GF(2)上的一个 n 维向量,共有 2^n 种可能的状态。每个时刻的状态可用 n 长序列 $a_1,a_2,\cdots,a_n$ 或 n 维向量 $a_1,a_2,\cdots,a_n$ 表示,其中 a_i 为当前时刻第 i 级存储器中的内容。

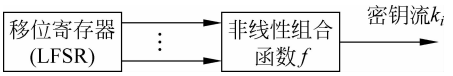

图 3.43　密钥流产生器

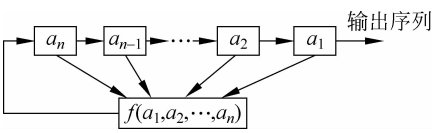

图 3.44　反馈移位寄存器

在主时钟确定的周期区间上,每一级存储器 a_i 都将其内容向下一级 a_{i-1} 传递,并根据寄存器当时的状态计算 $f(a_1,a_2,\cdots,a_n)$作为 a_n 的下一时间周期的内容,其中反馈函数 $f(a_1,a_2,\cdots,a_n)$是 n 元布尔函数。所以在时钟的每一脉冲下,总是从一个状态转移到另一个状态。

有反馈函数 $f(a_1,a_2,\cdots,a_n)=c_n a_1 \oplus c_{n-1}a_1 \oplus\cdots\oplus c_{n-1}a_n$,其中,$c_i$为 0 或者 1,⊕是模 2 加法。这个反馈函数是 $a_1,a_2,\cdots,a_n$的线性函数。称这种反馈移位寄存器为线性移位寄存器(LFSR),否则称为非线性移位寄存器。

如果反馈移位寄存器的状态为:

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

则 $a_{i+n}=f(a_i,a_{i+1},\cdots,a_{i+n-1})$,这个 a_{i+n}又是移位寄存器的输入。在 a_{i+n}的驱动下,移位寄存器的各个数据向前推移一位,使状态变为 $s_{i+1}=(a_{i+1},\cdots,a_{i+n})$,同时,整个移位寄存器的输出为 a_i。由此可以得到一系列数据 $a_1,a_2,\cdots,a_n,\cdots$。

例 3.12　图 3.45 是一个三级移位寄存器,初始状态是 $s_1=(a_1,a_2,a_3)=(1,0,1)$,写出它的输出序列的前 5 位。

从图 3.45 中可以发现反馈函数是 a_1 和 a_3 的异或,那么三级移位寄存器的输出如表 3.10 所示。

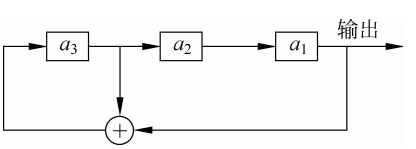

图 3.45　一个三级移位寄存器

表 3.10　三级移位寄存器的输出

状态(a_3,a_2,a_1)	输　出	状态(a_3,a_2,a_1)	输　出
101	1	010	0
010	0	110	0
101	1		

3.9.3 RC4算法

RC4是Ron Rivest在1987年为RSA数据安全公司设计的一种同步流密码。据分析显示该密码的周期大于10^{100}。每输出一字节的结果仅需要8～16条机器操作指令。RC4的应用很广,例如它被集成于Microsoft Windows、Lotus Notes、Apple AOCE和Oracle Secure SQL中,还被用于SSL/TLS、WEP(Wired Equivalent)等协议中。

RC4不是基于移位寄存器的流密码,而是一种基于非线性数据表变换的流密码,它以一个足够大的数据表为基础,对表进行非线性变换,产生非线性密钥流序列。RC4是一个可变密钥长度、面向字节操作的流密码,该字节的大小n可以根据用户需要来定义,一般应用中n都取8位。流密钥的生成需要两个处理过程:一个是密钥调度算法(Key-Scheduling Algorithm,KSA),用来设置数据表S的初始排列;另一个是伪随机产生算法(Pseudo Random-Generation Algorithm,PRGA),用来选取随机元素并修改S的原始排列顺序。

密钥流产生过程是用1～256个字节的可变长度的密钥初始化一个256个字节的数据表S,S的元素记为$S(i)$,$0 \leqslant i \leqslant 255$,大小为一个字节。加密和解密用的每一个密钥$K(i)$都是由$S$中的元素按照一定的方式选出一个元素而生成的。每生成一个$K(i)$值,S中的元素就被重新置换一次。

1. KSA初始化S

初始化时,先对S进行填充,即令$S(0)=0$,$S(1)=1$,$S(2)=2$,…,$S(255)=255$。用种子密钥填充一个256个字节的密钥表K,$K(0)$,$K(1)$,$K(2)$,…,$K(255)$,如果种子密钥的长度小于K的长度,则依次重复填充,直到将K填满,然后通过$S(i)$和$K(i)$置换S中的元素,其过程如下。

```
for i=0 to 255 do
        S(i)=i;
        T(i)=K(i mod keylen)
    j=0;
    for i=0 to 255 do
      j=(j+S(i)+T(i)) mod 256;
      Swap (S(i),S(j));
```

其中keylen为种子密钥长度,$T(i)$是一个临时数据表。上面过程对S操作仅仅是交换,交换后S所包含的值仍然是0～255的元素。

2. 密钥流生成

数据表S一旦完成初始化,将不再使用种子密钥。当KSA完成S的初始化后,PRGA就将接手工作,它为密钥流选取一个个的字节,即从S中选取随机元素,并修改S以便下一次选取。密钥流的生成是$S(0)$～$S(255)$,对每个$S(i)$,根据当前的S值,将$S(i)$与S中的另一个字节置换。当$S(255)$完成置换后,操作继续重复从$S(0)$开始。密钥流的选取过程如下。

```
i=0,  j=0;
while(true)
  i=(i+1) mod 256;
  j=(j+S(i)) mod 256;
  Swap(S(i),S(j));
  t=(S(i)+S(j)) mod 256;
  k=S(t);
```

例 3.13　假如使用 3 位(0～7)的 RC4,其操作是对 8 取模(而不是对 256 取模)。数据表 5 只有 8 个元素。初始化为:

	0	1	2	3	4	5	6	7
S	0	1	2	3	4	5	6	7

选取一个密钥,该密钥是由 0～7 的数以任意顺序组成的。例如选取 5、6 和 7 作为密钥。将该密钥如下填入密钥数据表中。

	5	6	7	5	6	7	5	6
K	0	1	2	3	4	5	6	7

利用如下循环构造实际 S 数据表。

```
j=0;
  for i=0 to 7 do
    j=(j+S(i)+K(i)) mod 8;
    Swap(S(i),S(j));
```

该循环以 $j=0$ 和 $i=0$ 开始,使用更新公式后 j 为:

$$j=(0+S(0)+K(0)) \bmod 8=(0+0+5) \bmod 8=5$$

因此,S 数据表的第一个操作是将 $S(0)$与 $S(5)$互换,互换结果如下所示。

	5	1	2	3	4	0	6	7
S	0	1	2	3	4	5	6	7

i 加 1 后,j 的下一个值为:

$$j=(5+S(1)+K(1)) \bmod 8=(5+1+6) \bmod 8=4$$

即将 S 数据表的 $S(1)$与 $S(4)$互换,互换结果如下所示。

	5	4	2	3	1	0	6	7
S	0	1	2	3	4	5	6	7

当该循环执行完后,数据表 S 就被随机化为:

	5	4	0	7	1	6	3	2
S	0	1	2	3	4	5	6	7

这样数据表 S 就可以用来生成随机的密钥流序列了。从 $j=0$ 和 $i=0$ 开始开始,RC4 将如下所示计算第一个密钥字:

$$i=(i+1) \bmod 8=(0+1) \bmod 8=1$$

$$j=(j+S(i)) \bmod 8=(0+S(1)) \bmod 8=(0+4) \bmod 8=4$$

$$\text{Swap}(S(1),S(4))$$

交换后数据表 S 变为:

	5	1	0	7	4	6	3	2
S	0	1	2	3	4	5	6	7

然后如下计算 t 和 k：

$$t=(S(i)+S(j)) \bmod 8=t=(S(1)+S(4)) \bmod 8=(1+4) \bmod 8=5$$

$$k=S(t)=S(6)=6$$

第一个密钥字是6,其二进制表示为110。重复该过程,直到生成的二进制位的数量等于明文位的数量。

常见的RC4实现是基于 $n=8$ 的(数字为0～255)。这种系统执行完KSA后的数据表S是0～255的一个排列,共有256!(即 2^{1600})种可能。这相当于使用一个1600位的密钥,这使得穷举攻击变得不可能。在RC4中,有一些弱点可用来破解该加密法。例如,RSA永远不会生成某类密钥,例如 $j=i+1$ 与 $S(j)=1$。事实证明,这类密钥的数量占到了所有可能密钥数的 2^{-2n}。当 $n=8$ 时,就是 $(256!/2^{16})$。

RC4算法在设计过程中采用了非线性的S盒,能够抵抗差分攻击和线性分析,但是RC4的安全性还是令人担忧。1999年,Kundanrewich等人设计了一个可编程逻辑电路用33天穷举了40位的RC4算法;2001年,Fluhrer等人则提出了分析RC4算法的有效方法,称为FMS的分析方法,该方法主要是针对WEP(Wired Equivalent Privacy)协议,该协议采用流密码算法RC4作为加密算法,为了克服流密钥重用的问题,在WEP协议中引入了初始向量,这样可以增强抗穷举攻击能力,但是也产生了FMS的分析方法,该方法是针对以明文传送的初始向量和对应的802.11帧的特点进行攻击的。

3.10 分组密码工作模式

分组密码算法是提供数据安全的一个基本构件。分组密码是针对固定大小的分组进行加密的。例如,DES是对64位的明文分组进行加密,AES是对128位分组操作。但需要保密传输的消息不一定刚好是一个分组大小,为了在实际中应用分组密码,我们定义了5种工作模式。任何一种对称分组密码算法都可以以这些方式进行应用。

3.10.1 电子密码本模式

电子密码本(Electronic Code Book,ECB)模式是分组密码的基本工作方式,它将明文分割成独立大小的分组 b,最后一组在必要时需要填充,一次处理 b 位的明文,每次使用相同的密钥加密,如图3.46所示。由于任意 b 位的明文,只有唯一的密文与之对应,就像密码本一样可以查到对应的密文,因此称为电子密码本模式。ECB对每组进行加密,加密后将各组密文合并成密文消息。在图3.46中,明文被分割成大小为 b 位的一串分组,记为 $P_1,P_2,\cdots,P_N$,对应的密文为 $C_1,C_2,\cdots,C_N$。

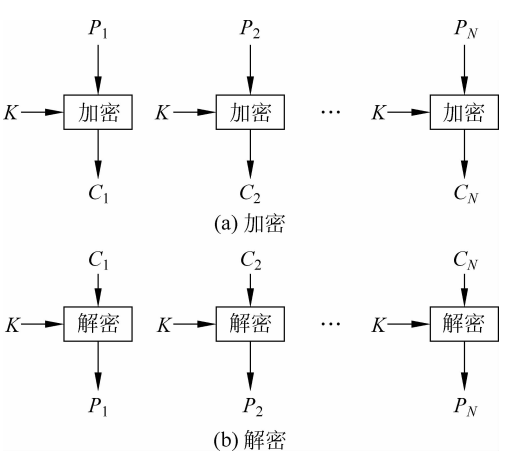

图3.46 电子密码本模式

在 ECB 模式下，每一个分组依次独立加密，产生独立的密文组，每一分组的加密结果均不受其他分组的影响。因此使用此种方式，可以利用并行处理来加速加密运算或解密运算，并且在传输时任意一个分组发生错误，不会影响其他分组，这是该模式的一个优点。但是，相同的明文组将产生相同的密文组，这样就会泄露明文的数据模式。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。但如果同一明文分组在消息中反复出现，产生的密文分组就会相同，因此，用于长消息时可能不够安全。如果消息有固定的结构，密码分析者就有可能会利用这种规律。例如，如果已知消息总是以某个事先规定的字段开始，那么分析者就有可能会得到许多明文-密文对。如果消息有重复的成分，而重复的周期是 b 位的倍数，那么这些成分都有可能会被密码分析者识别出来。因此 ECB 模式特别适合短数据（如加密密钥）。

3.10.2　密码分组链接模式

为了克服 ECB 的缺陷，人们希望设计一种方案使同一明文分组重复出现时产生的密文分组不同。一种简单的方案就是使用密码分组链接（Cipher Block Chaining，CBC）模式，如图 3.47 所示。这种模式和 ECB 模式一样，也要将明文分成 b 位的一串分组，最后一组不足 b 位要进行填充。但是 CBC 将这些分组链接在一起进行加密操作，加密输入是当前明文分组和前一密文分组的异或，它们形成一条链，每次加密使用相同的密钥，每个明文分组的加密函数输入与明文分组之间不再有固定的关系，所以明文分组的数据模式不会在密文中暴露。

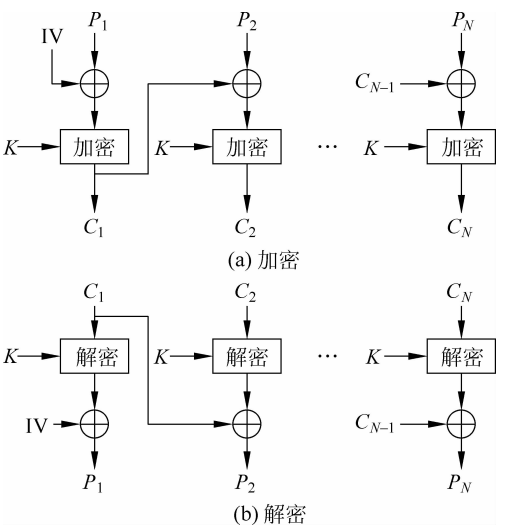

图 3.47　密码分组链接模式

在加密时，最开始一个分组先和一个初始向量（Initialization Vector，IV）进行异或，然后再用密钥加密，每一个分组的加密结果均会受到前面所有分组的影响，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。对每一个分组加密可以表示为：

$$C_i = E_K(P_i \oplus C_{i-1})$$

$$C_{-1} = \mathrm{IV}$$

解密时，将第一块密文解密结果与 IV 异或可恢复第一块明文。其他的每一个密文分组被解密后，再与前一个密文分组异或来产生出明文分组，即：

$$D_k[C_i] \oplus C_{i-1} = D_k[E_K[P_i \oplus C_{i-1}]] \oplus C_{i-1} = P_i \oplus C_{i-1} \oplus C_{i-1} = P_i$$

由于 CBC 模式的链接机制，可以避免像 ECB 模式下的那种明文数据模式的泄露。并且它对加密大于 b 位的明文非常合适。CBC 模式除了能够获得保密性外，还能用于认证，可以识别攻击者在密文传输中是否做了数据篡改，比如分组的重放、插入和删除等。但 CBC 模式同时也会导致错误传播，密文传输中任何一组发生错误不仅会影响该分组的正确解密，也会影响其下一分组的正确解密。该加密模式的另一个缺点是不能实时解密，也就是

说,必须等到每个 b 位都被接收到之后才能开始加密,否则就不能得到正确的结果。

IV 对于收发双方都应是已知的。为提高安全性,IV 应像密钥一样被保护。保护 IV 的原因如下:如果攻击者能欺骗接收方使用不同的 IV 值,攻击者就能够在第一个明文分组中改变某些选定的数据位,这是因为:

$$C_1 = E_K(\mathrm{IV} \oplus P_i)$$

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

用 $X(i)$表示分组 b 位 X 的第 i 位,那么 $P_1(i)=\mathrm{IV}(i)\oplus D_k[C_1](i)$,由异或的性质,有:

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

其中撇号表示取反。上式意味着如果攻击者篡改 IV 中的某些位,则接收方收到的 P_1 中相应的位也会发生变化。

3.10.3 密码反馈模式

如果需要将加密的明文必须按照一个字节或者一位进行处理,就可以采用密码反馈(Cipher Feed Back,CFB)模式或者输出反馈(Output Feed Back,OFB)模式。这两种模式实际上是将分组密码转换为流密码,流密码不需要明文长度是分组长度的整数倍,且可以实时操作。比较适合多媒体数据加密,每一个字节(或者一位)加密后可以立即发送,接送方也可以立即解密。

图 3.48 是密码反馈模式,假设它的输出是 s 位,s 位的大小可以是 1 位、8 位、64 位或者其他大小,表示为 CFB-1、CFB-8 和 CFB-64 等。因此密码反馈模式可以将分组密码转换成流密码。在加密时,用密钥 K 加密大小 b 位移位寄存器中的数据(其大小是该密码算法加密的明文分组长度。例如,若使用 DES 加密,则大小为 64 位;若使用 AES 加密,大小为 128 位)。移位寄存器的初始值为某个初始向量 IV。加密后输出的最左边 s 位与明文的 P_1(P_1 与 s 的大小相同)异或,产生出第一个密文单元 C_1,并将 C_1 单元传输出去。然后将移位寄存器中的内容左移 s 位,并将 C_1 送入移位寄存器的最右边 s 位。就这样持续进行,直到明文的所有单元都被加密为止。

在解密时,除了将收到的密文单元与加密函数的输出进行异或以产生明文单元外,其他与加密采用相同的方案。注意这里使用的是加密函数而不是解密函数。如假设 $S_s(X)$表示 X 的最左边 s 位,则:

$$C_1 = P_1 \oplus S_s(E(K,\mathrm{IV}))$$

所以 $P_1=C_1 \oplus S_s(E(K,\mathrm{IV}))$。

下面以 DES 为例,使用 CFB-8 工作模式说明加密过程。

(1) 加密:加密函数的输入是一个 64 位的移位寄存器,产生初始向量 IV。

(2) 对移位寄存器 64 位的数据用密钥进行加密,然后取加密数据最左边的 8 位与输入的明文最初的 8 位进行异或操作,得到的值作为 8 位密文单元。

(3) 这 8 比特密文被移至位寄存器的最右端,而其他位则向左移动 8 位,最左端 8 比特丢弃。

(4) 继续加密,与第 2 段明文输入异或,如此重复直到所有明文单元都完成加密。

有一点必须注意,这里的明文单元 P_i 和密文单元 C_i 与电子密码本模式和密码分组链接模式中的含义不一样。我们以 DES 为例说明这个问题,DES 中分组长度是 64 位(在本书中我们用 b 位表示)。在电子密码本模式和密码分组链接模式中,将明文分割成一个 64 位的分组,即 P_i 的大小,加密后输出一个 64 位的分组,即 C_i 的大小。在密码反馈模式中,移

位寄存器的大小是 64 位，加密后选取其中的一部分（大小可以是 1 位、8 位或者 64 位），假设我们选取 8 位，再与相同大小的明文单元 P_i 异或，输出相同大小的密文单元 C_i，显然这时 P_i 和 C_i 的大小是 8 位，而不是 64 位。

显然密码反馈模式具有流密码的优点，也拥有 CBC 模式的优点。但是它也拥有 CBC 模式的缺点，即也会导致错误传播。明文的一个错误会影响所有后面的密文以及在解密过程中的逆。另外密码反馈模式会降低数据加密速度。由于无论每次输出多少位，都需要事先用密钥 K 加密一次，再与相等的明文位异或，所以即使一次输出为 1 位，也要经过相同的过程，这就降低了加密速度。如果一次输出密文单元与移位寄存器相同的大小，那么密码反馈模式等同于密码分组链接模式。

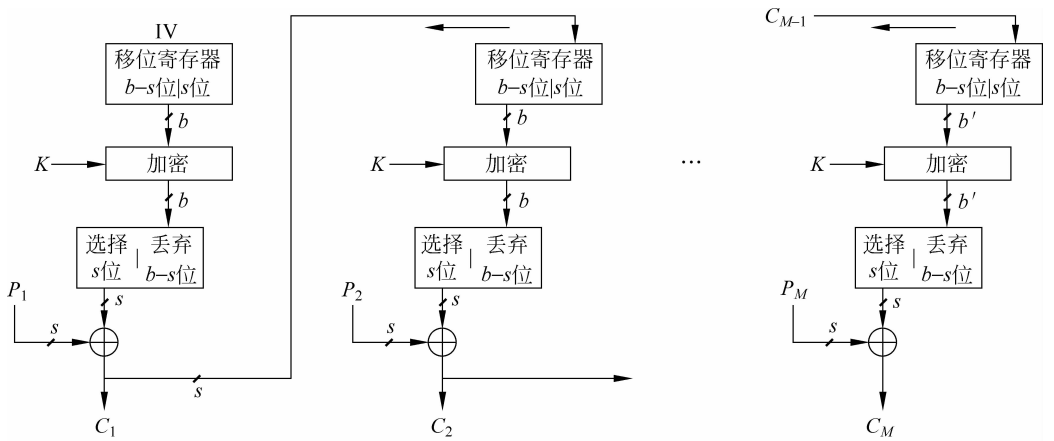

图 3.48　密码反馈模式

3.10.4　输出反馈模式

类似于密码反馈模式，不同之处在于输出反馈模式（OFB）是将加密算法的输出反馈到移位寄存器，而密码反馈模式是将密文单元反馈到移位寄存器，如图 3.49 所示。

与CFB相比,OFB模式的优点是传输过程中的位错误不会被传播。但是相对于其他模式,因为数据之间相关性小,这种加密模式是相对不安全的。如这种模式难于检测密文是否被篡改。如果在密文中某位取反,那么在恢复后的明文中相应位也取反。因此攻击者有可能通过对数据部分和校验部分同时进行篡改,导致纠错码无法检测。所以在应用的时候除非特别需要,一般不提倡应用OFB模式。

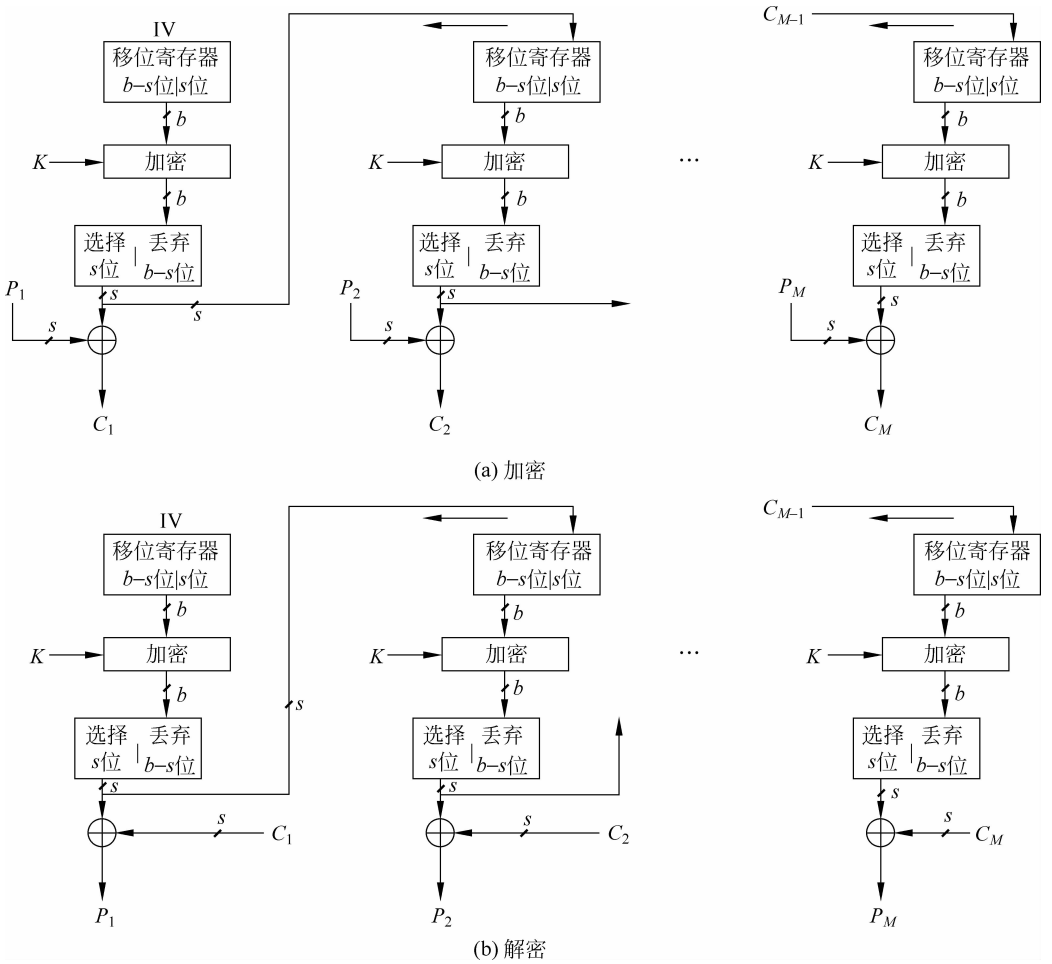

图 3.49 输出反馈模式

3.10.5 计数器模式

计数器(Counter,CTR)采用与明文分组相同的长度,但加密不同的明文组,计数器对应的值不同。计数器首先被初始化为一个值,然后随着消息块的增加,计数器的值依次递增1。计数器加1后与明文分组异或得到密文分组。解密是使用相同的计数器值序列,用加密后的计数器的值与密文分组异或来恢复明文,如图3.50所示。

计数器模式比较适合对实时性和速度要求比较高的场合,它具有以下优点。

(1) 处理效率:由于下一块数据不需要前一块数据的运算结果,所以CTR能够并行加密(解密)。这使其吞吐量可以大大提高。

（2）预处理：基本加密算法的执行不依赖明文或者密文的输入，因此可以事先处理。这样可以极大地提高吞吐量。

（3）随机访问：由于对某一密文分组的处理与其他密文分组无关，因此可以随机地对任意一个密文分组进行解密处理。

（4）简单性：计数器模式只要求实现加密算法，而不要求解密算法，加密阶段和解密阶段都使用相同的加密算法。

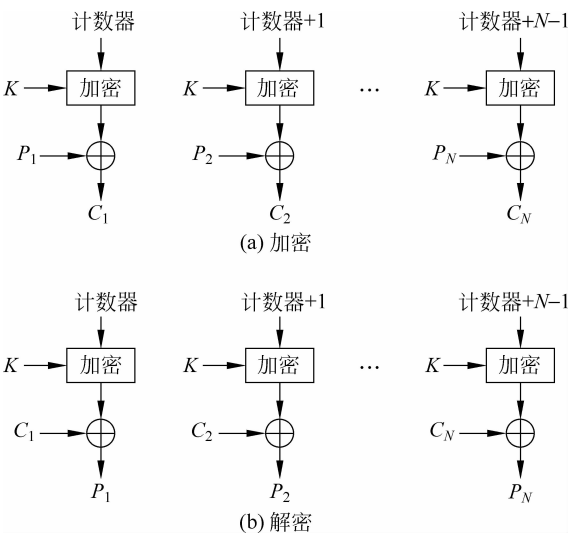

图 3.50　计数器模式

3.11　随机数的产生

随机数在信息安全中起着非常重要的作用。在很多场合需要用到随机数，如对称密码体制中密钥、非对称密码体制公钥以及用于认证的临时交互号等均使用了随机数。安全的随机数应该满足随机性和不可预测性。随机性有如下两个评价标准。

- 分布一致性：随机数分布是一致的，即每个数出现频率大约相等。
- 独立性：数据序列中的任何数不能由其他数导出。

一般来说，数据序列是否满足均匀分布可通过检测来判断，而是否满足独立性则是无法判断的。但却有很多检测方法能证明数据序列不满足独立性。如果对数据序列进行足够多次数的检测后都不能证明不满足独立性，就可比较有把握地相信该数据序列满足独立性。对产生的数据序列不仅要求其具有随机性而且要求其具有不可预测性，即根据数据序列的一部分不能推导出之前的部分序列，也不能预测后续序列。

一般说来，产生随机数的方式有两种：一是通过一个确定性的算法，由数字电路或是软件实现，把一个初值扩展成一个长的序列；二是选取真实世界的自然随机源，比如热噪声等。由前一种方法产生的序列通常被称为伪随机序列，而后者通常被称为真随机序列。

3.11.1 真随机数发生器

对于伪随机序列来说,因为使用的是确定的算法,有一定的规律所循,所以只要具备足够的计算能力,总能进行预测。能否产生真正的随机数,长期以来,这个问题一直都处于激烈的争论之中。但对于工程应用来说,只要产生的序列具有随机统计特性,并且不可再现,就可以被称为真随机序列。设计一个真随机数发生器包括两步:第1步是获取真随机源;第2步是利用真随机源依照特定的数学方法获得真随机数。真随机源广泛存在于现实世界中,比如计算机网络中IP包到达的时间、随机噪音、计算机当前的秒级时钟、键盘反应时间、热噪声、操作系统的进程信息、光量子的偏振等。获取方法可以通过调用系统函数或者硬件电路来实现。利用真随机源产生真随机数的方法有很多,一种最简单的方法是直接利用真随机源的奇偶特性来产生0和1序列。为了增加序列的随机性,往往还对产生的0和1序列进行一系列的变换,比如归一化、非线性映射、移位、加密等。图3.51是通过提取电路中的热噪声来产生随机数的方法。该方法将提取的热噪声进行放大,输入到一个比较器,与固定的参考电压进行比较,从而确定输出0、1序列。图3.52是基于自激振荡器频率不稳定性的真随机数发生器的原理示意图。

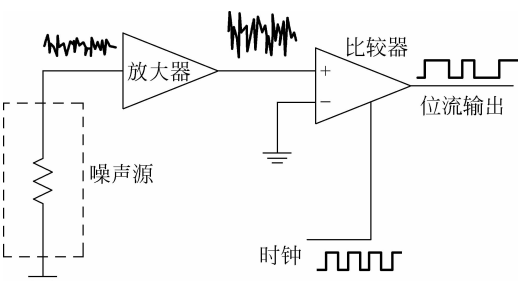

图3.51 利用热噪声的随机数发生器

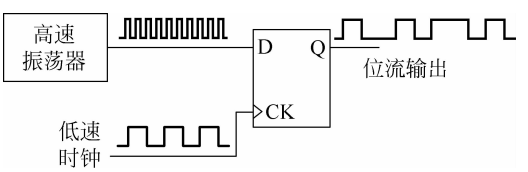

图3.52 振荡采样随机数发生器

3.11.2 伪随机数发生器

在理想情况下,密码算法和协议中需要的秘密数应该用一个真随机生成器来产生。然而,在大多数实际环境中,真随机比特的生成是一个效率较低的过程,并且安全地存储和传送一个大随机数也是不切实际的。因此,在信息安全中常常用伪随机发生器产生伪随机数。这些随机数尽管不是真正的随机,但能够通过许多随机测试。

1. 线性同余法

线性同余(Linear Congruential)法是一种广泛使用的伪随机数产生方法。其随机数序列$\{X_n\}$可通过下面的式子迭代获得:

$$X_{n+1} = (aX_n + c) \bmod m, \quad n \geqslant 0$$

其中:

- X_0称为种子或者初始值,并且$0 \leqslant X_0 < m$;
- 常数m称为模数,$m > 0$;
- 常数a称为乘子,$0 \leqslant a < m$;
- 常数c称为增量,$0 \leqslant c < m$。

当 $c=0$ 时，该算法称为乘同余法；当 $c\neq 0$ 时，该算法称为混合线性同余法。

为了得到 [0,1] 区间上分布的随机数，可以令：

$$R_n = \frac{X_n}{m}$$

其中 R_n 为满足要求的随机数。

a、c、m 的取值是产生高质量的随机数的关键。如当 $a=7$，$c=0$，$m=32$，$X_0=1$ 时，生成的数为{1,7,17,23,1,7,…}，该数列的周期为 4，而模为 32，结果不能令人满意。当随机数周期达到模时，则其周期称为满周期，也就是理论上的最大周期。所以我们在用同余算法生成随机数的时候，要尽可能地使周期达到满周期。合理地选择 a、c、m、X_0，可以使重复的周期充分长。如果满足下面条件，随机数发生器可以达到满周期。

(1) c 和 m 互素。

(2) 对 m 的任何一个素因子 p，$a\equiv 1 \bmod p$。

(3) 如果 4 是 m 的因子，则 $a\equiv 1 \bmod 4$。

对于上面的条件，这些参数一般取下面的值：$m=2^L$，其中 L 是计算机中存放一个整数值的二进制位数(称为整数的尾数字长)。这种方法有两个优点：一是 m 越大，随机数的周期就可能越大；二是算法上利用计算机的整数溢出原理，可简化计算。其他的参数取值为：

$$a = 4\alpha + 1$$
$$c = 2\beta + 1$$

其中 α 和 β 为任意正整数。

线性同余法的强度取决于乘子和模数的选择。但是除了初值 x_0 的选取具有随机性外，算法本身并不具有随机性，因为选定 x_0 后，以后的数就被确定性地产生了。这个性质可用于对该算法的密码分析，如果攻击者知道正在使用线性同余算法并知道算法的参数，则一旦获得数列的一个数，就可得到以后的所有数。甚至攻击者如果只知道正在使用线性同余算法以及产生的数列中极少一部分，就足以确定出算法的参数。假定攻击者能确定 x_0、x_1、x_2 和 x_3，就可通过以下方程组：

$$X_1 = (aX_0 + c) \bmod m$$
$$X_2 = (aX_1 + c) \bmod m$$
$$X_3 = (aX_2 + c) \bmod m$$

解出 a、c 和 m。

改进的方法是利用系统时钟修改随机数数列。一种方法是每当产生 N 个数后，就利用当前的时钟值模 m 后作为新种子。另一种方法是直接将当前的时钟值加到每个随机数上再对 m 取模。

2. 非线性同余法

非线性同余(Nonlinear Congruential)法的随机数序列{X_n}可通过下面的式子迭代获得：

$$X_{n+1} = f(X_n) \bmod m, \quad n \geqslant 0$$
$$R_n = \frac{X_n}{m}$$

其中,$X_n \in Z_m = \{0,1,2,\cdots,m-1\}$,$f$ 是 Z_m 上的一个整数函数。如果 $f(x)=ax+c$,则变为线性同余。在非线性同余法中的 f 通常是 Z_m 上的一个排列多项式。

下面是几种典型的非线性同余发生器,它们的区别主要是 f 函数不同。

1) 逆同余发生器

$$X_{n+1} = (aX'_n + b) \bmod m, \quad n \geqslant 0$$

$$R_n = \frac{X_n}{m}$$

其中,X'_n 是 X_n 关于模 m 的乘法逆预元。

2) 二次同余发生器

$$X_{n+1} = (aX_n^2 + bX_n + c) \bmod m, \quad n \geqslant 0$$

$$R_n = \frac{X_n}{m}$$

其中,a、b、c 为非负整数,且 $a \neq 0$。

3) BBS 发生器

BBS(Blum Blum Shub)发生器是由 Lenore Blum、Manuel Blum 和 Michae Shub 于 1986 年共同提出的一种随机数发生器,其递推公式为:

$$X_{n+1} = X_n^2 \bmod m, \quad n \geqslant 0$$

$$R_n = \frac{X_n}{m}$$

其中,$m=pq$,p 和 q 是两个大素数,且 $p \equiv q \equiv 3 \pmod 4$,选择随机数 s 与 m 互素,计算初始值 $X_0 = s^2 \bmod m$。

BBS 发生器最大的特点是可以直接计算任意一个 X_n 的值:

$$X_n = (X_0^{2^n \bmod (p-1)(q-1)}) \bmod m$$

BBS 发生器的安全性很好,并且通过了几乎所有的理论检验,但其运行速度较慢。

4) 幂同余发生器

幂同余发生器是 BBS 发生器的推广,其迭代公式为:

$$X_{n+1} = X_n^d \bmod m, \quad n \geqslant 0$$

$$R_n = \frac{X_n}{m}$$

其中,d 和 m 为正整数。一个重要的特殊情形是 $m=pq$,且 p 和 q 为两个大素数。

5) 指数同余发生器

指数同余发生器的迭代公式为:

$$X_{n+1} = g^{X_n} \bmod m, \quad n \geqslant 0$$

$$R_n = \frac{X_n}{m}$$

其中,g 和 m 为正整数。一个重要的特殊情形是 m 为一个大素数。

3. 混沌随机数发生器

在混沌区的数据具有两个显著的特性:迭代不重复性和初值敏感性。如果选定一个迭代方程和适当的系数,方程将进行无限制不循环地迭代。下式是混合光学双稳模型的迭代

方程：

$$X_{n+1} = A\sin^2(X_n - X_B)$$

A 和 X_B 是方程的系数，当 $A=4$，$X_B=2.5$ 时方程处于混沌状态。根据该方程生成混沌序列$\{X_i\}$，可以获得不同 0 和 1 序列 S_i。

$$S_i = \begin{cases} 1, & \text{如果 } X_i \geqslant \dfrac{2}{3}A \\ 0, & \text{其他情况} \end{cases}$$

4. 用密码学的方法产生随机数

单向函数可以用于产生伪随机数，方法是首先选取随机种子 s，然后再将函数应用于序列 s，$s+1$，$s+2$，…，进而输出序列 $f(s)$，$f(s+1)$，$f(s+2)$，…。该单向函数可以是 hash 函数（如 SHA-1），或者是对称分组密码（如 DES）。

1）循环加密

循环加密是一种非常简单的随机数产生方法，如图 3.53 所示。它用一个种子密钥循环加密计数器，从而产生一个随机序列。计数器的周期为 N。如要产生 56 位的 DES 密钥，可以用一个周期为 2^{56} 的计数器，每产生一个密钥，计数器的值增加 1，那么这种方法产生的伪随机序列是全周期的，所有的输出序列都是由不同的计数值而来，所以它们互不相同。由于种子密钥是保密的，所以由生成的随机数不能推出后续的随机数。有时为了增加强度，可以用一个全周期的伪随机数发生器来代替简单的计数器。

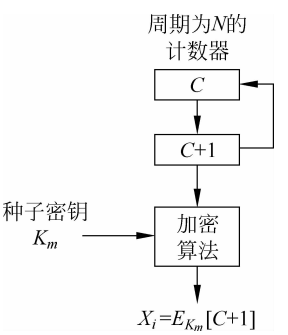

图 3.53　由计数器生成伪随机数

2）ANSI X9.17 随机数生成器

ANSI X9.17 基于 3DES 随机数生成标准，如图 3.54 所示。

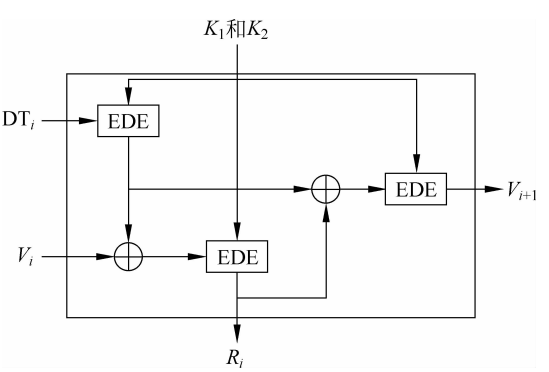

图 3.54　ANSI X9.17 伪随机数生成器

图 3.54 中的符号含义如下。

- EDE：表示用两个密码加密的三重 DES，即加密-解密-加密。
- DT_i：第 i 轮的初始日期和时间。
- V_i：第 i 轮的初始种子值，V_{i+1} 表示第 i 轮产生的新种子，并作为第 $i+1$ 轮的种子。

- R_i：第 i 轮的所产生的伪随机数。
- K_1 和 K_2：DES所使用的密钥。

ANSI X9.17伪随机数生成器采用的输入是 DT_i 和 V_i。DT_i 是一个64位数，代表当前的日期和时间，每产生一个伪随机数它均要改变。V_i 是种子值，可以是任意的64位数，并在随机数生成过程中被更新。从图3.54中很容易得到随机数序列和下一轮的种子。

$$R_i = \mathrm{EDE}_{[K_1,K_2]}(V_i \oplus \mathrm{EDE}_{[K_1,K_2]}(DT_i))$$

$$V_{i+1} = \mathrm{EDE}_{[K_1,K_2]}(R_i \oplus \mathrm{EDE}_{[K_1,K_2]}(DT_i))$$

其中 $\mathrm{EDE}_{[K_1,K_2]}(X)$ 表示使用两个密钥 K_1 和 K_2 的三重DES加密 X。

3.12 对称密码的密钥分配

3.12.1 密钥分配基本方法

对称密码体制要求双方共享一个共同的密钥，并且为防止攻击者得到密钥，还必须时常更新密钥。通常安全系统存在的问题是出在密钥分配(Key Distribution)上。如何安全地分发这个密钥是对称密码体制的核心问题。在两个用户(主机、进程、应用程序) A和B之间分配密钥的方法有以下几种。

(1) 密钥由A选取，并通过物理手段交给B。

(2) 密钥由第三方选取，并由第三方通过物理手段交给A和B。

(3) 如果A和B事先已有一密钥，则其中一方选取新密钥后，用已有的密钥加密新密钥并发送给另一方。

(4) 如果A和B与可信的第三方C分别有一保密通道，则C为A和B选取密钥后，分别在两个保密信道上发送给A和B。

前两种方法称为人工发送。在通信网中，若只有个别用户想进行保密通信，密钥的人工发送还是可行的。然而如果所有用户都要求支持加密服务，则任意一对希望通信的用户都必须有一共享密钥。如果有 n 个用户，则密钥数目为 $n(n-1)/2$。因此当 n 很大时，密钥分配的代价非常大，如当有1000个结点时，需要多达500 000个密钥，如果加密在应用层，则每个用户或者进程都需要一个密钥，那么密钥分配任务则更重。

对于第(3)种方法，攻击者一旦获得一个密钥就可获取以后所有的密钥，而且用这种方法为所有用户分配初始密钥时，代价仍然很大。

第(4)种方法比较常用，其中的第三方通常是一个负责为用户分配密钥的密钥分配中心(Key Distribution Center，KDC)。这时每一用户必须和密钥分配中心有一个共享密钥，称为主密钥(Master Key)。通过主密钥分配给一对用户的密钥称为会话密钥(Session Key)，用于这一对用户之间的保密通信。通信完成后，会话密钥即被销毁。如上所述，如果用户数为 n，则会话密钥数为 $n(n-1)/2$。但主密钥数却只需 n 个，所以主密钥可通过物理手段发送。

一个完整的密钥分配方案需要完成两个功能：一是将密钥分发给双方；二是双方互相认证，确保密钥一定只给了双方。图3.55是一个典型的密钥分配过程。由密钥分配中心(KDC)产生会话钥，然后分发给A和B。图3.55中字符的含义如下。

- K_a 和 K_b 分别是 A 和 B 各自拥有与 KDC 共享的主密钥。
- K_s 是分配给 A 和 B 的一次性会话钥。
- N_1 和 N_2 是临时交互号(Nonces)，可以是时间戳、计数器或随机数，主要用于防止重放攻击。
- ID_A 和 ID_B 分别是 A 和 B 的身份标识(例如 A 和 B 的网络地址)。
- $f(N_2)$是对 N_2 的某种变换(例如将 N_2 加 1)函数，目的是认证。
- ‖ 表示连接符，如 $ID_A \| ID_B \| N_1$ 表示同时传送了 ID_A、ID_B 和 N_1。

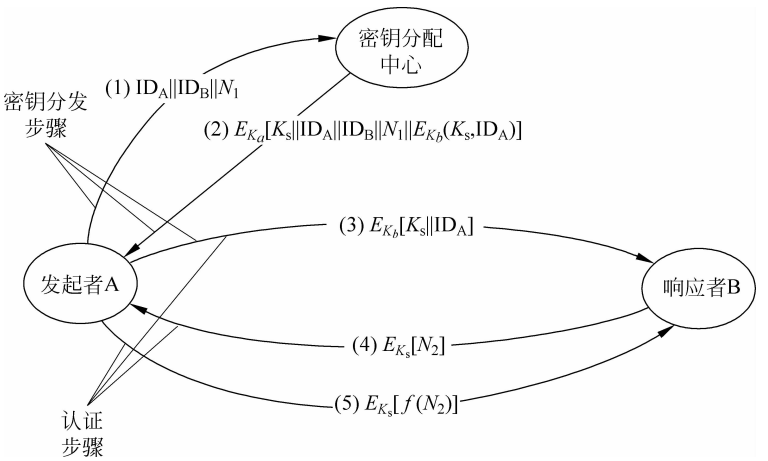

图 3.55　密钥分配过程

A 和 B 之间的会话密钥是通过以下几步来完成的。

(1) A 向 KDC 发出会话密钥请求。表示请求的消息由两个数据项组成，第 1 项是 A 和 B 的身份 ID_A 和 ID_B，第 2 项是这一步骤的唯一识别符 N_1，称为临时交互号。每次请求所用的都应不同，且为防止假冒，应使攻击者难以猜测。因此用随机数作为这个识别符最为合适。使用临时交互号的目的是防止重放攻击。

(2) KDC 为 A 的请求发出应答。应答是用 A 和 KDC 的共享的主密钥 K_a 加密，因此只有 A 才能成功地对这一消息解密，并且 A 可相信这一消息的确是由 KDC 发出的。消息中包括给 A 的两项内容：

- 一次性会话密钥 K_s。
- A 在第(1)步中发出的请求，包括一次性随机数 N_1，目的是使 A 将收到的应答与发出的请求相比较，看是否匹配。这样 A 能验证自己发出的请求在被 KDC 收到之前，是否被他人篡改。而且 A 可以确定收到的这个消息是否是对它请求的响应，而不是对以前消息的重放。

此外，该消息中还有给 B 的两项内容：

- 一次性会活密钥 K_s。
- A 的身份 ID_A。

这两项由 B 和 KDC 的共享的主密钥 K_b 加密，将由 A 转发给 B，以建立 A 和 B 之间的连接，并用于向 B 证明 A 的身份。

(3) A 存储会话密钥备用，并向 B 转发 $E_{K_b}[K_s \| ID_A]$。因为转发的是由 K_b 加密后的

密文,所以转发过程不会被窃听。B收到后,可得会话密钥 K_s,并且可知另一方是A,还从 K_b 知道 K_s 的确来自KDC。

完成这一步后,会话密钥就被安全地分配给了A和B。下面需要在A和B之间进行认证。

(4) B用会话密钥 K_s 加密另一个临时交互号 N_2,并将加密结果发送给A。

(5) A以 $f(N_2)$ 作为对B的应答,其中 f 是对 N_2 进行某种变换(例如加1)的函数,并将应答用会话密钥加密后发送给B。注意一点,如果不将 N_2 进行某种变换,直接以 N_2 作为应答,则会存在重放攻击。

这两步可使B相信第(3)步收到的消息不是一个重放。并且双方进行了认证。

第(4)和第(5)步是典型挑战/应答(Challenge/Response)认证方式。假设A期望从B获得一个新消息,首先发给B一个临时值(Challenge),并要求后续从B收到的消息(Response)中正确地包含这个临时值。

3.12.2 密钥的分层控制

在网络中,如果用户数目非常多而且分布的地域非常广,一个KDC就无法承担为用户分配密钥的重任。问题的解决方法是使用多个KDC的分层结构。例如,在每个小范围(如一个LAN或一个建筑物)内,都建立一个本地KDC。同一范围的用户在进行保密通信时,由本地KDC为他们分配密钥。如果两个不同范围的用户想获得共享密钥,则可通过各自的本地KDC,而两个本地KDC的沟通又需经过一个全局KDC。这样就建立了两层KDC。类似地,根据网络中用户的数目及分布的地域,可建立三层或多层KDC。

分层结构可减少主密钥的分布,因为大多数主密钥是在本地KDC和本地用户之间共享。此外,如果一个本地KDC出错或者被攻击,则危害只限制在一个局部区域,而不会影响全局。

3.12.3 会话密钥的有效期

会话密钥更换得越频繁,系统的安全性就越高。因为攻击者即使获得一个会话密钥,也只能获得很少的密文。但另一方面,会话密钥更换得太频繁,又将延迟用户之间的交换,同时还会造成网络负担。所以在决定会话密钥的有效期时,应权衡这两个方面。

对于面向连接的协议,在连接未建立前或断开时,会话密钥的有效期可以很长。而每次建立连接时,都应使用新的会话密钥。如果逻辑连接的时间很长,则应定期更换会话密钥。

对于无连接协议(如面向交易的协议),无法明确地决定更换密钥的频率。为安全起见,用户每进行一次交换,都用新的会话密钥。然而这又失去了无连接协议的主要优势,如延时了交易时间,每个交易都希望用最少的费用和最短的延迟。比较好的方案是在某一个固定周期内或者交易一定量内使用同一会话密钥。

3.12.4 无中心的密钥分配

用密钥分配中心(第三方)为用户分配密钥时,要求所有用户都信任KDC,同时还要求对KDC加以保护。如果密钥的分配没有这个中心,则不必有以上两个要求。在下面的分配方案中,每个用户事先和其他用户之间存在一个主密钥,然后使用这些主密钥产生会话钥。如果网络中有 n 个用户,则需有 $n(n-1)/2$ 个主密钥。当 n 很大时,整个网络中的主密钥很多,但每个结点最多只保存 $n-1$ 个主密钥,用这些主密钥可以产生很多会话钥。

图 3.56 是一个无中心的密钥分配过程。

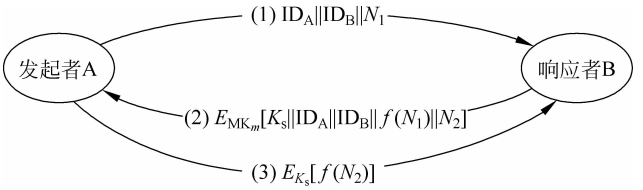

图 3.56　无中心的密钥分配过程

两个用户 A 和 B 建立会话密钥需经以下 3 个步骤。

(1) A 向 B 发出建立会话密钥的请求，包括 A 和 B 的身份标识和临时交互号 N_1。

(2) B 用与 A 共享的主密钥 MK_m 对应答的消息加密，并发送给 A。应答的消息中包括选取的会话密钥 K_s、B 的身份标识、$f(N_1)$和另一个临时交互号 N_2。

(3) A 使用新建立的会话密钥 K_s 对 $f(N_2)$加密后返回给 B。

以上过程也完成了两件事，一是在 A 和 B 之间分配了会话钥；二是 A 和 B 进行了相互认证。在过程 1 和过程 2 中，挑战(Challenge)是临时交互号 N_1，应答(Response)是 $f(N_1)$，完成了 A 对 B 的认证。在过程(2)和过程(3)中，挑战(Challenge)是临时交互号 N_2，应答(Response)是 $f(N_2)$，这样就完成了 B 对 A 的认证。

3.13　关 键 术 语

密码学(Cryptology)
密码编码学(Cryptography)
密码分析学(Cryptanalysis)
密码分析者(Cryptanalyst)
明文(Plaintext)
加密(Encryption)
密文(Ciphertext)
解密(Decryption)
密码(Cipher)
密码体制(Cryptosystem)
密钥(Key)
分组密码(Block Ciphers)
流密码(Stream Ciphers)
对称加密(Symmetric Encryption)
穷举攻击(Brute Force Search)
唯密文攻击(Ciphertext-Only Attack)
已知明文攻击(Known-Plaintext Attack)
选择明文攻击(Chosen-Plaintext Attack)

选择密文攻击(Chosen-Ciphertext Attack)
选择文本攻击(Chosen Text Attack)
绝对安全(Unconditional Security)
计算上安全(Computational Security)
代换(Substitution)
置换(Permutation)
单字母代换密码(Monogram Substitution Cipher)
多字母代换密码(Polygram Substitution Cipher)
单表代换密码(Monoalphabetic Substitution Cipher)
多表代换密码(Polyalphabetic Substitution Cipher)
代换密码(Substitution Cipher)
仿射密码(Affine Cipher)
置换密码(Permutation Cipher)
扩散(Diffusion)
混淆(Confusion)
数据加密标准(Data Encryption Standard,DES)
高级加密标准(Advanced Encryption Standards,AES)
同步流密码(Synchronous Stream Cipher)
自同步流密码(Self-Synchronous Stream Cipher)
线性同余(Linear Congruential)
密钥分配(Key Distribution)
主密钥(Master Key)
会话密钥(Session Key)
密钥分配中心(Key Distribution Center,KDC)

3.14 习 题 3

3.1 下式是仿射密码的加密变换:

$$c = (3m + 5) \bmod 26$$

试求:
(1) 该密码的密钥空间是多少?
(2) 求出消息 hello 对应的密文。
(3) 写出它的解密变换。
(4) 试对密文进行解密。

3.2 用 Playfair 密码加密下面的消息:

ciphers using substitutions or transpositions are not secure because of language

characteristics。密钥为 the playfair cipher was invented by Charles Wheatstone。

3.3　假设密钥为 encryption，用维吉尼亚密码加密消息 symmetric schemes require both parties to share a common secret key。

3.4　Hill 密码不能抵抗已知明文攻击，如果有足够多的明文和密文对，就能破解 Hill 密码。

(1) 攻击者至少有多少个不同的明文-密文对才能攻破该密码？

(2) 描述这种攻击方案。

3.5　用 Hill 密码加密消息 hill，密钥为：

$$k=\begin{pmatrix}11 & 8\\ 3 & 7\end{pmatrix}$$

并写出从密文恢复明文的解密过程。

3.6　用一次一密加密消息 0101101010110011110001010101010101101111000101000 1，选定的密钥是 1001010101111010110100010100000111110010010101001 0，试写出密文。

3.7　使用 DES 加密，假设明文和密钥都为 $(0123456789ABCDEF)_{16}=(00000001\ 00100011\ 01000101\ 01100111\ 10001001\ 10101011\ 11001101\ 11101111)_2$：

(1) 推导出第 1 轮的子密钥 K_1。

(2) 写出 R_0 和 L_0。

(3) 扩展 R_0 并计算 $E(R_0)\oplus K_1$。

(4) 将第(3)问的结果，输入到 8 个 S 盒，求出加密函数 F。

(5) 推导出 R_1 和 L_1。

3.8　在 $GF(2^8)$上{01}的逆是什么？并验证其在 S 盒中的输入。

3.9　假设 AES 的 State 矩阵的某一列分别是 $s_0=\{87\}$，$s_1=\{6E\}$，$s_2=\{46\}$，$s_3=\{A6\}$。经过列混淆变换后，$s_1=\{6E\}$映射为 $s_1'=\{37\}$，试验证这一结果。

3.10　采用 AES 加密，密钥为 2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C，明文为 32 43 F6 AD 88 5A 30 8D 3131 98 A2 E0 37 07 34。

(1) 写出最初的 State 的值。

(2) 写出密钥扩展数组中的前 8 个字节。

(3) 写出初始轮密钥加后 State 的值。

(4) 写出字节代换后 State 的值。

(5) 写出行移位后的 State 的值。

(6) 写出列混淆后 State 的值。

3.11　习题 3.10 的明文和密钥不变，采用 SMS4 加密。

(1) 求出第 1 轮的轮密钥 rk_0。

(2) 求第 1 轮加密后的明文输出是什么？

3.12　有一个四级线性移位寄存器的反馈函数为 $f(a_1,a_2,a_3,a_4)=a_1\oplus a_2$，其中初态为$(a_1,a_2,a_3,a_4)=(1000)$，求其输出序列的前 12 位。

3.13　假如使用 3 位(0～7)的 RC4，其操作是对 8 取模(而不是对 256 取模)，密钥是 326。

(1) 求初始化后 S 表的值。

(2) 计算第1个密钥字。

(3) 用上面生成的密钥加密明文100101。

3.14 在8位CFB模式中,如果在传输中一个密文字符发生错误,这个错误将被传送多远?

3.15 编写仿射密码的加密和解密程序。

3.16 写一个程序实现维吉尼亚(Vigenère)密码的加密和解密。

3.17 编程实现AES算法。

3.18 编程实现线性同余伪随机数生成算法。