本资源来自数缘社区

http://maths.utime.cn:81

欢迎来到数缘社区。本社区是一个高等数学及密码学的技术性论坛，由山东大学数学院研究生创办。在这里您可以尽情的遨游数学的海洋。作为站长，我诚挚的邀请您加入，希望大家能一起支持发展我们的论坛，充实每个版块。把您宝贵的资料与大家一起分享！

数学电子书库

每天都有来源于各类网站的与数学相关的新内容供大家浏览和下载，您既可以点击左键弹出网页在线阅读，又可以点右键选择下载。现在书库中藏书 1000 余本。如果本站没有您急需的电子书，可以发帖说明，我们有专人负责为您寻找您需要的电子书。

密码学论文库

国内首创信息安全专业的密码学论文库，主要收集欧密会（Eurocrypt）、美密会（Crypto）、亚密会（Asiacrypt）等国内外知名论文。现在论文库中收藏论文 4000 余篇（包括论文库版块 700 余篇、论坛顶部菜单“密码学会议论文集”3000 余篇）。如果本站没有您急需的密码学论文，可以发帖说明，我们有专人负责为您寻找您需要的论文。

提示：本站已经收集到 1981－2003 年欧密会、美密会全部论文以及 1997 年－2003 年五大会议全部论文（欧密会、美密会、亚密会、PKC、FSE）。

数学综合讨论区

论坛管理团队及部分会员来源于山东大学数学院七大专业（基础数学、应用数学、运筹学、控制论、计算数学、统计学、信息安全），在数学方面均为思维活跃、成绩优秀的研究生，相信会给您的数学学习带来很大的帮助。

密码学与网络安全

山东大学数学院的信息安全专业师资雄厚，前景广阔，具有密码理论、密码技术与网络安全技术三个研究方向。有一大批博士、硕士及本科生活跃于本论坛。本版块适合从事密码学或网络安全方面学习研究的朋友访问。

网络公式编辑器

数缘社区公式编辑器采用 Latex 语言，适用于任何支持图片格式的论坛或网页。在本论坛编辑好公式后，您可以将自动生成的公式图片的链接直接复制到你要发的帖子里以图片的形式发表。

如果您觉得本站对您的学习和成长有所帮助，请把它添加到您的收藏夹。如果您对本论坛有任何的意见或者建议，请来论坛留下您宝贵的意见。

附录 A：本站电子书库藏书目录

http://maths.utime.cn:81/bbs/dispbbs.asp?boardID=18&ID=2285

附录 B：版权问题

计算机科学组合学丛书

计算机密码学

—— 计算机网络中的数据保密与安全

（第3版）

卢开澄 编著

清华大学出版社

计算机科学组合学丛书

封面设计　卢开澄 傅瑞学

组合数学（第3版）
图论及其应用（第2版）
计算机算法导引 — 设计与分析
计算机密码学 — 计算机网络中的数据保密与安全（第3版）
单目标、多目标与整数规划
计算机纠错码
计算复杂性理论

ISBN 7-302-07536-0
9 787302 075363 >
定价：46.00元

计算机科学组合学丛书

计 算 机 密 码 学

——计算机网络中的数据保密与安全

（第 3 版）

卢开澄　编著

清华大学出版社
北　京

内容简介

在电子商务和电子政务的兴起和发展过程中，近代密码学扮演了十分活跃的角色。本书是在第2版的基础上，结合这几年密码学技术的发展改写而成。全书共13章，叙述了密码学基本概念、分组密码、公钥密码、大数运算、密码协议、密钥管理等，第3版比第2版增加了大数运算、数字签名、密钥管理、密码协议等内容，尤其对AES的加密标准及部分候选算法做了详细的介绍，并加强了与网络通信的保密安全相关的内容。

本书可作为计算机专业或其他专业关于"网络通信保密安全"相关课程的教材或参考书。

图书在版编目(CIP)数据

计算机密码学——计算机网络中的数据保密与安全/卢开澄编著. —3版. —北京：清华大学出版社，2003

(计算机科学组合学丛书)

ISBN 7-302-07536-0

Ⅰ.计… Ⅱ.卢… Ⅲ.计算机网络—密码术 Ⅳ.TP393.08

中国版本图书馆CIP数据核字(2003)第[redacted]号

出 版 者：清华大学出版社　　北京清华大学学研大厦

http://www.tup.com.cn　　100084

社 总 机：010-62770175　　010-62776969

组稿编辑：薛 慧 张 民

文稿编辑：付宇光 张 民

封面设计：卢开澄 傅瑞学

印 刷 者：北京昌平环球印刷厂

装 订 者：北京国马印刷厂

发 行 者：新华书店总店北京发行所

开 本：185×260 印张：31.75 字数：731千字

版 次：2003年12月第3版 2003年12月第1次印刷

书 号：ISBN 7-302-07536-0/TP·5548

印 数：1～5000

定 价：46.00元

计算机科学组合学丛书序

电子计算机的出现是20世纪的大事，它改变了我们这个世界的面貌。可以毫不夸张地说，它的影响遍及所有的角落，几乎无处不在。数学更不例外。严格地说，电子计算机本身就是近代数学的辉煌成就。将计算机与数学割裂开来，既不合理也不可能。组合学就是在计算机科学蓬勃发展的刺激下而崛起的，从而成为近若干年来最活跃的数学分支。它研究的问题有的可追溯到欧拉和哈密顿等18世纪的数学家，但它成为一新的分支还是近若干年的事。它从与计算机科学相结合中获得了广阔的发展空间，从而也为计算机科学奠定了理论基础。

什么是计算机科学呢？有的学者将它定义为研究算法的一门学科。研究算法无疑是计算机科学的重要领域，也是本丛书的核心内容，贯穿始终。组合学家在20世纪70年代初建立的算法复杂性"NP理论"，至今仍然令无数计算机科学工作者与数学工作者为之折腰。

计算机科学里的组合学内容十分广泛。本丛书涉及组合分析、图论、组合算法、近代密码学、组合优化、编码理论及算法复杂性等7部分。

组合分析是算法的理论基础。组合分析与组合算法犹如数学分析与计算数学，众所周知，前者是后者的理论根基。

图论原本是组合数学这个"家族"的主要成员，只因它已成长壮大，故自立门户独立出去。

算法复杂性的NP理论是近30年的一大成就。研究表明，对于一类叫做NPC类的困难问题，至今都不存在有效算法，但它们难度相当，只要其中任何一个找到多项式解法，则全体都获得解决；或证明它们根本不存在有效办法。不论是前者还是后者都还看不见露到海平面上的桅杆塔，它吸引了众多的有志之士。密码学是其中十分引人入胜的分支。如若设计好的密码，对它的破译等价于某一NPC类困难问题，无疑这样的密码将是牢不可破的。

在计算机网络深入普及的信息时代，信息本身就是时间，就是财富。信息的传输通道是脆弱的公共信道，信息存储于"不设防"的计算机系统中，如何保护信息的安全使之不被窃取及不至于被篡改或破坏，已成为当今普遍关注的重大问题。密码是有效而且可行的办法。在计算机网络的刺激下，近代密码学便在算法复杂性理论的基础上建立起来了。密码作为一种技术，自从人类有了战争，不久便有了它。但作为一门学科则是近20多年的事，甚至于它已成为其他学科的基础。密码也从此走出"军营"，进入百姓家。

实际中的"优化"问题是大量的，半个多世纪以来它曾经几度辉煌。近来在计算机科学的影响下，又出现了若干闪光点，十分耀眼，引人注目。

实际上密码也是一种编码。如果说密码学研究的编码是保证通信的保密与安全，则

编码理论研究的是通信中如何纠错与检错。计算机纠错码是既实用、理论上又饶有趣味的分支。

本丛书是作者在清华大学计算机科学与技术系长期工作的总结。它不是一部"长篇记述",而是互相关联又彼此相对独立的,因此难免有少量交叉。它们涉及的面非常广泛,囿于作者的水平,缺点和错误在所难免,敬请读者不吝指正。谢谢。

作　者

前　言

自从人类有了战争，就有了密码，所以密码作为一种技术源远流长，可以追溯到远古时代，而且还有过自己的辉煌经历。但成为一门学科则是近20余年的事，这是受计算机科学蓬勃发展的刺激结果。今天在计算机被广泛应用的信息时代，信息本身就是时间，就是财富。大量信息用数据形式存放在计算机系统里。信息的传输则通过公共信道。这些计算机系统和公共信道是不设防的，是很脆弱的，容易受到攻击和破坏，信息的丢失不容易被发现，而后果是极其严重的。如何保护信息的安全已不仅仅是军事和政府部门感兴趣的问题，各企事业单位也愈感迫切。因为在网络化的今天，计算机犯罪每年使他们遭受的损失极其巨大，而且还在发展中。密码是有效而且可行的保护信息安全的办法，有效是指密码能做到使信息不被非法窃取、不被篡改或破坏，可行是说它需要付出的代价是可以接受的。

密码形成一门新的学科是在20世纪70年代。它的理论基础之一应该首推1949年Shannon的一篇文章"保密通信的信息理论"，这篇文章过了30年后才显示出它的价值。现在，密码学有了突飞猛进的发展，而且成为有些学科的基础。特别是"电子商务"和"电子政府"的提出，使得近代密码学的研究成为热门的课题。也大大地扩大了它的发展空间。

在近代密码学上值得一书的大事有两件：一是1977年美国国家标准局正式公布实施了美国的数据加密标准(DES)，公开它的加密算法，并批准用于非机密单位及商业上的保密通信。密码学的神秘面纱从此被揭开。二是Diffie和Hellman联合写的一篇文章"密码学的新方向"，提出了适应网络上保密通信的公钥密码思想，掀起了公钥密码研究的序幕。受他们的思想启迪，各种公钥密码体制被提出，特别是RSA公钥密码的提出在密码学史上是一个里程碑。可以这么说："没有公钥密码的研究就没有近代密码学。"

在密码学的发展过程中，计算机科学和数学工作者作出了卓越的贡献。数学中许多分支如数论、概率统计、近世代数、信息论、椭圆曲线理论、算法复杂性理论、自动机理论、编码理论等都可以在其中找到各自的位置。它的踪影遍及数学许多分支，而且还推动了并行算法的研究，从而成为近若干年来非常引人入胜的领域。但还应该强调指出的是密码学毕竟不等于数学，它还有自己的空间。

中国不能没有自己的密码系统，中国也必须有自己的数据加密标准。近年来，我国引进了很多设备，惟有密码设备不能依靠引进，开展这方面的研究是当务之急。

本书是作者在清华大学计算机系从事数据安全的教学科研基础上写成的，文中有不妥之处，欢迎读者批评指正。

作　者

2003年9月

目　录

第 1 章　传统密码与密码学基本概念

1.1　引　　论

密码学以研究秘密通信为目的，即研究对传输信息采取何种秘密的变换以防止第三者对信息的窃取。

在今天的信息社会里，通信安全保密问题的研究已不仅仅出于军事、政治和外交上的需要。科学技术的研究和发展及商业等方面，无一不与信息息息相关。所以信息就是生命，信息就是时间，信息就是财富。由于信息是共享的，信息的扩散会产生社会影响，所以保护信息的安全是信息时代的迫切需要。

保护信息的安全无疑是十分重要的，然而信息的丢失不容易被发现。同时它又是具有时间性的。同一信息在不同时间里的价值也是不一样的，有时候获得信息的时间比信息本身还重要。

信息的存储和传输是通过载体进行的，例如，信使便是以人作为载体的。近代通信的载体有电波或电信号、磁盘等。

保密有载体保密和通信保密两种。密码学主要研究通信保密，而且仅限于数据通信保密。此外，还有语音保密和图像传输保密等。语音保密是研究电话通信的保密技术，不在本书讨论范围内。

近年来，密码学研究之所以十分活跃，主要原因是它与计算机科学的蓬勃发展息息相关。此外还由于电信事业以及防止日益严重的计算机犯罪的需要。由于公共和私人部门的一些机构愈来愈多地应用电子数据处理，将数据存储在数据库中，因此防止非法泄露、删除、修改等是必须正视的问题。特别是，电子资金传输系统是一个由通信网络互相联结的金融机构，并通过这种网络传输大量资金，这是密码通信通向民用的典型例子。因此，信息的安全性也成为全社会关心的问题，密码学从此也成为一门新的学科，引起了数学家和计算机科学工作者日益浓厚的兴趣。

一般说来，由数据库收集或存储的大量数据，或在传输过程中的数据，由于传输中的公共信道和存储的计算机系统非常脆弱，容易受到两种形式的攻击：一种是从传输信道上截取信息，或从存储的载体上偷窃或非法复制信息，我们称之为被动攻击，其结果是导致数据的暴露和对私有权的侵犯；另一种是对在传输过程中或对存储的数据进行非法删除、更改或插入等操作，这称之为主动攻击，其结果可能引起数据或文件的混乱，严重时可能导致信息处理系统完全失控。对于这两种可能遭受到的攻击除了制定法律外，还需要有合适的保护措施，密码技术就是一种有效的方法。事实证明，这也是最经济可行的方法，它使得在一种潜在不安全的环境中保证通信的安全。正是因为密码对于通信安全的极端重要性，所以应该强调说，不安全的密码技术比没有还要坏，因为它给人们以安全的假象。

密码技术还有效地被用于信息鉴别、数字签名等，用以防止电子欺骗，这对信息处理

系统的安全起到极其重要的作用。

近代密码学研究并非是传统密码技术的旧话重提，它有其自己的特点。快速电子计算机和现代数学方法一方面为加密技术提供了新的概念和工具，另一方面也给破译者以有力武器。总之，较之传统的密码系统有更丰富多彩的内容。

密码加密算法的对立面就是密码分析，也就是密码的破译技术研究。加密与破译是一对矛盾，了解破译对研究加密是非常必要的。

1.2 基本概念

什么是密码？简单地说它就是一组含有参数 k 的变换 E。设已知信息 m，通过变换 E_k 得密文 c，即

$$c=E_k(m)$$

这个过程称之为加密，参数 k 称之为密钥。加密算法 E 确定之后，由于密钥 k 不同，密文 c 也不同。

当然不是所有含参数 k 的变换都可以作为密码，它要求计算 $E_k(m)$ 不困难，而且若第三者不掌握密钥 k，即使截获了密文 c，他也无法从 c 恢复信息 m，也就是反过来从 c 求 m 极为困难。以后称 m 为明文。

通信双方一方为发信方，简称发方，另一方为收信方，简称收方。传统的保密通信机理可用图 1.1 表示。

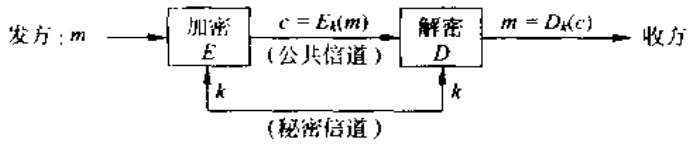


图 1.1

从密文 c 恢复明文 m 的过程称之为解密。解密算法 D 是加密算法 E 的逆运算，解密算法也是含参数 k 的变换。传统密码加密用的密钥 k 与解密用的密钥 k 是相同的，所以有时也叫对称密码。通信双方用的密钥 k 是通过秘密方式由双方私下约定产生的，只能由通信双方秘密掌握。如果丢失了密钥，则密码系统不攻自破。密钥的重要性可想而知。举一个最简单的例子。设已知明文 m 为

during the last twenty years there has been an explosion of public academic research in cryptography

明文的意思是“近 20 年对密码学的公开研究已急剧增加”。

先将明文分成 5 个字符一组得

durin gthel asttw entyy earst hereh asbee nanex plosi onofp ublic acade micre searc hincr yptog raphy

再将每组按相反顺序写下，例如 durin，倒过来写成 nirud，于是得密文 c 如下：

nirudlehtgwttsayytnetsraehereheebsaxenanisolppfonocilbuedacaercimcraes

rcnihgotpyyhpar

这里加密算法便是将明文先分组再逆序书写，密钥是每组的字符长。本例 $k=5$。若不知道加密算法，该密文相对于明文面目全非，从而达到加密的目的。当然这个加密算法不是很安全的，破译不难。

1.3 若干传统密码与其破译技术

在这一节将介绍若干经典、传统的密码，附带讨论其中若干破译技巧。本书重点讨论加密算法。加密与破译是一对矛和盾，整个密码学也分成两大分支：加密方法与密码分析。密码分析则是研究破译的一门技术。但了解破译技术对研究加密算法是必要的。加密是一门科学，密码分析也是一门学问。有的加密算法对不掌握密码分析方法的人乍一看十分神秘，似乎"牢不可破"，其实不堪一击。前面已强调过，不可靠的密码比没有密码还坏。

1.3.1 密码举例

最早的一种密码是在公元前两世纪，由一位希腊人提出来的。他将 26 个字母排列在一个 5×5 的方格里，其中 i 和 j 填在同一格，如下所示：

	1	2	3	4	5
1	a	b	c	d	e
2	f	g	h	ij	k
3	l	m	n	o	p
4	q	r	s	t	u
5	v	w	x	y	z

于是每个字母对应一数 $\xi\eta$，其中 ξ 是该字母所在行的标号，η 是列标号。如 c 对应 13，r 对应 42 等。使用这种密码可以将明文

secure message transmission is of extreme importance in information based society

转换为密文

43 15 13 45 42 15 32 15 43 43
11 22 15 44 42 11 33 43 32 24
43 43 24 34 33 24 43 34 21 15
53 44 42 15 32 15 24 32 35 34
42 44 11 33 13 15 24 33 24 33
21 34 42 32 11 44 24 34 33 12
11 43 15 14 43 34 13 24 15 44
54

在古代这种棋盘密码曾被广泛应用。

1.3.2 Kaiser 密码

Kaiser 密码是每一字母向前推移 k 位。例如 $k=5$ 便有明文和密文对应关系如下：

明文：a b c d e f g h i j k l m n o

密文：F G H I J K L M N O P Q R S T

明文：p q r s t u v w x y z

密文：U V W X Y Z A B C D E

于是对于明文：

data security has evolved rapidly

可加密得密文：

I F Y F X J H Z W N Y D M F X J A T Q A J I W F U N I Q D

不同的 k 可得不同的密文。

若令 26 个字母分别对应于整数 0～25，如表 1.1 所示。

表 1.1

a	b	c	d	e	f	g	h	i	j	k	l	m
1	2	3	4	5	6	7	8	9	10	11	12	13
n	o	p	q	r	s	t	u	v	w	x	y	z
14	15	16	17	18	19	20	21	22	23	24	25	0

则 Kaiser 加密变换实际上是

$$c \equiv m + k \bmod 26$$

其中 m 是明文对应的数据，c 是与明文对应的密文数据，k 是加密用的参数，也称为密钥。

例如

data security

对应于数据序列

4 1 20 1 19 5 3 21 18 9 20 25

$k=5$ 时得密文序列

9 6 25 6 24 10 8 0 23 14 25 4

对应的密文为

I F Y F X J H Z W N Y D

若选取 k_1，k_2 两个参数，其中$(k_1,26)=1$，即 k_1 和 26 互素，令

$$c \equiv k_1 m + k_2 \bmod 26$$

这种变换称之为仿射变换。$k_1=1$ 时便是 Kaiser 变换。

例如：$k_1=7$，$k_2=10$，则明文

please send moneys

的对应数据为

16 12 5 1 19 5 19 5 14 4

13 15 14 5 25 19

通过变换 $c \equiv 7m+10 \bmod 26$ 可得

18 16 19 17 13 19 13 19 4 12

23 11 4 19 3 13

对应的密文为

R P S Q M S M S D L W K D S C M

1.3.3 单表置换

下面引进更一般的变换形式，例如

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

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

其中 TSINGHUAVERY 是 Tsinghua University 略去前面已出现的字符依次写下的。后面 BCD…WXZ 则是按英文字母表顺序续上前面未出现的字母而构成的，故 Tsinghua University便称之为该密码的密钥词组。密钥词组可作为密码的标志，记住这个密钥词组就能掌握字母加密置换的全体。

前面已提到加密算法和破译技术是一对矛和盾。为了更好地研究加密方法，了解必要的破译技巧是有好处的。

下面举一例子说明密码分析是如何进行的，若已知密文

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

原来的明文经过加密变换显然掩盖了真面目，然而不是没有留下可以利用的蛛丝马迹。比如字母出现的频率和前后连缀的关系不会由于简单的置换而不留下痕迹。下面将从此入手，逐步剥去伪装恢复其原来面目。

从大量非技术性的英文书籍、报刊或文章中摘取足够长度的章节进行统计，发现英文字母出现的频率有惊人的相似之处。比如 e 出现的次数最多，其他如 t，a，o，n，i 等出现较多，几乎处处如此。不仅是单个字母如此，相邻的连缀字母也是如此。

出现频率较高的双字母有 th，he，in，er，an，re，ed，on，es，st，en，at，to，nt，ha，nd，ou，ea，ng，as，or，ti，is，et，it，ar 等。三字母出现频率高的有 the，ing，and，her，ent 等。

经过大量的统计，可得出英文字母出现的频率如表 1.2 所示。

表 1.2

a	b	c	d	e	f
0.0856	0.0139	0.0279	0.0378	0.1304	0.0289
g	h	i	j	k	l
0.0199	0.0528	0.0627	0.0013	0.0042	0.0339
m	n	o	p	q	r
0.0249	0.0707	0.0797	0.0199	0.0012	0.0677
s	t	u	v	w	x
0.0607	0.1045	0.0249	0.0092	0.0149	0.0017
y	z				
0.0199	0.0008				

连缀字的频率如表 1.3 所示。

表 1.3

	a	b	c	d	e	f	g	h	i	j	k	l	m
a	0.0011	0.0193	0.0388	0.0469	0.002	0.01	0.0233	0.002	0.048	0.002	0.0103	0.1052	0.0281
b	0.0931	0.0057	0.0016	0.0006	0.3219	0	0	0	0.0605	0.0057	0	0.1242	0.0049
c	0.1202	0	0.0196	0.0004	0.1707	0	0	0.1277	0.0761	0	0.0324	0.0369	0.0015
d	0.1044	0.002	0.0026	0.0218	0.3778	0.0007	0.0132	0.0007	0.1803	0.0033	0	0.0125	0.0178
e	0.066	0.0036	0.0433	0.1194	0.0438	0.0142	0.0125	0.0021	0.0158	0.0005	0.0036	0.0456	0.034
f	0.0838	0	0	0	0.1283	0.0924	0	0	0.1608	0	0	0.0299	0.0009
g	0.1078	0	0	0.0018	0.2394	0	0.0177	0.1281	0.0839	0	0	0.0203	0.0027
h	0.1769	0.0005	0.0014	0.0008	0.5623	0	0	0.0005	0.1167	0	0	0.0016	0.0016
i	0.038	0.0082	0.0767	0.0459	0.0437	0.0129	0.028	0.0002	0.0016	0	0.005	0.0567	0.0297
j	0.1259	0	0	0	0.1818	0	0	0	0.035	0	0	0	0
k	0.0359	0.0028	0	0.0028	0.5282	0.0028	0	0.0198	0.1582	0	0.0113	0.0198	0.0028
l	0.1342	0.0019	0.0022	0.0736	0.1918	0.0105	0.0108	0	0.1521	0	0.0079	0.1413	0.0082
m	0.1822	0.0337	0.0026	0	0.2975	0.001	0	0	0.1345	0	0	0.001	0.0654
n	0.055	0.0004	0.0621	0.1681	0.1212	0.0102	0.1391	0.0013	0.0665	0.0009	0.0066	0.0073	0.0104
o	0.0085	0.0101	0.0162	0.0231	0.0037	0.1299	0.0082	0.0025	0.0092	0.0014	0.0078	0.0416	0.0706
p	0.1359	0	0.0006	0	0.1747	0	0	0.0237	0.0423	0	0	0.0812	0.0073
q	0	0	0	0	0	0	0	0	0	0	0	0	0
r	0.1026	0.0033	0.0172	0.0282	0.2795	0.0031	0.0175	0.0017	0.1181	0	0.0205	0.0164	0.0303
s	0.0604	0.0012	0.0284	0.0027	0.1795	0.0024	0	0.0561	0.1177	0	1.0091	0.0145	0.0112
t	0.0619	0.0003	0.0036	0.0002	0.1417	0.0007	0.0002	0.3512	0.1406	0	0	0.0101	0.0044
u	0.0344	0.0415	0.0491	0.0243	0.0434	0.0052	0.0382	0.001	0.0258	0	0.0014	0.1097	0.0329
v	0.0749	0	0	0.0023	0.6014	0	0	0	0.2569	0	0	0	0.0012

续表

	a	b	c	d	e	f	g	h	i	j	k	l	m
w	0.2291	0.0008	0	0.0032	0.1942	0	0	0.1422	0.2104	0	0	0.0041	0
x	0.0672	0	0.1119	0	0.1269	0	0	0.0075	0.1119	0	0	0	0.0075
y	0.0586	0.0034	0.0103	0.0069	0.2897	0	0	0	0.069	0	0.0034	0.0172	0.0379
z	0.2278	0	0	0	0.4557	0	0	0	0.2152	0	0	0.0127	0

	n	o	p	q	r	s	t	u	v	w	x	y	z
a	0.1878	0.0008	0.0222	0	0.118	0.1001	0.1574	0.0137	0.0212	0.0057	0.0026	0.0312	0.0023
b	0	0.0964	0	0	0.0662	0.0229	0.0049	0.0727	0.0016	0	0	0.1168	0
c	0.0011	0.2283	0	0.0004	0.0426	0.0087	0.0893	0.0347	0	0	0	0.0094	0
d	0.0053	0.0733	0	0.0007	0.0324	0.0495	0.0013	0.0601	0.0099	0.004	0	0.0264	0
e	0.1381	0.004	0.0192	0.0034	0.1927	0.1231	0.0404	0.0048	0.0215	0.0205	0.0152	0.0121	0.0004
f	0.0009	0.2789	0	0	0.1215	0.0026	0.0496	0.0462	0	0	0	0.0043	0
g	0.0451	0.114	0	0	0.1325	0.0256	0.0247	0.0512	0	0	0	0.0053	0
h	0.0038	0.0786	0	0	0.0153	0.0027	0.0233	0.0085	0	0.0011	0	0.0041	0
i	0.2498	0.0893	0.01	0.0008	0.0342	0.1194	0.1135	0.0011	0.025	0	0.0023	0.0002	0.0079
j	0	0.3147	0	0	0.007	0	0	0.3357	0	0	0	0	0
k	0.0565	0.0198	0	0	0.0085	0.1102	0.0028	0.0028	0	0	0	0.0113	0
l	0.0004	0.0778	0.0041	0	0.0034	0.0389	0.0254	0.0269	0.0056	0.0011	0	0.0819	0
m	0.0042	0.1246	0.0722	0	0.0026	0.0244	0.0005	0.0337	0.0005	0	0	0.0192	0
n	0.0194	0.0528	0.0004	0.0007	0.0011	0.0751	0.1641	0.0124	0.0068	0.0018	0.0002	0.0157	0.0004
o	0.219	0.0222	0.0292	0	0.153	0.0357	0.0396	0.0947	0.0334	0.0345	0.0012	0.0041	0.0004
p	0.0006	0.1511	0.0581	0	0.2306	0.018	0.0287	0.0457	0	0	0	0.0017	0
q	0	0	0	0	0	0	0	1	0	0	0	0	0
r	0.0325	0.1114	0.0055	0	0.0212	0.0655	0.0596	0.0192	0.0142	0.0017	0.0002	0.0306	0
s	0.0021	0.0706	0.0386	0.0009	0.0027	0.0836	0.2483	0.0579	0	0.0039	0	0.0081	0
t	0.0015	0.1229	0.0003	0	0.0479	0.0418	0.0213	0.0195	0.0005	0.0088	0	0.0203	0.0005
u	0.1517	0.0019	0.0386	0	0.146	0.1221	0.1255	0.0029	0.0014	0	0.001	0.0014	0.0005
v	0	0.053	0	0	0	0.0023	0	0.0012	0.0012	0	0	0.0058	0
w	0.0357	0.1292	0	0	0.0106	0.0366	0.0016	0	0	0	0	0.0024	0
x	0	0.0075	0.3507	0	0	0	0.1716	0	0	0	0.0373	0	0
y	0.0172	0.2207	0.031	0	0.031	0.1517	0.0172	0.0138	0	0.0103	0	0.0069	0.0034
z	0	0.0506	0	0	0	0	0	0.0127	0	0	0	0	0.0253

表 1.3 中以 e 行为例，e 行 a 列为 0.066；e 行 e 列为 0.0438；即 ea 出现的频率占 0.066，ee 出现的频率占 0.0438。e 行的全体总和为 1。

现在回到对上面密文中 280 个字母进行统计，并将出现的频数列表如下：

A	B	C	D	E	F	G	H	I	J	K	L	M
16	5	13	0	7	17	23	26	5	12	3	2	2
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
36	25	1	5	0	0	22	20	4	6	9	14	7

若按频数从大到小依次排列得

N	H	O	G	T	U	F	A	Y	C	J	X
36	26	25	23	22	20	17	16	14	13	12	9
E	Z	W	B	I	Q	V	K	L	M	P	
7	7	6	5	5	5	4	3	2	2	1	

机械地依照频率的大小顺序确定明文和密文的对应关系是不恰当的。一般来说，e，t，a，o，n，i，r，s，h 在高频率字母群中出现；d，l，u，c 和 m 在中频率字母群中出现；p，f，y，w，g，b，v 属于低频部分。j，k，q，x，z 比较罕见。从频率表中可以看出，高频字母群和中频字母群之间有一明显的间断。也就是说，高频中频率最低为 h，它的频率为 0.0528，中频中频率最高为 d，它的频率为 0.0378，频率之差为

$$0.0528-0.0378=0.015$$

而 e 在高频字母群中特别突出。

这在密文的频数分析中也很明显。可以有理由假定高频字母群 e，t 等 9 个字符正好是密文中前 9 位高频字符的对应明文。而且其中密文字符 N 非常可能就是 e 的对应密文。

现将密文中频率高的 9 个字母的前后连缀关系组成表 1.4。表中每个格子中有上下两个数，上面一个数表示前缀关系，下面一个数则表示后缀关系。例如 N 行 Y 列只有上面一个数 9，这表示 YN 出现 9 次，而 NY 一次也没有。又如 G 行 N 列上面一个数为 5，下面一个数为 4，这说明 NG 出现 5 次，而 GN 出现 4 次。其他依此类推。

表 1.4

↓ \ →	N	H	O	G	T	U	F	A	Y	
N		3 1		4 5	4	5	3 7	5	9	e
H	1 3	2 2	5 4	2 1	4 1	1 1	2	1 1	10	t
O		5 5		6 10	1 7	1	1 1	2	3	i
G	5 4	1 2	4 6	2 2		2	2	2		?
T	4	1 4	7 1			4 3	1	2 3		n
U	5	1 1	1	2	3 4	2 3				a
F	7 3	2	1 1	3	1	3 2		3	1	?
A	5	1 1	2	2	3 2		4		1	o
Y	9	10	3				1	1		h

现在依次判断如下：

① N 是明文 e 的密文，用 N↔e 表示。

② 高频字母群中的 a，i，o 这 3 个元音互联的机会极小。由此可推断 O，U，A 可能是 a，i，o 的密文。H 显然很少同 U 和 A 相接触，然而经常和 O 相联，所以它不可能是

元音。

③ a，i，o 这 3 个元音互相联结的机会显然很少，但根据统计 i o 的机会相对稍高。从上面的统计表中可知 OA 出现两次，OU 出现一次。其他如 UO，UA，AO，AU 等均不出现。所以若 OA 分别是 i o 的密文，则应有

$$O\leftrightarrow i,\quad A\leftrightarrow o$$

④ 从上面的推断结果来看，U 可能是 a 的密文，OU 出现一次。特别考虑到 NU 出现 5 次，而 UN 不出现，说明 e a 出现次数较多，而 a e 很少，这些事实都证明 U↔a 的可能性很大。

⑤ 高频辅音中最容易识别的是字母 n，n 前面是元音的几率高达 0.80。从元音表中可见 T 前面和 N，O，U，A 连接的总次数为 17。

⑥ Y 的特点也很突出，YN 出现 9 次，然而 NY 不出现。根据 N↔e 和 h e 出现机会多，而 e h 出现机会极少，所以推断 Y↔h 是非常可能的。

⑦ 根据英文中 t h 经常出现，而 h t 很罕见的特点，断定可能有 H↔t。

⑧ 在高频率的字母群中，还有 r，s 两个未能判定。但 r 和 a，e，i，o 的连接多于 s，而 s 多于同辅音相连。在余下的高频密文中，G 和 F 中不容易得出有说服力的结论。虽有差别，但并不明显。现在已经在 280 个字母中识别出 159 个，占一半以上。先将它代入部分密文得：

G	J	X	X	N	G	G	O	T	Z	N	U	C	O	T	W
				e			i	n		e	a		i	n	
M	O	H	Y	J	T	K	I	A	M	T	X	O	B	Y	N
w	i	t	h		n		n	o	w	n		i		h	e
F	G	O	G	I	N	U	G	J	F	N	Z	V	Q	H	Y
		i			e	a				e				t	h
N	G	N	E	A	J	F	H	Y	O	T	W	G	O	T	H
e		e		o			t	h	i	n			i	n	t
Y	N	A	F	Z	N	F	T	U	I	N	Z	B	N	F	G
h	e	o			e		n	a		e			e		
N	L	N	F	U	T	X	N	X	U	F	N	E	J	C	I
e		e		a	n		e		a		e				
N	H	Y	A	Z	G	A	E	U	T	U	C	Q	G	O	G
e	t	h	o			o		a	n	a				i	
O	T	B	J	O	H	O	A	T	C	J	X	K	H	Y	N
i	n	t		i	t	i	o	n					t	h	e
U	V	O	C	O	H	Q	U	H	C	N	U	G	H	H	A
a		i		i	t		a	t		e	a		t	t	o
F	N	U	Z	H	Y										
	e	a		t	h										

⑨ _ i t h 可能是 with，故令 M↔w。

⑩ 若将 M 代以 w，代入上面密文，观察是否可以导出其他信息，请见上面有

"═══"的部分

with—n—nown

和

int—ition

可猜测分别是 with unknown 和 intuition，故令 J↔u，K↔k，现将已得到的置换排列如下：

a	b	c	d	e	f	g	h	i	j	k	l	m
U				N			Y	O		K		
n	o	p	q	r	s	t	u	v	w	x	y	z
T	A					H	J		M			

⑪ 从对应关系

t	u	v	w
H	J		M

可以推测

L↔v

⑫ 由⑧知 F 和 G 究竟如何对应于 r 和 s，无确切的根据。现在由于

H↔t，　J↔u

所以可以令

F↔r，　G↔s

⑬ 由于 U↔a，可令 V↔b。

⑭ 将以上结果代入密文得

G	J	X	X	N	G	G	O	T	Z	N	U	C	O	T	W
s	u			e	s	s	i	n		e	a		i	n	
M	O	H	Y	J	T	K	T	A	M	T	X	O	B	Y	N
w	i	t	h	u	n	k	n	o	w	n		i		h	e
F	G	O	G	I	N	U	G	J	F	N	Z	V	Q	H	Y
r	s	i	s		e	a	s	u	r	e		b		t	h
N	G	N	E	A	J	F	H	Y	O	T	W				
e	s	e		o	u	r	t	h	i	n					

⑮ 由 s u (XX)e s s 可令 X↔c，得 success。括号(　)里的 XX 是尚未破译的密文。

⑯ 由 ci (B) hers 可令 B↔p，得 ciphers。

⑰ 由(E)our 可令 E↔f 得 four。

⑱ thin(W)可令 W↔g 得 thing。

由此得置换的密钥词组为 NEW YORK CITY，得置换

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

全部明文为：

Success in dealing with unknown ciphers is measured by these four things in the order named, perseverance, careful methods of analysis, intuition, luck. The ability at least to read the language of the original text is very desirable but not essential. Such is the opening sentence of Parker Hitt's Manual for the Solution of Military Ciphers.

标点符号是加上的。这段原文很有意思，翻译为："破译一未知密码是否成功，可由以下 4 个因素来衡量，按其顺序为：毅力、审慎的分析方法、直观和运气。阅读原文的文字的起码能力是需要的，然而不是必不可少的。这是 Parker Hitt 的《军事密码破译指南》一书的开场白。"其实除 N↔e 外，8 个高频字符的对应只有 8! =40320 个。穷举也不难实现。

这个例子说明英文字符的频率差异是英文本身给密码带来多余的东西，为破译提供了可以利用的弱点。

前面介绍的密码体制都是属于单表置换。即一个明文字母对应的密文字母是确定的。正因为如此，频率分析对该密码体制进行了有效的攻击。下面将介绍一种多表密码，即一个明文字母可以表示成多个密文字母。

1.3.4 Vigenere 密码

Vigenere 是法国的密码专家，以他的名字命名的加密算法是多表密码的典型代表。方法如下：

设密钥 $k=k_1k_2\cdots k_n$，明文 $M=m_1m_2\cdots m_n$，加密变换

$$E_k(M)=c_1c_2\cdots c_n$$

其中 $c_i\equiv(m_i+k_i) \bmod 26$，$i=1,2,\cdots,n$。

例如，$M=$data security，$k=$best。首先将 M 分解成长为 4 的序列

data secu rity

每一节利用密钥 $k=$best 加密，得密文

$$c=E_k(M)=\text{EELT TIUN SMLR}$$

下面是 Vigenere 方阵，利用它可以进行加密和解密。

明文：	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

至于密钥 k 可以通过周期性地延长，周而复始，反复进行以至无穷。即令

$$k = k_1 k_2 k_3 \cdots$$

其中

$$k_{i+kn} \equiv k_i \bmod n$$

例如，利用密钥 k=best 对明文 data 加密得密文 EELT，第一个 E 是在 b 行 d 列上；第二个 E 是在 e 行 a 列上；同样的道理 L 在 s 行 t 列上；第四个 T 是在 t 行 a 列上。其余依此类推。解密时第一个明文 d 是 b 行含密文 E 的列，同理第二个明文 a 是 e 行含密文 E 的列，同理推出其他。

Vigenere 方阵给我们以启发，可用类似想法得到很多其他多表密码，其中最著名的有 Beaufort 方法，如下所示。

明文：	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	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A
b	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B
c	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C
d	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D
e	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E
f	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F
g	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G
h	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H
i	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I
j	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J
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	L	K

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	L
m	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
n	M	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
o	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O
p	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P
q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q
r	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R
s	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S
t	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T
u	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U
v	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V
w	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W
x	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X
y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y
z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z

它是由 Vigenere 方阵作如下修改而得到的：Beaufort 方阵的每一行恰好是 Vigenere 方阵对应行的逆序排列。利用 Beaufort 方阵进行加密，相当于已知密钥

$$k=k_1k_2\cdots k_n$$

明文

$$M=m_1m_2\cdots m_n$$

可得密文

$$c=E_k(M)=c_1c_2\cdots c_n$$

其中

$$c_i\equiv(k_i+m_i)\bmod 26,\quad i=1,2,\cdots,n$$

Vigenere 方阵也可用来作 Beaufort 密码的加密、解密。具体方法留给读者自己思考。

多表密码加密算法结果将使得对单表置换用的简单频率分析方法失效。

1.3.5 对 Vigenere 密码的分析

(1) 若随机地选取英文字母构成序列，任一字母被选上的概率为$\frac{1}{26}$。

若两个随机产生的英文字母序列并列，在特定的位置上出现相同字母的概率为

$$26\times\left(\frac{1}{26}\right)^2=1/26=0.0385$$

因特定的一字母重复的概率为$(1/26)^2$，但共有 26 个字母，所以特定位置上出现相同字母的概率为$\frac{1}{26}$。

如果任取两段英文并列，在特定位置上同时出现 a 的概率应为

$$(0.0856)^2=0.0073274$$

同时出现 b 的概率为

$$(0.0139)^2=0.0001932$$

其余可依此类推。

若 p_1 表示 a 出现的概率，p_2 表示 b 出现的概率，……，p_{26} 表示 z 出现的概率，那么

$$x=\sum_{i=1}^{26}p_i^2=0.0687$$

x 是 0.0385 的 1.784 倍。也就是说，若任取两段英文并列，则在特定位置上出现相同字母的概率是随机产生的两段字母序列的 1.784 倍。

若对某一密文出现的字母频率进行统计得

$$f_A,f_B,f_C,\cdots,f_Z$$

而且满足

$$f_A+f_B+f_C+\cdots+f_Z=1$$

所以频率的平均值仍然是$\frac{1}{26}$。

我们引进

$$k^2=\sum_{\xi=A}^{Z}\left(f_\xi-\frac{1}{26}\right)^2$$

$$=\sum_{\xi=A}^{Z}f_\xi^2-\frac{2}{26}\sum_{\xi=A}^{Z}f_\xi+\left(\frac{1}{26}\right)^2\cdot 26$$

由于

$$\sum_{\xi=A}^{Z}f_\xi=1$$

所以

$$k^2=\sum_{\xi=A}^{Z}f_\xi^2-\frac{2}{26}+\frac{1}{26}$$

$$=\sum_{\xi=A}^{Z}f_\xi^2-\frac{1}{26}$$

若当 $f_A=f_B=\cdots=f_Z=\frac{1}{26}$ 时，$k\equiv 0$。k 值愈大表示 $f_A,f_B,\cdots,f_Z$ 起伏比较大，或 $f_A,f_B,\cdots,f_Z$ 对于平均值$\frac{1}{26}$的偏离程度愈强烈。在计算 k^2 时需要计算 $\sum_{\xi=A}^{Z}f_\xi^2$，这个值实际上是任选两个位置，在这两个位置上出现相同字母的概率。

(2) 假定对于 $c_1c_2\cdots c_n$ 中，字母 A 的数目为 n_A，字母 B 的数目为 n_B，$\cdots$，字母 Z 的数目为 n_Z。那么

由字母 A 构成的字母对数目为$\frac{1}{2}n_A(n_A-1)$

由字母 B 构成的字母对数目为$\frac{1}{2}n_B(n_B-1)$

⋮

由字母 Z 构成的字母对数目为$\frac{1}{2}n_Z(n_Z-1)$

所以，相同字母构成的字母对数目为

$$S=\frac{1}{2}[n_A(n_A-1)+n_B(n_B-1)+\cdots+n_Z(n_Z-1)]$$

$$=\frac{1}{2}\sum_{\xi=A}^{Z}n_\xi(n_\xi-1)$$

两个位置上字母相同的概率为

$$IC=\frac{S}{C(n,2)}=\sum_{\xi=A}^{Z}n_\xi(n_\xi-1)/n(n-1)$$

称 IC 为重合指数(index of coincidence)，它可以看做是 $\sum_{\xi=A}^{Z}(f_\xi)^2$ 的近似值，是任意两位有相同文字的概率。

(3) 假定密钥 $k=k_1k_2\cdots k_m$ 的长度为 m，而且各位由不同的字母组成，则可将密文 c 分成 m 行，每行有 $\frac{n}{m}$ 个元素，使得每行是单表置换加密，不同的行由不同的密钥加密。按以上假设，同一行两个选定的位置上有相同字母的概率接近 0.0687。在不同行里选定两个位置，由于它是由不同的密码加密的，而且密钥 k 各位由不同的文字组成，所以在这个位置上出现相同文字的概率接近 0.0385。在长度为 n 的密文中，选择两个位置共有 $C(n,2)=\frac{1}{2}n(n-1)$种方案，故有相同文字对的数目为 $\frac{1}{2}n(n-1)IC$。

另一方面，假定 n 是 m 的倍数，在选了第 1 个位置后，在同一行中选第 2 个位置的方案数为 $\frac{1}{2}n\left(\frac{n}{m}-1\right)$，在不同行中选择第 2 个位置的方案数为 $\frac{1}{2}n\left(n-\frac{n}{m}\right)$。这样

$$\frac{1}{2}n(n-1)IC=\frac{1}{2}n\left(n-\frac{n}{m}\right)\times 0.0385+\frac{1}{2}n\left(\frac{n}{m}-1\right)\times 0.0687$$

所以

$$(n-1)IC-0.0385n+0.0687=(0.0687-0.0385)\frac{n}{m}=0.0302\frac{n}{m}$$

$$m=\frac{0.0302n}{(n-1)IC-0.0385n+0.0687}$$

这个公式可用来估计密钥 k 的长度。即从密文 c，可求得 IC，进而求出密钥 k 的长度 m 的估计值。确定 m 是破译 Vigenere 码的关键一步。这里所得仅是参考数值，如何确定在下面例子中再介绍。

(4) Kasiski 试验。两个相同的字母序列出现在一份明文中，一般说来，利用 Vigenere 密码加密，结果密文字母是不会相同的。但是，如若两个相同字母序列间距离正好是密钥长度的倍数时，也可能产生相同的密文序列。所以研究密文序列时，若发现有重复出现的字母序列，则它们间的距离很可能是密钥长度的倍数。寻找重复出现的字母序列，并求其长度的过程称为 Kasiski 试验。

例 1.1 已知密文如下：

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

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

字符数共 280 个，其中

A：9 个	B：4 个	C：10 个
D：7 个	E：14 个	F：15 个
G：14 个	H：10 个	I：11 个
J：7 个	K：9 个	L：13 个
M：10 个	N：3 个	O：7 个
P：12 个	Q：9 个	R：8 个
S：20 个	T：12 个	U：14 个
V：12 个	W：27 个	X：5 个
Y：6 个	Z：12 个	

$IC=0.0431$　　$m=6.802$

现作 Kasiski 试验，并将结果列于表 1.5。

先对表 1.5 作一些说明。如“位置”一栏，以 UI 为例，有 3 和 228，3 指的是第 1 个 UI 前面有 3 个字符，第 2 个 UI 出现在第 228 个字母之后，其间间隔为 228－3＝225，$225=3^2\times5^2$，“因子”栏为 3，3，5，5，其余类推。

表 1.5

字符	位置		距离	因子			
UI	3	228	225	3	3	5	5
IU	4	122	118	2	59		
IU	4	207	203	7	29		
IU	4	254	250	2	5	5	5
DW	6	111	105	3	5	7	
WF	7	57	50	2	5	5	
WF	7	112	105	3	5	7	
HG	9	142	133	7	19		
GL	10	145	135	3	3	3	5
LZ	11	106	95	5	19		
ZA	12	267	255	3	5	17	
AR	13	168	155	5	31		
HW	16	172	156	2	2	3	13
WL	17	132	115	5	23		
WL	17	155	138	2	3	23	

续表

字符	位置		距离	因子					
LL	18	63	45	3	3	5			
LL	18	148	130	2	5	13			
YY	21	25	4	2	2				
QA	27	215	188	2	2	47			
TJ	29	180	151	151					
PF	31	236	205	5	41				
PF	31	245	214	2	107				
FK	32	58	26	2	13				
FK	32	129	97	97					
FK	32	153	121	11	11				
MU	34	272	238	2	7	17			
SS	37	126	89	89					
SS	37	127	90	2	3	3	5		
SS	37	276	239	239					
SW	38	56	18	2	3	3			
WW	39	48	9	3	3				
WW	39	131	92	2	2	23			
WW	39	183	144	2	2	2	2	3	3
SV	42	233	191	191					
FA	44	75	31	31					
AE	45	203	158	2	79				
EV	46	161	115	5	23				
VW	47	162	115	5	23				
WW	48	131	83	83					
WW	48	183	135	3	3	3	5		
WG	49	79	30	2	3	5			
WG	49	199	150	2	3	5	5		
QC	51	250	199	199					
CM	52	271	219	3	73				
MV	53	239	186	2	3	31			
VS	55	125	70	2	5	7			
WF	57	112	55	5	11				
FK	58	129	71	71					
FK	58	153	95	5	19				
LL	63	148	85	5	17				
LG	64	149	85	5	17				
IT	69	179	110	2	5	11			
AS	76	118	42	2	3	7			
AS	76	216	140	2	2	5	7		
SJ	77	82	5	5					
SJ	77	277	200	2	2	2	5	5	

续表

字符	位置		距离	因子					
JW	78	278	200	2	2	2	5	5	
WG	79	199	120	2	2	2	3	5	
SJ	82	277	195	3	5	13			
EP	84	104	20	2	2	5			
EP	84	244	160	2	2	2	2	2	5
UE	88	140	52	2	2	13			
TP	90	188	98	2	7	7			
MP	93	219	126	2	3	3	7		
MP	93	235	142	2	71				
HZ	97	212	115	5	23				
HZ	97	252	155	5	31				
SF	99	128	29	29					
SF	99	217	118	2	59				
EP	104	244	140	2	2	5	7		
WK	109	184	75	3	5	5			
KD	110	210	100	2	2	5	5		
HA	117	202	85	5	17				
AS	118	216	98	2	7	7			
IU	122	207	85	5	17				
IU	122	254	132	2	2	3	11		
SS	126	127	1						
SS	126	276	150	2	3	5	5		
SS	127	276	149	149					
SF	128	217	89	89					
FK	129	153	24	2	2	2	3		
KW	130	154	24	2	2	2	3		
KW	130	198	68	2	2	17			
WW	131	183	52	2	2	13			
WL	132	155	23	23					
CC	134	270	136	2	2	2	17		
LR	146	193	47	47					
GW	150	222	72	2	2	2	3	3	
KW	154	198	44	2	2	11			
WS	158	163	5	5					
OT	187	265	78	2	3	13			
IU	207	254	47	47					
HZ	212	252	40	2	2	2	5		
ZI	213	253	40	2	2	2	5		
IQ	214	229	15	3	5				
MP	219	235	16	2	2	2	2		
PF	236	245	9	3	3				

续表

字符	位置		距离	因子				
ZM	238	257	19	19				
DWF	6	111	105	3	5	7		
EVW	46	161	115	5	23			
LLG	63	148	85	5	17			
SJW	77	277	200	2	2	2	5	5
FKW	129	153	24	2	2	2	3	
HZI	212	252	40	2	2	2	5	

分析因子出现的频率：

2：52 次　　3：32 次　　4：28 次
5：50 次　　6：18 次　　7：13 次
8：15 次　　9：11 次　　10：21 次
11：5 次　　12：8 次　　13：7 次
14：8 次　　15：16 次　　16：3 次
17：9 次　　18：5 次　　19：4 次
20：12 次　　21：5 次

密钥长为 2 的可能性不大，所以密钥长度可能为 5。现将密文按密文顺序分 5 行，且周而复始，即第 6 个字符归第 1 行。并求其重合指数 IC 如下：

第 1 段

U U G I W Y J U W A G V U G T F G P T O F P K W P V K C
U G G W H T C V T K G Q G N K Q P V Q M V P Q U K O C R

1：IC=0.0623

第 2 段

F D L H Y Y P X C E Q S T Z Y A G N P P D L D T W S W E
E L W L E T W T J C M E Y D D A R P Q P E F C Z Z T C S

2：IC=0.0506

第 3 段

Q W Z W Y Q F S S V C W B F O S S S T H C Z W H I S W Z
H R O U V U H G R O I S H I H S G B D F G O H Z B Z M S

3：IC=0.0649

第 4 段

U F A L F A K S V W M F L V E J J U M Z X Q F A U S L W
G L F W W A W N W I L K A U Z F W U S Z E D Z M G A U J

4：IC=0.0617

第 5 段

I H R L S T M W F W V K L I I W E E P S E W X S O F C E
V L K S S R B I W P R W E L I M R I V M E J I X J X M W

5：IC=0.0617

可见除第 2 行外，其余行的 IC 都在 0.060 以上，有力说明它们可能分别是单表置换的结果。

上面的 5 行假定它们各自通过

$$c \equiv m + k_i \bmod 26 \qquad i = 1,2,3,4,5$$

变换得到的，密钥 k_i 是不同的。假定第 i 行的密钥为 k_i，第 j 行的密钥为 k_j，令 $\delta_{ij} = k_j - k_i$，后面将设法求出 δ_{ij}，目的在于将它们联接起来，使之成为同一密钥 k 加密的单一的密文。

设第 i 行的字符数为 n，而第 j 行的字符数为 $\bar{n}$，同理，第 i 行的字符 A 的数目设为 n_A，第 j 行字符 A 的数目为 $\bar{n}_A$，其他依此类推。将两行合并起来的重合指数为

$$IC = \sum_{\xi=A}^{Z}(n_\xi + n_\xi)(n_\xi + \bar{n}_\xi - 1)/[(n+\bar{n})(n+\bar{n}-1)]$$

但

$$\begin{aligned}
&\sum_{\xi=A}^{Z}(n_\xi + \bar{n}_\xi)(n_\xi + \bar{n}_\xi - 1) \\
&\quad = \sum_{\xi=A}^{Z} n_\xi^2 - \sum_{\xi=A}^{Z}\bar{n}_\xi^2 + 2\sum_{\xi=A}^{Z} n_\xi\bar{n}_\xi - \sum_{\xi=A}^{Z} n_\xi - \sum_{\xi=A}^{Z}\bar{n}_\xi \\
&\quad = \sum_{\xi=A}^{Z} n_\xi^2 + \sum_{\xi=A}^{Z}\bar{n}_\xi^2 + 2\sum_{\xi=A}^{Z} n_\xi\bar{n}_\xi - n - \bar{n}
\end{aligned}$$

所以对第 i 行和第 j 行进行试配合时，可将第 i 行的每一元素作加 $k \bmod 26$ 的运算，这里 $k=0,1,2,\cdots,25$。观察使 IC 达到最大的 k 值，由于 $\sum_{\xi=A}^{Z} n_\xi^2$ 和 $\sum_{\xi=A}^{Z}\bar{n}_\xi^2$ 是常量，故其实也就是使 $\sum_{\xi=A}^{Z} n_\xi\bar{n}_\xi$ 达到最大的 k 值 k^*。例如，第 1 行

U U G I W Y J U W A G V U G T F G P T …

当 $k=2$ 时，便得相应的

W W I K Y A L W A C I X W I V H I R V …

现将第 i 行每位后推 k 位得到的新的序列和第 j 行联合，计算 $\sum_{\xi=A}^{Z} n_\xi\bar{n}_\xi$，并将结果列表如下：

$i=1$	104	101	112	127	106	131	104	109	131	209*	104	98	103	
$j=2$	140	112	108	125	103	107	108	124	101	141	139	139	110	$k^*=9$
$i=1$	112	152	142	122	100	112	121	93	139	91	110	128	226*	
$j=3$	129	102	91	124	94	121	120	90	90	116	148	129	134	$k^*=12$
$i=1$	133	125	107	134	125	159	151	108	70	100	159	126	126	
$j=4$	76	114	143	209*	117	92	90	135	95	95	90	116	141	$k^*=16$
$i=1$	94	131	218*	132	74	90	131	105	118	105	92	107	146	
$j=5$	115	113	154	161	125	104	94	111	112	122	107	164	111	$k^*=2$
$i=2$	107	109	121	214*	109	115	103	127	84	125	116	100	99	
$j=3$	96	165	134	143	92	153	107	118	114	147	106	95	137	$k^*=3$

续表

$i=2$	127	148	125	131	99	103	121	194*	114	124	93	115	89	
$j=4$	112	104	117	134	106	138	125	143	105	155	119	118	77	k^*-7
$i=2$	132	99	103	120	130	122	152	129	129	100	96	123	136	
$j=5$	105	108	158	119	91	147	206*	110	86	103	116	98	123	$k^*=19$
$i=3$	129	98	130	119	217*	113	125	94	134	88	95	95	118	
$j=4$	162	122	138	85	133	214	164	108	114	69	123	154	95	$k^*=4$
$i=3$	142	137	99	141	152*	145	94	77	97	122	120	97	155	
$j=5$	118	108	134	209*	109	94	111	128	88	115	150	95	99	$k^*=16$
$i=4$	140	113	113	110	108	111	104	93	141	117	120	128	188*	
$j=5$	125	107	93	120	133	156	112	58	102	149	146	104	145	$k^*=12$

其中 k^* 是 k 的最大值所在位置。

根据以上分析，可能的密钥有

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

现在把分的5段密文重新连接起来，以第1段为基准，第2段后退9步；第2段的第1个字符是F，后退9步应为W；第3段后退12步，第3段的第1个字母为Q，后退12步应为E；第4段后退16步，第4段的第1个字符为U，后退16即是E；第5段的第1个字符为I，后退两步为G，等等。按原来的顺序连接起来得密文：

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

统计26个英文字母出现的频率

A	B	C	D	E	F	G	H	I	J	K
5	0	20	4	9	7	36	7	6	14	25
L	M	N	O	P	Q	R	S	T	U	V
0	3	13	5	22	16	5	0	17	23	26

W	X	Y	Z
12	2	2	1

相应的重合指数 $IC=0.0654$，因而密文 c' 可看做是下面明文 m 通过 Kaiser 变换加密而成。

Success in dealing with unknown ciphers is measured by these four things in the order named perseverance, careful methods of analysis, intuition, luck. The ability at least to read the language of the original text is very desirable but not essential. Such is the opening sentence of Parker Hitt's Manual for the Solution of Military Ciphers.

这一段明文已见于前面的单表置换。

1.3.6 Vernam 密码

$$M=m_1m_2m_3\cdots \qquad m_i=0 \text{ 或 } 1, \quad i=1,2,\cdots$$
$$k=k_1k_2k_3\cdots \qquad k_j=0 \text{ 或 } 1, \quad j=1,2,\cdots$$

即明文和密钥都先将它化为二进制数码，加密过程是：

$$E_k(M)=c_1c_2c_3\cdots$$

其中

$$c_i\equiv(m_i+k_i) \bmod 2 \qquad i=1,2,3,\cdots$$

Vernam 加密算法实际上是在(0,1)字母上运用 Vigenere 加密算法。优点在于$(m_i+k_i) \bmod 2$ 运算非常容易实现。而且

$$c_i+k_i\equiv m_i \bmod 2$$

这里的加法都是指不进位的二进制加法。

若有两组明文 M_1 和 M_2，$M_i=m_1^{(i)}m_2^{(i)}m_3^{(i)}\cdots$，$i=1,2$，通过相同的密钥产生密文，分别用 C_1 和 C_2 表示，设

$$C_1=c_1^{(1)}c_2^{(1)}c_3^{(1)}\cdots$$
$$C_2=c_1^{(2)}c_2^{(2)}c_4^{(2)}\cdots$$
$$c_i^{(j)}\equiv(m_i^{(j)}+k_i) \bmod 2$$
$$j=1,2, \quad i=1,2,\cdots$$

则有

$$c_i^{(1)}+c_i^{(2)}\equiv(m_i^{(1)}+k_i)+(m_i^{(2)}+k_i) \bmod 2$$
$$=m_i^{(1)}+m_i^{(2)} \bmod 2 \quad i=1,2,3,\cdots$$

因而 Vernam 密码只要找到一组和密码相对应的明文便可破译。为了克服它固有的缺点，设想密钥取得非常长，然而这样又将带来诸多麻烦，以至于很难实现。有人想过用两个较短的密钥，用其中之一对另一个进行加密产生出新的密钥，叫做由这两个密钥派生的密钥。例如--密钥长 1000 比特，一密钥长 999 比特，派生密钥长可达 999 000 比特。

1.3.7 Playfair 密码

Playfair 密码的密钥是一个 5×5 矩阵。例如下面的矩阵便是由密钥 FIVESTARS

导出的，其中 J 当做 I 处理。

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

对每一对明文 m_1，m_2 加密方法如下：

① 若 m_1 和 m_2 在同一行，则密文 c_1 和 c_2 分别紧靠 m_1，m_2 右端的字母。其中第一列看做是最后一列的右方。

② 若 m_1 和 m_2 在同一列，则密文 c_1 和 c_2 分别是紧靠 m_1，m_2 下方的字母。这里第一行看做是最后一行的下方。

③ 若 m_1，m_2 不在同一行，也不在同一列，则 c_1 和 c_2 是由 m_1 和 m_2 确定的矩形的其他两角的字母，并且 c_1 和 m_1，c_2 和 m_2 同行。

④ 若 $m_1=m_2$，则插入空字母(比如 Q)于重复字母之间。

⑤ 若明文字母数为奇数，将空字母 Q 加在明文的末端。

例如：M=Playfair cipher was actually invented by wheatstone

先将明文 M 分解为两个字母一对：

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

通过加密可得密文：

QK	BW	IT	VA	AS	OK	VB	IG	IC
TA	WT	QZ	KZ	AW	ES	MA	FK	KE
ZG	IB	CF	RM	PI				

1.3.8 Hill 密码

Hill 加密算法的基本思想是将 l 个明文字母通过线性变换，将它们转换为 k 个密文字母。解密只要作一次逆变换就可以了。密钥就是变换矩阵本身。即

$$M=m_1m_2\cdots m_l$$

$$E_k(M)=c_1c_2\cdots c_l$$

其中

$$\begin{aligned}c_1&=k_{11}m_1+k_{12}m_2+\cdots+k_{1l}m_l\\c_2&=k_{21}m_1+k_{22}m_2+\cdots+k_{2l}m_l\\&\vdots\\c_l&=k_{l1}m_1+k_{l2}m_2+\cdots+k_{ll}m_l\end{aligned}$$

或写成

$$\boldsymbol{C}=\boldsymbol{K}\boldsymbol{M} \bmod n$$

其中

$$\boldsymbol{C}=\begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_l \end{pmatrix},\quad \boldsymbol{M}=\begin{pmatrix} m_1 \\ m_2 \\ \vdots \\ m_l \end{pmatrix}$$

$$K=(k_{ij})_{l\times l}$$

$$\boldsymbol{M}=\boldsymbol{K}^{-1}C \bmod n$$

例如，$l=4$，$n=26$

$$\boldsymbol{K}=\begin{pmatrix} 8 & 6 & 9 & 5 \\ 6 & 9 & 5 & 10 \\ 5 & 8 & 4 & 9 \\ 10 & 6 & 11 & 4 \end{pmatrix}$$

$$\boldsymbol{K}^{-1}=\begin{pmatrix} 23 & 20 & 5 & 1 \\ 2 & 11 & 18 & 1 \\ 2 & 20 & 6 & 25 \\ 25 & 2 & 22 & 25 \end{pmatrix}$$

不难验证

$$\begin{pmatrix} 8 & 6 & 9 & 5 \\ 6 & 9 & 5 & 10 \\ 5 & 8 & 4 & 9 \\ 10 & 6 & 11 & 4 \end{pmatrix}\begin{pmatrix} 23 & 20 & 5 & 1 \\ 2 & 11 & 18 & 1 \\ 2 & 20 & 6 & 25 \\ 25 & 2 & 22 & 25 \end{pmatrix}=\begin{pmatrix} 339 & 416 & 312 & 364 \\ 416 & 339 & 442 & 390 \\ 364 & 286 & 391 & 338 \\ 364 & 494 & 332 & 131 \end{pmatrix}$$

$$\equiv\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \bmod 26$$

$$\boldsymbol{M}=\text{Hill}$$

若采用从 A 到 Z 的 26 个字母依顺序从 0 到 25 编号，$\boldsymbol{M}$ 可分别得数字化后 4 个数字：7，8，11，11，这样

$$\begin{aligned} c_1 &= 8\times 7+6\times 8+9\times 11+5\times 11 \\ &= 258\equiv 24 \bmod 26 \\ c_2 &= 6\times 7+9\times 8+5\times 11+10\times 11 \\ &= 279\equiv 19 \bmod 26 \\ c_3 &= 5\times 7+8\times 8+4\times 11+9\times 11 \\ &= 242\equiv 8 \bmod 26 \\ c_4 &= 10\times 7+6\times 8+11\times 11+4\times 11 \\ &= 283\equiv 23 \bmod 26 \end{aligned}$$

因此　　$\boldsymbol{C}=$ YTIX

反之，已知 $C=$YTIX，则有

$$m_1=23\times24+20\times19+5\times8+23=995\equiv7 \bmod 26$$

$$m_2=2\times24+11\times29+18\times8+23=424\equiv8 \bmod 26$$

$$m_3=2\times24+20\times19+6\times8+25\times23=1051\equiv11 \bmod 26$$

$$m_4=25\times24+2\times19+22\times8+25\times23=1389\equiv11 \bmod 26$$

故得 $M=$Hill

当然字母和数字的对应还可以改变，使得更不容易攻击成功。一般说来，Hill 密码能比较好地抵抗频率法的分析。

1.3.9 密码分析学

前面已提到密码分析学是密码学的一个分支，并举了一些传统密码及其破译技巧的例子。总的说来，密码分析学是研究在不掌握密钥的情况下，利用密码体制的弱点来恢复明文的一门学科。上面所举例子都只是掌握若干密文对密码的破译。对密码的攻击主要可分为以下几种：

(1) 惟密文攻击。即密码分析者仅仅掌握若干密文。不言而喻，这些密文都是用同一个加密算法和密钥加密的。密码分析者的任务是尽可能多地恢复明文或推算出密钥。找出其密钥就可以一劳永逸地解出其他被加密的信息。即已知 $c_i=E_k(m_i)$，$i=1,2,\cdots,l$，推出 $m_1,m_2,\cdots,m_l$ 或推出 l。或从 $c_{l+1}=E_k(m_{l+1})$ 推出 m_{l+1}。

(2) 已知明文攻击。密码分析者不仅掌握若干的密文，还知道对应的明文本身。密码分析者利用它推出用来加密的密钥，或导出加密算法。即已知 $c_i=E_k(m_i)$ 及 m_i，$i=1,2,\cdots,l$，推出 k，或从 $c_{l+1}=E_k(m_{l+1})$ 推出 m_{l+1}。

显然，已知明文攻击较之惟密文攻击有更强的攻击力，掌握的关于该密码的信息也更多。

(3) 选择明文攻击。密码分析者不仅获得若干密文及其相应的明文，而且掌握的明文还加以挑选。不用说，比已知明文攻击条件更苛刻。明文是经过选择的，必然提供了更多可供破译的信息，攻击力更强了。密码分析者利用其来推出加密的密钥或加密算法，或由用同一加密算法及密钥加密的新密文推出对应的明文。

一般说来，好的加密算法是可以公开的，也是不怕公开的。公开了不会从根本上有利于攻击。只要敌方不掌握密钥，谁也没有有效的办法从密文恢复明文。请注意这里指的是有效的算法。即在有限资源的条件下，在允许的时间内将密码破译的算法。所谓"在允许的时间内"是指破译的时间如果太长，破译时已失去意义，这方法也就没用了。有的密码将算法本身也加以保密，目的在于除了给破译者增加许多困难外，主要还在于保护他的研究，不愿与他人共享。

数据加密标准 DES 就是这样。不过新拟议中的美国加密标准还是打算将算法隐蔽起来，这样当然会给攻击者增加麻烦。但目的不是全靠算法保密来达到安全。

实际上所有的加密算法都是可以破译的，后面将介绍的一次一密加密算法除外。理论上一次一密是不可破译的，但它又是不实用的。所以实际上不存在不可破译的密码。如果破译所需的计算能力和时间是现实所不能实现的，则称这样的密码是安全的，或计算

上是安全的。比如破译所需时间要几十个世纪，事实上不可能做到。退一步讲，若保密有效时间为1年，破译要5年，即使破译了也没意义了。

破译一密码需要的计算时间和计算能力的总和(实际上也就是破译算法的时间复杂度和空间复杂度)称为工作因子。

进一步讨论密码分析超出本书的要求，这里介绍若干密码的破译技术，是出于帮助加深理解密码加密算法的需要。

最后顺便谈一谈什么样的密码算是好的？

首先应该是保密的要求，确定加密、解密的工作量，也就是两者要求相匹配，保密要求高的加密算法要求也随之要复杂。

其次密钥和加密算法不要过分复杂，尽可能简单，还要求误差不要扩散以引起混乱。

最后还要求密文不要比原来的明文更长。

第 2 章　数学的准备

近代密码学越来越多地用到数学。在这一章里我们将介绍数论和有限域理论。为了讲有限域理论，还不得不涉及群论，对于熟悉这部分内容的读者可以略去。

近代密码学涉及的数学不仅仅是数论和有限域理论，只是它们在讨论下一章分组密码及以后的公钥密码时是必要的。如介绍 AES 不能没有有限域理论，故先作必要的数学准备。至于还需要的其他数学，则分别散于其他各章。近代密码学用到数学之多，几乎遍及许多数学分支，如概率统计、信息论、数论、有限域理论、复杂性理论、编码理论，甚至于代数几何等都在近代密码学中扮演了重要的角色，不过应该强调指出的是数学在这里仅仅是一种工具，一种不可或缺的工具。

2.1　数　　论

2.1.1　数的 m 进制表示

人们习惯于十进制数的表示，例如 3721，实际上是

$$3721=3\times10^3+7\times10^2+2\times10+1$$

人们习惯于用十进制表示一个数，是因为人的手指是 10 个。但在计算机里用十进制数表示一个数就很不方便。使用任何进制表示一个数应该都是可以的。比如一个数 n 可表示为 m 进制数如下，设 $n=(a_k a_{k-1}\cdots a_1 a_0)_m$：

$$n=a_k m^k+a_{k-1}m^{k-1}+\cdots+a_2 m^2+a_1 m+a_0$$
$$0\leqslant a_i<m,\quad i=0,1,\cdots,k \tag{2-1}$$

其中 m 是一正整数，如何求得 $a_0,a_1,\cdots,a_k$ 呢？方法如下：

令 $n_0=n$，

$$n_1=\left\lfloor\frac{n_0}{m}\right\rfloor=a_k m^{k-1}+a_{k-1}m^{k-2}+\cdots+a_1$$

余数

$$r_1=a_0$$

$\left\lfloor\frac{n}{m}\right\rfloor$ 表示 n 除以 m 后，取其整数部分，也就是比 $\frac{n}{m}$ 小的最大整数。

同样道理

$$n_2=\left\lfloor\frac{n_1}{m}\right\rfloor=a_k m^{k-2}+a_{k-1}m^{k-3}+\cdots+a_2$$

余数

$$r_2=a_1$$

即若 $n_i>0$，令

$$n_{i+1}=\left\lfloor\frac{n_i}{m}\right\rfloor=a_k m^{k-i-1}+a_{k-i}m^{k-i-2}+\cdots+a_{i+1}$$

余数

$$r_{i-1}=a_i,\quad i=0,1,2,\cdots$$

直到

$$n_{k+1}=\left\lfloor\frac{n_k}{m}\right\rfloor=0,\quad a_k=r_{k+1}$$

即到 $n_k<m$ 为止。

数 n 表示成(2.1)式是惟一的。如若不然，设又有

$$n=b_lm^l+b_{l-1}m^{l-1}+\cdots+b_1m+b_0$$

只要证明 $b_j=a_j, j=1,2,\cdots,k$，而且 $l=k$，因有

$$a_km^k+a_{k-1}m^{k-1}+\cdots+a_2m^2+a_1m+a_0=b_lm^l+b_{l-1}m^{l-1}+\cdots+b_1m+b_0$$

不妨设 $l>k$，则有

$$a_0-b_0=(b_lm^l+b_{l-1}m^{l-1}+\cdots+b_1m)-(a_km^k+a_{k-1}m^{k-1}+\cdots+a_1m)$$

上式的右端可用 m 除尽，故

$$m\mid(a_0-b_0)$$

即 m 可除尽(a_0-b_0)的意思。由于 $0\leqslant a_0,b_0<m$，故只有一种可能

$$a_0=b_0$$

即

$$b_lm^l+b_{l-1}m^{l-1}+\cdots+b_1m=a_km^k+a_{k-1}m^{k-1}+\cdots+a_1m$$

上式除以 m 得

$$b_lm^{l-1}+b_{l-1}m^{l-2}+\cdots+b_1=a_km^{k-1}+a_{k-1}m^{k-2}+\cdots+a_1$$

$$b_1-a_1=a_km^{k-1}+a_{k-1}m^{k-2}+\cdots+a_2m-(b_lm^{l-1}+b_{l-1}m^{l-2}+\cdots+b_2m)$$

和前面理由一样有 $m\mid(b_1-a_1)$，$0\leqslant b_1,a_1<m$，故

$$b_1=a_1$$

依此类推，可得：

$$b_2=a_2,\ b_3=a_3,\ \cdots,\ b_k=a_k$$

而且

$$l=k$$

令

$$n=(a_k\quad a_{k-1}\quad\cdots\quad a_1\quad a_0)_m$$

表示数 n 的 m 进制表示。

例如 $n=389, m=5$

$$n_0=n=389$$

$$n_1=\left\lfloor\frac{389}{5}\right\rfloor=77,\quad r_1=4=a_0$$

$$n_2=\left\lfloor\frac{n_1}{5}\right\rfloor=\left\lfloor\frac{77}{5}\right\rfloor=15,\quad r_2=2=a_1$$

$$n_3=\left\lfloor\frac{n_2}{5}\right\rfloor=\left\lfloor\frac{15}{5}\right\rfloor=3,\quad r_3=0=a_2$$

$$n_4=\left\lfloor\frac{n_3}{5}\right\rfloor=\left\lfloor\frac{3}{5}\right\rfloor=0,\quad r_4=3=a_3$$

故

$$389 = 3 \cdot 5^3 + 2 \cdot 5^1 + 4 = (3024)_5$$

计算机里特别适宜于用二进制表示，即 $m=2$，

$$\begin{aligned}
n_0 &= 389 \\
n_1 &= \lfloor n_0/2 \rfloor = \lfloor 389/2 \rfloor = 194, \ c_0 = 1 \\
n_2 &= \lfloor n_1/2 \rfloor = \lfloor 194/2 \rfloor = 97, \ c_1 = 0 \\
n_3 &= \lfloor n_2/2 \rfloor = \lfloor 97/2 \rfloor = 48, \ c_2 = 1 \\
n_4 &= \lfloor n_3/2 \rfloor = \lfloor 48/2 \rfloor = 24, \ c_3 = 0 \\
n_5 &= \lfloor n_4/2 \rfloor = \lfloor 24/2 \rfloor = 12, \ c_4 = 0 \\
n_6 &= \lfloor n_5/2 \rfloor = \lfloor 12/2 \rfloor = 6, \ c_5 = 0 \\
n_7 &= \lfloor n_6/2 \rfloor = \lfloor 6/2 \rfloor = 3, \ c_6 = 0 \\
n_8 &= \lfloor n_7/2 \rfloor = \lfloor 3/2 \rfloor = 1, \ c_7 = 1 \\
n_9 &= \lfloor n_8/2 \rfloor = \lfloor 1/2 \rfloor = 0, \ c_8 = 1
\end{aligned}$$

故

$$389 = 2^8 + 2^7 + 2^2 + 1$$

或

$$389 = (1\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 1)_2$$

2.1.2 数的因数分解

整数 1 只能被 1 除尽。其他整数至少可被两个整数除尽，一个是 1，另一个是这个数本身。只能被 1 和该数自身除尽的数称为素数。

不是 1 且非素数的正整数称为合数。a 除尽 b 表示为 $a \mid b$。以后不特别说明英文字母 a,b,c 等都是表示正整数。若 $a \mid b$，且 $a \mid c$，就说 a 是 b 和 c 的公因子。若 a 是 b 和 c 的公因子，且 b 和 c 的每一个公因子都除尽 a，则称 a 是 b 和 c 的最大公因子，用 $\gcd\{b,c\}$ 或 (b,c) 表示它，即

$$a = \gcd\{b,c\} \quad 或 \quad a = (b,c)$$

例如，$12 = \gcd\{12,60\}$。

若 $a \mid c$，则称 c 是 a 的倍数，若 $a \mid c, b \mid c$，则称 c 是 a 和 b 的公倍数；如果 a 和 b 的公倍数 c 除尽 a 和 b 的任一个公倍数，则称 c 是 a 和 b 的最小公倍数，表示为

$$c = \mathrm{lcm}\{a,b\} \quad 或 \quad c = [a,b]$$

例如，$60 = \mathrm{lcm}\{15,20,30\}$。

定理 2.1 若 $a = qb + r$，则

$$\gcd\{a,b\} = \pm \gcd\{b,r\}$$

证明 设 $d = (a,b)$，$d' = (b,r)$，则

$$d \mid (a - qb), \text{ 即 } d \mid r$$

所以 d 是 b 和 r 的公因数，即 $d \mid d'$。类似地，由 $d' \mid (qb + r)$ 可得 $d' \mid a$，所以 $d' \mid d$。

由 $d \mid d'$，可令 $d' = hd$；同理由 $d' \mid d$，可令 $d = kd'$。即 $d' = hkd'$，因此有

$$hk = 1$$

$$h = k = \pm 1, \ d = \pm d'$$

定理 2.2 每一对不全为零的整数，必有一个正的最大公因数。

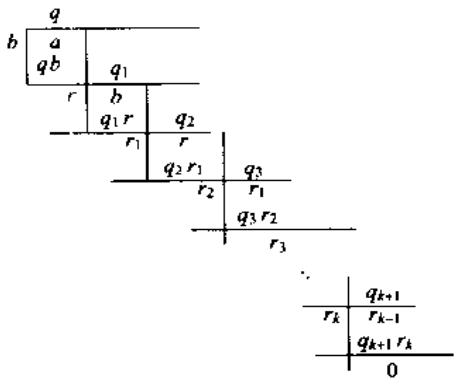
证明 不失一般性，设 a 和 b 是正整数，且 $a>b$。令 $a=qb+r, 0\leqslant r<b$。由定理 2.1，可得

$$(a,b)=(b,r)$$

又令 $b=q_1r+r_1, 0\leqslant r_1<r$。那么

$$(a,b)=(b,r)=(r,r_1)$$

依此类推直到出现余数为零时终止。若用下面辗转相除的形式表示，直观而且一目了然。



$$\begin{aligned}
a&=qb+r, & (a,b)&=(b,r)\\
b&=q_1r+r_1, & (b,r)&=(r,r_1)\\
r&=q_2r_1+r_2, & (r,r_1)&=(r_1,r_2)\\
&\vdots\\
r_{k-1}&=q_{k+1}r_k, & (r_{k-1},r_k)&=r_k
\end{aligned}$$

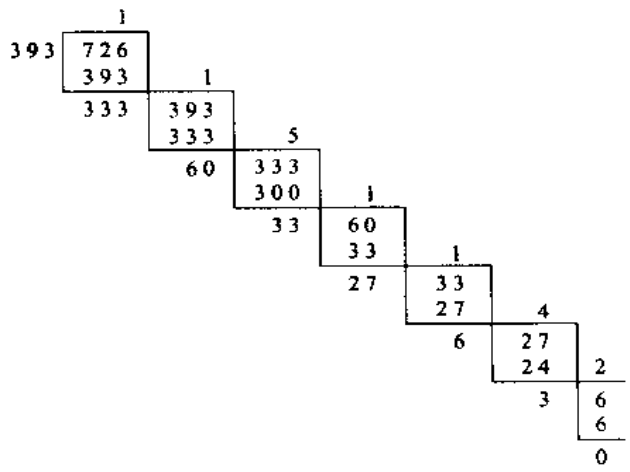
所以

$$(a,b)=(b,r)=(r,r_1)=(r_1,r_2)=\cdots=(r_{k-1},r_k)=r_k$$

$$r>r_1>r_2>\cdots>r_{k-1}>r_k$$

$$(r_{k-1},r_k)=r_k$$

例 2.1 求 gcd{726,393}。



辗转相除，直到出现 r_{k+1}, r_k，即$(r_{k-1}, r_k) = r_k$ 为止。

则 $726 = 1 \times 393 + 333$

$$333 = 726 - 393$$

$393 = 333 + 60$

$$60 = 393 - 333$$

$333 = 5 \times 60 + 33$

$$33 = 333 - 5 \times 60$$

$60 = 33 + 27$

$$27 = 60 - 33$$

$33 = 27 + 6$

$$6 = 33 - 27$$

$27 = 4 \times 6 + 3$

$$3 = 27 - 4 \times 6$$

$6 = 2 \times 3$

所以
$$\begin{aligned}\gcd\{726,393\} &= \gcd\{393,333\}\\ &= \gcd\{333,60\} = \gcd\{60,33\}\\ &= \gcd\{33,27\} = \gcd\{27,6\}\\ &= \gcd\{6,3\} = 3\end{aligned}$$

定理 2.3 若 $a = \gcd\{a,b\}$，则存在整数 p 和 q，使得

$$d = pa + qb$$

证明 设 $\gcd\{r_{k-1}, r_k\} = r_k = d$.

即

$$\begin{aligned}
&r_{k-1} = q_{k+1} r_k && \\
&r_{k-2} = q_k r_{k-1} + r_k, && r_k = r_{k-2} - q_k r_{k-1}\\
&r_{k-3} = q_{k-1} r_{k-2} + r_{k-1}, && r_{k-1} = r_{k-3} - q_{k-1} r_{k-2}\\
&r_1 = q_3 r_2 + r_3, && r_3 = r_1 - q_3 r_2\\
&r = q_2 r_1 + r_2, && r_2 = r - q_2 r_1 + r_2\\
&b = q_1 r + r_1, && r_1 = b - q_1 r\\
&a = qb + r, && r = a - qb
\end{aligned}$$

所以
$$d = r_k = r_{k-2} - q_k r_{k-1}$$

再用 $r_{k-1} = r_{k-3} - q_{k-1} r_{k-2}$ 代入消去 r_{k-1} 得

$$d = r_{k-2} - q_k(r_{k-3} - q_{k-1} r_{k-2}) = (1 + q_k q_{k-1}) r_{k-2} - q_k r_{k-3}$$

接着依次消去 $r_{k-2}, r_{k-3}, \cdots, r$，最后得

$$d = r_k = pa + qb$$

还是以 $3 = \gcd\{726,393\}$ 为例

$$\begin{aligned}
3 &= 27 - 4 \times 6 \quad (\text{以 } 6 = 33 - 27 \text{ 代入})\\
&= 27 - 4 \times (33 - 27)\\
&= -4 \times 33 + 5 \times 27 \quad (\text{以 } 27 = 60 - 33 \text{ 代入})\\
&= -4 \times 33 + 5 \times (60 - 33)
\end{aligned}$$

$$=5\times60-9\times33 \quad (\text{以 } 33=333-5\times60 \text{ 代入})$$
$$=5\times60-9\times(333-5\times60) \quad (\text{以 } 60=393-333 \text{ 代入})$$
$$=-9\times333+50\times(393-333)$$
$$=50\times393-59\times333 \quad (\text{以 } 333=726-393 \text{ 代入})$$
$$=50\times393-59\times(726-393)$$
$$=-59\times726+109\times393$$

所以 $$3=-59\times726+109\times393$$

若 $\gcd\{a,b\}=1$，则称 a 和 b 互素，1 和任何整数都互素。

若 a 和 b 互素，必存在整数 p 和 q，使得

$$1=pa+qb$$

例 2.2 求 $5^{-1}\bmod 13$。

$$\begin{array}{r|l} 5\overline{)13} & 2 \\ \underline{10} & \\ 3\overline{)5} & 1 \\ \underline{3} & \\ 2\overline{)3} & 1 \\ \underline{2} & \\ 1\overline{)2} & 2 \\ \underline{2} & \\ 0 & \end{array}$$

$$13=2\times5+3$$
$$3=13-2\times5$$
$$5=1\times3+2$$
$$2=5-3$$
$$3=2+1$$
$$1=3-2$$
$$1=3-(5-3)=2\times3-5$$
$$=2(13-2\times5)-5$$
$$=2\times13-5\times5$$
$$5(-5)\equiv1 \bmod 13$$

所以 $$5^{-1}\equiv-5\equiv8 \bmod 13$$

例 2.3 求 $11^{-1}\bmod 13$。

$$\begin{array}{r|l} 11\overline{)13} & 1 \\ \underline{11} & \\ 2\overline{)11} & 5 \\ \underline{10} & \\ 1\overline{)2} & 2 \\ \underline{2} & \\ 0 & \end{array}$$

$$13=11+2$$
$$2=13-11$$
$$11=5\times2+1$$
$$1=11-5\times2$$
$$=11-5(13-11)$$
$$=6\times11-5\times13$$

所以 $$11\times6\equiv1 \bmod 13$$
$$11^{-1}\equiv6 \bmod 13$$

定理 2.4 $ac\equiv bc \bmod m$，若 $(c,m)=d$，则

$$a\equiv b \bmod (m/d)$$

证明 $$ac=km+bc$$

若 $$c=c'd,\ m=m'd$$
$$ac'd=km'd+bc'd$$

$$ac' = bc' + km'$$

$$ac' \equiv bc' \bmod m'$$

由于 $(c', m') = 1$

所以 $a \equiv b \bmod m'$

例 2.4 $42 \equiv 7 \bmod 5$

$6 \times 7 \equiv 1 \times 7 \bmod 5,\ (7,5) = 1$

所以 $6 \equiv 1 \bmod 5$

例 2.5 $63 \equiv 3 \bmod 12$

$63 = 3 \times 21$

所以 $3 \times 21 \equiv 3 \bmod 12,\ (3, 12) = 3$

所以 $21 \equiv 1 \bmod (12/3)$

定理 2.5 $ax \equiv b \bmod m$，若 x_1 满足 $ax_1 \equiv b \bmod m$，则对所有 x_2 满足

$$x_2 \equiv x_1 \bmod m$$

恒有

$$ax_2 \equiv b \bmod m$$

定理 2.6 $ax \equiv b \bmod m$ 有解的充要条件是

$$d \mid b, \quad \text{其中} (a,m) = d$$

证明 必要条件：

若 $ax \equiv b \bmod m$

$$ax = km + b, \quad (a,m) = d$$

设

$$a = a'd, m = m'd$$

$$(a', m') = 1$$

则

$$a'dx = km'd + b$$

$$(a'x - km')d = b, \quad \text{所以 } d \mid b$$

充分条件：设 $d \mid b$，$d = (a,m)$，令 $b = b'd$，$a = a'd$，$m = m'd$，由于 a' 和 m' 互素，存在整数 p 和 q，使

$$1 = pa' + qm'$$

$$b = pa'b + qm'b$$

$$= pa'b'a + qm'b'd$$

$$= pb'a + qb'm$$

$$a(pb') \equiv b \bmod m$$

故 $x \equiv pb' \bmod m$ 是 $ax \equiv b \bmod m$ 的解，即 $ax \equiv b \bmod m$ 有解。

定理 2.7 $\begin{cases} x \equiv b_1 \bmod m_1 \\ x \equiv b_2 \bmod m_2 \end{cases}$ 有解的充要条件是

$$b_1 \equiv b_2 \bmod d, \quad d = (m_1, m_2)$$

证明必要条件。

$$x = k_1 m_1 + b_1 = k_2 m_2 + b_2$$

$$k_1 m_1 - k_2 m_2 = b_2 - b_1$$

若 $$d = (m_1, m_2), m_1 = m_1' d, m_2 = m_2' d$$

则 $$(k_1 m_1' - k_2 m_2') d = b_2 - b_1$$

$$d \mid (b_2 - b_1)$$

即 $$b_1 - b_2 \equiv 0 \bmod d$$

$$b_1 \equiv b_2 \bmod d$$

充分条件。设 $b_1 \equiv b_2 \bmod d$，$b_1 = kd + b_2$，n 是任意整数。

$x \equiv b_1 \bmod m_1$，则 $x = k_1 m_1 - b_1$。k_1 是任意整数，只要证明存在 k_1，使 x 满足 $x \equiv b_2 \bmod m_2$

$$x = k_2 m_2 + b_2 = k_1 m_1 + b_1$$

即 $$k_2 m_2' d + b_2' d = k_1 m_1' d + b_1' d$$

$$k_2 m_2' - k_1 m_1' = b_1' - b_2'$$

m_1' 和 m_2' 互素，故

$$m_1' k_1 \equiv b_1' - b_1 \bmod m_1'$$

有解。

定理 2.8 若 $a \mid bc$，$\gcd(a, b) = 1$，则 $a \mid c$。

证明 若 a 和 b 互素，则将 $pa + qb = 1$ 的两端同乘以 c，得

$$pac + qbc = c$$

由假定，$a \mid bc$，所以 a 除尽等式的左端，因此 $a \mid c$。

推论 2.1 若素数 p 除尽 $a_1 a_2 \cdots a_n$，则必存在 k：$1 \leqslant k \leqslant n$，使得 $p \mid a_k$。

证明 若 p 与 a_1 互素，则 $p \mid a_2 \cdots a_n$；若 p 也和 a_2 互素，则 $p \mid a_3 \cdots a_n$；…。

若 p 和 $a_1, a_2, \cdots, a_{n-1}$ 的每一个互素，则最后有 $p \mid a_n$。

定理 2.9 每一个正合数，可表示为正素数的乘积，并且不考虑乘积的顺序时，表示法是惟一的。

证明 若 c 是合数，则存在 a 和 b，使得 $c = ab$；若 a 和 b 是合数，可设 $a = a_1 a_2$，$b = b_1 b_2$，从而 $c = a_1 a_2 b_1 b_2$；继续以上的步骤，直到不能再分解因子为止。这个过程可在有限步内完成，故有

$$c = p_1 p_2 \cdots p_n$$

若这个分解不是惟一的，设

$$c = q_1 q_2 \cdots q_m$$

那么 $p_1 p_2 \cdots p_n = q_1 q_2 \cdots q_m$。

素数 $p_1 \mid q_1 q_2 \cdots q_m$，故存在 q_t，$p_1 \mid q_t$，由于 q_t 也是素数，因此 $p_1 = q_t$。不妨设 $p_1 = q_1$，由 $p_1 \neq 0$ 可知

$$p_2 p_3 \cdots p_n = q_2 q_3 \cdots q_m$$

继续这个步骤，直到取尽所有的 p_i 为止。所以

$$n = m$$

由这个定理，若 c 有素数因子 $p_1, p_2, \cdots, p_n$，那么

$$c = p_1^{a_1} p_2^{a_2} \cdots p_n^{a_n}$$

2.1.3 同余类

若 $m|a-b$，即 $a-b=km$，我们就说 a 和 b 模 m 同余，记以

$$a \equiv b \bmod m$$

m 被称为这个同余式的模。

同余关系和通常意义的相等颇为相似，但实质不同。a 和 b 模 m 同余表示 a 和 b 除以 m 的余数相同，或 $a-b$ 是 m 的倍数。

例如 $5\equiv 2 \bmod 3$，$8\equiv 2 \bmod 3$。

也就是说 $a\equiv b \bmod m$，实际上是说明存在一整数 k，使得

$$a \equiv km + b$$

定理 2.10 模 m 的同余关系满足

(1) 自反的，即 $a\equiv a \bmod m$

(2) 对称的，即若 $a\equiv b \bmod m$，则 $b\equiv a \bmod m$

(3) 传递的，即若 $a\equiv b \bmod m$，$b\equiv c \bmod m$，则 $a\equiv c \bmod m$

证明从略。

定理 2.11 若 $a\equiv b \bmod m$，$c\equiv d \bmod m$，则

(1) $a\pm c\equiv b\pm d \bmod m$

(2) $ac\equiv bd \bmod m$

证明 因 $a\equiv b \bmod m$，$c\equiv d \bmod m$，所以

$$a = km + b,\ c = hm + d$$

$$a \pm c = (k \pm h)m + (b \pm d)$$

从而

$$a \pm c \equiv b \pm d \bmod m$$

同理可证：$ac\equiv bd \bmod m$

定理 2.12 若 $ac\equiv bc \bmod m$，且 c 和 m 互素，则

$$a \equiv b \bmod m$$

证明 由 $ac\equiv bc \bmod m$，可知

$$ac = km + bc$$

即

$$c(a-b) = km$$

由于 c 和 m 互素，因此 $c|k$。设 $k=hc$，则

$$c(a-b) = hcm$$

$$a - b = hm \quad 或 \quad a \equiv b \bmod m$$

定理 2.13 若 $ac\equiv bc(\bmod m)$，$d=(c,m)$，则

$$a\equiv b \bmod m/d$$

证明留作习题。

2.1.4 线性同余方程

若整数 x_1 满足线性同余式

$$ax \equiv b \bmod m$$

即
$$ax_1 \equiv b \bmod m$$

可证明模 m 与 x_1 同余的所有整数都满足这个线性同余式，即若 $x_2 \equiv x_1 \bmod m$，则

$$ax_2 \equiv b \bmod m$$

模 m 和 x_1 同余的整数构成同余式

$$ax \equiv b \bmod m$$

的同余解。

例如：$2x \equiv 3 \bmod 5$

$x \equiv 4 \bmod 5$ 是这个线性同余式的解。

2.1.5 联立同余方程和中国剩余定理

定理 2.14 下列两个同余式

$$x \equiv b_1 \bmod m_1,\ x \equiv b_2 \bmod m_2 \tag{2-2}$$

有一个共同解的充要条件是

$$b_1 \equiv b_2 \bmod d,\ d = (m_1, m_2)$$

证明 设 x_1 满足(2-3)的两个同余式，则

$$x_1 = b_1 + k_1 m_1 = b_2 + k_2 m_2$$

从而
$$k_2 m_2 \equiv b_1 - b_2 \bmod m_1 \tag{2-3}$$

从定理 2.14 可知，(2-4)式中 k_2 存在的充要条件是

$$(m_1, m_2) \mid (b_1 - b_2)$$

对于 n 个联立同余式同样有类似结果。

定理 2.15 对于联立同余式

$$x \equiv b_i \bmod m_i, \quad i = 1, 2, \cdots, n$$

有一共同的解的充要条件是

$$(m_i, m_j) \mid (b_i - b_j), \quad i \neq j, i, j = 1, 2, \cdots, n$$

证明从略。

定理 2.16 若 $a \equiv b \bmod m_1$，$a \equiv b \bmod m_2$，$\cdots$，$a \equiv b \bmod m_k$

则
$$a \equiv b \bmod [m_1, m_2, \cdots, m_k]$$

证明 因 $a \equiv b \bmod m_i$，$i = 1, 2, \cdots, k$，所以

$$m_i \mid (a - b), i = 1, 2, \cdots, k$$

故
$$[m_1, m_2, \cdots, m_k] \mid (a - b)$$

从而
$$a \equiv b \bmod [m_1, m_2, \cdots, m_k]$$

中国剩余定理 设 $m_1, m_2, \cdots, m_k$ 是两两互素的正整数，则

$$x \equiv b_i \bmod m_i, \quad i = 1, 2, \cdots, k \tag{2-4}$$

模$[m_1, m_2, \cdots, m_k]$有惟一解。$[m_1, m_2, \cdots, m_k]$表示 $m_1, m_2, \cdots, m_k$ 的最小公倍数。

证明 令 $M = m_1 m_2 \cdots m_k$

$$M_j = M / m_j = m_1 m_2 \cdots m_{j-1} m_{j+1} \cdots m_k$$

求 y_j 使

$$M_j y_j \equiv 1 \bmod m_j \quad j=1,2,\cdots,k$$

由于$(M_j, m_j)=1$，所以解 y_j 是存在的。令

$$x = b_1 M_1 y_1 + b_2 M_2 y_2 + \cdots + b_k M_k y_k \tag{2-5}$$

可证明 x 便是(2-4)的解。为证明这一点，请注意 $j \neq h$ 时，$m_j \mid M_h$。故

$$M_j \equiv 0 \bmod m_h$$

即 x 中各项除第 h 项外，其余都模 m_h 与 0 同余。又 $M_h y_h \equiv 1 \bmod m_h$

所以

$$x \equiv b_h M_h y_h \bmod m_h$$

$$\equiv b_h \bmod m_h$$

后面证明(2-5)式是模 $m_1 m_2 \cdots m_k$ 的惟一解。如若不然，设 x 和 $\bar{x}$ 是(2-4)式模 M 的两个解，即

$$x \equiv \bar{x} \equiv b_j \bmod m_j \quad j=1,2,\cdots,k$$

那么

$$x - \bar{x} \equiv 0 \bmod m_j$$

即

$$m_j \mid (x-\bar{x})$$

因此

$$M \mid (x-\bar{x})$$

或

$$x - \bar{x} \equiv 0 \bmod M$$

这就证明了对于模 M 的解是惟一的。

定理的证明是构造性的，它提供了求解的一种步骤。

例 2.6 求解

$$x \equiv 1 \bmod 2$$

$$x \equiv 2 \bmod 3$$

$$x \equiv 3 \bmod 5$$

$$M = 2 \times 3 \times 5 = 30$$

$$M_1 = 15,\ M_2 = 10,\ M_3 = 6$$

$$15 y_1 \equiv 1 \bmod 2,\ y_1 = 1$$

$$10 y_2 \equiv 1 \bmod 3,\ y_2 = 1$$

$$6 y_3 \equiv 1 \bmod 5,\ y_3 = 1$$

$$x = 15 + 20 + 18 = 53 \equiv 23 \bmod 30$$

例 2.7 求解

$$x \equiv 0 \bmod 2$$

$$x \equiv 0 \bmod 3$$

$$x \equiv 1 \bmod 5$$

$$x \equiv 6 \bmod 7$$

$$M = 2 \times 3 \times 5 \times 7 = 210$$

$$M_1 = 105, M_2 = 70, M_3 = 42, M_4 = 30$$

$$105 y_1 \equiv 1 \bmod 2,\ y_1 = 1$$

$$70 y_2 \equiv 1 \bmod 3,\ y_2 = 1$$

$$42 y_3 \equiv 1 \bmod 5$$

则 y_3 如下求解

$$42 = 8 \times 5 + 2,\ 5 = 2 \times 2 + 1$$
$$1 = 5 - 2 \times 2 = 5 - 2 \times (42 - 8 \times 5) = 17 \times 5 - 2 \times 42$$
$$42 \times (-2) \equiv 1 \bmod 5$$

故

$$y_3 \equiv -2 \bmod 5,\ \text{即}\ y_3 = 3$$
$$30y_4 \equiv 1 \bmod 7$$

则 y_4 如下求解

$$30 = 7 \times 4 + 2,\ 7 = 3 \times 2 + 1$$
$$1 = 7 - 3 \times 2 = 7 - 3 \times (30 - 7 \times 4) = 13 \times 7 - 3 \times 30$$
$$30 \times (-3) \equiv 1 \bmod 7$$

故

$$y_4 \equiv 4 \bmod 7$$
$$x = 3 \times 42 + 6 \times 4 \times 30$$
$$= 126 + 720$$
$$= 846 \equiv 6 \bmod 210$$

2.1.6 欧拉(Euler)定理和费尔玛(Fermat)定理

(1) 欧拉函数

所有模 m 和 r 同余的整数组成一个剩余类$[r]$。显然这个剩余类中的任何一个数可以确定这个剩余类。每个整数总是和 m 个数

$$0,1,\cdots,|m|-1$$

中的一个数同余,而且仅和其中的一个数同余。$0,1,\cdots,|m|-1$ 中任意两个数模 m 不同余。

一组整数 $r_1,r_2,\cdots,r_m$,分别属于不同的同余类时,则称它构成一个模 m 的完全剩余集。例如,$0,1,\cdots,|m|-1$ 便是一组完全的正剩余集。

剩余类$[r]$中的每一个数可表示为

$$r+km \quad k=0,\pm1,\pm2,\cdots$$

所以剩余类$[r]$中的每一个数和 m 互素的充要条件是 r 和 m 互素。和 m 互素的同余类的数目用 $\phi(m)$ 表示之,称为是 m 的欧拉函数。

定理 2.17 若 m_1 和 m_2 互素,则

$$\phi(m_1m_2)=\phi(m_1)\phi(m_2)$$

证明 设 x 是一个正整数,满足

$$0 \leqslant x < |m_1m_2|$$

设

$$x \equiv r_1 \bmod m_1,\ x \equiv r_2 \bmod m_2$$

其中

$$0 \leqslant r_1 < m_1,\ 0 \leqslant r_2 < m_2$$

则 r_1,r_2 为 x 所惟一确定。

反之,给定 r_1,r_2 可惟一确定 x,即 x 和一对数偶(r_1,r_2)一一对应。x 和 m_1m_2 互素,

当且仅当 x 分别和 m_1, m_2 都互素，同时 x 和 m_i 互素的充要条件是 r_i 和 m_i 互素，$i=1,2$。

这就证明了与 m_1m_2 互素的数 x 的数目等于数偶 (r_1, r_2) 的数目，其中 r_1 和 m_1 互素，r_2 和 m_2 互素。比 m_1 小而与 m_1 互素的 r_1 的数目为 $\phi(m_1)$，小于 m_2 而与 m_2 互素的 r_2 的数目为 $\phi(m_2)$，这样的数偶 (r_1, r_2) 的数目为 $\phi(m_1)\phi(m_2)$。

定理 2.18 若 $m=p_1^{a_1}p_2^{a_2}\cdots p_k^{a_k}$，则

$$\phi(m)=m\left(1-\frac{1}{p_1}\right)\left(1-\frac{1}{p_2}\right)\cdots\left(1-\frac{1}{p_k}\right)$$

证明 首先证明对于素数 p 有

$$\phi(p^a)=p^a\left(1-\frac{1}{p}\right)$$

在区间 $0\leqslant x<p^a$ 上的整数，或是 p 的倍数，或与 p^a 互素，其中 p 的倍数为

$$0, p, 2p, \cdots, (p^{a-1}-1)p$$

共为 p^{a-1} 个。所以和 p^a 互素的数目为

$$p^a-p^{a-1}=p^a(1-1/p)$$

根据定理 2.17 得

$$\begin{aligned}\phi(m)&=p_1^{a_1}\left(1-\frac{1}{p_1}\right)p_2^{a_2}\left(1-\frac{1}{p_2}\right)\cdots p_k^{a_k}\left(1-\frac{1}{p_k}\right)\\&=m\left(1-\frac{1}{p_1}\right)\left(1-\frac{1}{p_2}\right)\cdots\left(1-\frac{1}{p_k}\right)\end{aligned}$$

(2) 欧拉定理

定理 2.19 若 a 和 m 互素，则

$$a^{\phi(m)}\equiv 1 \bmod m$$

证明 设 $\phi(m)=k$，并令 $r_1, r_2, \cdots, r_k$ 是与 m 互素且模 m 的剩余集。显然，由于 a 和 m 互素，$ar_1, ar_2, \cdots, ar_k$ 也是和 m 互素，且两两不相同余，否则若

$$ar_i\equiv ar_j \bmod m$$

则

$$r_i\equiv r_j \bmod m$$

与假定矛盾。所以

$$a^k r_1r_2\cdots r_k\equiv r_1r_2\cdots r_k \bmod m$$

由于 $r_1, r_2, \cdots, r_k$ 与 m 互素，故得

$$a^k\equiv 1 \bmod m$$

推论 2.2(Fermat) 若 p 是素数，$(a,p)=1$，则

$$a^{p-1}\equiv 1 \bmod p$$

由此推论可得

定理 2.20(Fermat) 若 p 是素数，则

$$a^p\equiv a \bmod p$$

从 $a^{\phi(m)}\equiv 1 \bmod m$ 可知，若 a 和 m 互素，则相对于模 m，a 的逆元素为 $a^{\phi(m)-1}$，即 $a^{-1}\equiv a^{\phi(m)-1} \bmod m$。

例 2.8 求解 $3^{102}\equiv ? \bmod 11$。

因为 $3^{10}\equiv 1 \bmod 11$，$(3^{10})^{10}\equiv 1 \bmod 11$

即 $3^{100}\equiv 1 \bmod 11$

所以 $3^{102}\equiv 3^2\equiv 9 \bmod 11$

即 $3^{102}\equiv 9 \bmod 11$

例 2.9 $2^9=512\equiv 6 \bmod 11$，所以 6 是 2 模 11 的逆。

例 2.10 若 a、b 是正整数，p 是素数，且 a 与 p 互素，则

$$ax\equiv b \bmod p$$

的解是 $x\equiv a^{p-2}b \bmod p$

(3) 定理应用

$$ax\equiv b \bmod m$$

有解的充要条件是 $d|b$，其中 $d=(a,m)$。且当求得 $\bar{a}\equiv a^{-1} \bmod m$ 时，可利用它求问题的解：

$$x\equiv \bar{a}b \bmod m$$

$\bar{a}=a^{-1} \bmod m$ 表示 $\bar{a}a\equiv 1 \bmod m$。

例 2.11 $7x\equiv 22 \bmod 31$

$$7^{-1}\equiv 9 \bmod 31$$

所以 $$x\equiv 9\times 22\equiv 198\equiv 12 \bmod 31$$

2.1.7 威尔逊(Wilson)定理

定理 2.21(Wilson) 若 p 是素数，则

$$(p-1)!\equiv -1 \bmod p$$

反之，若整数 p 满足

$$(p-1)!\equiv -1 \bmod p$$

则 p 是一个素数。

证明 必要性。

$p=2$ 时，$(p-1)!\equiv 1\equiv -1 \bmod 2$。定理为真。$p>2$ 时，对整数 a：$1\leqslant a<p$，总存在 a 的逆 $\bar{a}$，满足 $a\bar{a}\equiv 1 \bmod p$。实际上，它是 $ax\equiv 1 \bmod p$ 的解。下面证明当且仅当 $x\equiv 1$ 或 $x\equiv -1 \bmod p$ 时，它的逆是自身。因若

$$x^2\equiv x\cdot x\equiv 1 \bmod p$$

则 $$p|(x^2-1)$$

所以 $$p|(x-1) \quad 或 \quad p|(x+1)$$

即 $$x\equiv 1 \bmod p \quad 或 \quad x\equiv -1 \bmod p$$

这就证明了 $x^2\equiv 1 \bmod p$ 当且仅当

$$x\equiv 1 \bmod p \quad 或 \quad x\equiv -1 \bmod p$$

所以，2，3，…，$p-2$ 间的 $p-3$ 个数分成 $(p-3)/2$ 时，互为逆元素。故

$$(p-2)!\equiv 1 \bmod p$$

$$(p-1)!\equiv (p-1)(p-2)!\equiv p-1\equiv -1 \bmod p$$

充分性。假定

$$(p-1)! \equiv -1 \bmod p$$

但 $p=ab$，其中

$$1<a<p, \quad 1<b<p$$

因 $a<p$，故 $a\mid(p-1)!$，根据假定

$$(p-1)! \equiv -1 \bmod p$$

$$(p-1)!+1 \equiv 0 \bmod p$$

所以 $$p\mid[(p-1)!+1]$$

即 $$a\mid[(p-1)!+1]$$

由$[(p-1)!+1]-(p-1)!=1$可知$a\mid 1$，这跟$a>1$的假定矛盾，故p是素数。

2.1.8 平方剩余

只要考虑下列同余式

$$x^2 \equiv 1,\ x^2 \equiv 2,\ x^2 \equiv 3,\ x^2 \equiv 4 \bmod 5$$

不难发现

$$x=1,\ x=4 \text{ 满足 } x^2 \equiv 1 \bmod 5$$

$$x=2,\ x=3 \text{ 满足 } x^2 \equiv 4 \bmod 5$$

不存在整数满足

$$x^2 \equiv 2 \bmod 5 \text{ 或 } x^2 \equiv 3 \bmod 5$$

求解 $x^2 \equiv a \bmod p$，其中a与p互素，且p是奇素数。若问题有解，则称a是模p的平方剩余，否则称为非平方剩余。

引理 2.1 若p是奇素数，$p\nmid a$，则

$$x^2 \equiv a \bmod p$$

或无解或有两个模p不同余的解。

证明 若$x^2 \equiv a \bmod p$有解x'，则不难验证$-x'$是另一个与x'不同余的解。因

$$(-x')^2 = x'^2 \equiv a \bmod p$$

同时 $$x' \not\equiv -x' \bmod p$$

否则 $$2x' \equiv 0 \bmod p$$

这跟p是奇素数，且$p\nmid a$的假定相矛盾。

再没有其他不同余的解了。如若x''和x'都是问题$x^2 \equiv a \bmod p$的解，则

$$x'^2 \equiv x''^2 \equiv a \bmod p$$

从而 $$x'^2-x''^2=(x'-x'')(x'+x'') \equiv 0 \bmod p$$

故 $$p\mid(x'-x'') \text{ 或 } p\mid(x'+x'')$$

即 $$x' \equiv x'' \bmod p \text{ 或 } x' \equiv -x'' \bmod p$$

故只有两个不同余的解。

定理 2.22 若p是奇素数，则整数$1,2,\cdots,p-1$中正好有$(p-1)/2$个是模p的平方剩余，其余的$(p-1)/2$个是非平方剩余。

证明 为从$1,2,\cdots,p-1$中求出模p的所有平方剩余，计算$1,2,\cdots,p-1$的平方模p的最小正剩余，因为共有$p-1$个非零剩余，且

$$x^2 \equiv a \bmod p$$

的解的数目或为零或有两个，故正好有$(p-1)/2$个模p的平方剩余在$1,2,\cdots,p-1$中，其余$(p-1)/2$个为非平方剩余。

2.2 群　　论

2.2.1 群的概念

定义 2.1 给定一集合$G=\{a,b,\cdots\}$和该集合上的运算$*$，满足下列四条件的代数系统$<G,*>$称为群：

(a) 封闭性：若$a,b\in G$，则存在$c\in G$，使$a*b=c$

(b) 结合律成立：$a,b,c\in G$，恒有$(a*b)*c=a*(b*c)$

(c) 存在单位元素e：即存在$e\in G$，对$\forall a\in G$，恒有$a*e=e*a=a$

(d) 存在逆元素：对于$\forall a\in G$，恒有$b\in G$使$a*b=b*a=e$

元素b称为a的逆元素，用a^{-1}表示，即$b=a^{-1}$。

若对$\forall a,b\in G$，有$a*b=b*a$，则称$<G,*>$为阿贝尔(Abel)群。为了简便起见，$a*b$简记为ab。由于$(ab)c=a(bc)$，故可记为abc。

例 2.12 $G=\{1,-1\}$在乘运算下构成群，而且是阿贝尔群。

① 封闭性：$(1)(1)=(-1)(1)=-1$，$(1)(1)=1$，$(-1)(-1)=1$。

② 显然结合律成立。

③ 单位元$e=1$。

④ 逆元存在。1的逆元素为1本身，即$(1)(1)=1$，或$1^{-1}=1$。同理$(-1)^{-1}=-1$。

例 2.13 $G=\{0,1,\cdots,n-1\}$，关于mod n的加法是一个阿贝尔群。

① 封闭性：$a\in G$，$b\in G$，$(a+b) \bmod n$只能在G中。

② 结合律显然成立。

③ 单位元$e=0$。

④ $a\in G$，$a^{-1}=n-a$，$a+a^{-1}=n\equiv 0 \bmod n$。

例 2.14 $G=\{1,2,\cdots,p-1\}$，p是一素数，则mod p关于乘法构成阿贝尔群。

① 封闭性：$a\in G$，$b\in G$，$ab=ba\in G$。

② 结合律成立，而且$ab\equiv ba \bmod p$。

③ 单位元$e=1$。

④ 逆元素。$a\in G$，$\gcd\{i,p\}=1$，故存在整数l和m使$la+mp=1$，故

$$la \equiv 1 \bmod p,\ l=a^{-1}$$

例 2.15 $G=\left\{\begin{pmatrix}a & b\\ c & d\end{pmatrix}\middle|\ \text{其中}a,b,c,d\text{都是实数，且存在}\begin{pmatrix}a & b\\ c & d\end{pmatrix}^{-1}\right\}$关于矩阵乘法，$<G,*>$成群，但不是阿贝尔群。

单位元素$e=\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}$。

例 2.16 已知 T_α：

$$T_\alpha:\begin{cases}\xi = x\cos\alpha + y\sin\alpha\\ \eta = -x\sin\alpha + y\cos\alpha\end{cases}$$

或

$$\begin{pmatrix}\xi\\ \eta\end{pmatrix} = \begin{pmatrix}\cos\alpha & \sin\alpha\\ -\sin\alpha & \cos\alpha\end{pmatrix}\begin{pmatrix}x\\ y\end{pmatrix}$$

设

$$T_\beta:\begin{pmatrix}x\\ y\end{pmatrix} = \begin{pmatrix}\cos\beta & \sin\beta\\ -\sin\beta & \cos\beta\end{pmatrix}\begin{pmatrix}x'\\ y'\end{pmatrix}$$

定义 $T_\alpha \cdot T_\beta$ 为

$$\begin{aligned}\begin{pmatrix}\xi\\ \eta\end{pmatrix} &= \begin{pmatrix}\cos\alpha & \sin\alpha\\ -\sin\alpha & \cos\alpha\end{pmatrix}\begin{pmatrix}x\\ y\end{pmatrix}\\ &= \begin{pmatrix}\cos\alpha & \sin\alpha\\ -\sin\alpha & \cos\alpha\end{pmatrix}\begin{pmatrix}\cos\beta & \sin\beta\\ -\sin\beta & \cos\beta\end{pmatrix}\begin{pmatrix}x'\\ y'\end{pmatrix}\\ &= \begin{pmatrix}\cos\alpha\cos\beta - \sin\alpha\sin\beta & \cos\alpha\sin\beta + \sin\alpha\cos\beta\\ -\sin\alpha\cos\beta - \cos\alpha\sin\beta & \cos\alpha\cos\beta - \sin\alpha\sin\beta\end{pmatrix}\begin{pmatrix}x'\\ y'\end{pmatrix}\\ &= \begin{pmatrix}\cos(\alpha+\beta) & \sin(\alpha+\beta)\\ -\sin(\alpha+\beta) & \cos(\alpha+\beta)\end{pmatrix}\begin{pmatrix}x'\\ y'\end{pmatrix}\end{aligned}$$

即

$$T_\alpha \cdot T_\beta = T_{\alpha+\beta}$$

① 封闭性，由 $T_\alpha \cdot T_\beta = T_{\alpha+\beta}$ 可知

② 结合律可由 $T_\alpha(T_\beta T_\gamma) = (T_\alpha T_\beta)T_\gamma = T_{\alpha+\beta+\gamma}$ 得证

③ $e = T_0$

④ T_α 的逆元素为 $T_{-\alpha}$。

故坐标变换 T_α 成群，而且元素个数为无穷。

2.2.2 群的性质

1. 群的单位元 e 是惟一的。

用反证法。若 e_1 和 e_2 都是群 G 的单位元，则根据定义有 $e_1 e_2 = e_1$，$e_1 e_2 = e_2$，所以 $e_1 = e_2$。

2. $a, b, c \in G$，若 $ab = ac$，则 $b = c$；若 $ab = cb$，则 $a = c$。

现证其一。若 $ab = ac$，用 a^{-1} 互乘等式两端得 $a^{-1}(ab) = a^{-1}(ac)$，根据结合律

$$a^{-1}(ab) = (a^{-1}a)b = (a^{-1}a)c$$

所以

$$b = c$$

3. G 的每一元素的逆元是惟一的。

仿 1. 用反证法。

4. 群 $<G, *>$ 的元素 a，若

$$(a)^n = (\underbrace{a * a * \cdots * a}_{n\text{次}}) = e$$

使上式成立的最小正整数 n 称为元素 a 的阶。

定理 2.23 有限群 G 的每一元素的阶是有限的。

证明 设 G 的元素个数为 g，作 $a,a^2,\cdots,a^g,a^{g+1}$，由于 $a^h\in G,h=1,2,\cdots,g,g+1$，而 G 的元素只有 g 个，故存在 $1\leqslant i<j\leqslant g+1$ 使得

$$a^i = a^j$$

即

$$a^{j-i} = e$$

故 a 的阶是有限的。

定义 2.2 $<G,*>$是一群，$H\subset G$，$<H,*>$也是一群，则称 H 为 G 的一个子群。

$a\in G$，设 a 的阶为 n，则 $H=\{a,a^2,\cdots,a^n\}$是 G 的子群，且为阿贝尔群。

若$<H,*>$是群$<G,*>$的子群。设群$<G,*>$的阶为 n，$<H,*>$的阶为 m，则 $m|n$，即 m 可整除 n。

证明 $b\in G$，$bH\triangleq\{b*h,h\in H\}$。

先证 bH 的阶等于 m，即 bH 的阶等于 H 的阶。因若 $bh_1=bh_2$ 则 $h_1=h_2$。

再证 $b_1H=b_2H$ 或 $b_1H\cap b_2H=\varnothing$。若 $b_1h_1=b_2h_2$，则 $b_1=b_2h_2h_1^{-1}$，即 $b_1H\subset b_2H$。同理可证 $b_2=b_1h_1h_2^{-1}$，即 $b_2H\subset b_1H$，所以 $b_1H=b_2H$。若取 $b_1=e$，取 b_2 属于 $G\backslash b_1H$，则 $b_2H\cap H=\varnothing$，取 $b_2\in G\backslash(H\cup b_1H)$则 $b_2H\cap(H\cup b_1H)=\varnothing$。依此反复进行直至

$$G = H\cup b_2H\cup\cdots\cup b_tH$$

即

$$n=km$$

2.3 有限域理论

2.3.1 域的概念

定义 2.3 F 是至少含有两个元素的集合，对 F 定义了两种运算“+”和“*”，并且满足以下三条件的代数系统$<F,+,*>$称为域。

(1) F 的元素关于运算“+”构成阿贝尔群，设其单位元为 0。

(2) $F\backslash\{0\}$关于运算“*”构成阿贝尔群。

(3) 对于 $a,b,c\in F$，分配律成立。即

$$(a+b)*c=a*c+b*c$$

$$c*(a+b)=c*a+c*b$$

若域 F 的元素有限个，则称之为有限域或伽罗瓦(Galois)域。$F\backslash\{0\}$表示集合 F 除去元素$\{0\}$后的元素。

例 2.17 实数的全体、复数的全体关于通常的加法、乘法都是域，分别称为实数域和复数域。

F 集合的元素除去元素 0 外关于乘法成阿贝尔群，其单位元素用 1 表示。在不引起混乱的情况下就用 0 和 1 分别表示 F 关于“+”和 $F\backslash\{0\}$关于“*”运算构成的交换群(阿贝尔群)的单位元。

若 p 是素数，则 $F=\{0,1,\cdots,p-1\}$ 在 mod p 的意义下关于加法“+”，乘法“·”成域。

所谓 mod p 意义下相等，指的是 $a=b \bmod p$，即 $a-b=sp$。

F 关于加法运算构成交换群，容易给予证明。要证明$\{1,2,\cdots,p-1\}$关于乘法运算构成阿贝尔群，只要证属于集合中的任一元素 a 存在逆元素就可。由于 p 是素数，a 和 p 的公因子为 1，故惟一存在整数 b 和 c，使得

$$ab+pc=1$$

所以

$$ab=1 \bmod p$$

b 就是 a 的逆元素。

·	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

·	1	2	3	4
1	1	2	3	4
2	2	4	1	3
3	3	1	4	2
4	4	3	2	1

上例验证了 $p=5$ 时，$F=\{0,1,2,3,4\}$在 mod 5 意义下关于“+”，“·”运算成域。

然而 p 不为素数时则不然，这只要从下面的例子便可看出。$p=6$ 时有

+	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

·	1	2	3	4	5
1	1	2	3	4	5
2	2	4	0	2	4
3	3	0	3	0	3
4	4	2	0	4	2
5	5	4	3	2	1

p 是素数，则 $F=\{0,1,2,\cdots,p-1\}$在 mod p 的意义下关于“+”，“·”运算构成的域用 $GF(p)$表示。

2.3.2 伽罗瓦域 $GF(p^n)$

系统讨论 $GF(p^n)$域属于近世代数的内容，下面仅简略介绍。

多项式

$$p(x)=a_0+a_1x+\cdots+a_kx^k,\quad a_i\in F,\ i=0,1,\cdots,k$$

如果 F 的元素个数为 p，则不同的多项式 $p(x)$ 个数为 p^{k+1}，即 $p(x)$ 和 $k+1$ 位 p 进制数 $a_ka_{k-1}\cdots a_2a_1a_0$ 一一对应。

$$a_i\in F\quad i=0,1,\cdots,p$$

则 $p(x)$ 和 $(a_0,a_1,\cdots,a_p)$ 一一对应。特别当

$$F=GF(2)=\{0,1\}$$

设

$$p_{000}(x)=0$$
$$p_{100}(x)=1$$
$$p_{010}(x)=x$$
$$p_{110}(x)=1+x$$
$$p_{001}(x)=x^2$$
$$p_{101}(x)=1+x^2$$
$$p_{011}(x)=x+x^2$$
$$p_{111}(x)=1+x+x^2$$

这些多项式的乘积可得次方超过 2 的多项式。若引进在 $GF(2)$ 不可化约多项式 $m(x)=1+x+x^3$。

则 mod $m(x)$，上面多项式除去 $p_{000}(x)$ 外关于乘法构成交换群。

	1	x	$1+x$	x^2	$1+x^2$	$x+x^2$	$1+x+x^2$
1	1	x	$1+x$	x^2	$1+x^2$	$x+x^2$	$1+x+x^2$
x	x	x^2	$x+x^2$	$1+x$	1	$1+x+x^2$	$1+x^2$
$1+x$	$1+x$	$x+x^2$	$1+x^2$	$1+x+x^2$	x^2	1	x
x^2	x^2	$1+x$	$1+x+x^2$	$x+x^2$	x	$1+x^2$	1
$1+x^2$	$1+x^2$	1	x^2	x	$1+x+x^2$	$1+x$	$x+x^2$
$x+x^2$	$x+x^2$	$1+x+x^2$	1	$1+x^2$	$1+x$	x	x^2
$1+x+x^2$	$1+x+x^2$	$1+x^2$	x	1	$x+x^2$	x^2	$1+x$

以 $(1+x^2)(x+x^2)$ 为例

$$(1+x^2)(x+x^2)=x+x^2+x^3+x^4$$

$$\begin{array}{r|l} & x+1 \\ \hline x^3+x+1 & x^4+x^3+x^2+x \\ & x^4+\ \ +x^2+x \\ \hline & x^3 \\ & x^3+\ \ x+1 \\ \hline & x+1 \end{array}$$

所以 $$(1+x^2)(x+x^2)\equiv 1+x \bmod (x^3+x+1)$$

又如 $$(1+x+x^2)(1+x+x^2)=1+x+x^2+x+x^2+x^3+x^2+x^3+x^4$$
$$=1+x^2+x^4$$

$$x^3+x+1\overline{)\begin{array}{r}x\\ x^4+x^2+x\\ \hline x^4+x^2+x\\ \hline x+1\end{array}}$$

所以 $$(1+x+x^2)(1+x+x^2)\equiv 1+x \bmod (x^3+x+1)$$

而且

$$x^3 \equiv 1+x$$
$$x^4 \equiv x+x^2$$
$$x^5 \equiv x^2+x^3 \equiv x^2+x+1$$
$$x^6 \equiv x^3+x^2+x \equiv x^2+1$$
$$x^7 \equiv x^3+x \equiv 1$$

可见$\{0,1,x,1+x,x^2,1+x^2,x+x^2,1+x+x^2\}$在 mod$(x^3+x+1)$的意义下，关于加法"+"和乘法"·"构成有限域。更确切地说：在 $GF(2)$域上，mod $m(x)$构成有限域。可表示成 $GF(2,x^3+x+1)$。这里 $m(x)=x^3+x+1$。有的也表示成 $F_2[x]/m(x)$。

将上面的讨论进一步推广到次方不超过 4，系数在 $GF(2)$的 $2^4=16$ 个多项式。

定义 2.4 一个多项式在 F 域上，若不能表为两个次方低的多项式之积，则称该多项式在 F 域上是不可化约的。

例如：因在 $GF(2)$域$(1+x)^2=1+x^2$，故 $1+x^2$ 在 $GF(2)$域上不是不可化约的。虽然 $1+x^2$ 在实数域是不可化约的，在虚数域也是可化约的。

可以证明 $p(x)=1+x+x^5$ 在 $GF(2)$是不可化约的，证明将在后面给出。若不特别声明，本章以后都假定在 $GF(2)$域上讨论。

定理 2.24 $p(x)$是 4 次不可化约的多项式，则次方不超过 3 的某一多项式 $A(x)$，必存在惟一的 $A^{-1}(x)$，使得

$$A(x)A^{-1}(x) \equiv 1 \bmod p(x)$$

证明 若 $A(x)B_1(x)=A(x)B_2(x) \bmod p(x)$，其中 $B_1(x),B_2(x)$属于

$$F: 1,x,x+1,x^2,\cdots,x^3+x^2+x+1$$

中的一个，则有 $$B_1(x) \equiv B_2(x)$$

因为 $$p(x) \mid A(x)[B_1(x)-B_2(x)]$$

由于 $A(x)$和 $B_1(x),B_2(x)$的次方都小于 $p(x)$的次方 4，这只能导致 $B_1(x)\equiv B_2(x)$，否则应有

$$p(x) \mid A(x), \quad p(x) \mid [B_1(x)-B_2(x)]$$

这是不可能的。若 $B(x)$遍历 F 中所有元素，由于 mod $p(x)$，F 关于乘法换成群，故 $A(x)B(x)$的结果是 F 的元素的某一种排列，即存在 $B(x)$使 $A(x)B(x)\equiv 1 \bmod p(x)$。$A^{-1}(x)\equiv B(x) \bmod p(x)$。

例 2.18 设不可化约多项式 $p(x)=x^4+x+1$。

(1) 求 x 的逆 x^{-1}，使 $xx^{-1}\equiv 1 \bmod p(x)$。

由于

$$1+x+x^4 \equiv 0 \bmod p(x)$$

$$1 \equiv x + x^4 \equiv x(1+x^3), \mod p(x)$$

所以
$$x^{-1} \equiv 1 + x^3, \mod(1+x+x^4)$$

(2) 求 $x+x^2$ 的逆

$$\begin{array}{r|l} & x^2+x+1 \\ \hline x^2+x & x^4+x+1 \\ & x^4+x^3 \\ \hline & x^3+x+1 \\ & x^3+x^2 \\ \hline & x^2+x+1 \\ & x^2+x \\ \hline & 1 \end{array}$$

所以
$$x^4+x+1 = (x^2+x)(x^2+x+1)+1$$

即
$$(x^2+x)(x^2+x+1) \equiv 1 \bmod (x^4+x+1)$$

所以
$$(x^2+x)^{-1} = x^2+x+1$$

另一种方法：设

$$(x^2+x)^{-1} = ax^2+bx+c$$

$$(x^2+x)(ax^2+bx+c) \equiv 1 \bmod (x^4+x+1)$$

$$ax^4+(b+a)+(c+b)x^2+cx \equiv 1$$

或
$$a(x+1)+(b+a)x^3+(b+c)x^2+cy \equiv 1$$

所以
$$a = 1$$

$$a+c = 0$$

$$b+c = 0$$

所以
$$a = 1, \quad c = 1, \quad b = 1$$

即
$$(x+x^2)^{-1} = x^2+x+1$$

在 $GF[2, x^4+x+1]$ 中

$$x^0 = 1, x^1 = x, (x)^2 = x^2, (x)^3 = x^3,$$

$$(x)^4 = x+1, x^5 = x^2+x, x^6 = x^3+x^2$$

$$x^7 = x^4+x^3 = x^3+x+1$$

$$x^8 = x^4+x^2+x = x^2+1$$

$$x^9 = x^3+x$$

$$x^{10} = x^4+x^2 = x^2+x+1$$

$$x^{11} = x^3+x^2+x$$

$$x^{12} = x^4+x^3+x^2 = x^3+x^2+x+1$$

$$x^{13} = x^4+x^3+x^2+x = x^3+x^2+1$$

$$x^{14} = x^4+x^3+x = x^3+1$$

$$x^{15} = x^4+x = 1$$

$GF[2, x^4+x+1]$ 构成一阶数为 2^4 的 Galois 域，表示成 $GF(2^4)$。

2.3.3 有限域的基本理论

定理 2.25 设 $p(x)$ 是系数在 $GF(p)$ 域的 m 次不可化约多项式，则次方 $\leqslant m-1$，系

数在 $GF(p)$ 上所有多项式关于 mod $p(x)$ 构成一阶为 p^m 的 Galois 域 $GF(p^m)$。

$GF(p^m)$ 的元素可表示为元素在有限域 $GF(p)$ 的 m 维向量，也可以看成是系数在 $GF(p)$ 的多项式类 mod $p(x)$ 的同余类。

令 α 满足 $p(\alpha)=0$，故在 $GF(p^m)$ 域方程 $p(x)=0$ 存在一个根。

$GF(p^m)$ 包含 α 的所有次方 $\leqslant m-1$ 的多项式，其系数在 $GF(p)$。即

$$\alpha_i = \alpha_0 \alpha = \alpha_0 \alpha^0 + \cdots + \alpha_{m-1}\alpha^{m-1} \in GF(p^m)$$

其中 $\alpha_i \in GF(p)$，$i=0,1,2,\cdots,m-1$，且

$$p(\alpha) = 0$$

例 2.19 设在 $GF(3)$ 上不可化约的多项式

$$p(x) = x^2 + x + 2$$

则有 $\alpha^n = 1$，$(\alpha)^1 = \alpha$

在 $GF(3)$ 上有

$$\alpha^2 = -\alpha - 2 = 2\alpha + 1$$
$$\alpha^3 = 2\alpha^2 + \alpha = 2(2\alpha+1) + \alpha = \alpha + 2 + \alpha = 2\alpha + 2$$
$$\alpha^4 = 2\alpha^3 + 2\alpha = 2(2\alpha+1) + 2\alpha = 2$$
$$\alpha^5 = 2\alpha$$
$$\alpha^6 = 2\alpha^2 + 2(2\alpha+1) = \alpha + 2$$
$$\alpha^7 = \alpha^3 + 2\alpha = 2\alpha + 1 + 2\alpha = \alpha + 1$$
$$\alpha^8 = \alpha^4 + \alpha = 2\alpha + 1 + \alpha = 1$$

即存在一 $GF(3^2)$ 域：

$$0,1,2,x,x+1,x+2,2x,2x+1,2x+2$$

令 $p_{hk}(x) = h + kx$，$h,k = 0,1,2$

例如

$$p_{00}(x) = 0, \quad p_{01}(x) = x, \quad p_{02}(x) = 2x$$
$$p_{10}(x) = 1, \quad p_{11}(x) = 1+x, \quad p_{12}(x) = 1+2x$$
$$p_{20}(x) = 2, \quad p_{21}(x) = 2+x, \quad p_{22}(x) = 2+2x$$

2.3.4 域的特征

设 F 是阶为 p 的有限域，单位元素为 $1, 1+1=2, 1+1+1=2+1=3, \cdots$，故存在一最小整数 p，使得

$$p = \underbrace{1+1+\cdots+1}_{p\text{次}} \equiv 0$$

p 必是一素数，否则若 $p=r \cdot s=0$，则

$$r = 0 \quad \text{或} \quad s = 0$$

与 p 是最小整数的假定相矛盾。称 p 为域 F 的特征。

p 是 F 域的特征，则对于 $\forall q \in F$，恒有 $qp=0$。

故 F 包含有元素：

$$0,1,1+1=2,2+1=3,\cdots,\underbrace{1+1+\cdots+1}_{p-1\text{次}}=p-1$$

这些元素构成 F 的子域 $GF(p)$。

若 F 没有其他元素，则 $F=GF(p)$。

若 F 有其他元素 q，则 F 有最大的线性无关元素，设为 $\alpha_0=1,\alpha_1,\alpha_2,\cdots,\alpha_{m-1}$。

若 F 包含有元素

$$a_0\alpha_0+a_1\alpha_1+\cdots+a_{m-1}\alpha_{m-1}$$

$$a_i\in GF(p),\quad i=1,2,\cdots,m$$

并无其他元素，则 F 是 m 维向量，向量的分量在 $GF(p)$。

2.3.5 本原元素

令 F^* 为 F 非零元素的集合，即 $F^*=F\{0\}$。

F^* 是阶为 $r=p^m-1$ 关于乘法的循环群。若该群含有有限元素：

$$1,\alpha,\alpha^2,\cdots,\alpha^{r-1},\quad \alpha^r=1$$

则称 α 为域 F 的生成元素。

F^* 显然关于乘法构成群。$\beta\in F^*$，$\{\beta^h\}$最多有 p^m-1 个元素，设存在最小整数 r，使

$$\beta^{n+r}=\beta^r,\quad 1\leqslant r\leqslant p^m-1$$

或 $\beta^r=1$，则称 r 为 β 的阶。

现在 F^* 中选一元素 β，使它的阶 r 达到最大，设为 r^*，可证明对子 F^* 中任一元 α，设 α 的阶为 l，必有

$$l\mid r^*$$

假定对于某一素数 p，使

$$r^*=p^a r_1,\quad l=p^b l_1$$

l_1 和 r_1 不被 p 除尽。则

$$(\beta^{p^a})^{r_1}=\beta^{r^*}=1$$

因 l 是 α 的阶，故有

$$(\alpha^{l_1})^{p^b}=\alpha^{p^b l_1}=\alpha^l=1$$

所以

$$(\alpha^{l_1}\beta^{p^a})^{p^b r_1}=(\beta^{p^a r_1})^{p^b}(\alpha^{p^b l_1})^{p^a}=1$$

即 $\beta^{p^a}\alpha^{l_1}$ 的阶为 $p^b r_1$，所以

$$b\leqslant a$$

否则跟 β 的阶 $p^a r_1$ 最大的假定矛盾。

上面假定 p 是某一素数，这就证明了：F^* 的任一元素满足方程

$$x^r-1=0$$

这说明 x^r-1 可被

$$\prod_{\beta\in F^*}(x-\beta)$$

除尽。由于 F^* 有 p^m-1 个元素，$r\geqslant p^m-1$。但 $r\leqslant p^m-1$，故

$$r=p^m-1$$

即
$$\prod_{\beta\in F^*}(x-\beta)=x^{p^m-1}-1$$
即 F 域中非零元素构成循环群：
$$\alpha,\alpha^2,\alpha^3,\cdots,\alpha^{p^m-1}(=1)$$
这样的元素 α 称为域 F 的本原元素。本原元素 α 的阶为 p^m-1，使
$$\alpha^{p^m-1}=1$$
推论 2.3 元素个数为 p^m 的域 F 的某一元素 β 满足
$$\beta^{p^m}=\beta$$
或 β 满足方程
$$x^{p^m}=x$$
即
$$x^{p^m}-x=\prod_{\beta\in F}(x-\beta)$$
定理 2.26 任意有限域 F 都有一本原元素。这个本原元素即为循环群 F^* 的生成元素。

引理 2.2 对于特征为 p 的有限域有
$$(x+y)^p=x^p+y^p$$
证明
$$(x+y)^p=\sum_{k=0}^{p}\binom{p}{k}x^{p-k}y^k$$
$$\binom{p}{0}=\binom{p}{p}=1$$
$$1\leqslant k\leqslant p-1$$
$$\binom{p}{k}=p(p-1)\cdots(p-k+1)/k\equiv 0 \bmod p$$
对于特征为 2 的域有
$$(x+y)^2=x^2+y^2$$
$$(x+y)^4=x^4+y^4$$
$$(x+y)^8=x^8+y^8$$
$$\cdots$$
推论 2.4 对于域 $GF(p)$，任意整数 b，恒有
$$b^{p-1}\equiv 1 \bmod p$$

第3章 分组密码

分组密码即对固定长度的一组明文进行加密的一种加密算法。这一章主要介绍其中比较突出有代表性的若干算法，也是加密算法中比较重要的一种类型。

3.1 Feistel 加密算法

3.1.1 概述

前面已论及信息安全不仅仅是各国政府和军事部门关切的问题，企事业单位同感迫切需要，所以美国国家标准局(NIST)于 1977 年公布由 IBM 公司研制的一种加密算法，批准它作为非机要部门使用的数据加密标准，简称 DES。DES 是 Data Encryption Standard 的缩写。自从公布以来，它一直超越国界，成为国际上商用保密通信和计算机通信的最常用的加密算法。原先规定 DES 的使用期为 10 年，可能是 DES 尚未受到严重威胁，更主要是新的数据加密标准研制工作尚未完成，或意见尚未统一，所以当时的美国政府宣布延长它的使用期。因而 DES 超期服役到 2000 年。二十多年来它一直活跃在国际保密通信的舞台上，扮演了十分突出的角色。进入 20 世纪 90 年代以后，以色列的密码学家 Shamir 等人提出一种"差分分析法"，以后日本人也提出了类似的方法，这才称得上对它有了攻击的方法。严格地说 Shamir 的"差分分析法"也只是有理论上的价值。至少到目前为止是这样，比如后来的"线性逼迫法"，它是一种已知明文的攻击，需要 $2^{43}\approx 4.398\times 10^{12}$ 个明、密文对，在这样苛刻的要求下，还要付出很大的代价才能解出一个密钥。早在 DES 提出不久，就有人提出造一专用的装置来对付 DES，其基本思想无非是借用硬设备来实现对所有的密钥进行遍历搜索。由于电子技术的突飞猛进，这样的专门设备的造价大大降低，速度有质的飞跃，对 DES 形成了实际的威胁。DES 确实辉煌过，它的弱点在于专家们一开始就指出的，即密钥太短。新的加密标准取代 DES 的时间已经到来。

3.1.2 Feistel 加密网络

DES 是 Feistel 加密算法的一种实现，不仅如此，就是后来 AES 候选算法中不少还是采用了 Feistel 网络方法，所以它已成为一种有效的构造密码方法。

Shannon 曾经建议交替地使用搅乱和扩散方法来设计密码，所谓搅乱也就是打乱明文，使密文和明文的统计关系尽可能地复杂化。扩散则是使密文和密钥的关系变得毫无统计规律。扩大密钥和明文对密文的影响。Feistel 加密网络是一个典型，它在密码学方法论方面影响比较大，至今许多加密算法也或多或少地利用了它的思想。所谓 Feistel 网络，是一种乘积型的加密算法。它将明文分成 L_0 和 R_0 两部分，各 b 比特，经过 n 轮的迭

代，最后归并成密文块，第 i 轮的输入为 L_{i-1} 和 R_{i-1}，为前一轮的输出，输出为 L_i 和 R_i，各轮的结构相同，但子密钥 k_i 不同，由密钥 k 产生。它的流程图如图 3.1 和图 3.2 所示。

密文 C 可表示为

$$C = F(m)$$

或

$$F(m) = C = \underbrace{(F_n)}_{\text{第}n\text{轮}}\underbrace{(IF_{n-1})}_{\text{第}n-1\text{轮}}\cdots\underbrace{(IF_2)}_{\text{第2轮}}\underbrace{(IF_1)}_{\text{第1轮}}(m)$$

其中

$$F_i(L_i, R_i) = (L_{i-1} \oplus f(R_{i-1}, k_i), R_i) \quad i = 1, 2, \cdots, n$$

$$I(L, R) = (R, L)$$

其中 $f(R_{i-1}, k_i)$ 称为轮函数，k_i 是子密钥。

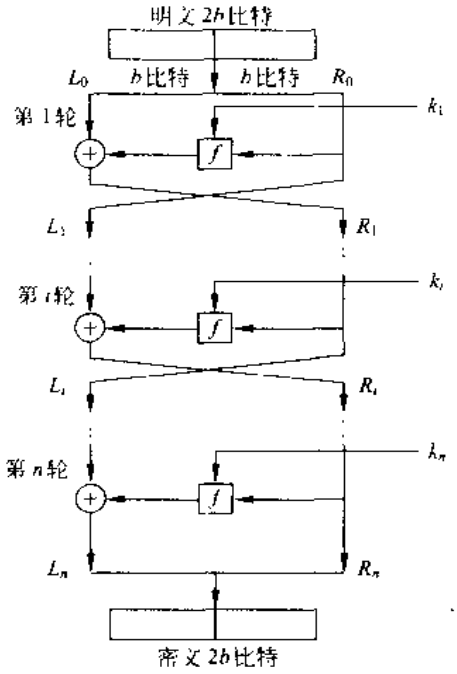


图 3.1

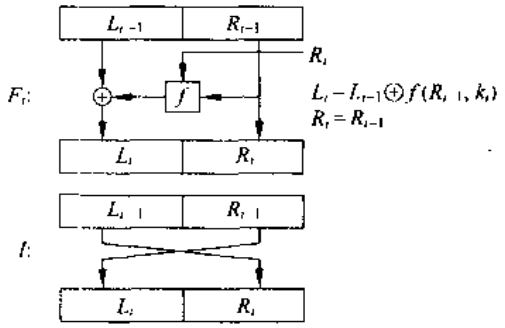


图 3.2

解密过程和加密类似，但子密钥的顺序颠倒过来，即

$$m = F^{-1}(C) = F_1(IF_2)\cdots(IF_{n-1})(IF_n)(C)$$

算法的正确性在于证明：

(1) $F_i(F_i(L_{i-1},R_{i-1}))=(L_{i-1},R_{i-1})$

(2) $I(I(L,R))=(L,R)$

即 F_i 和 I 都是以自身作为逆变换。

$$\begin{aligned}F_i(L_{i-1},R_{i-1}) &= (L_{i-1}\oplus f(R_{i-1},k_i),R_{i-1})\\ F_i(F_i(L,R)) &= F_i(L_{i-1}\oplus f(R_{i-1},k_i),R_{i-1})\\ &= (L_{i-1}\oplus f(R_{i-1},k_i)\oplus f(R_{i-1},k_i),R_{i-1})\\ &= (L_{i-1},R_{i-1})\end{aligned}$$

同样有 $I(L,R) = (R,L)$

所以 $I(I(L,R)) = I(R,L) = (L,R)$

所以 $F^{-1}(C) = F_1(IF_2)(IF_3)\cdots(IF_{n-1})(IF_n)(C)$

由于

$$C = F_n(IF_{n-1})\cdots(IF_1)(m)$$

$$F_nF_n(L_{n-1},R_{n-1}) = (L_{n-1},R_{n-1})$$

$$II(L_{n-1},R_{n-1}) = (L_{n-1},R_{n-1})$$

所以

$$\begin{aligned}F^{-1}(C) &= F_1(IF_2)\cdots(IF_{n-1})F_{n-1}\cdots(IF_1)(m)\\ &= F_1(IF_2)\cdots(IF_{n-2})(F_{n-2}I\cdots IF_1)(m)\\ &= \cdots = F_1F_1(m) = m\end{aligned}$$

3.1.3 DES 加密标准

数据加密标准 DES 虽然已走完它的生命历程，但是作为一种 Feistel 加密算法的例子仍然有讨论的价值。它的缺点在于密钥太短，号称 64 位，实际上只有 56 位。

设

$$m = m_1m_2\cdots m_{64}$$

$$k = k_1k_2\cdots k_{64}$$

后面将看到其中 $k_8,k_{16},k_{24},k_{32},k_{40},k_{48},k_{56},k_{64}$ 实际上是不起作用的。

DES 的加密过程可表示为

$$DES(m) = IP^{-1}T_{16}\cdot T_{15}\cdots T_2\cdot T_1\cdot IP(m)$$

DES 的流程图如图 3.3 所示。

(1) IP 是初始置换，IP^{-1} 是它的逆置换。

$$m \longrightarrow \boxed{IP} \longrightarrow \tilde{m}$$

设 $m=m_1m_2\cdots m_{64}$，$\tilde{m}=\tilde{m}_1\tilde{m}_2\cdots\tilde{m}_{64}$

$\tilde{m}=m_{58}m_{50}m_{42}\cdots m_{23}m_{15}m_7$

即 $\tilde{m}_1=m_{58}$，$\tilde{m}_2=m_{50}$，$\cdots$，$\tilde{m}_{64}=m_7$

$\tilde{m}$ 的下标依次如下：

IP：

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

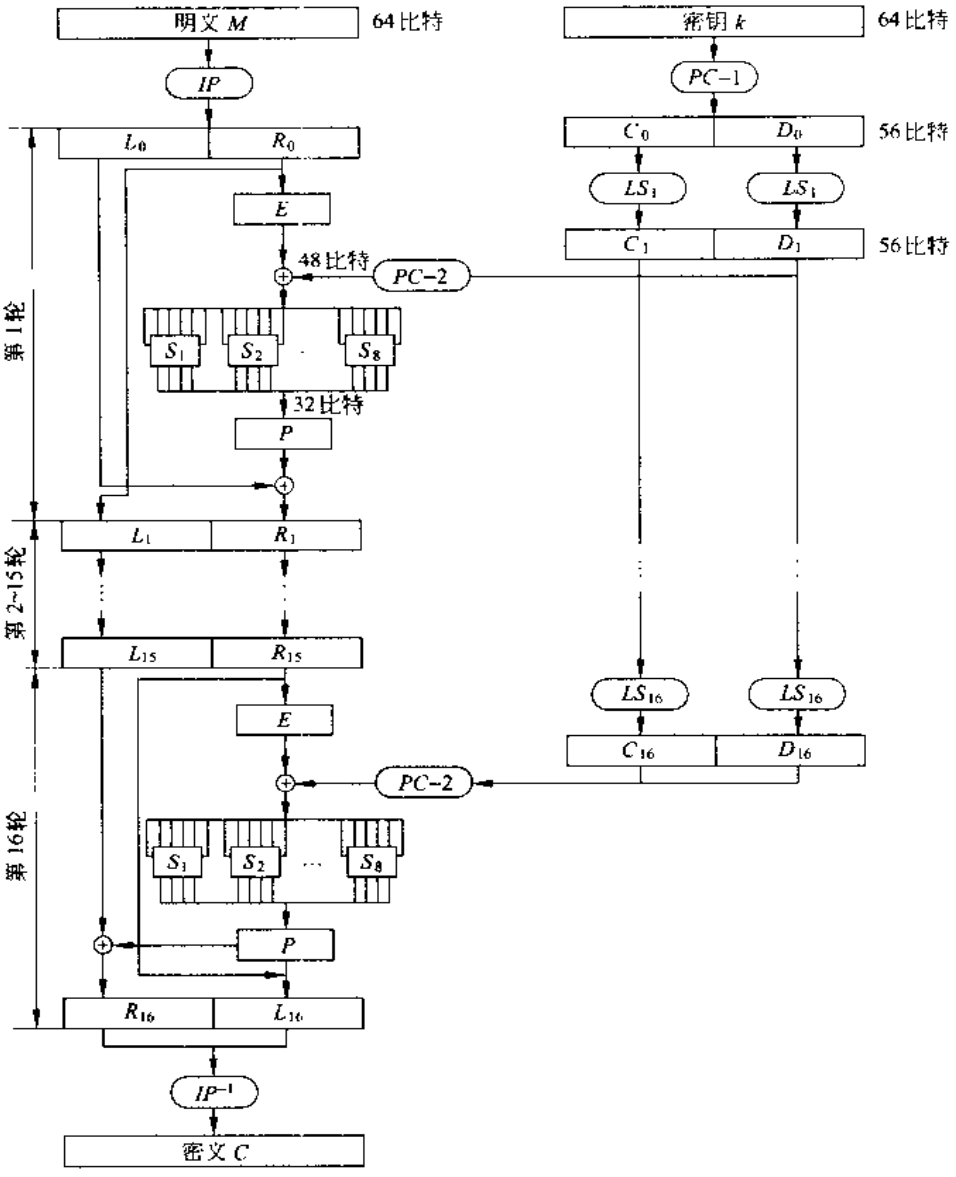
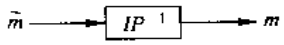


图 3.3

IP^{-1}满足

$$IPIP^{-1}=IP^{-1}IP=I$$



若 $\tilde{m}=\tilde{m}_1\tilde{m}_2\cdots\tilde{m}_{64}$

则 $m=\tilde{m}_{40}\tilde{m}_8\tilde{m}_{48}\cdots\tilde{m}_{17}\tilde{m}_{57}\tilde{m}_{25}$

IP^{-1}关于$\tilde{m}$的下标如下：

IP^{-1}：

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

（2）DES的迭代过程。为了叙述DES的迭代过程，还可以将它的流程图形象地表示如图3.4所示。

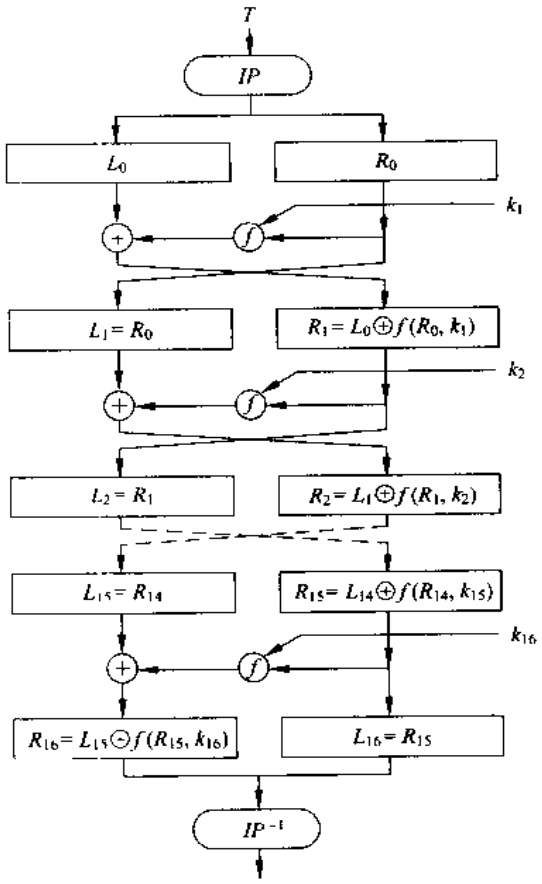


图 3.4

现将第 i 次迭代表示为如图 3.5($i=1,2,\cdots,16$)所示：

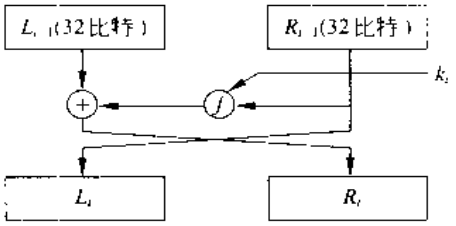


图 3.5

图 3.5 中 L_{i-1} 和 R_{i-1} 分别是第 $i-1$ 次迭代结果的左右两部分，各 32 比特。则

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

L_0, R_0 是初始输入经 IP 置换的结果。

$\oplus$ 是按位作不进位加法运算，即

$$1 \oplus 0 = 0 \oplus 1 = 1,\ 0 \oplus 0 = 1 \oplus 1 = 0$$

k_i 是由 64 比特的密钥产生的子密钥，k_i 是 48 比特，k_i 如何产生在后面介绍。

DES 的关键在于 $f(R_{i-1}, k_i)$ 的功能。f 是将 32 比特的输入转化为 32 比特的输出，见图 3.6。

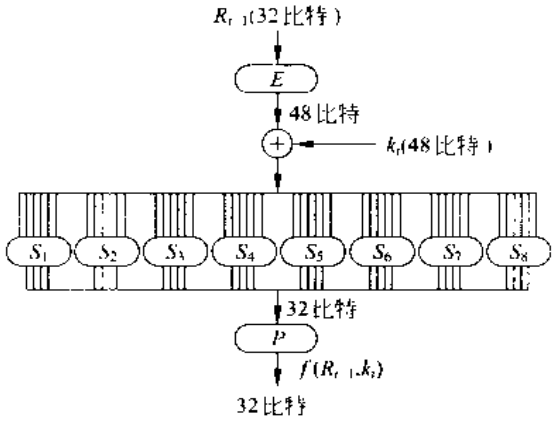


图 3.6

E 的作用是将 32 比特的输入膨胀为 48 比特，若输入为

$$r_1^{(i-1)}, r_2^{(i-1)}, \cdots, r_{32}^{(i-1)}$$

则它的输出为

$$r_{32}^{(i-1)}, r_1^{(i-1)}, r_2^{(i-1)}, \cdots, r_{32}^{(i-1)}, r_1^{(i-1)}$$

输出的 48 比特的下标是

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

E 输出的 48 比特顺序分成 8 组，每组 6 比特，分别通过 $S_1, S_2, \cdots, S_8$ 盒后又缩为 32 比特，即每盒输入为 6 比特，输出为 4 比特。这个过程见图 3.7。

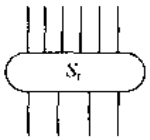


图 3.7

假定 S_i 盒的 6 个输入端为 $b_1b_2b_3b_4b_5b_6$，在 S_i 表中找出 b_1b_5 行、$b_2b_3b_4b_5$ 列的元素

$$S_i(b_1b_6,\ b_2b_3b_4b_5)$$

便是 S_i 盒的输出。S 表如表 3.1 所示。

表 3.1 S 表

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

续表

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

不难发现 S 表中的元素每行都是 0～15 这 16 个数的某一种排列。

例 3.1 S_1 的输入为 101100，

$$b_1=1,\ b_6=0,\ b_1b_6=(10)_2=2$$

$$(b_2b_3b_4b_5)_2=(0110)_2=6$$

查 $S_1(2,6)=7$

故 S_1 的输出为 0111，依此类推。

S 盒输出的 32 比特经 P 置换，最后得

$$f(R_{i-1},k_i)$$

$f(R_{i-1},k_i)$ 也是 32 比特。置换 P 将 32 比特的输入，改变的顺序如下所示：

$$P:\begin{matrix}16 & 7 & 20 & 21\\ 29 & 12 & 28 & 17\\ 1 & 15 & 23 & 26\\ 5 & 18 & 31 & 10\\ 2 & 8 & 24 & 14\\ 32 & 27 & 3 & 9\\ 19 & 13 & 30 & 6\\ 22 & 11 & 4 & 25\end{matrix}$$

即若输入 32 比特为 $a_1,a_2,\cdots,a_{32}$。输出为

$$a_{16},a_7,a_{20},\cdots,a_{11},a_4,a_{25}$$

(3) 子密钥的生成。图 3.8 给出了子密钥的产生流程图。

$PC-1$ 是对 56 比特输入进行重新排序，输出顺序如下：

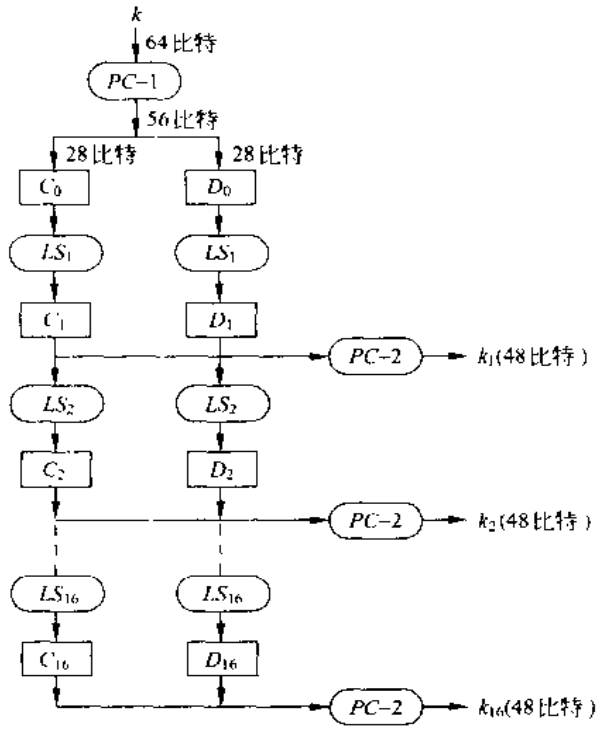


图 3.8

$PC-1$：

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

密钥 k 应是 64 比特，请注意 $PC-1$ 表中不出现 8，16，24，32，40，48，56，64，所以实际上有 56 位有效。也可以看作是输入 64 比特，而只选取其中 56 位，8，16，…，64 位舍去，它本来就是奇偶校验位。

C_0、D_0 是输出 56 比特的左右两个一半，即

$$C_0 = k_{57}k_{49}k_{41}\cdots k_{52}k_{44}k_{36}$$

$$D_0 = k_{63}k_{55}k_{47}\cdots k_{20}k_{12}k_4$$

LS_i 是循环左移。左移的位数如下：

迭代顺序	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
左移位数	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

设 $C_1 = c_1c_2\cdots c_{28}$，$D_1 = d_1d_2\cdots d_{28}$

由于是第 2 次迭代，应循环左移 1 位，故

但 $C_2 = c_2c_3\cdots c_{28}c_1$，$D_2 = d_2d_3\cdots d_{28}d_1$

$$C_3=c_4c_5\cdots c_{28}c_1c_2c_3,\ D_3=d_4d_5\cdots d_{28}d_1d_2d_3$$

因为第 3 次迭代，应循环左移 2 位。

$PC-2$ 是从 56 位中选出 48 位输出，如下所示：

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

3.1.4 DES 的解密过程

DES 解密过程和 DES 完全类似，只不过将 16 轮的子密钥序列 $k_1,k_2,\cdots,k_{16}$ 的顺序倒过来，即第一轮用第 16 个子密钥 k_{16}，第 2 轮用 k_{15}，依此类推，即

$$DES^{-1}=IP^{-1}T_1T_2\cdots T_{15}T_{16}IP$$

容易验证

$$DES^{-1}(DES(m))=m$$

$$DES(DES^{-1}(m))=m$$

由于

$$DES(m)=IP^{-1}T_{16}T_{15}\cdots T_2T_1IP$$

因为

$$IPIP^{-1}=I$$

所以

$$DES^{-1}(DES(m))=IP^{-1}T_1T_2\cdots T_{16}(T_{16}T_{15}\cdots T_2T_1IP(m))$$

我们来看一看

$$T_{16}(T_{15}\cdots T_1IP(m))$$

的结果。

$T_{16}(T_{15}\cdots T_1IP(m))$ 表示 $DES(m)$ 第 16 轮迭代后 R_{16} 和 L_{16} 的结果。从图 3.9 和图 3.10 可知

图 3.9

$$L=R_{15},\ R=L_{15}\oplus f(R_{15},k_{16})\oplus f(R_{15},k_{16})=L_{15}$$

即 $DES^{-1}(C)$ 第 1 轮迭代后有 $L=R_{15}$，$R=L_{15}$。同理

$$R_{15}=L_{14}\oplus f(R_{14},k_{15}),\ L_{15}=R_{14}$$

故 $DES^{-1}(C)$ 的第 2 轮的结果如图 3.11 所示。

依此类推

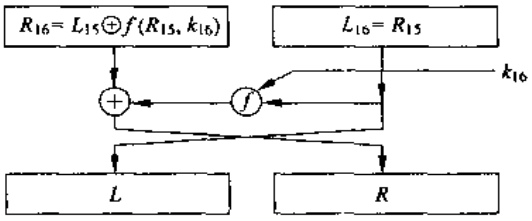


图 3.10

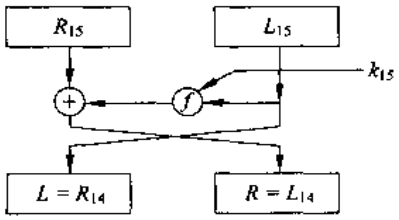


图 3.11

$$DES^{-1}(DES(m))=m$$

得证。

3.1.5 DES 的解密过程和举例

设明文 $m=$computer，密钥为 program，它们用 ASCII 码表示

$$\begin{aligned} m=&01100011\ \ 01101111\ \ 01101101\\ &01110000\ \ 01110101\ \ 01110100\\ &01100101\ \ 01110010 \end{aligned}$$

或用十六进制表示为

$$\begin{aligned} m=&\ 63\ \ 6F\ \ 6D\ \ 70\ \ 75\ \ 74\ \ 65\ \ 72\\ k=&01110000\ \ 01110010\ \ 01101111\\ &01100111\ \ 01110010\ \ 01101101\\ &01101101 \end{aligned}$$

或用十六进制表示为

$$\begin{aligned} k=&70\ \ 72\ \ 6F\ \ 67\ \ 72\ \ 61\ \ 6D\\ =&h_1h_2\cdots h_{56} \end{aligned}$$

请注意：这里 k 只有 56 位，由于第 8,16,24,32,40,48,56,64 位不起作用，故不予赋值。也可以让密钥是 64 比特，如 Programs，不过其中的 8 位 $k_8,k_{16},k_{24},k_{32},k_{40},k_{48},k_{56},k_{64}$ 不起作用。若令

$$k=k_1k_2k_3\cdots k_{64}$$

即把 $h_1h_2\cdots h_{56}$ 插入保持前后顺序不变，比如 k_8 需设

$$k_9=h_8,k_{10}=h_9,k_{11}=h_{10},k_{12}=h_{11},k_{13}=h_{12}$$

$$k_{14}=h_{13},k_{15}=h_{14},k_{16}$$

需设 $k_{17}=h_{15},k_{18}=h_{16},\cdots$

即实际的密钥为 64 比特如下：

$$\begin{aligned} k=&0111000*0011100*1001101*1110110*\\ &0111011*1001001*1000010*1101101* \end{aligned}$$

* 是插入的 $k_8,k_{16},k_{24},k_{32},k_{40},k_{48},k_{56},k_{64}$ 需设的值。

可得

$C_0 = 11101100\ \ 10011001\ \ 00011011\ \ 1011$

$D_0 = 10110100\ \ 01011000\ \ 10001110\ \ 0111$

$C_1 = 11011001\ \ 00110010\ \ 00110111\ \ 0111$

$D_1 = 01101000\ \ 10110001\ \ 00011100\ \ 1111$

$C_2 = 10110010\ \ 01100100\ \ 01101110\ \ 1111$

$D_2 = 11010001\ \ 01100010\ \ 00111001\ \ 1110$

$C_3 = 11001001\ \ 10010001\ \ 10111011\ \ 1110$

$D_3 = 01000101\ \ 10001000\ \ 11100111\ \ 1011$

$C_4 = 00100110\ \ 01000110\ \ 11101111\ \ 1011$

$D_4 = 00010110\ \ 00100011\ \ 10011110\ \ 1101$

$C_5 = 10011001\ \ 00011011\ \ 10111110\ \ 1100$

$D_5 = 01011000\ \ 10001110\ \ 01111011\ \ 0100$

$C_6 = 01100100\ \ 01101110\ \ 11111011\ \ 0010$

$D_6 = 01100010\ \ 00111001\ \ 11101101\ \ 0001$

$C_7 = 10010001\ \ 10111011\ \ 11101100\ \ 1001$

$D_7 = 10001000\ \ 11100111\ \ 10110100\ \ 0101$

$C_8 = 01000110\ \ 11101111\ \ 10110010\ \ 0110$

$D_8 = 00100011\ \ 10011110\ \ 11010001\ \ 0110$

$C_9 = 10001101\ \ 11011111\ \ 01100100\ \ 1100$

$D_9 = 01000111\ \ 00111101\ \ 10100010\ \ 1100$

$C_{10} = 00110111\ \ 01111101\ \ 10010011\ \ 0010$

$D_{10} = 00011100\ \ 11110110\ \ 10001011\ \ 0001$

$C_{11} = 11011101\ \ 11110110\ \ 01001100\ \ 1000$

$D_{11} = 01110011\ \ 11011010\ \ 00101100\ \ 0100$

$C_{12} = 01110111\ \ 11011001\ \ 00110010\ \ 0011$

$D_{12} = 11001111\ \ 01101000\ \ 10110001\ \ 0001$

$C_{13} = 11011111\ \ 01100100\ \ 11001000\ \ 1101$

$D_{14} = 00111101\ \ 10100010\ \ 11000100\ \ 0111$

$C_{14} = 01111101\ \ 10010011\ \ 00100011\ \ 0111$

$D_{14} = 11110110\ \ 10001011\ \ 00010001\ \ 1100$

$C_{15} = 11110110\ \ 01001100\ \ 10001101\ \ 1101$

$D_{15} = 11011010\ \ 00101100\ \ 01000111\ \ 0011$

$C_{16} = 11101100\ \ 10011001\ \ 00011011\ \ 1011$

$D_{16} = 10110100\ \ 01011000\ \ 10001110\ \ 0111$

从 C_0, D_0 通过 PC_2 可得 16 个子密钥：

k_1 = 00111101 10001111 11001101 00110111 00111111 01001000

k_2 = 10101011 00111101 10011000 01001110 00010110 01000111

k_3 = 01011100 00101110 11101101 11011110 11000001 11101100

k_4 = 11010011 11111100 00011000 00000000 11011111 11001001

k_5 = 01001100 10101111 11100110 11011010 10110100 00110001

k_6 = 11110010 11111100 00001111 11101011 01001111 00101000

k_7 = 01101001 10100111 01100010 00011000 01111011 00011010

k_8 = 11100000 11011100 10111111 11110101 01010000 00110100

k_9 = 10001100 11010110 11100010 10001011 11001010 11010100

k_{10} = 11110010 01011011 01111110 01010001 11100111 10010001

k_{11} = 10101100 11110011 01000001 10111011 00000100 00001101

k_{12} = 00000011 01011111 01111111 11001010 01110011 10000110

k_{13} = 11101101 01110001 11010001 00110100 01100011 10101101

k_{14} = 00010111 11001111 11101001 11110010 00011000 11000011

k_{15} = 11011011 01110001 10010011 11000110 10100011 00111011

k_{16} = 00011111 01101010 00101111 10100101 11000101 10001011

从而得 L_i, R_i, $i=0,1,2,\cdots,16$ 如下：

L_0 =11111111 10111000 01110110 01010111

R_0 =00000000 11111111 00000110 10000011

L_1 =00000000 11111111 00000110 10000011

R_1 =10111011 10011001 11101001 11001100

L_2 =10111011 10011001 11101001 11001100

R_2 =10000011 11110101 10001100 10100011

L_3 =10000011 11110101 10001100 10100011

R_3 =00010001 11011011 11001111 11011100

L_4 =00010001 11011011 11001111 11011100

R_4 =10011010 11110110 01100010 01011001

L_5 =10011010 11110110 01100010 01011001

R_5 =00100111 10100001 00011011 01011101

L_6 =00100111 10100001 00011011 01011101

R_6 =10110110 11100110 11001011 11011010

L_7 =10110110 11100110 11001011 11011010

R_7 =00101011 00111100 00010000 10101110

L_8 =00101011 00111100 00010000 10101110

R_8 =00011000 11111111 00010001 01010010

L_9 =00011000 11111111 00010001 01010010

R_9 =01111101 00001100 10100001 00101111

L_{10} =01111101 00001100 10100001 00101111

R_{10} =00011100 10111110 11100110 11101110

L_{11} =00011100 10111110 11100110 11101110

R_{11} =00000110 10001101 00100001 01011101

L_{12} =00000110 10001101 00100001 01011101

R_{12} =00111011 10000010 00101110 01001011

L_{13} =00111011 10000010 00101110 01001011

R_{13} =00100000 11111001 00001101 11110001

L_{14} =00100000 11111001 00001101 11110001

R_{14} =01101110 01001001 10001101 00010000

L_{15} =01101110 01001001 10001101 00010000

R_{15} =11101000 10000011 01111000 01001100

L_{16} =01010010 10011000 11000001 01011010

R_{16} =11101000 10000011 01111000 01001100

最后通过 IP^{-1} 及密文(64 比特)：

00100100 01100001 00000010

10011011 01011001 10001000

11001111 10110100

设明文 m 为 computer，密钥 k 改为 security，请注意现在 k 是 64 比特。它们用 ASCII 码表示为

m: 01100011 01101111 01101101

01110000 01110101 01110100

01100101 01110010

k: 01110011 01100101 01100011

01110101 01110010 01101001

01110100 01111001

或用十六进制表示

m = 63 6F 6D 70 75 74 65 72

k = 73 65 63 75 72 69 74 79

m 经 IP 置换得

L_0 = 11111111 10111000 01110110 01010111

R_0 = 00000000 11111111 00000110 10000011

或用十六进制表示

L_0 = FFB87657, R_0 = 00FF0683

k 通过 PC^{-1} 得

C_0 = 00000000 11111111 11111111 1101

D_0 = 00010101 01001010 10100000 1001

或表示为十六进制数

C_0 = 00FFFFD D_0 = 154AA09

C_0 和 D_0 各向左旋转移位一比特，

C_1：01FFFFA，即 00000001 11111111 11111111 1010

D_1：2A95412，即 00101010 10010101 01000001 0010

通过 $PC-2$ 产生 k_1 为 F0BE6E752830，即

11110000 10111110 01101110

01110101 00101000 00110000

R_0 经 E 膨胀为 48 比特：8017FE80D406，即

100000 000001 011111 111110

100000 001101 010000 000110

它与 k 作异或运算得 70A990F5FC36

011100 001010 100110 010000

111101 011111 110000 110110

(1) 011100 进入 S_1 盒，从 S_1 盒的第 0 行第 14 列的元素 0，可知其输出为 0000；

(2) 001010 进入 S_2 盒，从 S_2 盒的第 0 行第 5 列的元素 11，可知其输出为 1011；

(3) 100110 进入 S_3 盒，S_3 盒的第 2 行第 3 列的元素 9，可知其输出为 1001；

(4) 010000 进入 S_4 盒，从 S_4 盒的第 0 行第 8 列元素 1，可知其输出为 0001；

(5) 111101 进入 S_5 盒，从 S_5 盒的第 3 行第 14 列元素 5，可知其输出为 0101；

(6) 011111 进入 S_6 盒，从 S_6 盒的第 1 行第 15 列的元素 8，可知其输出为 1000；

(7) 110000 进入 S_7 盒，从 S_7 盒的第 2 行第 8 列的元素 10，可知其输出为 1010；

(8) 110110 进入 S_8 盒，从 S_8 盒的第 2 行第 11 列的元素 13，可知输出为 1101。

故 8 个 S 盒的输出为 0B9158AD，即

00001011 10010001 01011000 10101101

最后通过 P 得 $f(R_0,k_1)$ = FC0C4D21，即

11111100 00001100 01001101 00100001

$L_0 \oplus f(R_0,k_1)$ 得 03B43B76

00000011 10110100 00111011 01110110

故第 1 轮的输出，即第 2 轮的输入为

$L_1 = R_0$ = 00FF0683

$R_1 = L_0 \oplus f(R_0,k_1)$ = 03B43B76

现在 16 轮迭代过程列表于下：

明文：636F6D7075746572， 密钥：7365637572697479

C_0 = 00FFFFD, D_0 = 154AA09

L_0 = FFB87657, R_0 = 00FF0683

C_1 = 01FFFFA, D_1 = 2A95412, k_1 = F0BE6E752830

E_R = 8017FE80D106, $E+k$ = 70A990F5FC36, $S_{F,k}$ = 0B9158AD, $f_{K,1}$ = FC0C4D21

L_1 = 00FF0683, R_1 = 03B13B76

C_2 = 03FFFF4, D_2 = 552A824, k_2 = F0BEF682C285

E_R = 007DA81F6BAC, $E+k$ = F0C35E9DA929, $S_{E,2}$ = 536F77B4, $f_{K,2}$ = E265F1EF

L_2 = 03B43B76, R_2 = E29AF26C

C_3 = 0FFFFD0, D_3 = 54AA091, k_3 = F4F676D20781

E_R = 7054F57A4359, $E+k$ = 81A283A844D8, $S_{F,3}$ = FB38DA35, $f_{K,3}$ = 77ACCE66

L_3 = E29AF26C, R_3 = 7418F510

C_4 = 3FFFF40, D_4 = 52A8245, k_4 = E6D7769A0309

E_R = 3A80F17AA8A0, $E+k$ = DC5787E0ABA9, $S_{F,4}$ = E48562E4, $f_{K,4}$ = 80B497B1

L_4 = 7418F510, R_4 = 622E65DD

C_5 = FFFFD00, D_5 = 1AA0915, k_5 = EED377527300

E_R = B0415C30BEFA, $E+k$ = 5E922B62CDFA, $S_{F,5}$ = B361DCF3, $f_{K,5}$ = F3971E0F

L_5 = 622E65DD, R_5 = 878FBB1F

C_6 = FFFF403, D_6 = 2A82455, k_6 = AFD37B702128

E_R = C0FC5FDF68FF, $E+k$ = 6F2F24AF49D7, $S_{F,6}$ = 58E9E48B, $f_{K,6}$ = 890F89CF

L_6 = 878FBB1F, R_6 = EB21EC12

C_7 = FFFD00F, D_7 = AA09154, k_7 = AF53FBF0380A

E_R = 756903F580A0, $E+k$ = DA3AF815B8AF, $S_{F,7}$ = 78952B4D, $f_{K,7}$ = 9C38BBA2

L_7 = EB21EC12, R_7 = 1BB700BD

C_8 = FFF403F, D_8 = A824552, k_8 = BF5BD961323A

E_R = 8F7DAE8015FA, $E+k$ = 302677E427C0, $S_{F,8}$ = B1CBA16D, $f_{K,8}$ = 89D16FE2

L_8 = 1BB700BD, R_8 = 62F083F0

C_9 = FFE807F, D_9 = 5048AA5, k_9 = 3F59DB8AC501

E_R = 3057A1407FA0, $E+k$ = 0F0E7ACABAA1, $S_{F,9}$ = F5B29532, $f_{K,9}$ = 27C2E71E

L_9 = 62F083F0, R_9 = 3C75E7A3

C_{10} = FFA01FF, D_{10} = 4122A95, k_{10} = 3F69DD4A4704,

E_R = 9F83ABF0FD06, $E+k$ = A0EA76BABA02, $S_{E,10}$ = D46E85C2, $f_{K,10}$ = 01D3B05F

L_{10} = 3C75E7A3, R_{10} = 632333AF

C_{11} = FE807FF, D_{11} = 5048AA55, k_{11} = 1F6D9DD84188

E_R = B069069A7D0E, $E+k$ = AF049B423CD6, $S_{F,11}$ = 95DA835E, $f_{K,11}$ = 0FF36172

L_{11} = 632333AF, R_{11} = 338686D1

C_{12} = FA01FFF, D_{12} = 122A954, k_{12} = 5F2DB3DC05209

E_R = 9A7C0D40D6A2, $E+k$ = C551B08084AB, $S_{F,12}$ = 51EF49CA, $f_{K,12}$ = 9857F147

L_{12} = 338686D1, R_{12} = FB74C2E8

C_{13} = E807FFF, D_{13} = 48AA550, k_{13} = DFACADD23228

E_R = 7F6BA9605751, $E+k$ = A0C701B26579, $S_{E,13}$ = D32E7553, $f_{K,13}$ = 62D6F8CE

L_{13} = FB74C2E8, R_{13} = 51505E1F

C_{14} = A01FFFF, D_{14} = 22A9541, k_{14} = DBAEAEB01B28

E_R = AA2AA03FC0FE, $E+k$ = 71810E8FDBD6, $S_{F,14}$ = 0C1A887E, $f_{K,14}$ = 1F5A0470

L_{14} = 51505E1F, R_{14} = E42EC698

C_{15}=807FFFE，D_{15}=8AA5504，k_{15}=F8BEAE103A32

E_R=70815D60D4F1，$E+k$=883FF370EEC3，$S_{E,k}$=1DC4E82F，$f_{R,k}$=190F5DB2

L_{15}=E42EC698，R_{15}=485F23AD

C_{16}=00FFFFD，D_{16}=154AA09，k_{16}=F1BEA68841C5

E_R=A502FE907D5A，$E+k$=54BC58183C9F，$S_{E,k}$=C24B1FF2，$f_{R,k}$=F2E3A449

L_{16}=485F23AD，R_{16}=16DD62D1

最后得 L_{16}=485F23AD，R_{16}=16DD62D1。

进行 IP^{-1} 运算得密文 C=379CB171B20D7A23。

附带说明一下 DES 算法，当明文或密钥每改变一比特，都对密文产生显著的影响，当然密钥 64 比特中的第 8,16,24,32,40,48,56,64 比特的改变，对密文不会产生影响。比如明文为 computer，密钥为 security，通过 DES 加密的密文为 379CB171B20D7A23，密钥不变，明文为 computes，则密文改为 BC41F776B294379E，改变的位数达到 30 个，又如明文为 domputer，则密文改变为 FB23B211D2E38F61，密文改变了 29 位，这两个例子都是明文改变一比特引起的密文的位改变将近一半，同样若密钥改变一比特也有类似的结果，这说明密文的每一位都是明文和密钥每一位十分复杂而敏感的关系，牵一发而动全身。称这种现象为雪崩现象。

3.1.6 关于 DES 的若干问题

1. 弱密钥

许多密码都有坏密钥，DES 也一样。比如若出现子密钥满足

$$k_1=k_2=\cdots=k_{16}$$

则有

$$DES_k(m)=DES_k^{-1}(m)$$

或

$$DES_k(DES_k(m))=m$$

这样的密钥 k 称为弱密钥。不难知

$$k_{57}=k_{49}=k_{41}=\cdots=k_{44}=k_{36}=0 \quad 或\ 1$$

$$k_{63}=k_{55}=k_{47}=\cdots=k_{12}=k_4=0 \quad 或\ 1$$

时将是弱密钥。故弱密钥有以下 4 种

01	01	01	01	01	01	01	01
1F	1F	1F	1F	0E	0E	0E	0E
E0	E0	E0	E0	F1	F1	F1	F1
FE	FE	FE	FE	FE	FE	FE	FE

其中 01 代表 00000001，1F 代表 00011111，E0 代表 11100000，FE 代表 11111110。

还有一种称为半弱密钥的，即存在 k 和 k' 使得

$$DES_k(m)=DES_{k'}^{-1}(m)$$

或

$$DES_k(DES_{k'}(m))=m$$

k 和 k' 成对构成半弱密钥。即 C_0 为 1010…10，而 D_0 为 00…0，或 11…1，或 0101…01，或 1010…10 的密钥，分别和 C_1 为 0101…01，D_0 为 00…0，或 11…11，或 0101…01，或 1010…10 的密钥。

半弱密钥有下面 12 个

01	FE	01	FE	01	FE	01	FE
FE	01	FE	01	FE	01	FE	01
1F	E0	1F	E0	0E	F1	0E	F1
E0	1F	E0	1F	F1	0E	F1	0E
01	E0	01	E0	E0	F1	01	F1
E0	01	E0	01	F1	01	F1	01
1F	FE	1F	FE	0E	FE	0E	FE
FE	1F	FE	1F	FE	0E	FE	0E
01	1F	01	1F	01	0E	01	0E
1F	01	1F	01	0E	01	0E	01
E0	FE	E0	FE	F1	FE	F1	FE
FE	E0	FE	E0	FE	F1	FE	F1

其中 F1 表示 11110001，0E 表示 00001110，其余同上。其中 A，B，C，D，E，F，分别是 10，11，12，13，14，15 的二进制表示。E 为 14，它的二进制表示为 1110，F 为 15，它的二进制表示为 1111。

2. DES 的安全性

人们普遍感兴趣的问题是 DES 的抗攻击强度怎么样？或直截了当地说密钥长度是不是够了。上面已经说过它的密钥号称 64 比特，实际上只有 56 比特，$2^{56}\approx 7\times 10^{16}$。普遍的印象密钥长仅 56 比特是短了些。Diffie 和 Hellman 曾设想花千万美金造一台专用机，可望一天内找到一个密钥，基本思想还是穷举，即强行攻击。上面也提到 1990 年 Eli Biham 和 Adi Shamir 提出一种"差分分析法"。在选择明文条件下，复杂度有所下降。比如在选择明密文对 2^{47} 对的要求情况下，攻击复杂度虽降到 $O(2^{47})$，但是代价也是可观的。

顺便提一句，若迭代次数不是 16 次，而是 8 次，选择明文 $2^{14}=16\,384$，攻击复杂性降至仅 $O(2^{7})$。总之，DES 密钥太短，超期服役的时间也太长。新的攻击手段不断出现，DES 已面临着实实在在的威胁。直接的威胁还是在于专用设备，由于芯片的速度越来越快，造价越来越便宜，专用设备的造价也大大地降低。

3.1.7 DES 的变形

1. 变形之一：3-DES

前面已经提及 DES 的密钥 k 为 56 比特，长度不够。若采用如图 3.12 所示的三重加密形式，它的解密流程如图 3.13 所示，其正确性留给读者自己来证明。

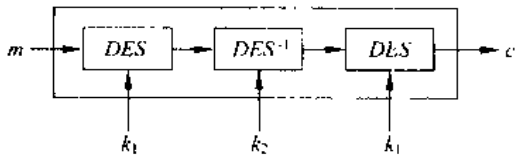


图 3.12

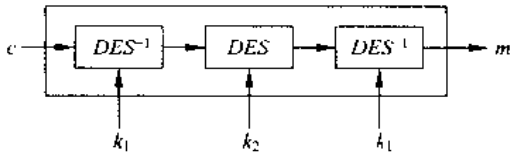


图 3.13

2. 变形之二

若明文

$$m = m_1 m_2 \cdots m_n$$

m_i 是 64 比特的分组，则一般加密、解密如图 3.14 所示，通常称之为 ECB 模式，ECB 即为 Electronic Code Book 的缩写。

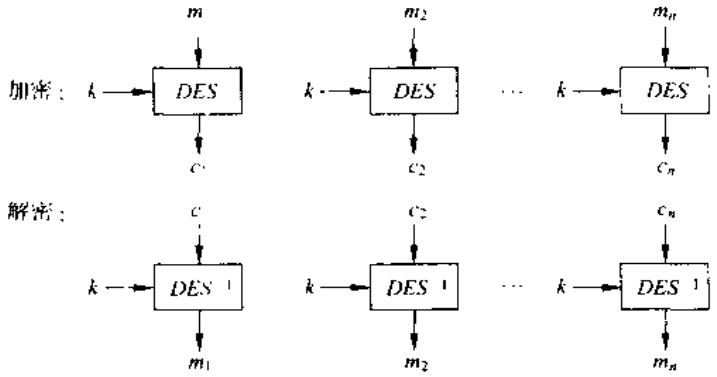


图 3.14

下面介绍：

(1) CBC 模式。CBC 是 Cipher Block Chaining 的缩写，意即由密文块链接流程图，如图 3.15 所示。

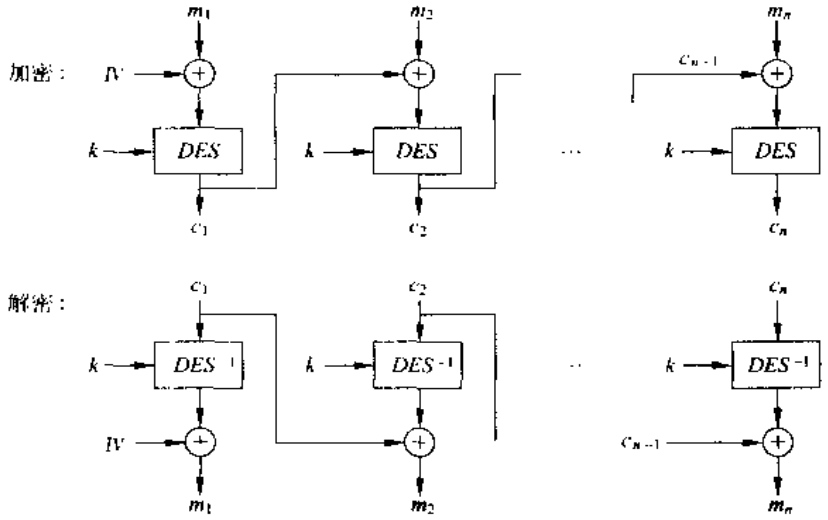


图 3.15

其中 IV 是 Initial Vector 的缩写，即已知的 64 比特的初始向量。类似于密钥由通信双方秘密约定，或通信前通过密码传输。

即

$$c_h = DES_k(c_{h-1} \oplus m_h), \quad h = 1,2,\cdots,n,$$
$$c_0 = IV$$

$$DES_k^{-1}(c_n) \oplus c_{n-1}$$
$$= DES_k^{-1}(DES_k(c_{n-1} \oplus m_n)) \oplus c_{n-1}$$
$$= c_{n-1} \oplus m_n \oplus c_{n-1} = m_n$$

同理证

$$DES_k^{-1}(c_h) \oplus c_{h-1} = m_h$$
$$h = n, n-1, \cdots, 1$$

(2) CFB 模式。CFB 是 Cipher Feedback 的缩写，意即密文反馈。它是利用 DES 作为序列密码的工作。其工作的流程图如图 3.16 所示，图中 SR 是 Shift Register 的缩写，即移位寄存器；S 是 Seed 的缩写，即 64 比特的种子。

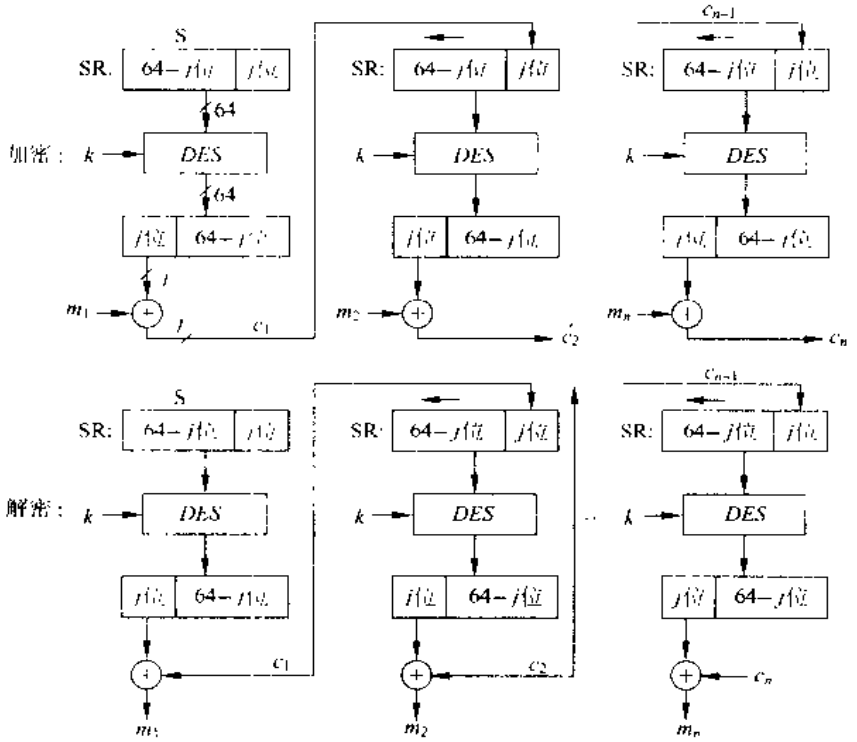


图 3.16

图 3.16 中 j 一般取 8 比特，由流程图可以看出每组密文都是前面所有明文的函数关系。

首先输入到移位寄存器中去的是种子，加密后最左边(也就是高位)的 j 位和明文 m_1 作⊕运算，产生 c_1，与此同时移位寄存器向左移 j 位，而 c_1 正好放在输出的其间(最右边低位)，过程持续下去，直到所有明文块都加密完毕为止。

设 $M_j(x)$ 表示 x 的高位 j 位，则

$$c_1 = m_1 \oplus M_j(DES(S))$$

所以

$$m_1 = c_1 \oplus M_j(DES(S))$$

(3) OFB 模式。OFB 是 Output Feedback 的缩写，意为输出反馈。

OFB 的流程图如图 3.17 所示。

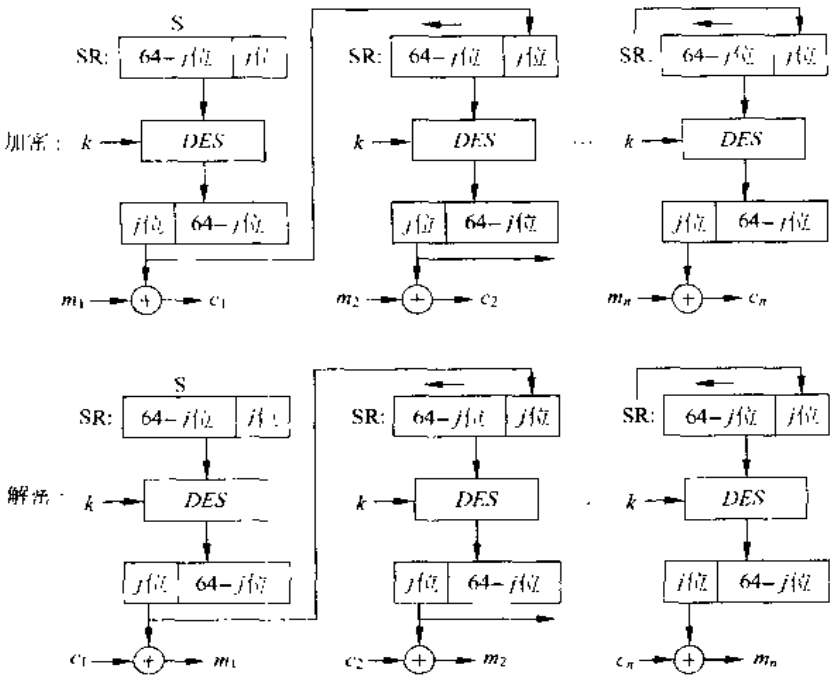


图 3.17

3. 随机化

加密过程的随机因素是很多的，比如通信的日期能双方核准，可利用它对 8 个 S 盒进行置换。

首先介绍若干组合数学的一个结果。比如一个正整数 A_1，$0 \leqslant A_1 \leqslant n! - 1$ 可表示成

$$A_1 = a_{n-1}(n-1)! + a_{n-2}(n-2)! + \cdots + a_2 2! + a_1$$

其中 $0 \leqslant a_i \leqslant i$，$i = 1,2,\cdots,n-1$

$$\left\lfloor \frac{A_1}{2} \right\rfloor = A_2 = a_{n-1}\frac{(n-1)!}{2} + a_{n-2}\frac{(n-2)!}{2} + \cdots + a_2$$

余数

$$r_1 = a_1, \quad r_1 = A_1 - 2A_2$$

$$\left\lfloor \frac{A_2}{3} \right\rfloor = A_3 = a_{n-1}\frac{(n-1)!}{3!} + a_{n-2}\frac{(n-2)!}{3!} + \cdots + a_3$$

余数

$$r_2 = a_2, \quad r_2 = A_2 - 3A_3$$

令 $A_i = \left\lfloor \frac{A_{i-1}}{i} \right\rfloor$，余数 $r_{i-1} = a_i$，$r_{i-1} = A_{i-1} - iA_i$，$i = 1,2,\cdots,n-1$

因此正整数 A 可用 $(a_{n-1},a_{n-2},\cdots,a_2,a_1)$ 来表示，其中 $0 \leqslant a_i \leqslant i$，$i = 1,2,\cdots,n-1$。

下面介绍一个 $(a_{n-1},a_{n-2},\cdots,a_2,a_1)$，其中

$$0 \leqslant a_i \leqslant i, \quad i = 1,2,\cdots,n-1$$

和一个 $1,2,\cdots,n$ 的排列相对应的办法。

令 a_i 表示某一排列，其中 $i+1$ 这个数前(或后)面比它小的数的个数。

举例说明如下，假如 $n=4$，设已知

$$(a_3,a_2,a_1) \quad (2,2,0)$$

由于 $a_3=2$，表明 4 所在的位置，后面有两个数比它小，故有状态：

再考虑 $a_2=2$，说明 3 所在的位置后面也有两个数比它小，故有

最后 $a_1=0$，即 2 后面没有比它小的数，故

最后一个空位便是 1，故(2,2,0)对应一排列

$$3 \quad 4 \quad 1 \quad 2$$

若将"后面"改为"前面"，可得排列：

$$2 \quad 1 \quad 4 \quad 3$$

DES 的密钥虽有 64 比特，设 $k=k_1k_2\cdots k_{64}$，其实 $k_8,k_{16},\cdots,k_{64}$ 在加密过程中没起作用。引进

$$h=\sum_{i=1}^{8}k_i10^{i-1} \bmod 40320$$

其中

$$40320=8!, \quad 0\leqslant k\leqslant 8!-1$$

可将 h 对应一 1,2,3,4,5,6,7,8 的排列：$p=p_1p_2p_3p_4p_5p_6p_7p_8$，作 S 盒的顺序，达到将密钥扩大到 64 比特的作用。

又如利用通信的时间作为随机数，例如，2002 年 8 月 25 日可记为

$$250802$$

$$8!=40320$$

$$250802\equiv 242 \bmod 40320$$

$$\left\lfloor\frac{242}{2}\right\rfloor=121, \quad r_1=0$$

$$\left\lfloor\frac{121}{3}\right\rfloor=40, \quad r_2=1$$

$$\left\lfloor\frac{40}{4}\right\rfloor=10, \quad r_3=0$$

$$\left\lfloor\frac{10}{5}\right\rfloor=2, \quad r_4=0$$

$$\left\lfloor\frac{2}{6}\right\rfloor=0, \quad r_5=2$$

$$r_6=r_7=0$$

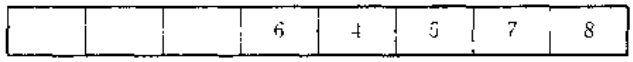
故 $242=2\cdot5!+2!$ 对应于

$$(0,0,2,0,0,1,0)$$

$a_7-a_7=0$，故有

$a_6=2$，故有

$a_5=a_5=0$，故有



$a_4=1,a_4=0$，故有

即对应一排列

$$1\quad 3\quad 2\quad 6\quad 4\quad 5\quad 7\quad 8$$

这可作为 S 盒的排列顺序，即 S_2 与 S_3 对换，然后是 S_0,S_1,S_3,S_2,S_4，即 S 盒的顺序是

$$S_1\quad S_3\quad S_2\quad S_6\quad S_4\quad S_5\quad S_7\quad S_8$$

通过时间来驱动选择 S 盒的排列顺序来提高抗攻击的能力。

3.2 IDEA 密 码

3.2.1 IDEA 加密算法

DES 数据加密标准的出现在近代密码学的历史上是一件大事，它将走完自己的发展历程，面临更新与提高。近年来分组密码新的算法不断涌现，IDEA 是其中的佼佼者。我们说它是佼佼者，是指它在方法论上是新颖的，而且速度快。IDEA 是 International Data Encryption Algorithm 的缩写，意为国际数据加密算法。由中国学者来学嘉博士与著名密码学家 James Massey 于 1990 年联合提出，后经修改于 1992 年最后完成。它的明文与密文块都是 64 比特，但密钥长 128 比特。加密与解密也相同，只是密钥各异。无论用软件或硬件来实现都不难。加密、解密运算速度都非常快。

IDEA 加密算法如图 3.18 所示。

64 比特的数据块分成 4 个子块，每一子块 16 比特，令这 4 个子块为 X_1,X_2,X_3 和 X_4 作为迭代第 1 轮的输入，全部共 8 轮迭代。每轮迭代都是 4 个子块彼此间以及 16 比特的子密钥进行异或，$\bmod 2^{16}$ 作加法运算，$\bmod(2^{16}+1)$ 作乘运算。任何一轮迭代第 3 和第 4 子块互换，每一轮迭代运算步骤如下：

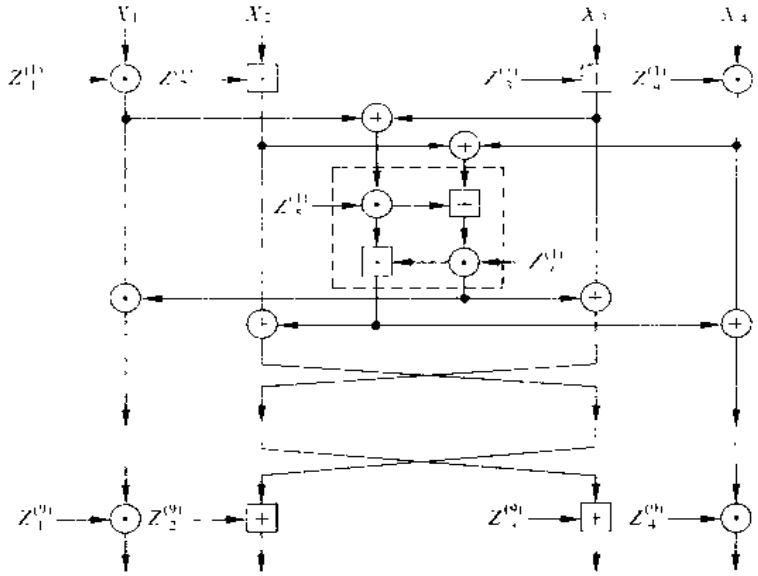


图 3.18

⊕ 16 比特子块间逐位作异或运算

⊙ 16 比特的整数作 mod($2^{16}+1$)的乘法运算

⊞ 16 比特的整数作 mod 2^{16}的加法运算

(1) X_1 和第 1 个子密钥 $Z_1^{(1)}$ 作乘法运算。

(2) X_2 和第 2 个子密钥 $Z_2^{(1)}$ 作加法运算。

(3) X_3 和第 3 个子密钥 $Z_3^{(1)}$ 作加法运算。

(4) X_4 和第 4 个子密钥 $Z_4^{(1)}$ 作乘法运算。

(5) (1)和(3)的结果作异或运算。

(6) (2)和(4)的结果作异或运算。

(7) (5)的结果与第 5 子密钥块作乘法运算。

(8) (6)和(7)的结果作加法运算。

(9) (8)的结果与第 6 个子密钥块作乘法运算。

(10) (7)和(9)的结果作加法运算。

(11) (1)和(9)的结果作异或运算。

(12) (3)和(9)的结果作异或运算。

(13) (2)和(10)的结果作异或运算。

(14) (4)和(10)的结果作异或运算。

结果的输出为(11)、(13)、(12)、(14)。除最后一轮(第 9 轮)外,第 2 和第 3 块交换。

第 8 轮结束后,最后输出变换有:

(1) X_1 和第 1 子密钥块作乘法运算。

(2) X_2 和第 2 子密钥块作加法运算。

(3) X_3 和第 3 子密钥块作加法运算。

(4) X_4 和第 4 子密钥块作乘法运算。

3.2.2 IDEA 密码的子密钥生成

子密钥的产生过程也很容易，1～8 轮，每轮有 6 个子密钥，最后一轮只有 4 个，共 52 个子密钥。每个子密钥 16 比特。设密钥

$$k = k_1 k_2 \cdots k_{128}$$

将它分成 8 段，依次为

$$Z_1^{(1)} = k_1 k_2 \cdots k_{16}, \quad Z_2^{(1)} = k_{17} k_{18} \cdots k_{32},$$
$$Z_3^{(1)} = k_{33} k_{34} \cdots k_{48}, \quad Z_4^{(1)} = k_{49} k_{50} \cdots k_{64},$$
$$Z_5^{(1)} = k_{65} k_{66} \cdots k_{80}, \quad Z_6^{(1)} = k_{81} k_{82} \cdots k_{96},$$
$$Z_1^{(2)} = k_{97} k_{98} \cdots k_{112}, \quad Z_2^{(2)} = k_{113} k_{114} \cdots k_{128}$$

再将 k 向左旋转移位 25 比特得

$$k_{26} k_{27} \cdots k_{128} k_1 k_2 \cdots k_{25}$$

如上所说，分 8 段，前 4 段正好是第 2 轮的子密钥 $Z_3^{(2)}, Z_4^{(2)}, Z_5^{(2)}, Z_6^{(2)}$；后 4 段依次是 $Z_1^{(3)}, Z_2^{(3)}, Z_3^{(3)}, Z_4^{(3)}$。继续向左旋转移位 25 比特，依次是 $Z_5^{(3)}, Z_6^{(3)}, Z_1^{(4)}, Z_2^{(4)}, Z_3^{(4)}, Z_4^{(4)}, Z_5^{(4)}, Z_6^{(4)}$。

继续以上步骤，直到将 52 个子密钥生成完毕为止。

3.2.3 IDEA 密码的解密运算

IDEA 的解密过程和加密完全一样，只不过解密用的子密钥如下：

轮次	加密子密钥块	解密子密钥块
1	$Z_1^{(1)} Z_2^{(1)} Z_3^{(1)} Z_4^{(1)} Z_5^{(1)} Z_6^{(1)}$	$(Z_1^{(9)})^{-1} - Z_2^{(9)} - Z_3^{(9)} (Z_4^{(9)})^{-1} Z_5^{(8)} Z_6^{(8)}$
2	$Z_3^{(1)} Z_2^{(1)} Z_1^{(2)} Z_4^{(1)} Z_5^{(2)} Z_6^{(2)}$	$(Z_1^{(8)})^{-1} - Z_3^{(8)} - Z_2^{(8)} (Z_4^{(8)})^{-1} Z_5^{(7)} Z_6^{(7)}$
3	$Z_1^{(3)} Z_2^{(3)} Z_3^{(3)} Z_4^{(3)} Z_5^{(3)} Z_6^{(3)}$	$(Z_1^{(7)})^{-1} - Z_3^{(7)} - Z_2^{(7)} (Z_4^{(7)})^{-1} Z_5^{(6)} Z_6^{(6)}$
4	$Z_1^{(4)} Z_2^{(4)} Z_3^{(4)} Z_4^{(4)} Z_5^{(4)} Z_6^{(4)}$	$(Z_1^{(6)})^{-1} - Z_3^{(6)} - Z_2^{(6)} (Z_4^{(6)})^{-1} Z_5^{(5)} Z_6^{(5)}$
5	$Z_1^{(5)} Z_2^{(5)} Z_3^{(5)} Z_4^{(5)} Z_5^{(5)} Z_6^{(5)}$	$(Z_1^{(5)})^{-1} - Z_3^{(5)} - Z_2^{(5)} (Z_4^{(5)})^{-1} Z_5^{(4)} Z_6^{(4)}$
6	$Z_1^{(6)} Z_2^{(6)} Z_3^{(6)} Z_4^{(6)} Z_5^{(6)} Z_6^{(6)}$	$(Z_1^{(4)})^{-1} - Z_3^{(4)} - Z_2^{(4)} (Z_4^{(4)})^{-1} Z_5^{(3)} Z_6^{(3)}$
7	$Z_1^{(7)} Z_2^{(7)} Z_3^{(7)} Z_4^{(7)} Z_5^{(7)} Z_6^{(7)}$	$(Z_1^{(3)})^{-1} - Z_3^{(3)} - Z_2^{(3)} (Z_4^{(3)})^{-1} Z_5^{(2)} Z_6^{(2)}$
8	$Z_1^{(8)} Z_2^{(8)} Z_3^{(8)} Z_4^{(8)} Z_5^{(8)} Z_6^{(8)}$	$(Z_1^{(2)})^{-1} - Z_3^{(2)} - Z_2^{(2)} (Z_4^{(2)})^{-1} Z_5^{(1)} Z_6^{(1)}$
9	$Z_1^{(9)} Z_2^{(9)} Z_3^{(9)} Z_4^{(9)}$	$(Z_1^{(1)})^{-1} - Z_2^{(1)} - Z_3^{(1)} (Z_4^{(1)})^{-1}$

这里 Z^{-1} 表示 $Z \bmod (2^{16}+1)$ 乘法的逆，即

$$Z \odot Z^{-1} \equiv 1 \bmod (2^{16}+1)$$

$-Z$ 表示 $Z \bmod 2^{16}$ 加法运算的逆，即

$$Z \boxplus (-Z) \equiv 0 \bmod 2^{16}$$

也就是说解密运算的第 1 轮用到的 6 个子密钥依次是

$$(Z_1^{(9)})^{-1}, -Z_2^{(9)}, -Z_3^{(9)}, (Z_4^{(9)})^{-1}, Z_5^{(8)}, Z_6^{(8)}$$

依此类推。特别是如何求 Z^{-1} 使

$$Z \odot Z^{-1} \equiv 1 \bmod (2^{16}+1)$$

后面通过例子来说明。

下面讨论解密运算的正确性，也就是说通过解密运算恢复明文。

图 3.19 中的 A 表达步骤(1)～(4)的操作，B 表达(5)～(14)的步骤，包括第 2、第 3 两部分的交换。

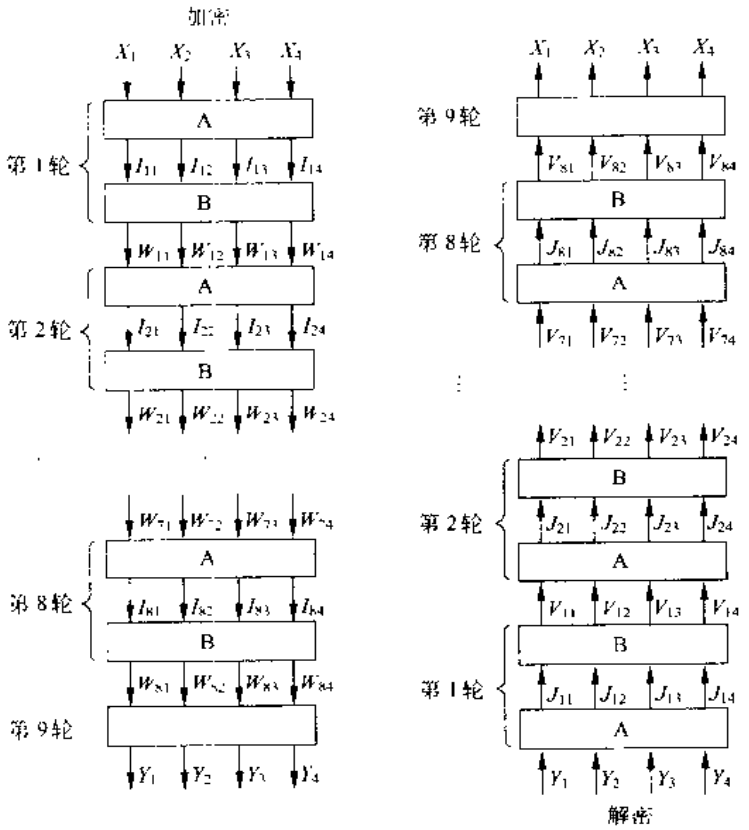


图 3.19

明文设为(X_1, X_2, X_3, X_4)，密文为(Y_1, Y_2, Y_3, Y_4)。

加密运算的第 9 轮有：

$$Y_1 = W_{91} \odot Z_1^{(9)}, \qquad Y_2 = W_{93} \boxplus Z_2^{(9)},$$
$$Y_3 = W_{92} \boxplus Z_3^{(9)}, \qquad Y_4 = W_{94} \odot Z_4^{(9)}$$

Y_1, Y_2, Y_3, Y_4 作为解密运算的输入，解密运算的第 1 轮有

$$J_{11} = Y_1 \odot (Z_1^{(9)})^{-1} = W_{81} \odot Z_1^{(9)} \odot (Z_1^{(9)})^{-1} = W_{81}$$
$$J_{12} = Y_2 \boxplus (-Z_2^{(9)}) = W_{83} \boxplus Z_2^{(9)} \boxplus (-Z_2^{(9)}) = W_{83}$$
$$J_{13} = Y_3 \boxplus (-Z_3^{(9)}) = W_{82} \boxplus Z_3^{(9)} \boxplus (Z_3^{(9)}) = W_{82}$$
$$J_{14} = Y_4 \odot (Z_4^{(9)})^{-1} = W_{84} \odot Z_4^{(9)} \odot (Z_4^{(9)})^{-1} = W_{84}$$

可见解密的第 1 轮 A 的输出恰好是加密第 8 轮的输出。

MA 是 Multiplication Addition 的缩写，它是 IDEA 加密算法的主要构成模块。如图

3.20 所示，$MA^{(0)}(U,V)$ 表示 MA 的左边的输出，$MA^{(1)}(U,V)$ 表示 MA 的右边输出，则有

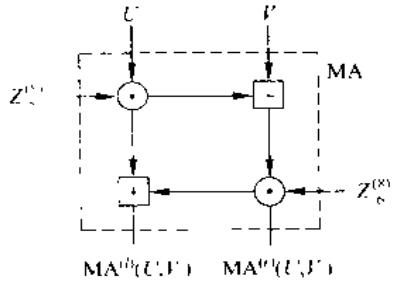


图 3.20

$$
\begin{aligned}
\boldsymbol{W}_{81} &= I_{81} \boxplus MA^{(1)}(I_{81} \boxplus I_{83}, I_{82} \boxplus I_{84}) \\
\boldsymbol{W}_{82} &= I_{82} \boxplus MA^{(0)}(I_{81} \boxplus I_{83}, I_{82} \boxplus I_{84}) \\
\boldsymbol{W}_{83} &= I_{82} \boxminus MA^{(0)}(I_{81} \boxplus I_{83}, I_{82} \boxplus I_{84}) \\
\boldsymbol{W}_{84} &= I_{84} \boxplus MA^{(0)}(I_{83} \boxplus I_{83}, I_{82} \boxplus I_{84})
\end{aligned}
$$

而

$$
\begin{aligned}
V_{11} &= J_{11} \boxplus MA^{(1)}(J_{11} \boxplus J_{13}, J_{12} \boxplus J_{14}) \\
&= \boldsymbol{W}_{81} \boxplus MA^{(1)}(\boldsymbol{W}_{81} \boxplus \boldsymbol{W}_{83}, \boldsymbol{W}_{82} \boxplus \boldsymbol{W}_{84}) \\
&= I_{81} \boxplus MA^{(1)}(I_{81} \boxplus I_{83}, I_{82} \boxplus I_{84}) \\
&\quad \boxplus MA^{(1)}[I_{81} \boxplus MA^{(1)}(I_{81} \boxplus I_{83}, I_{82} \boxplus I_{84}) \\
&\quad \boxminus I_{82} \boxplus MA^{(0)}(I_{81} \boxminus I_{83}, I_{82} \boxplus I_{84}), \\
&\quad I_{82} \boxplus MA^{(0)}(I_{81} \boxplus I_{83}, I_{82} \boxplus I_{84}) \\
&\quad \boxplus MA^{(0)}(I_{81} \boxplus I_{83}, I_{82} \boxplus I_{84})] \\
&= I_{81} \boxplus MA^{(1)}(I_{81} \boxplus I_{83}, I_{82} \boxplus I_{84}) \\
&\quad \boxplus MA^{(1)}(I_{81} \boxplus I_{83}, I_{82} \boxplus I_{84}) = I_{81}
\end{aligned}
$$

同样可证

$$V_{12} = I_{83}, \quad V_{13} = I_{82}, \quad V_{14} = I_{84}$$

这就证明了解密运算第 1 轮 B 的输出，正好是加密运算第 8 轮的 A 的输入，不过第 2 轮和第 3 轮互换位置。

依此类推，可证

$$V_{81} = I_{11}, \quad V_{82} = I_{13}, \quad V_{83} = I_{12}, \quad V_{84} = I_{14}$$

最后一轮的输出依次是

$$
\begin{aligned}
V_{81} \odot (Z_1^{(1)})^{-1} &= I_{11} \odot (Z_1^{(1)})^{-1} = X_1 \odot Z_1^{(1)} \odot (Z_1^{(1)})^{-1} = X_1 \\
V_{83} \boxplus (-Z_2^{(1)}) &= I_{12} \boxplus (-Z_2^{(1)}) = X_2 \boxplus (Z_2^{(1)}) \boxplus (-Z_2^{(1)}) = X_2 \\
V_{82} \boxplus (-Z_3^{(1)}) &= I_{13} \boxplus (-Z_3^{(1)}) = X_3 \boxplus (Z_3^{(1)}) \boxplus (-Z_3^{(1)}) = X_3 \\
V_{84} \odot (Z_4^{(1)})^{-1} &= I_{14} \odot (Z_4^{(1)})^{-1} = X_4 \odot Z_4^{(1)} \odot (Z_4^{(1)})^{-1} = X_4
\end{aligned}
$$

故输出是明文 (X_1, X_2, X_3, X_4)。

3.2.4 举例

假定已知密钥 k 为 computersecurity

明文 m 为 Tsinghua

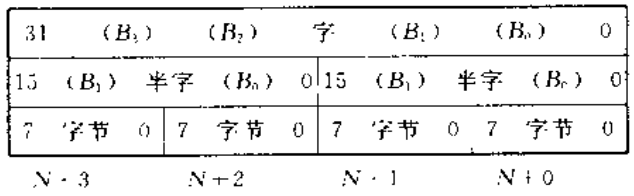
它们的 ASCII 码分别为

$$
\begin{aligned}
k=&\ 11000110\quad 11110110\quad 10110110\quad 00001110\\
&\ 10101110\quad 00101110\quad 10100110\quad 01001110\\
&\ 11001110\quad 10100110\quad 11000110\quad 10101110\\
&\ 01001110\quad 10010110\quad 00101110\quad 10011110\\
m=&\ 00101010\quad 11001110\quad 10010110\quad 01110110\\
&\ 11100110\quad 00010110\quad 10101110\quad 10000110
\end{aligned}
$$

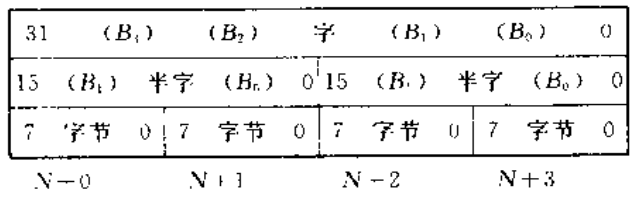
在讨论这个例子之前，应说明一点的是：$k=k_1k_2\cdots k_{128}$，其中 k_1 是最低位，k_{128} 是最高位。

在大多数计算机系统中，存储器是以字节为单位编址的。假设一个字由 4 个字节组成，我们使用 B_3，B_2，B_1，B_0 来分别表示这 4 个字节。其中 B_3 是字的最高有效字节，B_0 是最低有效字节。B_3，B_2，B_1，B_0 在以字节为单位的存储器中存放的次序有两种方法：Little-Endian(小端)和 Big-Endian(大端)。

Little-Endian：



Big Endian：



使用 Little-Endian 方法时，假设字的存储器地址为 N，则字节 B_3，B_2，B_1，B_0 依次存放在地址为 $N+3$，$N+2$，$N+1$，$N+0$ 的存储器单元，即，字地址等于最低有效字节地址。采用 Little-Endian 方法的处理机有 Intel 80x86/Pentium，DEC VAX，DEC Alpha 等。

使用 Big-Endian 方法时，字节 B_3，B_2，B_1，B_0 依次存放在地址为 $N+0$，$N+1$，$N+2$，$N+3$ 的存储器单元，即，字地址等于最高有效字节地址。采用 Big-Endian 方法的处理机有 IBM 360/370，Motorola 68K，MIPS，SPARC，HPPA 等。

$$
\begin{aligned}
Z_1^{(1)}&=11000110\quad 11110110\\
Z_2^{(1)}&=10110110\quad 00001110\\
Z_3^{(1)}&=10101110\quad 00101110\\
Z_4^{(1)}&=10100110\quad 01001110
\end{aligned}
$$

$$Z_3^{(1)}=11001110\quad 10100110$$
$$Z_7^{(1)}=11000110\quad 10101110$$
$$Z_1^{(2)}=01001110\quad 10010110$$
$$Z_8^{(2)}=00101110\quad 10011110$$

$Z_3^{(2)}$ 到 $Z_4^{(3)}$ 为将 k 向左旋转 25 比特，然后依次取 16 比特一组得

$$Z_1^{(2)}=00011101\quad 01011100$$
$$Z_4^{(2)}=01011101\quad 01001100$$
$$Z_5^{(2)}=10011101\quad 10011101$$
$$Z_6^{(2)}=01001101\quad 10001101$$
$$Z_1^{(3)}=01011100\quad 10011101$$
$$Z_2^{(3)}=00101100\quad 01011101$$
$$Z_7^{(3)}=00111101\quad 10001101$$
$$Z_4^{(3)}=11101101\quad 01101100$$

$Z_5^{(3)}$ 到 $Z_6^{(4)}$ 为继续向左旋转 25 比特，然后分别取 16 比特一组，如此进行下去，这里不一一列举。

将明文 m(64 比特)分成 4 组，每组 16 比特：

$$X_1=00101010\quad 11001110$$
$$X_2=10010110\quad 01110110$$
$$X_3=11100110\quad 00010110$$
$$X_4=10101110\quad 10000110$$

加密的第 1 步，即步骤(1)为计算

$$X_1\odot Z_1^{(1)}$$

但
$$X_1=2^{14}+2^{13}+2^{12}+2^9+2^8+2^6+2^4+2^2=29524$$
$$=(0111001101010100)_2$$
$$Z_1^{(1)}=(0110111101100011)_2=28515$$
$$=2^{14}+2^{13}+2^{11}+2^{10}+2^5+2^3+2^6+2^5+2+1$$
$$X_1\odot Z_1^{(1)}=X_1\cdot Z_1^{(1)}\bmod(2^{16}+1)$$
$$=29524\times 28515\bmod(2^{16}+1)$$
$$=54091$$
$$=(1101001101001111)_2$$

故步骤(1)的结果为

$$1101001101001111$$

步骤(2)为计算

$$X_2\boxplus Z_2^{(1)}$$

$$X_2=(0110111001101001)_2$$
$$=2^{14}+2^{13}+2^{11}+2^{10}+2^9+2^6+2^5+2^3+1$$
$$=28265$$

$$Z_2^{(1)} = (0111000001101101)_2$$
$$= 2^{14} + 2^{13} + 2^{12} + 2^6 + 2^5 + 2^4 + 2^3 + 1$$
$$= 28781$$
$$X \boxplus Z^{(1)} = X_2 + Z_2^{(1)} \bmod 2^{16}$$
$$= 28265 + 28781 \bmod 2^{16}$$
$$= 57046 \bmod 2^{16}$$
$$= (1101111011010110)_2$$

故步骤(2)的结果为

1101111011010110

现在将第一趟的 14 个步骤的结果列于下：

第 1 趟

Step 1：,1101,0011,0100,1111
Step 2：,1101,1110,1101,0110
Step 3：,1101,1100,1101,1100
Step 4：,0110,0001,1001,1101
Step 5：,0000,1111,1001,0011
Step 6：,1011,1111,0100,1011
Step 7：,1111,0111,1101,1110
Step 8：,1011,0111,0010,1001
Step 9：,0011,1101,1101,1111
Step 10：,0011,0101,1011,1101
Step 11：,1110,0110,1111,0010
Step 12：,0110,1110,0110,1001
Step 13：,1110,0011,0000,1001
Step 14：,0101,1100,0100,0010

8 趟后得密文

11011000 00110011 01101000 11010010
10010101 01111100 10110000 10010011

若改变密钥 k 为

1100011011110110101110110
0000111010101110001011110
1010011001001110110011110
1010011011000110101011110
*1100111010010110001011110
10011110

* 为改变的位，明文不变，结果得密文

0* 1 1* 1 0* 0 0 0 1* 1* 1 1 0 0 1 1

1* 0* 1 1* 0* 1* 0 1* 0* 1 1* 1 0 0 1 0
1 0 0 0* 0 1 0 0* 1* 0* 1 1 0* 0* 0 1*
0* 0 1 1 0 1* 1* 0 0* 0 1* 0* 0 1* 0* 0*

* 是改变位的标志，共有 29 位改变。

同样若明文改变为

00101010 11001110
10010110 1*1110110
11100110 00010110
10101110 10000110

* 是改变位标志，密钥不变，结果得密文：

1 1 1* 1 1 0 0 1* 0 1* 1 1 0 0 1 1
1* 1 0* 0 1 0 0 1* 0* 1 0 0* 0 1* 1 0
0* 1* 1* 1 1* 1 0 0* 0 1 0* 0* 1 0* 0* 0
0* 1* 0* 0* 0 1* 1* 1* 1* 1* 1* 1* 0 1* 1 0*

共有 31 位改变。

最后讨论解密密钥的计算，以密钥 k 为

computersecurity

明文 m 为

Tsinghua

为例，计算解密子密钥 $\overline{Z}_1^{(1)}$，但

$$\overline{Z}_1^{(1)}=(Z_1^{(9)})^{-1}$$

而

$$Z_1^{(9)}=(1101010111000001)_2$$

即

$$\begin{aligned}Z_1^{(9)}&=2^{15}+2^{14}+2^{12}+2^{10}+2^8+2^7+2^6+1\\&=54721\end{aligned}$$

求 $\overline{Z}_1^{(1)}$ 导致求

$$(54721)^{-1}\bmod(2^{16}+1)$$

但

$$(54721)^{-1}\bmod(2^{16}+1)=46929$$

$$46929=(1011011101010001)_2$$

所以第 1 个解密子密钥 $\overline{Z}_1^{(1)}$ 为

1011011101010001

其他子密钥不一一计算。

现在将求$(54721)^{-1}\bmod(2^{16}+1)$过程如下所示：

$$2^{16}+1=65537$$
$$65537=54721+10816$$
$$10816=65537-54721$$
$$54721=10816\times5+641$$

$$
\begin{aligned}
&641 - 54721 - 10816 \times 5\\
&10816 = 641 \times 16 + 560\\
&560 = 10816 - 16 \times 641\\
&641 = 560 + 81\\
&81 - 611 - 560\\
&560 = 6 \times 81 + 74\\
&74 = 560 - 6 \times 81\\
&81 = 74 + 7\\
&7 = 81 - 74\\
&74 = 10 \times 7 + 4\\
&4 = 74 - 10 \times 7\\
&7 = 4 + 3\\
&3 = 7 - 4\\
&4 = 3 + 1\\
&1 = 4 - 3
\end{aligned}
$$

$$
\begin{aligned}
1 &= 4 - 3 = 4 - (7 - 4) = 2 \times 4 - 7\\
&= 2(74 - 10 \times 7) - 7 = 2 \times 74 - 21 \times 7\\
&= 2 \times 74 - 21(81 - 74) = 23 \times 74 - 21 \times 81\\
&= 23(560 - 6 \times 81) - 21 \times 81\\
&= 23 \times 560 - 159 \times 81\\
&= 23 \times 560 - 159(641 - 560)\\
&= 182 \times 560 - 159 \times 641\\
&= 182(10816 - 16 \times 641) - 159 \times 641\\
&= 182 \times 10816 - 3071 \times 641\\
&= 182 \times 10816 - 3071(54721 - 5 \times 10816)\\
&= 15537 \times 10816 - 3071 \times 54721\\
&= 15537(65537 - 54721) - 3071 \times 5471\\
&= 15537 \times 65537 - 18608 \times 54721
\end{aligned}
$$

或　　$(-18608) \times 54721 \equiv 1 \bmod (2^{16}+1)$

所以　　$(54721)^{-1} = -18608 \bmod (2^{16}+1)$

所以　　$(54721)^{-1} \equiv 46929 \bmod (2^{16}+1)$

又因　$\overline{Z}_2^{(1)} = -Z_2^{(9)}$，　　$Z_2^{(9)} = (1001010111010001)_2$

$\overline{Z}_2^{(1)} + Z_2^{(9)} = 2^{16}$，　所以　$\overline{Z}_2^{(1)} = (0110101000101111)_2$

其他解密子密钥不一一细述。

3.3　AES 新的加密标准

AES 是 advanced encryption standard 的缩写，是取代 DES 的新的加密标准。前面已提过 DES 由于密钥太短，已走完它的生命历程，参加竞选的有 15 个加密算法，最后由

Rijndael 算法胜出。

3.3.1 准备知识

在讨论 AES 之前必须先介绍 AES 用到的 $GF(2^8)$域的运算。AES 的 $GF(2^8)$用到的系数在 $GF(2)$的不可化约多项式，即本原多项式。

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

系数在 $GF(2)$域的多项式全体，mod $p(x)$关于加法及乘法构成 $GF(2^8)$域。

一个字节，8 个比特

$$b_7 \quad b_6 \quad b_5 \quad b_4 \quad b_3 \quad b_2 \quad b_1 \quad b_0$$

对应一多项式

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

例如十六进制数‘57’，即

$$01010111$$

对应于多项式

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

又‘83’即 10000011 对应于

$$x^7 + x + 1$$

$$\text{‘57’} + \text{‘83’} = x^6 + x^4 + x^2 + x + 1 + x^7 + x + 1$$

$$= x^7 + x^6 + x^4 + x^2$$

‘57’+‘83’等于 1101 0100，即

$$\text{‘57’} + \text{‘83’} = \text{D4}$$

在 $GF(2^8)$域上的乘法由一个不可化约的 8 次多项式或 $m(x)$所确定。在 Rijndael 加密算法中取

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

对应于

$$1\ 0001\ 1011$$

或改为十六进制表示

$$0001\ 0001\ 1011$$

即

$$11B$$

下面讨论两个十六进制数的乘法。

例如‘57’·‘83’对应于

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

$$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$$

$$= (x^5 + x^3)(x^8 + x^4 + x^3 + x + 1) + x^7 + x^6 + 1$$

$$\equiv x^7 + x^6 + 1 \bmod (x^8 + x^4 + x^3 + x + 1)$$

也可以看作

$$\text{'57'} \cdot \text{'83'} = x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x + 1$$

由于

$$x^8 = x^4 + x^3 + x + 1, \qquad x^9 = x^5 + x^4 + x^2 + x,$$
$$x^{10} = x^6 + x^5 + x^3 + x^2, \qquad x^{11} = x^7 + x^6 + x^4 + x^3,$$
$$x^{12} = x^5 + x^4 + x^3 + x^2 = x^7 + x^5 + x^3 + x + 1$$

故

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

所以

$$\begin{aligned} \text{'57'} \cdot \text{'83'} = {} & (x^6 + x^3 + x^2 + 1) + (x^7 + x^6 + x^4 + x^3) \\ & + (x^7 + x^6 + x^4 + x) + (x^4 + x^3 + x + 1) \\ & + x^6 + x^5 + x^4 + x^3 + 1 \\ = {} & x^7 + x^6 + 1 \end{aligned}$$

例'57'·'13'

'13'即 0001 0011，对应于多项式 $x^4 + x + 1$。

$$\begin{aligned} & (x^6 + x^4 + x^2 + x + 1)(x^4 + x + 1) \\ & = x^{10} + x^5 + x^7 + x + 1 \\ & x^{10} = x^6 + x^5 + x^3 + x^2 \end{aligned}$$

故'57'·'13'对应于

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

对应于二进制数 1111 1110，用十六进制数表示为'57'·'13'=FE。

3.3.2 系数在 $GF(2^8)$ 的多项式

一个 4 字节的向量对应一系数在 $GF(2^8)$ 上次方低于 4 的多项式。这样的多项式相加时，只要对相应的系数作简单的加法。

乘法则比较复杂。假定 $GF(2^8)$ 上有两个多项式

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

令 $c(x) = c_6 x^6 + c_5 x^5 + c_4 x^4 + c_3 x^3 + c_2 x^2 + c_1 x + c_0$，则有

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

$$a_5 = a_3 b_2 \oplus a_2 b_3$$

$$a_6 = a_3 b_3$$

$c(x)$已不再用4字节向量表示。令 $M(x)=x^4+1$。mod $M(x)$，$c(x)$可降为4次以下。

$$x^n \bmod (x^4+1) \equiv x^{n \bmod 4}$$

例如 $x^7 = x^3 \cdot x^4 \equiv x^3 \bmod (x^4+1)$，　$x^7 \equiv x^3 \bmod (x^4+1)$。

令　　$$d(x) \equiv a(x)b(x) \bmod (x^4+1)$$

用　　$$d(x) = a(x) \otimes b(x)$$

表示它，

$$d(x) = d_3 x^3 + d_2 x^2 + d_1 x + d_0$$

$$d_0 = a_0 b_0 \oplus a_3 b_1 \oplus a_2 b_2 \oplus a_1 b_3$$

$$d_1 = a_1 b_0 \oplus a_0 b_1 \oplus a_3 b_2 \oplus a_2 b_3$$

$$d_2 = a_2 b_0 \oplus a_1 b_1 \oplus a_0 b_2 \oplus a_3 b_3$$

$$d_3 = a_3 b_0 \oplus a_2 b_1 \oplus a_1 b_2 \oplus a_0 b_3$$

或$b(x)$乘以固定多项式$a(x)$的运算可表示为以下矩阵形式：

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

$(x^4+1)^2 = x^4+1$ 由于在$GF(2)$域上，所以 x^4+1 在$GF(2^8)$上，不是不可化约多项式。

所以乘以固定多项式不一定是可逆的。Rijndael密码选一有逆运算的固定多项式。

若$b(x)$乘以 x 有

$$b_3 x^4 + b_2 x^3 + b_1 x^2 + b_0 x$$

mod (x^4+1)，即令 $x^4=1$，得结果

$$b_2 x^3 + b_1 x^2 + b_0 x + b_3$$

或

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 00 & 00 & 00 & 01 \\ 01 & 00 & 00 & 00 \\ 00 & 01 & 00 & 00 \\ 00 & 00 & 01 & 00 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

即 $a_1=01$，其余 $a_i=00$ 的结果。

所以 $x \otimes b(x)$对应于对向量的字节作循环移位，这个结果可推广到一般的形式

$$x^n \otimes b(x)$$

3.3.3 若干说明

Rijndael 加密算法是分组长度可变、密钥长度也可变的分组密码。分组长度、密钥长度彼此独立地确定为 128、192、256 比特。

称加密过程的中间结果为状态 State。状态 State 可用类似于元素为一字节的矩阵来表示，矩阵的行数为 4，列数为 N_b，N_b 等于分组长度除以 32，如下所示。

a_{00}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}
a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}

密钥也是一个长方形字节数组，4 行，列数为密钥长度除以 32，如下所示。

k_0	k_1	k_2	k_3
k_{10}	k_{11}	k_{12}	k_{13}
k_{20}	k_{21}	k_{22}	k_{23}
k_{30}	k_{31}	k_{32}	k_{33}

加密算法的输入是明文，转换为状态字节，依顺序 a_{00}、a_{10}、a_{20}、a_{30}、a_{01}、a_{11}、a_{21}、a_{31}…。加密结束时，输出也是从状态字节中按相同的顺序提取。

加密的轮数用 N_r 表示，N_r 和 N_b、N_k 的关系如表 3.2 所示。

表 3.2

N_k \ N_b	4	6	8
4	10	12	14
6	12	12	14
8	14	14	14

其中 N_b 是分组的列数，密钥 k 的列数为 N_k。

3.3.4 轮变换

轮变换由以下 4 个不同的变换组成。或

```
Round(State, RoundKey)
{
(1) ByteSub(State);
(2) ShiftRow(State);
```

(3) MixColumn(State);
(4) AddRoundKey(State, RoundKey);
}

最后一轮的密码算法略有不同。

```
FinalRound(State, RoundKey)
{
ByteSub(State);
ShiftRow(State);
AddRoundKey(State, RoundKey);
}
```

可见最后一轮 Mix Column 消失了。

现在对 AES 的流程图及 4 种变换分别介绍，如图 3.21 所示。

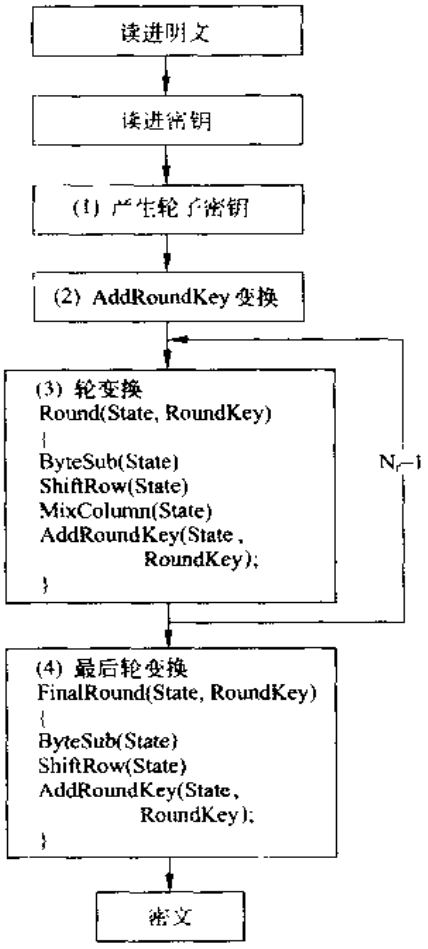


图 3.21

1. ByteSub 变换

ByteSub 即 Byte Substitution 是一个独立作用于状态字节的非线性变换，它由以下两个步骤组成。

(a) 在 $GF(2^8)$域，求乘法的逆运算，即对于

$$a \in GF(2^8)$$

求 $\beta \in GF(2^8)$，使

$$a \cdot \beta - \beta \cdot a \equiv 1 \bmod(x^8 + x^4 + x^2 + x + 1)$$

(b) 在 $GF(2)$域作变换：

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

由于所有的运算都在 $GF(2^8)$域上进行，最后的结果都在 $GF(2^8)$上。若 $g \in GF(2^8)$是 $GF(2^8)$的本原元素，则对于 $\alpha \in GF(2^8)$，$\alpha \neq 0$，则存在 $\beta \in GF(2^8)$，使得

$$\beta \equiv g^{\alpha} \bmod(x^8 + x^4 + x^2 + x + 1)$$

由于

$$g^{255} \equiv 1 \bmod(x^8 + x^4 + x^2 + x + 1)$$

所以

$$g^{255-\alpha} \equiv \beta^{-1} \bmod(x^8 + x^4 + x^2 + x + 1)$$

例如十六进制数'03'，即 00000011 就是一个本原元素。利用 $g =$ '03' 建立起 $GF(2^8)$域上元素 ab 与'xy'的对应关系，使

$$ab \equiv g^{xy} \bmod(x^8 + x^4 + x^2 + x + 1)$$

以及'xy'的逆元素，最后落实到求

$$\text{ByteSub}(xy)$$

的表格。

例如求 FE 的逆

查表 B，得

$$\text{FE} \equiv (03)^{20}$$

$$70 = 01110000$$

$$\text{FE} - 70 = (11111111) - (01110000)$$

$$= (10001111) = 8\text{F}$$

所以

$$(\text{FE})^{-1} = (03)^{8\text{F}}$$

查表 A

$$(03)^{8\text{F}} = 41$$

即

$$(FE)^{-1} = 41$$

故 C 表也可查得

$$(FE)^{-1} = 41$$

表 3.3 是计算

$$z \equiv g^{ab} \bmod p(x)$$

的结果。其中 $p(x) = x^8 + x^4 + x^3 + x + 1$，$a$，$b$ 都是十六进制数；$g =$ '03'。

表 3.3

a \ b	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	03	05	0F	11	33	55	FF	1A	2E	72	96	A1	F8	13	35
1	5F	E1	38	48	D8	73	95	A4	F7	02	06	0A	1E	22	66	AA
2	E5	34	5C	E4	37	59	EB	26	6A	BE	D9	70	90	AB	E6	31
3	53	F5	04	0C	14	3C	44	CC	4F	D1	68	B8	D3	6E	B2	CD
4	4C	D4	67	A9	E0	3B	4D	D7	62	A6	F1	08	18	28	78	88
5	83	9E	B9	D0	6B	BD	DC	7F	81	98	B3	CE	49	DB	76	9A
6	B5	C4	57	F9	10	30	50	F0	0B	1D	27	69	BB	D6	61	A3
7	FE	19	2B	7D	87	92	AD	EC	2F	71	93	AE	E9	20	60	A0
8	FB	16	3A	4E	D2	6D	B7	C2	5D	E7	32	56	FA	15	3F	41
9	C3	5E	E2	3D	47	C9	40	C0	5B	ED	2C	74	9C	BF	DA	75
A	9F	BA	D5	64	AC	EF	2A	7E	82	9D	BC	DF	7A	8E	89	80
B	9B	B6	C1	58	E8	23	65	AF	EA	25	6F	B1	C8	43	C5	54
C	FC	1F	21	63	A5	F4	07	09	1B	2D	77	99	B0	CB	46	CA
D	45	CF	4A	DE	79	8B	86	91	A8	E3	3E	42	C6	51	F3	0E
E	12	36	5A	EE	29	7B	8D	8C	8F	8A	85	94	A7	F2	0D	17
F	39	4B	DD	7C	84	97	A2	FD	1C	24	6C	B4	C7	52	F6	01

例如求 $z \equiv g^{A5} \bmod p(x)$，查表中 A 行，5 列得 EF，即

$$g^{A5} \bmod p(x) = EF$$

又如

$$g^{CA} \bmod p(x) = 77$$

显然 $z \in GF(2^8)$。

表 3.4 是已知 $z = XY$，X、Y 都是十六进制的数，$z \in GF(2^8)$，求 a，b，使

$$z \equiv g^{ab} \bmod p(x)$$

是前面表的逆运算。如果说前面表 A 是求 $g^{ab} \bmod p(x)$ 的指数运算，表 B 是在 $GF(2^8)$ 上求离散对数，例如 E 行 F 列的元素为 A5，即求 EF 的逆

$$g^{A5} \bmod p(x) \equiv EF$$

表 3.4

X \ Y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	FF	19	01	32	02	1A	C6	4B	C7	1B	68	33	EE	DF	03
1	64	04	E0	0E	34	8D	81	EF	4C	71	08	C8	F8	69	1C	C1
2	7D	C2	1D	B5	F9	B9	27	6A	4D	E4	A6	72	9A	C9	09	78
3	65	2F	8A	05	21	0F	E1	24	12	F0	82	45	35	93	DA	8E
4	96	8F	DB	BD	36	D0	CE	94	13	5C	D2	F1	40	46	83	38
5	66	DD	FD	30	BF	06	8B	62	B3	25	E2	98	22	88	91	10
6	7E	6E	48	C3	A3	B6	1E	42	3A	6B	28	54	FA	85	3D	BA
7	2B	79	0A	15	9B	9F	5E	CA	4E	D4	AC	E5	F3	73	A7	57
8	AF	58	A8	50	F4	EA	D6	74	4F	AE	E9	D5	E7	E6	AD	E8
9	2C	D7	75	7A	EB	16	0B	F5	59	CB	5F	B0	9C	A9	51	A0
A	7F	0C	F6	6F	17	C4	49	EC	D8	43	1F	2D	A4	76	7B	B7
B	CC	BB	3E	5A	FB	60	B1	86	3B	52	A1	6C	AA	55	29	9D
C	97	B2	87	90	61	BE	DC	FC	BC	95	CF	CD	37	3F	5B	D1
D	53	39	84	3C	41	A2	6D	47	14	2A	9E	5D	56	F2	D3	AB
E	44	11	92	D9	23	20	2E	89	B4	7C	B8	26	77	99	E3	A5
F	67	4A	ED	DE	C5	31	FE	18	0D	63	8C	80	C0	F7	70	07

表 3.5 是 Rijndael 的 S 置换结果，例如求 ByteSub(0C)，查表的 0 行 C 列得 FE。

表 3.5

X \ Y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

即　　ByteSub(0C)＝FE

又如　　ByteSub(DA)＝57

2. ShiftRow 变换

在 ShiftRow 中状态 State 的最后 3 行循环移位的位数如表 3.6 所示。

表 3.6

N_b \ N_r	c_1	c_2	c_3
4	1	2	3
6	1	2	3
8	1	3	4

即第 1 行移位 c_1 字节，第 2 行移位 c_2 字节，第 3 行移位 c_3 字节。c_1，c_2，c_3 与分组长 N_0 有关。

State 的最后 3 行移位运算表示为

ShiftRow(State)

State 在 ShiftRow 作用下的结果如图 3.22 所示。

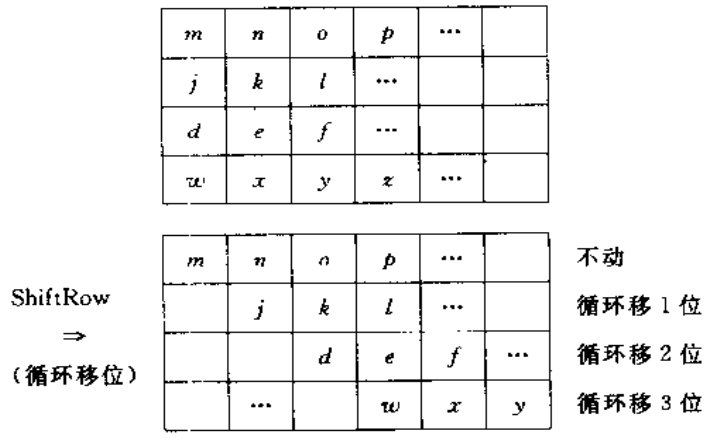


图　3.22

3. MixColumn 变换

在 MixColumn 作用下，State 的列看作是域 $GF(2^8)$ 的多项式，mod(x^4+1) 乘以 $c(x)$ 的结果

$$c(x) = (03)x^3 + (01)x^2 + (01)x + (02)$$

这里(03)为十六进制表示，依此类推。$c(x)$ 与 x^4+1 互素，故存在逆 $d(x)$＝'0B' x^3 + '0D' x^2 + '0G' x + '0E' 使 $c(x) \cdot d(x) \equiv$ 'D1' mod(x^4+1)。

设 $b(x)=c(x)\otimes a(x)$，有

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 01 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

MixColumn 作用于 State 可表示为

MixColumn(State)

4. RoundKeyAddition

轮子密钥对 State 作按位的 EXOR 运算，表示为

AddRoundKey(State, RoundKey)

即

$$(a_{ij})_{4\times 6} \oplus (k_{ij})_{4\times 6} = (b_{ij})_{4\times 6}$$

或

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} & a_{05} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \end{bmatrix} \oplus \begin{bmatrix} k_{00} & k_{01} & k_{02} & k_{03} & k_{04} & k_{05} \\ k_{10} & k_{11} & k_{12} & k_{13} & k_{14} & k_{15} \\ k_{20} & k_{21} & k_{22} & k_{23} & k_{24} & k_{25} \\ k_{30} & k_{31} & k_{32} & k_{33} & k_{34} & k_{35} \end{bmatrix} =$$

$$\begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} & b_{04} & b_{05} \\ b_{10} & b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{20} & b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ b_{30} & b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \end{bmatrix}$$

3.3.5 子密钥的生成

子密钥是由密钥导出的，这个过程包含了两个组成部分，一个是密钥膨胀，另一个是轮子密钥的选取，基本原理为如下 3 点：

(1) 轮子密钥的比特数总和等于分组长度乘以轮的数目加 1，即每分组长 128 比特，轮数等于 10 时，轮子密钥的比特数总和为

$$128 \times (10 + 1) = 1408 \text{ 比特}$$

(2) 密钥扩充为膨胀密钥(Expanded Key)。

(3) 轮子密钥是由这些膨胀密钥中取出来的，第 1 轮子密钥由最先 N_b 个字组成，第 2 轮子密钥为其次 N_b 个字，等等。

1. 膨胀密钥

膨胀密钥是 4 字节字的数组，用

$$W[N_b \times (N_r + 1)]$$

表示，最先 N_k 个字包含密钥，所有其他字用最小下标递归地定义，密钥安排依赖于 N_k 的值。

现采用类似于 C 语言形式描述如下：

若 $N_k \leqslant 6$，则有

```
KeyExpansion (byte Key[4 * Nk] word W[Nb * (Nr+1)])
{
    for(i=0; i<Nk; i++)
        W[i]=(Key[4*i],Key[4*i+1],Key[4*i+2],Key[4*i+3]);
    for(i=Nk; i<Nb*(Nr+1);i++)
    {
        temp=W[i-1];
        if(i%Nk==0)
            temp=SubByte(RotByte(temp))^Rcon[i/Nk];
        W[i]=W[i-Nk]^temp;
    }
}
```

若 $N_k>6$，则有

```
KeyExpansion(byte Key[4 * Nk] word W[Nb * (Nr+1)])
{
    for(i=0;i<Nk;i++)
        W[i]=(Key[4*i],Key[4*i-1],Key[4*i+2],Key[4*i+3]);
    for(i=Nk;i<Nb*(Nr+1);i++)
    {
        temp=W[i-1];
        if(i%Nk==0)
            temp=SubByte(RotByte(temp))^Rcon[i/Nk];
        else if (i%Nk==4)
            temp=SubByte(temp);
        W[i]=W[i-Nk]^temp;
    }
}
```

其中

Rcon[i]=(RC[i],'00','00','00')

RC[1]=1(即'01')

$RC[i]=x\cdot(RC[i-1])=x^{(i-1)}$

2. 子密钥的选取

第 i 轮子密钥是由 $W[N_b*i]$到 $W[N_b*(i+1)]$间的位构成。以 $N_b=6$，$N_k=4$ 为例，如图 3.23 所示。

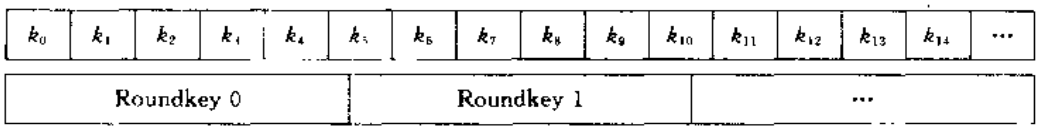


图 3.23

3.3.6 加密算法的形式化叙述

Rijndael 加密算法用伪 C 语言表示如下：

```
Rijndael (State,CipherKey)
{
    KeyExpansion(CipherKey,ExpandedKey);
    AddRoundKey(State,ExpandedKey);
    For(i=1;i<Nr;i++)Round(State,ExpandedKey+Nb*i);
    FinalRound(State,ExpandedKey+Nb*Nr);
}
```

KeyExpansion 必须事先执行，或

```
Rijndael(State,ExpandedKey)
{
    AddRoundKey(State,ExpandedKey);
    For(i=1;i<Nr;i++)Round(State,ExpandedKey+Nb*i);
    FinalRound(State,ExpandedKey+Nb*Nr);
}
```

3.3.7 解密

1. 准备知识

(1) ShiftRow 是一循环移位，它的逆也是对底下 3 行分别做 N_b-C_1，N_b-C_2，N_b-C_3 字节移位。

(2) ByteSub 的逆是利用 Rijndael 的 S 盒的逆做字节置换，见表 3.7。

表 3.7

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E

续表

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

这是解密用的 S 盒的逆。

(3) MixColumn 的逆与 MixColumn 类似，每列乘以 $d(x)$。

$$d(x) = \text{'0B'}x^3 + \text{'0D'}x^2 + \text{'0G'}x + \text{'0E'}$$

下列等式成立

$$(\text{'03'}x^3 + \text{'01'}x^2 + \text{'01'}x + 02) \otimes d(x) = \text{'01'}$$

(4) AddRoundKey 的逆是它自身。

2. 两轮 Rijndael 的逆

两轮的逆为

```
InvRound(State,RoundKey)
{
    AddRoundKey(State,RoundKey)
    InvMixColumn(State);
    InvShiftRow(State);
    InvByteSub(State);
}
```

FinalRound 的逆为；

```
InvFinalRound(State,RoundKey)
{
    AddRoundKey(State,RoundKey)
    InvShiftRow(State);
    InvByteSub(State);
}
```

两轮的 Rijndael 变异解密算法包含有 FinalRound 的逆，跟着是 Round 的逆，最后是 RoundKey Addition。有：

```
AddRoundKey(State,ExpandedKey+2 * Nb);
InvShiftRow(State);
```

```
InvByteSub(State);
AddRoundKey(State,ExpandedKey+Nb);
InvMixColumn(State);
InvShiftRow(State);
InvByteSub(State);
AddRoundKey(State,ExpandedKey);
```

3.3.8 代数性质

首先注意到,ShiftRow 对字节的值无影响,仅仅改变字节的位置;而 ByteSub 仅仅对各字节进行作用而不问它的位置,所以 ShiftRow 和 ByteSub 与它们的次序无关。

其次

```
AddRoundKey(State,RoundKey);
InvMixColumn(State);
```

可以改为

```
InvMixColumn(State);
AddRoundKey(State,InvRoundKey);
```

InvRoundKey 是由 InvMixColumn 作用于对应的 RoundKey 所得的,这是由于线性变换 A 有

$$A(x+k) = A(x) + A(k)$$

所以两轮 Rijndael 变异的逆可以变换为

```
AddRoundKey(State,ExpandedKey+2*Nb);
InvByteSub(State);
InvShiftRow(State);
InvMixColumn(State);
AddRoundKey(State,I_ExpandedKey+Nb);
InvByteSub(State);
InvShiftRow(State);
AddRoundKey(State,ExpandedKey);
```

两轮的讨论可推及一般。

下面定义解密的 Round 和 FinalRound 如下:

```
I_Round(State,I_RoundKey)
{
InvByteSub(State);
InvShiftRow(State);
InvMixColumn(State);
```

```
AddRoundKey(State,I_RoundKey);
}
I_FinalRound(State,I_RoundKey)
{
InvByteSub(State);
InvShiftRow(State);
AddRoundKey(State,RoundKey);
}
```

Rijndael 密码的逆可表示为：

```
I_Rijndael(State,CipherKey)
{
I_KeyExpansion(CipherKey,I_ExpandedKey);
AddRoundKey(State,I_ExpandedKey + Nb * Nr);
For(r = Nr - 1;r>0;r--)Round(State,I_ExpandedKey + Nb * r);
FinalRound(State,I_ExpandedKey);
}
```

解密的膨胀密钥定义为

(1) 作用 KeyExpansion；

(2) 作用 InvMixColumn 于各轮子密钥，最先与最后一个除外。

```
I_KeyExpansion(CipherKey,I_ExpandedKey)
{
    KeyExpansion(Cipher,I_ExpandedKey);
    for(i=1;i<Nr;i++)
    MixColumn(I_ExpandedKey+Nb*i);
}
```

3.3.9 举例

假定分组长度为 128 比特，轮数 $N_r=10$。

$B=00\ 01\ 02\ 03\ 04\ 05\ 06\ 07\ 08\ 09\ 0A\ 0B\ 0C\ 0D\ 0E\ 0F$

$K=00\ 01\ 02\ 03\ 04\ 05\ 06\ 07\ 08\ 09\ 0A\ 0B\ 0C\ 0D\ 0E\ 0F$

$$\boldsymbol{B}[ij]=\begin{bmatrix}00 & 04 & 08 & 0C\\ 01 & 05 & 09 & 0D\\ 02 & 06 & 0A & 0E\\ 03 & 07 & 0B & 0F\end{bmatrix}$$

第一个子密钥：

$$K_0=(\boldsymbol{W}_0,\boldsymbol{W}_1,\boldsymbol{W}_2,\boldsymbol{W}_3)$$

$$W_0 = 0\ 0\ 0\ 1\ 0\ 2\ 0\ 3$$
$$W_1 = 0\ 4\ 0\ 5\ 0\ 6\ 0\ 7$$
$$W_2 = 0\ 8\ 0\ 9\ 0\ A\ 0\ B$$
$$W_3 = 0\ C\ 0\ D\ 0\ E\ 0\ F$$

所以 N_b—$\boldsymbol{B}$ 矩阵的列数为 4。

第一步计算子密钥

$$W_i = \begin{cases} W_{i-N_k} \oplus \text{tempt}, & i \bmod N_k = 0 \\ W_{i-N_k} \oplus W_{i-1}, & i \bmod N_k \neq 0 \end{cases}$$

其中 N_k = 密钥长度 / 32 = 4

$\text{tempt} = \text{ByteSub}[S_1(W_{i-1})] \oplus r_{con_k}$

$r_{con_k} = (RC_i, 00, 00, 00)$

$RC_1 = 1$

$W_4 = W_0 \oplus W_3$

$= (04, 05, 06, 07) \oplus (D6, AA, 74, FD)$

由于

$$04 \oplus D6 = (00000100) \oplus (11010110)$$
$$= (11010010) = D2$$
$$05 \oplus AA = (00000101) \oplus (10101010)$$
$$= (10101111) = AF$$
$$06 \oplus 74 = (00000110) \oplus (01110100)$$
$$= (01110010) = 72$$
$$07 \oplus FD = (00000111) \oplus (11111101)$$
$$= (11111010) = FA$$

所以 $W_4 = D2\ AF\ 72\ FA$

类似方法可得

$$W_5 = DA\ A6\ 78\ F1$$
$$W_6 = D6\ AB\ 76\ FE$$

所以子密钥

$$\boldsymbol{K}_1 = D6\ AA\ 74\ FD\ D2\ AF\ 72\ FA\ DA\ A6\ 78\ F1\ D6\ AB\ 76\ FE$$

或写成

$$\boldsymbol{K}_1 = \begin{pmatrix} D6 & D2 & DA & D6 \\ AA & AF & A6 & AB \\ 74 & 72 & 78 & 76 \\ FD & FA & F1 & FE \end{pmatrix}$$

$$W_7 = W_4 \oplus \text{ByteSub}(S_1 W_6)$$
$$= B6\ 92\ CF\ 0B$$

$$W_9 = 64\ 3D\ BD\ F1$$

$$W_{10} = BE\ 9B\ C5\ 00$$

$$W_{11} = 68\ 30\ B3\ FE$$

所以子密钥

$$K_2 = B6\ 92\ CF\ 0B\ 64\ 3D\ BD\ F1\ BE\ 9B\ C5\ 00\ 68\ 30\ B3\ FE$$

或写成

$$K_2 = \begin{pmatrix} B6 & 64 & BE & 68 \\ 92 & 3D & 9B & 30 \\ CF & BD & C5 & B3 \\ 0B & F1 & 00 & FE \end{pmatrix}$$

$$W_{12} = B6\ FF\ 74\ 4E$$

$$W_{13} = D2\ C2\ C9\ BF$$

$$W_{14} = 6C\ 59\ 0C\ BF$$

$$W_{16} = 04\ 69\ BF\ 41$$

得密钥

$$K_3 = \begin{pmatrix} B6 & D2 & 6C & 04 \\ FF & C2 & 59 & 69 \\ 74 & C9 & 0C & BF \\ 4E & BF & BF & 41 \end{pmatrix}$$

类似可得

$$K_4 = \begin{pmatrix} 47 & 95 & F9 & FD \\ F7 & 35 & 6C & 05 \\ F7 & 3E & 32 & 8D \\ BC & 03 & BC & FD \end{pmatrix}$$

$$K_5 = \begin{pmatrix} 3C & A9 & 50 & AD \\ AA & 9F & F3 & F6 \\ A3 & 9D & AF & 22 \\ E8 & EB & 57 & AA \end{pmatrix}$$

$$K_6 = \begin{pmatrix} 5E & F7 & A7 & 0A \\ 39 & A6 & 55 & A3 \\ 0F & 92 & 3D & 1F \\ 7D & 96 & C1 & 6B \end{pmatrix}$$

$$K_7 = \begin{pmatrix} 14 & E3 & 44 & 4E \\ F9 & 5F & 0A & A9 \\ 70 & E2 & DF & C0 \\ 1A & 8C & 4D & 26 \end{pmatrix}$$

$$K_8=\begin{pmatrix}47 & A4 & E0 & AE\\ 43 & 1C & 16 & BF\\ 87 & 65 & BA & 7A\\ 35 & B9 & F4 & D2\end{pmatrix}$$

$$K_9=\begin{pmatrix}54 & F0 & 10 & BE\\ 99 & 85 & 93 & 2C\\ 32 & 57 & ED & 97\\ D1 & 68 & 9C & 4E\end{pmatrix}$$

$$K_{10}=\begin{pmatrix}13 & E3 & F3 & 4D\\ 11 & 94 & 07 & 2B\\ 1D & 4A & A7 & 30\\ 7F & 17 & 8B & C5\end{pmatrix}$$

1. 初始轮 B[ij]⊕K[ij]

$$\begin{pmatrix}00 & 04 & 08 & 0C\\ 01 & 05 & 09 & 0D\\ 02 & 06 & 0A & 0E\\ 03 & 07 & 0B & 0F\end{pmatrix}\ominus\begin{pmatrix}00 & 04 & 08 & 0C\\ 01 & 05 & 09 & 0D\\ 02 & 06 & 0A & 0E\\ 03 & 07 & 0B & 0F\end{pmatrix}=\begin{pmatrix}00 & 00 & 00 & 00\\ 00 & 00 & 00 & 00\\ 00 & 00 & 00 & 00\\ 00 & 00 & 00 & 00\end{pmatrix}=B[ij]$$

2. 第 1 轮

(1) 作 ByteSub 运算

由于 00 和 00 对应，故第一步对 b_0 求乘法的逆得

$$\begin{pmatrix}00 & 00 & 00 & 00\\ 00 & 00 & 00 & 00\\ 00 & 00 & 00 & 00\\ 00 & 00 & 00 & 00\end{pmatrix}$$

第二步作变换

$$\begin{pmatrix}y_0\\ y_1\\ y_2\\ y_3\\ y_4\\ y_5\\ y_6\\ y_7\end{pmatrix}=\begin{pmatrix}1 & 0 & 0 & 0 & 1 & 1 & 1 & 1\\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1\\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1\\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1\\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0\\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0\\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0\\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0\end{pmatrix}\begin{pmatrix}0\\ 0\\ 0\\ 0\\ 0\\ 0\\ 0\\ 0\end{pmatrix}+\begin{pmatrix}1\\ 1\\ 0\\ 0\\ 0\\ 1\\ 1\\ 0\end{pmatrix}$$

得

$$\begin{Bmatrix} 63 & 63 & 63 & 63 \\ 63 & 63 & 63 & 63 \\ 63 & 63 & 63 & 63 \\ 63 & 63 & 63 & 63 \end{Bmatrix}$$

(2) ShiftRow

$$\begin{Bmatrix} 63 & 63 & 63 & 63 \\ 63 & 63 & 63 & 63 \\ 63 & 63 & 63 & 63 \\ 63 & 63 & 63 & 63 \end{Bmatrix} \xrightarrow{\text{ShiftRow}} \begin{Bmatrix} 63 & 63 & 63 & 63 \\ 63 & 63 & 63 & 63 \\ 63 & 63 & 63 & 63 \\ 63 & 63 & 63 & 63 \end{Bmatrix}$$

(3) MixColumn

$$\begin{Bmatrix} 63 & 63 & 63 & 63 \\ 63 & 63 & 63 & 63 \\ 63 & 63 & 63 & 63 \\ 63 & 63 & 63 & 63 \end{Bmatrix} \xrightarrow{\text{MixColumn}} \begin{Bmatrix} 63 & 63 & 63 & 63 \\ 63 & 63 & 63 & 63 \\ 63 & 63 & 63 & 63 \\ 63 & 63 & 63 & 63 \end{Bmatrix}$$

(4) KeyAddition

$$\begin{Bmatrix} 63 & 63 & 63 & 63 \\ 63 & 63 & 63 & 63 \\ 63 & 63 & 63 & 63 \\ 63 & 63 & 63 & 63 \end{Bmatrix} \oplus \begin{Bmatrix} D6 & D2 & DA & D6 \\ AA & AF & A6 & AB \\ 74 & 72 & 78 & 76 \\ FD & FA & F1 & FE \end{Bmatrix} = \begin{Bmatrix} B5 & B1 & B9 & B5 \\ C9 & CC & C5 & C8 \\ 17 & 11 & 1B & 15 \\ 9E & 99 & 92 & 9D \end{Bmatrix}$$

第 1 轮的密文

$$C_1 = \text{B5 C9 17 9E B1 CC 11 99 B9 C5 1B 92 B5 C8 15 9D}$$

或写成

$$\begin{Bmatrix} B5 & B1 & B9 & B5 \\ C9 & CC & C5 & C8 \\ 17 & 11 & 1B & 15 \\ 9E & 99 & 92 & 9D \end{Bmatrix}$$

3. 第 2 轮

(1) ByteSub

查 ByteSub 表可得

$$\begin{Bmatrix} D5 & C8 & 56 & D5 \\ DD & 4B & A6 & E8 \\ F0 & 82 & AF & 59 \\ 0B & EE & 4F & 5E \end{Bmatrix}$$

ByteSub(B5) = D5

ByteSub(C9) = DD

ByteSub(17) = F0

(2) ShiftRow

$$\begin{bmatrix} D5 & C8 & 56 & D5 \\ DD & 4B & A6 & E8 \\ F0 & 82 & AF & 59 \\ 0B & EE & 4F & 5E \end{bmatrix} \xrightarrow{\text{ShiftRow}} \begin{bmatrix} D5 & C8 & 56 & D5 \\ 4B & A6 & E8 & DD \\ AF & 59 & F0 & 82 \\ 5E & 0B & EE & 4F \end{bmatrix}$$

(3) MixColumn

$$\begin{bmatrix} D5 & C8 & 56 & D5 \\ 48 & A6 & E8 & DD \\ AF & 59 & F0 & 82 \\ 5E & 0B & EE & 4F \end{bmatrix} \xrightarrow{\text{MixColumn}} \begin{bmatrix} 9D & 28 & 91 & 00 \\ F7 & 7F & 78 & A6 \\ 39 & C1 & 6C & C6 \\ 3C & AA & 25 & A5 \end{bmatrix}$$

假如第一列对应一系数在 $GF(2^8)$ 的多项式

$$D5x^3 + 48x^2 + AFx + 5E$$

作

$$(D5x^3 + 48x^2 + AFx + 5E) \cdot (03x^3 + 01x^2 + 01x + 02) \bmod (x^4 + 1)$$
$$\equiv 9Dx^3 + F7x^2 + 39x + 3C$$

类似有

$$(C8x^3 + A6x^2 + 59x + 0B) \cdot (03x^3 + 01x^2 + 01x + 02) \bmod (x^4 + 1)$$
$$\equiv 28x^3 + 7Fx^2 + C1x + AA$$

(4) KeyAddition

$$\begin{bmatrix} 9D & 28 & 91 & 00 \\ F7 & 7F & 78 & A6 \\ 39 & C1 & 6C & C6 \\ 3C & AA & 25 & A5 \end{bmatrix} \oplus \begin{bmatrix} B6 & 64 & BE & 68 \\ 92 & 3D & 9B & 30 \\ CF & BD & C5 & B3 \\ 0B & F1 & 00 & FE \end{bmatrix} = \begin{bmatrix} 2B & 4C & 2F & 68 \\ 65 & 42 & E3 & 96 \\ F6 & 7C & A9 & 75 \\ 37 & 5B & 25 & 5B \end{bmatrix}$$

故第 2 轮的结果为

$$C_2 = 2B\ 65\ F6\ 37\ 4C\ 42\ 7C\ 5B\ 2F\ E3\ A9\ 25\ 68\ 96\ 75\ 5B$$

或写成

$$\begin{bmatrix} 2B & 4C & 2F & 68 \\ 65 & 42 & E3 & 96 \\ F6 & 7C & A9 & 75 \\ 37 & 5B & 25 & 5B \end{bmatrix}$$

4. 第 3 轮

(1) ByteSub

$$\begin{bmatrix} 2B & 4C & 2F & 68 \\ 65 & 42 & E3 & 96 \\ F6 & 7C & A9 & 75 \\ 37 & 5B & 25 & 5B \end{bmatrix} = \begin{bmatrix} F1 & 29 & 15 & 45 \\ 4D & 2C & 11 & 90 \\ 42 & 10 & D3 & 9D \\ 9A & 39 & 3F & 39 \end{bmatrix}$$

(2) ShiftRow

$$\begin{bmatrix} F1 & 29 & 15 & 45 \\ 4D & 2C & 11 & 90 \\ 42 & 10 & D3 & 9D \\ 9A & 39 & 3F & 39 \end{bmatrix} \xrightarrow{\text{ShiftRow}} \begin{bmatrix} F1 & 29 & 15 & 45 \\ 2C & 11 & 90 & 4D \\ D3 & 9D & 42 & 10 \\ 39 & 9A & 39 & 3F \end{bmatrix}$$

(3) MixColumn

$$\begin{bmatrix} F1 & 29 & 15 & 45 \\ 2C & 11 & 90 & 4D \\ D3 & 9D & 42 & 10 \\ 39 & 9A & 39 & 3F \end{bmatrix} \xrightarrow{\text{MixColumn}} \begin{bmatrix} 67 & 66 & FA & 72 \\ FE & 2D & D1 & D0 \\ 2B & AC & 4A & 69 \\ 85 & D8 & 9F & EC \end{bmatrix}$$

(4) KeyAddition

$$\begin{bmatrix} 67 & 66 & FA & 72 \\ FE & 2D & D1 & D0 \\ 2B & AC & 4A & 69 \\ 85 & D8 & 9F & EC \end{bmatrix} \oplus \begin{bmatrix} B6 & D2 & 6C & 04 \\ FF & C2 & 59 & 69 \\ 74 & C9 & 0C & BF \\ 4E & BF & BF & 41 \end{bmatrix} = \begin{bmatrix} D1 & B4 & 96 & 76 \\ 01 & EF & 88 & B9 \\ 5F & 65 & 46 & D6 \\ CB & 67 & 20 & AD \end{bmatrix}$$

故第 3 轮的结果为

$$C_3 = \text{D1 01 5F CB B4 EF 65 67 96 88 46 20 76 B9 D6 AD}$$

第 4 轮到第 9 轮的结果分别是

$$C_4 = \text{8E 17 06 4A 2A 35 A1 83 72 9F E5 9F F3 A5 91 F1}$$

$$C_5 = \text{D7 55 7D D5 59 99 DB 32 59 E2 18 3D 55 8D CD D2}$$

$$C_6 = \text{73 A9 6A 5D 77 99 A5 F3 11 1D 2B 63 68 4B 1F 7F}$$

$$C_7 = \text{1B 6B 85 30 69 EE FC 74 9A FE FD 7B 57 A0 4C D1}$$

$$C_8 = \text{10 7E EA DF B6 F7 79 33 B5 45 7A 6F 08 FD 46 B2}$$

$$C_9 = \text{8E C1 66 48 1A 67 7A A9 6A 14 FF 6E CE 88 C0 10}$$

5. 最后一轮

(1) ByteSub

$$\begin{bmatrix} 8E & 1A & 6A & CE \\ C1 & 67 & 14 & 88 \\ 66 & 7A & FF & C0 \\ 48 & A9 & 6E & 10 \end{bmatrix} = \begin{bmatrix} 19 & A2 & 02 & 8B \\ 78 & 85 & FA & C4 \\ 33 & DA & 16 & BA \\ 52 & D3 & 9F & CA \end{bmatrix}$$

(2) ShiftRow

$$\begin{bmatrix} 19 & A2 & 02 & 8B \\ 78 & 85 & FA & C4 \\ 33 & DA & 16 & BA \\ 52 & D3 & 9F & CA \end{bmatrix} \rightarrow \begin{bmatrix} 19 & A2 & 02 & 8B \\ 85 & FA & C4 & 78 \\ 16 & BA & 33 & DA \\ CA & 52 & D3 & 9F \end{bmatrix}$$

(3) KeyAddition

$$\begin{bmatrix} 19 & A2 & 02 & 8B \\ 85 & FA & C4 & 78 \\ 16 & BA & 33 & DA \\ CA & 52 & D3 & 9F \end{bmatrix} \oplus \begin{bmatrix} 13 & E3 & F3 & 4D \\ 11 & 94 & 07 & 2B \\ 1D & 4A & A7 & 30 \\ 7F & 17 & 8B & C5 \end{bmatrix} = \begin{bmatrix} 0A & 41 & F1 & C6 \\ 94 & 6E & C3 & 53 \\ 0B & F0 & 94 & EA \\ B5 & 45 & 58 & 5A \end{bmatrix}$$

所以　　C=0A 94 0B B5 41 6E F0 45 F1 C3 94 58 C6 53 EA 5A

3.4 RC5 加密算法

RC5 是由以下 3 个参数确定的一种加密算法。

1. w：RC5 分组长度的比特数，比如 w 取 64 比特一组，也可以是 32 比特一组。

2. r：轮数，比如 64 比特的 w，取 $r=12$ 或 16。

3. b：密钥 K 的 8 比特字节数，比如 8 字节，128 比特，一般来说应表示为 RC5 $w/r/b$。比如 RC5-32/12/16 即表示字为 32 比特，加密的分组为 $2w$，即明文、密文的分组长：64 比特。迭代 12 轮。密钥长 16 字节，即 128 比特。根据 w 的不同，定义 P、Q 如下：

w	16	32	64
P	B7E1	B7E15163	B7E151628AED2A6B
Q	9E37	9E3779B9	9E3779B97F4A7C15

其中

P=取最接近于$(e-2)2^w$ 的奇整数。e 是 2.71828…

Q=取最接近于$(\phi-1)2^w$ 的奇整数，其中

$$\phi = \left(\frac{1+\sqrt{5}}{2}\right) = 1.618\cdots$$

假定明文块、密文块都是 64，此时，加密用到由密钥产生的 $2(r+2)$ 个于密钥：

$$S_0, S_1, \cdots, S_{2r+1}$$

其中 r 是轮数。S_i 的规模为 32 比特，$i=0,1,2,\cdots,2r+1$。把输入分成 A、B 两部分，各 32 比特。按 Little Endian 规定，A 的第一字节进入寄存器 A 的低位，第 4 字节进入最高位。RC5 的加密、解密过程如图 3.24 所示。

1. 加密算法

$$A \leftarrow A + S_0$$

$$B \leftarrow B + S_1$$

i 从 1 到 r 做

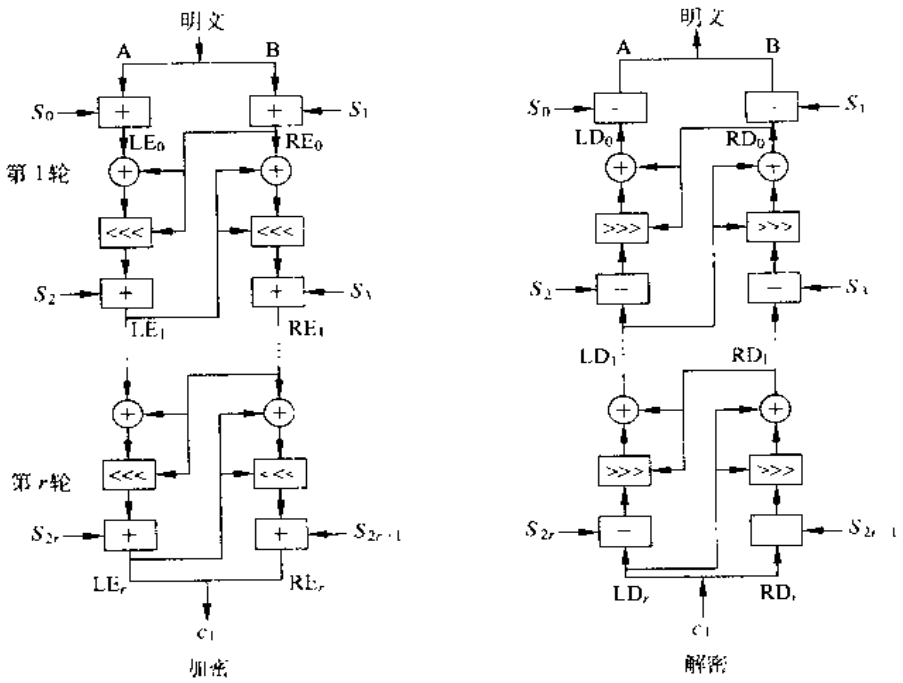


图 3.24

始

$$A \leftarrow ((A \oplus B) <<<< B) + S_{2r}$$
$$B \leftarrow ((B \oplus A) <<<< A) + S_{2r+1}$$

终。

其中"+"为 mod 2^{32}的加法。$x<<<<y$ 表示字 x 向左旋转移位 y 比特。$x>>>>y$ 表示向右旋转移位。

2. 解密算法

i 从 r 到 1 做

始

$$B \leftarrow ((B - S_{2i+1}) >>>> A) \oplus A$$
$$A \leftarrow ((A - S_{2i}) >>>> B) \oplus B$$
$$B \leftarrow B - S_1$$
$$A \leftarrow A - S_0$$

终。

$>>>>$为向右旋转移位 5 比特。"$-$"为 mod 2^{32}的减法。

3. 子密钥的生成

假定密钥 k 有 c 个字，即 $32c$ 比特。必要时补以零。将密钥 k 记入 L 数组，即

$$L = (L_0, L_1, \cdots, L_{c-1})$$
$$S_0 \leftarrow P$$

i 从 1 到 $2r+1$ 做

$$S_i \leftarrow (S_{i-1} + Q) \bmod 2^{32}$$

其中

$$P = \text{B7E15163}, \quad Q = \text{9E3779B9}$$

最后做：

$$i \leftarrow 0, \quad j \leftarrow 0$$
$$A \leftarrow 0, \quad B \leftarrow 0$$
$$n \leftarrow \max(2(r+1), c)$$

做 $3n$ 次：

始

$$A \leftarrow S_i \leftarrow (S_i + A + B) <<< 3$$
$$B \leftarrow L_j \leftarrow (L_j + A + B) <<< (A + B)$$
$$i \leftarrow (i+1) \bmod (2r+2)$$
$$j \leftarrow (j+1) \bmod c$$

终。

3.5 RC6 加密算法

RC6 和 RC5 类似，也是以分组大小、密钥的长短、以及轮数作为可变参数。RC6 更准确的写法应该是 RC6-w/r/b，w 是分组的字数。r 是轮数，b 是密钥的字节数。例如 w=32 字，即 128 比特，r=20，b=16，24 或 32 字节，则写为 RC6-32/20/16，RC6-32/20/24 或 RC6-32/20/32。

RC6 里用到的基本运算有

$a+b$　指 mod 2^w 的整数加法运算；

$a-b$　指 mod 2^w 的整数减法运算；

$a \oplus b$　w 比特字的按位异或运算；

$a \times b$　指 mod 2^w 的乘法运算；

$a <<< b$　对 w 比特字 a 作左旋转移位，移的位数为 b 的最小有效数字 $\log_2 w$ 位；

$a >>> b$　对 w 比特字 a 作右旋转移位，移的位数为 b 的最小有效数字的 $\log_2 w$ 位。

其中对数 $\log_2 w$ 位是以 2 为底，即 $\log_2 w$。

3.5.1 子密钥的生成

RC6 的子密钥生成和 RC5 完全一样。

用户供 b 字节的密钥，必要时补足以 0，依次分配给 c 个长 w 比特的字：

$$L[0], L[1], \cdots, L[c-1]$$

子密钥数目为 $2r+4$，储存在数组 $S[0,1,2,\cdots,2r+3]$ 中。

令

$$P_w = P_{32} = \text{B7E15163}$$
$$Q_w = Q_{32} = \text{9E3779B9}$$

P_w 是从 e--2 的二进制表示导出的，Q_w 是从 ϕ -1 的二进制表示导出的，ϕ 是黄金分割比，子密钥的产生过程如下：

$$S[0] \leftarrow P_w$$

i 从 1 到 $2r+3$ 作：

$$S[i] \leftarrow S[i-1] + Q_w$$
$$A \leftarrow B \leftarrow i \leftarrow j \leftarrow 0$$
$$V = 3 \times \max\{c, 2r+4\}$$

s 从 1 到 V 作：

始

$$A \leftarrow S[i] \leftarrow (S[i] + A + B) <<< 3$$
$$B \leftarrow L[j] \leftarrow (L[j] + A + B) <<< (A + B)$$
$$i \leftarrow (i+1) \bmod (2r+4)$$
$$j \leftarrow (j+1) \bmod c$$

终。

3.5.2 加、解密算法

明文 m 依次存储于 A, B, C, D 4 个寄存器中，每个占 w 比特。w 比特的子密钥为 $S[0,1,\cdots,2r+3]$。算法过程：轮数为 r。

$$B \leftarrow B + S[0]$$
$$D \leftarrow D + S[1]$$

i 从 1 到 r 作：

始

$$t \leftarrow (B \times (2B+1)) <<< \lg w$$
$$u \leftarrow (D \times (2D+1)) <<< \lg w$$
$$A \leftarrow ((A \oplus t) <<< u) + S[2i]$$
$$C \leftarrow ((C \oplus u) <<< t) + S[2i+1]$$
$$(A, B, C, D) \leftarrow (B, C, D, A)$$

终。

RC6 加密算法流程图如图 3.25 所示。

解密算法

密文存放在 w 比特的寄存器 A, B, C, D 中，子密钥储存在 $S[0,1,\cdots,2r+3]$，每一个占 w 比特轮数为 r，

$$C \leftarrow C - S[2r+3]$$
$$A \leftarrow A - S[2r+2]$$

i 从 r 到 1 作：

始

$$(A, B, C, D) \leftarrow (D, A, B, C)$$
$$u \leftarrow (D \times (2D+1)) <<< \lg w$$
$$t \leftarrow (B \times (2B+1)) <<< \lg w$$

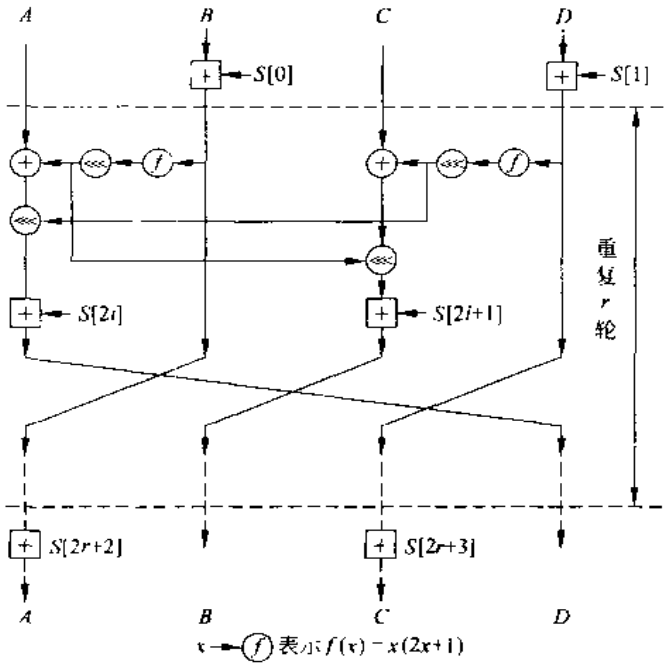


图 3.25

$$C \leftarrow (((C - S[2i+1]) >>> t) \oplus u$$
$$A \leftarrow ((A - S[2i]) >>> u) \oplus t$$

终。

$$D \leftarrow D - S[1]$$
$$B \leftarrow B - S[0]$$

明文存放在 A,B,C,D。

3.6 Serpent 密码

3.6.1 Serpent 加密算法

Serpent 也是 AES 候选算法之一，是 32 轮的 SP 网络作用于 4 个 32 比特字(128 比特明文长度)的分组的加密算法。SP 是 Substition Permutation 的缩写，即置换和排列的意思。32 比特字的各位的下标从 0 到 31。128 比特的分组的下标从 0 到 127。256 比特的密钥的下标从 0 到 255。在外部每个分组都用十六进制表示。

Serpent 将 128 比特的明文 m 加密成 128 比特的密文，设为 c，是在 33 个 128 比特的子密钥 $k_0,k_1,\cdots,k_{32}$ 控制下通过 32 次迭代后得到的。

用户的密钥长度是可变的，规定为 128、192 或 256 比特，少于 256 比特的，在最后有效位后加一个 1，再补以足够多的 0 使达到 256 位。

加密过程包含有

(1) 初始置换 IP。

(2) 32 轮迭代，每轮与子密钥作混合运算，再通过一次 S 盒，最后(最后一轮除外)作一次线性变换。最后一轮线性变换代以附加与密钥作混合运算。

(3) 最后作置换 FP。

IP 和 FP 在后面介绍。

IP 作用于明文 m 得到 B_0，它是第 1 轮的输入。轮次以 0 到 31 记序。即第 1 轮为 0 轮，最后一轮为第 31 轮。最先一轮(即 0 轮)的输出为 B_1，下面一轮(即第 1 轮)的输出为 B_2，第 i 轮的输出为 $B_i(i=0,1,2,\cdots,31)$。最后一轮的线性变换代以附加的密钥混合。输出的结果为 B_{32}。FP 作用于 B_{32}得密文 c。

轮 $R_i(i=0,1,2,\cdots,31)$只利用一个可重复的 S 盒。比如 R_0 用到 S_0。它的 32 个拷贝可以并行应用。比如第 1 个 S_0 的拷贝取 $B_0\oplus k_0$ 的 0,1,2,3 比特作为输入，取中间向量的前面 4 个比特作为输出；S_0 的第 2 个拷贝取 $B_0\oplus k_0$ 的 4～7 比特作为输入，返回后 4 比特，依此类推。中间结果利用线性变换给出 B_1，类似地 R_1 行利用 S_1 的 32 个拷贝，并行地对 $B_1\oplus k_1$ 的输出作线性变换，给出 B_2。

8 个 S 盒用上 4 次。即在第 7 轮用上 S_7 后，第 8 轮重复用 S_0。第 9 轮用上 S_1，等等，最后一轮 R_{31}稍稍有点差异，利用 S_7 于 $B_{31}\oplus k_{31}$，结果对 k_{32}作异或运算，而不是作线性变换。结果 B_{32}作 FP 置换，给出密文。

32 轮，8 个不同的 S 盒，各将 4 比特的输入转换为 4 比特的输出。每一 S 盒用上 4 轮，如 S_7 在第 7 轮用上，如 S_0 又在第 8 轮出现，如此等等。Serpent 的 S 盒如图 3.26 所示。

FP 是 IP 的逆。

现将加密过程形式地叙述如下：

$$B_0 \leftarrow IP(m)$$

$$B_{i+1} \leftarrow R_i(B_i) \quad i=0,1,2,\cdots,31$$

$$C \leftarrow FP(B_{32})$$

其中，

$$R_i(x)=L_i(\hat{S}_i(X\oplus\hat{k}_i)); \quad i=0,1,2,\cdots,30$$

$$R_i(x)=\hat{S}_i(X\oplus\hat{k}_i)\oplus\hat{k}_{32}, \quad i=31$$

这里 $\hat{S}_i$ 即是 $S_{i \bmod 8}$盒并行运算 32 次，L 是线性变换。

线性变换

$$X_0,X_1,X_2,X_3 = S_i(B_i\oplus k_i)$$

$$X_0 \leftarrow X_0 <<< 13$$

$$X_2 \leftarrow X_2 <<< 3$$

$$X_1 \leftarrow X_1\oplus X_0\oplus X_2$$

$$X_3 \leftarrow X_3\oplus X_2\oplus (X_0 << 3)$$

$$X_1 \leftarrow X_1 <<< 1$$

$$X_3 \leftarrow X_3 <<< 7$$

$$X_0 \leftarrow X_0\oplus X_1\oplus X_3$$

$$X_2 \leftarrow X_2\oplus X_3\oplus (X_1 << 7)$$

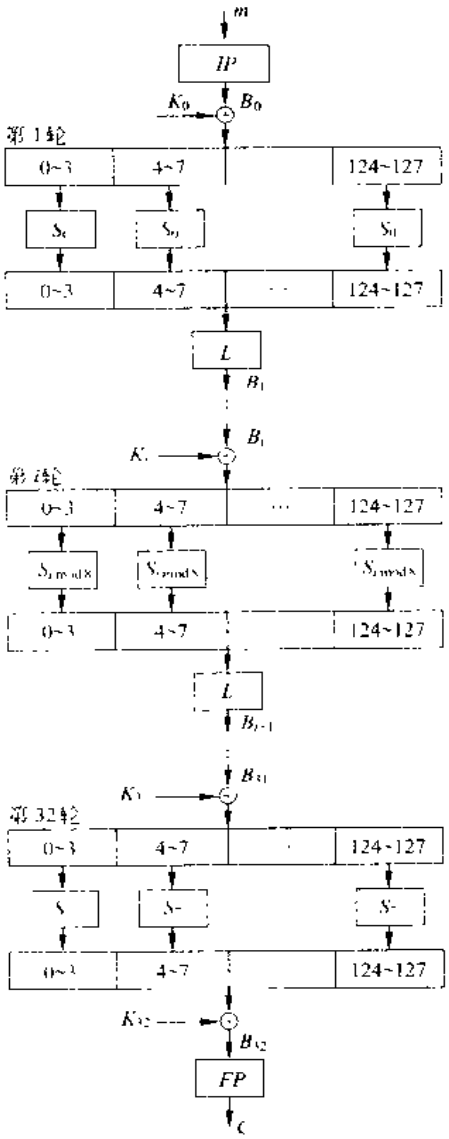


图 3.26

$$X_0 \leftarrow X_0 <<<< 5$$

$$X_2 \leftarrow X_2 <<<< 22$$

$$B_{i+1} \leftarrow X_3, X_1, X_2, X_0$$

其中$<<<$表示旋转，$<<$表示移位。

在最后一轮，上面线性变换被代以附加的密钥混合。

$$B_{32} \leftarrow S_i(B_{31} \oplus k_{31}) \oplus k_{32}$$

每一步有

$$IP(B_i) = \hat{B}_i, \quad IP(k_i) = \hat{k}_i$$

3.6.2 解密运算和子密钥的生成

解密与加密不同之点在于用 S 盒的逆取代 S 盒，顺序颠倒，线性变换用上它的逆，而且子密钥的顺序倒过来，如图 3.27 所示。

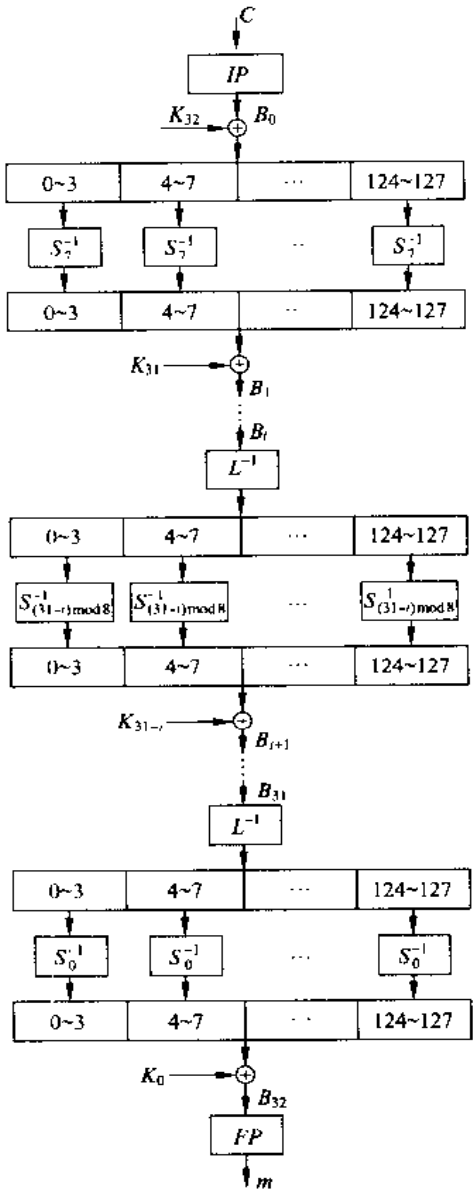


图 3.27

S 盒的逆：

InvS0：	13	3	11	0	10	6	5	12	1	14	4	7	15	9	8	2
InvS1：	5	8	2	14	15	6	12	3	11	4	7	9	1	13	10	0

InvS2:	12	9	15	4	11	14	1	2	0	3	6	13	5	8	10	7
InvS3:	0	9	10	7	11	14	6	13	3	5	12	2	4	8	15	1
InvS4:	5	0	8	3	10	9	7	14	2	12	11	6	4	15	13	1
InvS5:	8	15	2	9	4	1	13	14	11	6	5	3	7	12	10	0
InvS6:	15	10	1	13	5	3	6	0	4	9	14	7	2	12	8	11
InvS7:	3	0	6	13	9	14	15	8	5	12	11	7	10	1	4	2

以 InvS0 为例，其运算过程：a,b,c,d 为各 4 比特的输入，e,f,g,h 为输出。

$$t_1 \leftarrow a,\quad t_2 \leftarrow a \oplus b,\quad t_3 \leftarrow t_1 \vee t_2,\quad t_4 \leftarrow d \oplus t_3$$

$$t_7 \leftarrow d \wedge t_2,\quad t_5 \leftarrow c \oplus t_4,\quad t_8 \leftarrow t_1 \oplus t_4$$

$$g \leftarrow t_2 \oplus t_5,\quad t_{11} \leftarrow a \wedge t_4,\quad t_9 \leftarrow g \wedge t_8$$

$$f \leftarrow t_4 \oplus t_9,\quad t_{12} \leftarrow t_{11} \oplus f,\quad h \leftarrow t_{11} \oplus t_{12}$$

$$e \leftarrow h \oplus t_{14}$$

子密钥的生成：

128 比特密钥 K 分成 8 个 32 比特块

$$W_7, W_6, \cdots, W_0$$

再将它扩大到 $W_0, W_1, \cdots, W_{131}$

$$W_i \leftarrow (W_{i-8} \oplus W_{i-5} \oplus W_{i-3} \oplus W_{i-1} \oplus \phi \oplus i) \lll 11$$

其中 ϕ=9E3779B9。

$$\{h_0, h_1, h_2, h_3\} \leftarrow S_3(W_0, W_1, W_2, W_3)$$

$$\{h_4, h_5, h_6, h_7\} \leftarrow S_2(W_4, W_5, W_6, W_7)$$

$$\{h_8, h_9, h_{10}, h_{11}\} \leftarrow S_1(W_8, W_9, W_{10}, W_{11})$$

$$\{h_{12}, h_{13}, h_{14}, h_{15}\} \leftarrow S_0(W_{12}, W_{13}, W_{14}, W_{15})$$

$$\{h_{16}, h_{17}, h_{18}, h_{19}\} \leftarrow S_7(W_{16}, W_{17}, W_{18}, W_{19})$$

$$\vdots$$

$$\{h_{124}, h_{125}, h_{126}, h_{127}\} \leftarrow S_4(W_{124}, W_{125}, W_{126}, W_{127})$$

$$\{h_{128}, h_{129}, h_{130}, h_{131}\} \leftarrow S_3(W_{128}, W_{129}, W_{130}, W_{131})$$

定义：

$$k_i \leftarrow \{h_{4i}, h_{4i+1}, h_{4i+2}, h_{4i+3}\} \quad i=1,2,\cdots,r$$

3.6.3 附表

IP 表如下所示。

0	32	64	96	1	33	65	97	2	34	66	98	3	35	67	99
4	36	68	100	5	37	69	101	6	38	70	102	7	39	71	103
8	40	72	104	9	41	73	105	10	42	74	106	11	43	75	107
12	44	76	108	13	45	77	109	14	46	78	110	15	47	79	111
16	48	80	112	17	49	81	113	18	50	82	114	19	51	83	115
20	52	84	116	21	53	85	117	22	54	86	118	23	55	87	119
24	56	88	120	25	57	89	121	26	58	90	122	27	59	91	123
28	60	92	124	29	61	93	125	30	62	94	126	31	63	95	127

FP 表如下所示。

0	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60
64	68	72	76	80	84	88	92	96	100	104	108	112	116	120	124
1	5	9	13	17	21	25	29	33	37	41	45	49	53	57	61
65	69	73	77	81	85	89	93	97	101	105	109	113	117	121	125
2	6	10	14	18	22	26	30	34	38	42	46	50	54	58	62
66	70	74	78	82	86	90	94	98	102	106	110	114	118	122	126
3	7	11	15	19	23	27	31	35	39	43	47	51	55	59	63
67	71	75	79	83	87	91	95	99	103	107	111	115	119	123	127

线性变换：下表给出输出的 128 位和输入各位的关系。例如输入的第 72,114,125 位的异或为第 2 位。

116	52	56	70	83	94	105	72	114	125	2	9	15	30	76	84	126	36	90	103
120	56	60	74	87	98	109	1	76	118	2	6	13	19	34	80	88	40	94	107
24	60	64	78	91	102	113	5	80	122	6	10	17	23	38	84	92	44	98	111
28	64	68	82	95	106	117	9	84	126	10	14	21	27	42	88	96	48	102	115
32	68	72	86	99	110	121	2	13	88	14	18	25	31	46	92	100	52	106	119
36	72	76	90	103	114	125	6	17	92	18	22	29	35	50	96	104	56	110	123
1	40	76	80	94	107	118	10	21	96	22	26	33	39	54	100	108	60	114	127
5	14	80	84	98	111	122	14	25	100	26	30	37	43	58	104	112	3	118	
9	48	84	88	102	115	126	18	29	104	30	34	41	47	62	108	116	7	122	
2	13	52	88	92	106	119	22	33	108	34	38	45	51	66	112	120	11	126	
6	17	56	92	96	110	123	26	37	112	38	42	49	55	70	116	124	12	15	76
10	21	60	96	100	114	127	30	41	116	0	42	46	53	59	74	120	16	19	80
3	14	25	100	104	118		34	45	120	4	46	50	57	63	78	124	10	23	84
7	18	29	104	108	122		38	49	124	0	8	50	54	61	67	82	11	27	88
11	22	33	108	112	126		0	42	53	4	12	54	58	65	71	86	18	31	92
2	15	26	37	76	112	116	4	46	57	8	16	58	62	69	75	90	22	35	96
6	19	30	41	80	116	120	8	50	61	12	20	62	66	73	79	94	26	39	100
10	23	31	45	84	120	124	12	54	65	16	24	66	70	77	83	98	30	43	104
0	14	27	38	49	88	124	16	58	69	20	28	70	74	81	87	102	34	47	108
0	4	18	31	42	53	92	20	62	73	24	32	74	78	85	91	106	38	51	112
4	8	22	35	46	57	96	24	66	77	28	36	78	82	89	95	110	42	55	116
8	12	26	39	50	61	100	28	70	81	32	40	82	86	93	99	114	46	59	120
12	16	30	43	54	65	104	32	74	85	36	90	103	118				50	63	124
16	20	34	47	58	69	108	36	78	89	40	94	107	122				0	54	67
20	24	38	51	62	73	112	40	82	93	44	98	111	126				4	58	71
24	28	42	55	66	77	116	44	86	97	2	48	102	115				8	62	75
28	32	46	59	70	81	120	48	90	101	6	52	106	119				12	66	79
32	36	50	63	74	85	124	52	94	105	10	56	110	123				16	70	83
0	36	40	54	67	78	89	56	98	109	14	60	114	127				20	74	87
4	40	44	58	71	82	93	60	102	113	3	18	72	114	118	125		24	78	91
8	44	48	62	75	86	97	64	106	117	1	7	22	76	118	122		28	82	95
12	48	52	66	79	90	101	68	110	121	5	11	26	80	122	126		32	86	99

逆线性变换如下所示。

{53	55	72}	{1	5	20	90	}	‹	15	102}	{3	31	90				}
{57	59	76}	{5	9	24	94	}	‹	19	106}	{7	35	94				}
{61	63	80}	{9	13	28	98	}	‹	23	110}	{11	39	98				}
{65	67	84}	{13	17	32	102	}	‹	27	114}	'1	3	15	20	43	102	}
{69	71	88}	{17	21	36	106	}	‹1	31	118}	‹5	7	19	24	47	106	}
{73	75	92}	{21	25	40	110	}	‹5	35	122}	‹9	11	23	28	51	110	}
{77	79	96}	{25	29	44	114	}	‹9	39	126}	‹13	15	27	32	55	114	}
{81	83	100}	{1	29	33	48	118}	‹2	13	43}	‹1	17	29	31	36	59	118}
{85	87	104}	{5	33	37	52	122}	‹6	17	47}	‹5	21	23	35	40	63	122}
{89	91	108}	{9	37	41	56	126}	‹10	21	51}	‹9	25	27	39	44	67	126}
{93	95	112}	{2	13	41	45	60}	‹14	25	55}	‹2	13	29	31	43	48	71}
{97	99	116}	{6	17	45	49	64}	‹18	29	59}	‹6	17	33	35	47	52	75}
{101	103	120}	{10	21	49	53	68}	{22	33	63}	{10	21	37	39	51	56	79}
‹105	107	124}	{14	25	53	57	72}	{26	37	67}	{14	25	41	43	55	60	83}
‹0	109	111}	{18	29	57	61	76}	{30	41	71}	{18	29	45	47	59	64	87}
{4	113	115}	‹22	33	61	65	80}	{34	45	75}	{22	33	49	51	63	68	91}
{8	117	119}	‹26	37	65	69	84}	{38	49	79}	{26	37	53	55	67	72	95}
{12	121	123}	{30	41	69	73	88}	{42	53	83}	{30	41	57	59	71	76	99}
{16	125	127}	{34	45	73	77	92}	{46	57	87}	{34	45	61	63	75	80	103}
{1	3	20}	‹38	49	77	81	96}	{50	61	91}	{38	49	65	67	79	84	107}
{5	7	24}	‹42	53	81	85	100}	{54	65	95}	{42	53	69	71	83	88	111}
{9	11	28}	‹46	57	85	89	104}	{58	69	99}	{46	57	73	75	87	92	115}
{13	15	32}	‹50	61	89	93	108}	{62	73	103}	{50	61	77	79	91	96	119}
{17	19	36}	‹54	65	93	97	112}	{66	77	107}	{54	65	81	83	95	100	123}
{21	23	40}	{58	69	97	101	116}	{70	81	111}	{58	69	85	87	99	104	127}
{25	27	44}	{62	73	101	105	120}	{74	85	115}	{3	62	73	89	91	103	108}
{29	31	48}	{66	77	105	109	124}	{78	89	119}	{7	66	77	93	95	107	112}
{33	35	52}	{0	70	81	109	113}	‹82	93	123}	{11	70	81	97	99	111	116}
{37	39	56}	{4	74	85	113	117}	{86	97	127}	{15	74	85	101	103	115	120}
{41	43	60}	{8	78	89	117	121}	‹	3	90}	{19	78	89	105	107	119	124}
‹45	47	64}	{12	82	93	121	125}	‹	7	94}	‹0	23	82	93	109	111	123}
{49	51	68}	{1	16	86	97	125}	‹	11	98}	{4	27	86	97	113	115	127}

3.7 Twofish 密码

3.7.1 算法说明

Twofish 加密算法是明文块大小为 128 比特，密钥长度可为 128 比特、192 比特或 256 比特的一种分组密码。它的结构是 16 轮 Feistel 网络，先将明文分解成 4 组，每组 32 比特，分别和 4 个密钥字作异或运算。接着有 16 轮的迭代。其中左端两个 32 位各为 g

函数的输入，其中有一个先旋转 8 比特，g 函数包含有 4 字节宽的依赖于密钥的 S^* 盒，随以基于 MDS 矩阵的线性混合，其中 MDS 是一种变换，后面将专门介绍。两个 g 函数的结果再结合一起通过伪 Hadamard 变换（PHT），再和两个密钥字相加，这样左边的两个结果分别和右边的 2 个 32 比特作异或运算，其中一个先作左旋 1 比特，另一个是右旋 1 比特在后，左右两半换位，作为下一轮的开始，但最后一轮左右换位后和 4 个密钥字作异或运算，产生密文。加密流程见图 3.28。

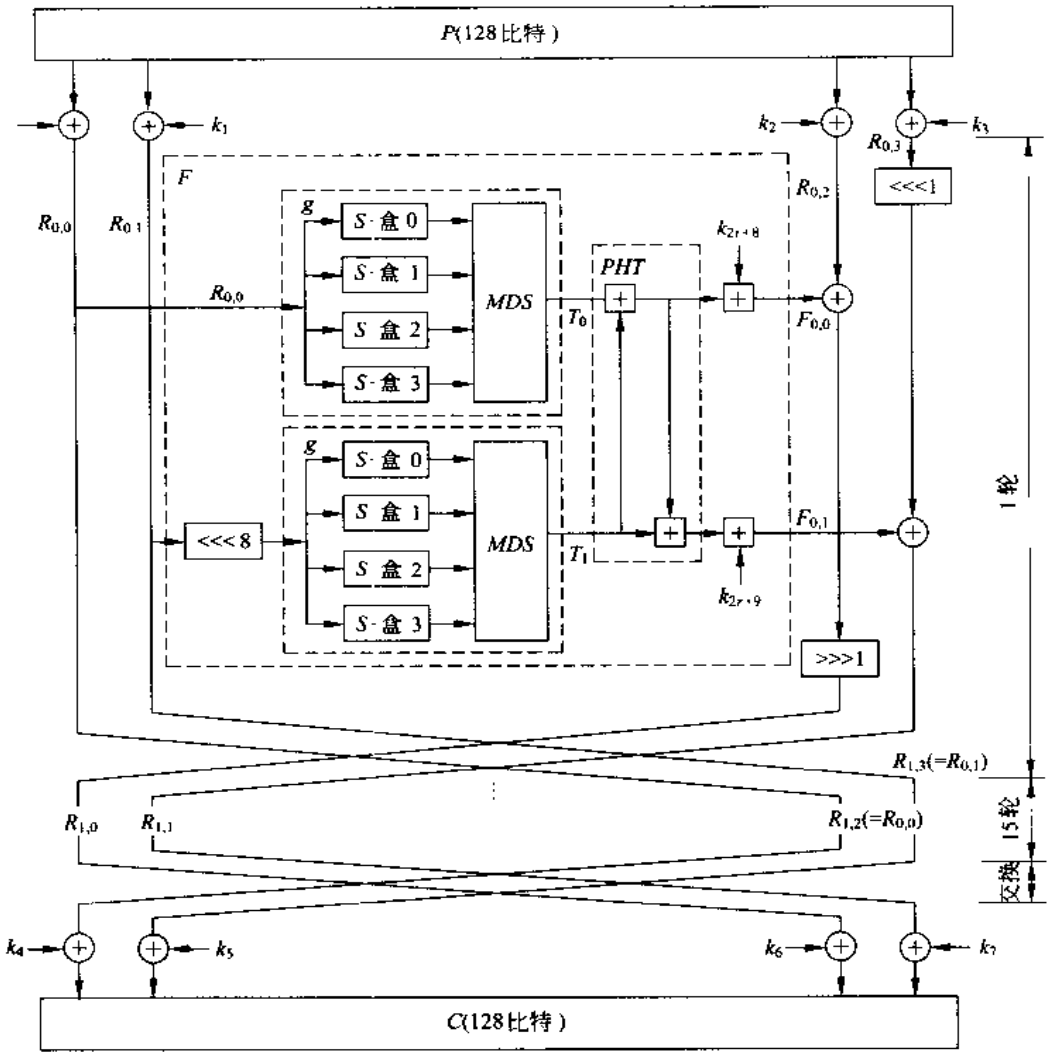


图 3.28

为了形式化叙述说明起见，设明文 126 比特，分为 16 字节：$p_0, p_1, \cdots, p_{15}$，又分为 4 字 P_0, P_1, P_2, P_3

$$P_i = \sum_{j=0}^{3} p(4i+j) \cdot 2^{8j}, \quad i = 0,1,2,3$$

令

$$R_{0,i} = P_i \oplus K_i, \quad i = 0,1,2,3$$

即 P_i 和 K_i 作异或运算

16 轮中每一轮，前面两字先作为函数 F 的输入，第 3 个字和 F 函数输出的第 1 个字作异或运算，再右旋转 1 比特。F 函数输出的第 2 个字和左旋转 1 比特后的第 4 个字作异或运算。最后左右两半互换，但第 16 轮不作左右互换，即

$$(F_{r,0}, F_{r,1}) = F(R_{r,0}, R_{r,1}, r)$$
$$R_{r+1,0} = ROR(R_{r,2} \oplus F_{r,0}, 1)$$
$$R_{r+1,1} = ROL(R_{r,3}, 1) \oplus F_{r,1}$$
$$R_{r+1,2} = R_{r,0}$$
$$R_{r+1,3} = R_{r,1}$$
$$r = 0,1,2,\cdots,15$$

ROR 和 ROL 分别是对第 1 个变元(32 比特字)作右或左旋转，旋转的位数，即第 2 个变元所示的数。输出密文也是 4 个字

$$C_i = R_{16,(i-2) \bmod 4} \oplus K_{i+4}, \quad i = 0,1,2,3$$

密文 C 也可以表示为 16 个字节 $c_0, c_1, \cdots, c_{15}$，$c_i = \left\lfloor \frac{C\lfloor i/4 \rfloor}{2^{8(i \bmod 4)}} \right\rfloor \bmod 2^8$，$i = 0,1,\cdots,15$。图 3.26 中的 PHT 是作 $(a+b) \bmod 2^{32}$。

3.7.2 函数 F

函数 F 是由密钥控制的 64 比特的置换，输入为 R_0，R_1 及轮数 r。

$$T_0 = g(R_0)$$
$$T_1 = g(ROL(R_1, 8))$$
$$F_0 = (T_0 + T_1 + K_{2r+8}) \bmod 2^{32}$$
$$F_1 = (T_0 + 2T_1 + K_{2r+9}) \bmod 2^{32}$$

(F_0, F_1) 是函数 F 的输出结果。

3.7.3 函数 g

g 是 Twofish 算法的核心，输入分成 4 字节，每字节通过各自由密钥控制的 S^* 盒，每个 S^* 盒的输入和输出都是 8 比特。4 个结果看作 $GF(2^8)$ 域上长度为 4 的向量乘以 4×4 的 MDS 矩阵。最后向量是 32 比特字便是 g 的结果。

$$x_i = \lfloor Z/2^{8i} \rfloor \bmod 2^8, \quad i = 0,1,2,3$$
$$y_i = S_i[x_i], \quad i = 0,1,2,3$$

其中 S_i 是与密钥有关的 S 盒。

$$\begin{bmatrix} Z_0 \\ Z_1 \\ Z_2 \\ Z_3 \end{bmatrix} = [MDS] \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

$$Z = \sum_{i=0}^{3} Z_i \cdot 2^{8i}$$

其中 MDS 如下

$$[MDS] = \begin{pmatrix} 01 & EF & 5B & 5B \\ 5B & EF & EF & 01 \\ EF & 5B & 01 & EF \\ EF & 01 & EF & 5B \end{pmatrix}$$

其中各元素都是用十六进制表示。Z 是函数 g 的结果。这里字节对应于有限域 $GF(2^8)$，或 $GF(2)[x]/v(x)$，其中本原多项式是 $v(x)=x^8+x^6+x^5+x^3+1$ 是 $GF(2)$ 上 8 次本原多项式。$GF(2^8)$ 上的元素

$$a = \sum_{i=0}^{7} a_i x^i, \quad a_i \in GF(2)$$

对应于字节的值

$$\sum_{i=0}^{7} a_i 2^i$$

$GF(2^8)$ 上元素的加法对应于字节的异或。

3.7.4 子密钥生成

密钥长度有 $N=128,192,256$ 比特，将产生 $k_0,k_1,\cdots,k_{39}$ 以及 g 函数所需的受密钥控制的 4 个 S^* 盒。对于比 256 比特短的密钥，可加上 0 直到满 128、192 或 256 比特。

令

$$k=N/64$$

密钥 K 有 $8k$ 个字节：

$$k_0, k_1, \cdots, k_{8k-1}$$

将它们转换为 $2k$ 个 32 比特的字

$$K_i = \sum_{j=0}^{3} k_{(4i+j)} 2^{8j}, \quad i = 0,1,\cdots,2k-1$$

$$K_e = (K_0, K_2, \cdots, K_{2k-2})$$

$$K_o = (K_1, K_3, \cdots, K_{2k-1})$$

构造在 $GF(2^8)$ 上的向量。

$$\begin{pmatrix} s_{i,0} \\ s_{i,1} \\ s_{i,2} \\ s_{i,3} \end{pmatrix} = [RS] \begin{pmatrix} k_{8i} \\ k_{8i+1} \\ k_{8i+2} \\ k_{8i+3} \\ k_{8i+4} \\ k_{8i+5} \\ k_{8i+6} \\ k_{8i+7} \end{pmatrix}$$

其中

$$[RS]=\begin{bmatrix}01 & A4 & 55 & 87 & 5A & 58 & DB & 9E\\ A4 & 56 & 82 & F3 & 1E & C6 & 68 & E5\\ 02 & A1 & FC & C1 & 47 & AE & 3D & 19\\ A4 & 55 & 87 & 5A & 58 & DB & 9E & 03\end{bmatrix}$$

令

$$S_i=\sum_{j=0}^{3}s_{i,j}2^{8j},\quad i=0,1,\cdots,k-1$$

$$S=(s_{k-1},s_{k-2},\cdots,s_1,s_0)$$

因此有 3 个向量：K_e，K_o 和 S，其中 S 是由密钥 k 产生的。

3.7.5 关于密钥的补充

Twofish 的密钥长度任意，可长达 256 位，若密钥长度不是 128、192、256 比特，可补以 0 使达到最靠近上述的数。例如密钥长 80 比特，可补以 0 使达到 128 比特。

3.7.6 函数 h

h 函数与 g 函数有关系 $g(x)=h(x,s)$，即 g 函数中的 S 盒由 $g(x)=h(x,s)$ 定义。

函数 h 的输入有二：一个 32 比特字 X 和每个元素长度为 32 比特字的 L。

$$L=(L_0,L_1,\cdots,L_{k-1})$$

L 的定义见本节最后。输出是一个字。函数有 k 步，每一步 4 个字节通过一固定的 S^* 盒，并与 L 中的一字节作异或运算。最后字节再一次通过一固定的 S^* 盒，4 个字节乘以 MDS 矩阵，如函数 g 一样。

形式化地说明如下。将字分裂为字节如下：

$$y_{i,j}=\lfloor L_i/2^{8j}\rfloor \bmod 2^8$$

$$x_j=\lfloor Z/2^{8j}\rfloor \bmod 2^8$$

$$i=0,1,\cdots,k-1$$

$$j=0,1,2,3$$

$$y_{k,j}\leftarrow x_j\quad j=0,1,2,3$$

请注意：密钥长 128 比特，则 $k=2$；若密钥长 192 比特，则 $k=3$；又若密钥长 256 比特，则 $k=4$。

现在分别讨论如下：

1. 若 $k=4$，已知 $y_{4,0}$，$y_{4,1}$，$y_{4,2}$，$y_{4,3}$，则计算 S_1：

$$S_1:\begin{aligned}y_{3,0}&=q_1[y_{4,0}]\oplus l_{3,0}\\ y_{3,1}&=q_0[y_{4,1}]\oplus l_{3,1}\\ y_{3,2}&=q_0[y_{4,2}]\oplus l_{3,2}\\ y_{3,3}&=q_1[y_{4,3}]\oplus l_{3,3}\end{aligned}$$

接着计算 S_2：

$$S_2:\begin{aligned} y_{2,0} &= q_1[y_{3,0}]\oplus l_{2,0}\\ y_{2,1} &= q_1[y_{3,1}]\oplus l_{2,1}\\ y_{2,2} &= q_0[y_{3,2}]\oplus l_{2,2}\\ y_{2,3} &= q_0[y_{3,3}]\oplus l_{2,3}\end{aligned}$$

再接着计算 S_3：

$$S_3:\begin{aligned} y_0 &= q_1[q_0[q_0[y_{2,0}]\oplus l_{1,0}]\oplus l_{0,0}]\\ y_1 &= q_0[q_0[q_1[y_{2,1}]\oplus l_{1,1}]\oplus l_{0,1}]\\ y_2 &= q_1[q_1[q_0[y_{2,2}]\oplus l_{1,2}]\oplus l_{0,2}]\\ y_3 &= q_0[q_1[q_1[y_{2,3}]\oplus l_{1,3}]\oplus l_{0,3}]\end{aligned}$$

这里 q_0、q_1 是由 8 比特到 8 比特的置换，将在后面介绍。

最后计算

$$\begin{bmatrix} Z_0\\ Z_1\\ Z_2\\ Z_3\end{bmatrix} = [MDS]\begin{bmatrix} y_0\\ y_1\\ y_2\\ y_3\end{bmatrix}$$

$$Z=\sum_{i=0}^{3} Z_i\cdot 2^{8i}$$

2. 若 $k=3$，则已知 $y_{3,0}$，$y_{3,1}$，$y_{3,2}$，$y_{3,3}$，直接越过 S_1，进入 S_2，其余步骤相同，从略。

3. 若 $k=2$，已知 $y_{2,0}$，$y_{2,1}$，$y_{2,2}$，$y_{2,3}$，可从 S_3 开始，现将详细流程图列于图 3.29 中。

Twofish 的 S 盒是依赖于密钥的，这是它的特点，即 S_i 由将 x_i 映射到 y_i 的函数 h 所确定，$i=0,1,2,3$。

3.7.7 扩展的密钥 K_j

$$\begin{aligned} \rho &= 2^{24}+2^{16}+2^8+2^0\\ A_i &= h(2i\rho, Me)\\ B_i &= ROL(h((2i+1)\rho, M_o), 8)\\ K_{2i} &\equiv (A_i+B_i) \bmod 2^{32}\\ K_{2i+1} &= ROL((A_i+2B_i) \bmod 2^{32}, 9)\end{aligned}$$

3.7.8 q_0 和 q_1

q_0 和 q_1 是由 4 比特到 4 比特的置换。

对于输入 x，输出 y 的定义如下：

$$\begin{aligned} a_0 &= \left\lfloor \frac{x}{16} \right\rfloor; & b_0 &\equiv x \bmod 16\\ a_1 &= a_0\oplus b_0; & b_1 &\equiv a_0\oplus ROL_4(b_0,1)\oplus 8a_0 \bmod 16\\ a_2 &= t_0[a_1]; & b_2 &= t_1[b_1]\\ a_3 &= a_2\oplus b_2; & b_3 &\equiv a_2\oplus ROL_4(b_2,1)\oplus 8a_2 \bmod 16\end{aligned}$$

$$a_4 = t_2[a_3]; \qquad b_4 = t_3[b_3]$$
$$y = 16b_4 + a_4$$

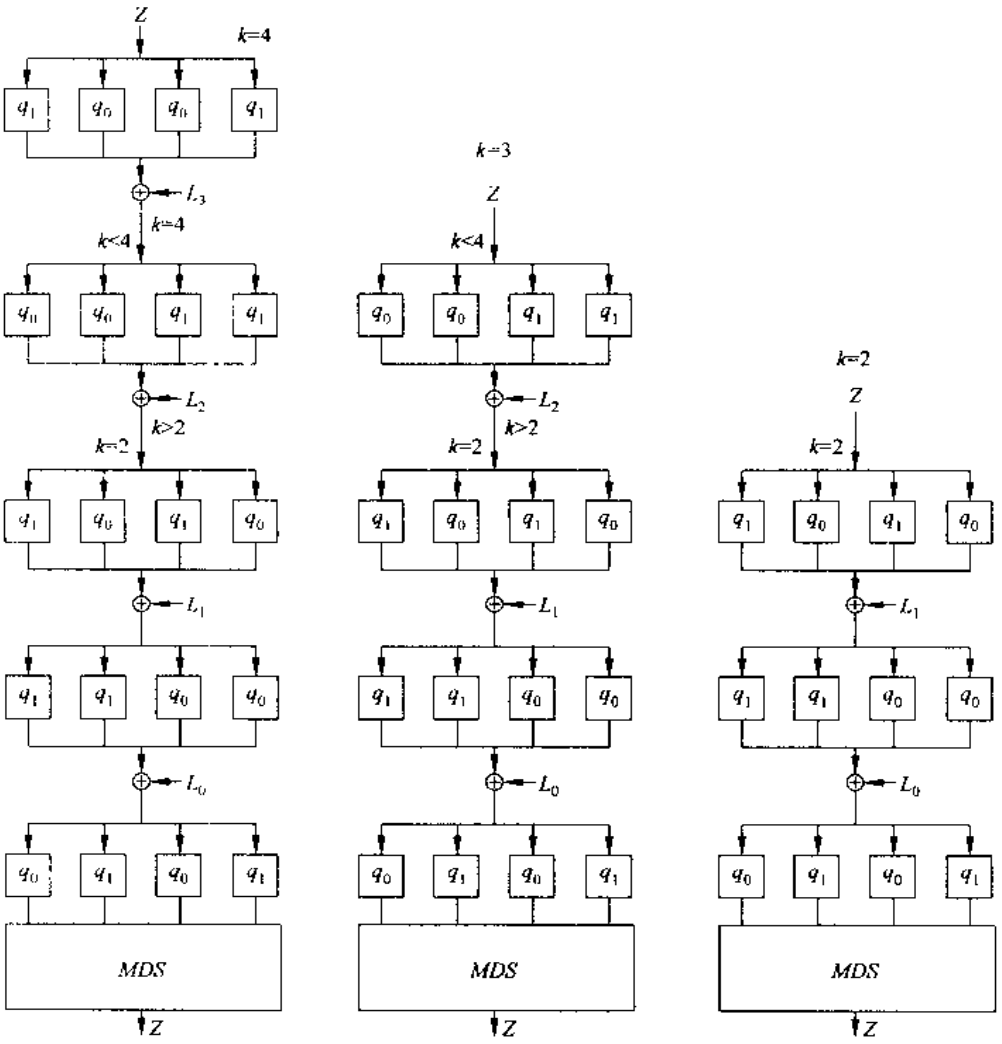


图 3.29

其中 ROR_4 和 ROR 类似，是旋转 4 比特的结果。

对于 q_0，4 比特的 S 盒为

$t_0 = [8\ 1\ 7\ D\ 6\ F\ 3\ 2\ 0\ B\ 5\ 9\ E\ C\ A\ 4]$

$t_1 = [E\ C\ B\ 8\ 1\ 2\ 3\ 5\ F\ 4\ A\ 6\ 7\ 0\ 9\ D]$

$t_2 = [B\ A\ 5\ E\ 6\ D\ 9\ 0\ C\ 8\ F\ 3\ 2\ 4\ 7\ 1]$

$t_3 = [D\ 7\ F\ 4\ 1\ 2\ 6\ E\ 9\ B\ 3\ 0\ 8\ 5\ C\ A]$

对于 q_1，4 比特的 S 盒为

$t_0 = [2\ 8\ B\ D\ F\ 7\ 6\ E\ 3\ 1\ 9\ 4\ 0\ A\ C\ 5]$

$t_1 = [1\ E\ 2\ B\ 4\ C\ 3\ 7\ 6\ D\ A\ 5\ F\ 9\ 0\ 8]$

t_2 = [4 C 7 5 1 6 9 A 0 E D 8 2 B 3 F]

t_3 = [B 9 5 1 C 3 D E 6 4 7 F 2 0 8 A]

128 比特的 F 函数的流程图如图 3.30 所示。

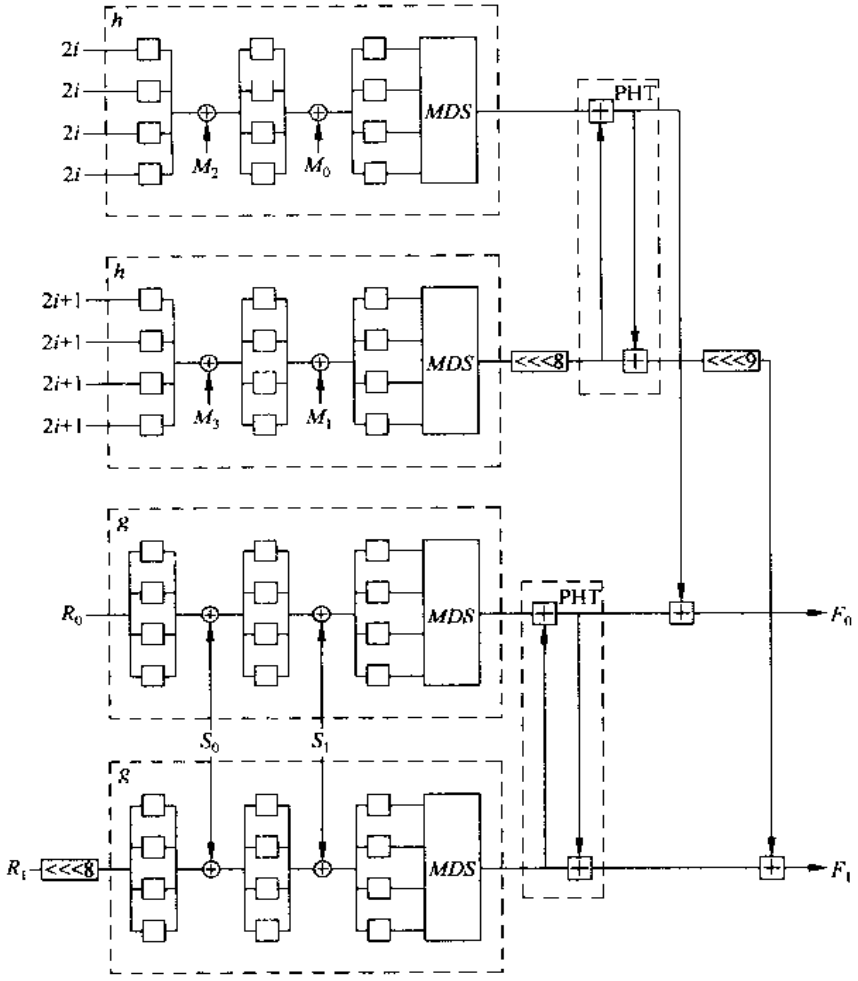


图 3.30　一轮 F 函数 128 比特的表示图

3.8 CAST-256 密码

CAST-256 算法是加拿大的 Carlisle M. Adams 等人提出的，是在原有的 CAST-126 加密算法基础上设计的。该算法是采用了推广的 Feistel 网络。最大的特点是：它能直接采用来自 Feistel 网络的轮函数。一般说来，增加轮函数将增加它的 S-盒的规模或扩大扩散层，这往往是比较困难，推广的 Feistel 网络无疑是给设计带来许多方便。

CAST-256 的轮函数有以下特点：

(1) S-盒的设计具有很高的非线性性、低的差分均匀性、良好的高阶严格雪崩特性及高阶的输出比特独立准则。

(2) 采用隐蔽密钥和循环移位密钥确保每圈中密钥熵高于数据熵，使差分分析变得更加困难。

(3) 来自不同群运算的混合，有效地减少高阶差分攻击的可能性。

3.8.1 若干记号

(1) f_1, f_2, f_3 是含有密钥的轮函数。

f_1：

$$I = ((k_{m_i} + D) \lll k_{r_i})$$

$$O = ((S_1[I_a] \oplus S_2[I_b]) - S_3[I_c]) + S_4[I_d]$$

f_2：

$$I = ((k_{m_i} \oplus D) \lll k_{r_i})$$

$$O = ((S_1[I_a] - S_2[I_b]) + S_3[I_c]) \oplus S_4[I_d]$$

f_3：

$$I = ((k_{m_i} - D) \lll k_{r_i})$$

$$O = ((S_1[I_a] + S_2[I_b]) \oplus S_3[I_c]) - S_4[I_d]$$

其中：k_{r_i} 为 5 比特，是第 i 轮旋转子密钥；

k_{m_i} 为 32 比特，第 i 轮的隐蔽子密钥；

I_a, I_b, I_c, I_d 为由输入 I 导出的子串，见后面内容；

S_i 为第 i 个 S-盒；

$+, -$ 是 mod 2^{32} 的加、减法运算；

$\lll$ 是左循环移位运算；

I 是输入数据，O 是结果输出。

(2) $m \leftarrow Q_i(m)$：其中 $m=(ABCD)$ 为 128 比特块，A, B, C, D 各 32 比特，表示下面的运算：

$$C \leftarrow C \oplus f_1(D, k_{r_0}^{(i)}, k_{m_0}^{(i)})$$

$$B \leftarrow B \oplus f_2(C, k_{r_1}^{(i)}, k_{m_1}^{(i)})$$

$$A \leftarrow A \oplus f_3(B, k_{r_2}^{(i)}, k_{m_2}^{(i)})$$

$$D \leftarrow D \oplus f_1(A, k_{r_3}^{(i)}, k_{m_3}^{(i)})$$

图 3.31 形象地表达了上述运算。

$M=(ABCD)$ 是 128 比特，A, B, C, D 各 32 比特。

(3) $M \leftarrow \bar{Q}_i(M)$ 表示下面的运算：

$$D \leftarrow D \oplus f_1(A, k_{r_0}^{(i)}, k_{m_1}^{(i)})$$

$$A \leftarrow A \oplus f_3(B, k_{r_2}^{(i)}, k_{m_2}^{(i)})$$

$$B \leftarrow B \oplus f_2(C, k_{r_1}^{(i)}, k_{m_1}^{(i)})$$

$$C \leftarrow C \oplus f_1(D, k_{r_0}^{(i)}, k_{m_0}^{(i)})$$

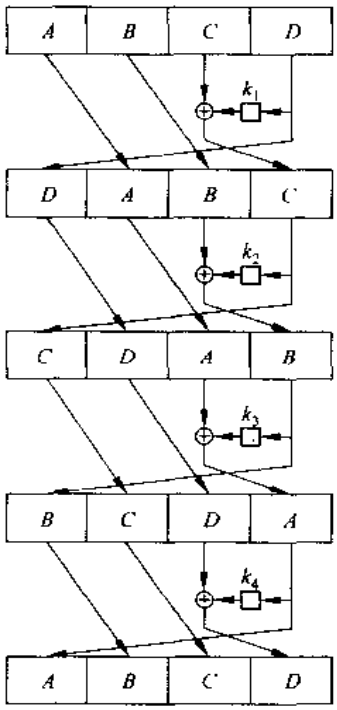


图 3.31

令

$$k_r^{(i)} = \{k_{r_0}^{(i)}, k_{r_1}^{(i)}, k_{r_2}^{(i)}, k_{r_3}^{(i)}\}$$

$k_{r_j}^{(i)}$：5 比特，　$j = 0,1,2,3$

$$k_m^{(i)} = \{k_{m_0}^{(i)}, k_{m_1}^{(i)}, k_{m_2}^{(i)}, k_{m_3}^{(i)}\}$$

$k_{m_j}^{(i)}$：32 比特，　$j = 0,1,2,3$

(4) $k \leftarrow w_i(k)$表示下面的运算：

$$G \leftarrow G \oplus f_1(H, t_{r_0}^{(i)}, t_{m_0}^{(i)})$$

$$F \leftarrow F \oplus f_2(G, t_{r_1}^{(i)}, t_{m_1}^{(i)})$$

$$E \leftarrow E \oplus f_3(F, t_{r_2}^{(i)}, t_{m_2}^{(i)})$$

$$D \leftarrow D \oplus f_1(E, t_{r_3}^{(i)}, t_{m_3}^{(i)})$$

$$C \leftarrow C \oplus f_2(D, t_{r_4}^{(i)}, t_{m_4}^{(i)})$$

$$B \leftarrow B \oplus f_3(C, t_{r_5}^{(i)}, t_{m_5}^{(i)})$$

$$A \leftarrow A \oplus f_1(B, t_{r_6}^{(i)}, t_{m_6}^{(i)})$$

$$H \leftarrow H \oplus f_2(A, t_{r_7}^{(i)}, t_{m_7}^{(i)})$$

其中 $k = (ABCDEFGH)$：256 比特，$A, B, \cdots, H$ 各 32 比特。

(5) $k_r^{(i)} \leftarrow k$ 表示下面的运算：

$$k_{r_0}^{(i)} = 5\ LSB(A),\ k_{r_1}^{(i)} = 5\ LSB(C)$$

$$k_{r_2}^{(i)} = 5\ LSB(E),\ k_{r_3}^{(i)} = 5\ LSB(G)$$

LSB 是 Least Significant Bits 的缩写。$5\ LSB(A)$表示 A 的最小 5 位有效数字，即低位 5 位数字，依此类推。

$k_{m_j}^{(i)} \leftarrow$表示：

$$k_{m_0}^{(i)} = H, k_{m_1}^{(i)} = F, k_{m_2}^{(i)} = D, k_{m_3}^{(i)} = B$$

3.8.2　加密、解密算法

1. 加密运算

m 为 128 比特的明文

i 从 0 到 5 做

$$m \leftarrow Q_i(m)$$

i 从 6 到 11 做

$$C \leftarrow \bar{Q}_i(m)$$

C 为 128 比特的密文。

2. 解密过程

解密运算和加密过程完全一样，只是密钥的顺序相反：从加密密钥到解密密钥的过程是：

加密密钥为 256 比特

i 从 0 到 11 做

始

$$k_{r_{mu}}^{(i)} \leftarrow k_r^{(11-i)}$$

$$k_{m_{mu}}^{(i)} \leftarrow k_m^{(11-i)}$$

终。

3. 密钥安排

S1. $C_m \leftarrow 2^{40}\sqrt{2} = 5A827999$

$M_m \leftarrow 2^{50}\sqrt{2} = 6ED9EBA1$

$C_r \leftarrow 19$

$M_r \leftarrow 17$

S2. i 从 0 到 23 做

j 从 0 到 7 做

始

$t_{m_j}^{(i)} \leftarrow C_m$

$C_m \leftarrow (C_m + M_m) \bmod 2^{32}$

$t_{r_j}^{(i)} \leftarrow C_r$

$C_r \leftarrow (C_r + M_r) \bmod 32$

终。

S3. k = ABCDEFGH = 主密钥 k 的 256 比特。

i 从 0 到 11 做

始

$k \leftarrow w_{2i}(k)$

$k \leftarrow w_{2i+1}(k)$

$k_r^{(i)} \leftarrow k$

$k_m^{(i)} \leftarrow k$

终。

注意：若 $|k| = 128$，则 E = F = G = H = 0；

若 $|k| = 160$，则 F = G = H = 0；

若 $|k| = 192$，则 G = H = 0；

若 $|k| = 224$，则 H = 0。

3.8.3 S 盒

S_1：

30fb40d4	9fa0ff0b	6beccd2f	3f258c7a	1e213f2f	9c004dd3	6003e540	cf9fc949
bfd4af27	88bbbdb5	e2034090	98d09675	6e63a0e0	15c361d2	c2e7661d	22d4ff8e
28683b6f	c07fd059	ff2379c8	775f50e2	43c340d3	df2f8656	887ca41a	a2d2bd2d
a1c9e0d6	346c4819	61b76d87	22540f2f	2abe32e1	aa54166b	22568e3a	a2d341d0
66db40c8	a784392f	004dff2f	2db9d2de	97943fac	4a97c1d8	527644b7	b5f437a7
b82cbaef	d751d159	6ff7f0ed	5a097a1f	827b68d0	90ecf52e	22b0c054	bc8e5935
4b6d2f7f	50bb64a2	d2664910	bee5812d	b7332290	e93b159f	b48ee411	4bff345d

fd45c240	ad31973f	c4f6d02e	55fc8165	d5b1caad	a1ac2dae	a2d4b76d	c19b0c50
882240f2	0c6e4f38	a4e4bfd7	4f5ba272	564c1d2f	c59c5319	b949e354	b04669fe
b1b6ab8a	c71358dd	6385c545	110f935d	57538ad5	6a390493	c63d37e0	2a54f6b3
3a787d5f	6276a0b5	19a6fcdf	7a42206a	29f9d4d5	f61b1891	bb72275e	aa508167
38901091	c6b505eb	84c7cb8c	2ad75a0f	874a1427	a2d1936b	2ad286af	aa56d291
d7894360	425c750d	93b39e26	187184c9	6c00b32d	73e2bb14	a0bebc3c	54623779
64459eab	3f328b82	7718cf82	59a2cea6	04ee002e	89fe78e6	3fab0950	325ff6c2
81383f05	6963c5c8	76cb5ad6	d49974c9	ca180dcf	380782d5	c7fa5cf6	8ac31511
35e79e13	47da91d0	f40f9086	a7e2419e	31366241	051ef495	aa573b04	1a805d8d
548300d0	00322a3c	bf64cddf	ba57a68e	75c6372b	50afd341	a7c13275	915a0bf5
6b54bfab	2b0b1426	ab4cc9d7	449ccd82	f7fbf265	ab85c5f3	1b55db94	aad4e324
cfa4bd3f	2deaa3e2	9e204d02	c8bd25ac	eadf55b3	d5bd9e98	e31231b2	2ad5ad6c
954329de	adbe4528	d8710f69	aa51c90f	aa786bf6	22513f1e	aa51a79b	2ad344cc
7b5a41f0	d37cfbad	1b069505	41ece491	b4c332e6	032268d4	c9600acc	ce387e6d
bf6bb16c	6a70fb78	0d03d9c9	d4df39de	e01063da	4736f464	5ad328d8	b347cc96
75bb0fc3	98511bfb	4ffbcc35	b58bcf6a	e11f0abc	bfc5fe4a	a70aec10	ac39570a
3f04442f	6188b153	e0397a2e	5727cb79	9ceb418f	1cacd68d	2ad37c96	0175cb9d
c69dff09	c75b65f0	d9db40d8	ec0e7779	4744ead4	b11c3274	dd24cb9e	7e1c54bd
f01144f9	d2240eb1	9675b3fd	a3ac3755	d47c27af	51c85f4d	56907596	a5bb15e6
580304f0	ca042cf1	011a37ea	8dbfaadb	35ba3e4a	3526ffa0	c37b4d09	bc306ed9
98a52666	5648f725	ff5c569d	0ced63d0	7c63b2cf	700b45e1	d5ea50f1	85a92872
af1fbda7	d4234870	a7870bf3	2d3b4d79	42e04198	0cd0ede7	26470db8	f881814c
474d6ad7	7c0c5e5c	d1231959	381b7298	f5d2f4db	ab838653	6e2f1e23	83719c9e
bd91e046	9a56456e	dc39200c	20c8c571	962bda1c	e1e696ff	b141ab08	7cca89b9
1a69e783	02cc4843	a2f7c579	429ef47d	427b169c	5ac9f049	dd8f0f00	5c8165bf

S_2:

1f201094	ef0ba75b	69e3cf7e	393f4380	fe61cf7a	eec5207a	55889c94	72fc0651
ada7ef79	4e1d7235	d55a63ce	de0436ba	99c430ef	5f0c0794	18dcdb7d	a1d6eff3
a0b52f7b	59e83605	ee15b094	e9ffd909	dc440086	ef944459	ba83ccb3	e0c3cdfb
d1da4181	3b092ab1	f997f1c1	a5e6cf7b	01420ddb	e4e7ef5b	25a1ff41	e180f806
1fc41080	179bee7a	d37ac6a9	fe5830a4	98de8b7f	77e83f4e	79929269	24fa9f7b
e113c85b	acc40083	d7503525	f7ea615f	62143154	0d554b63	5d681121	c866c359
3d63cf73	cee234c0	d4d87e87	5c672b21	071f6181	39f7627f	361e3084	e4eb573b
602f64a4	d63acd9c	1bbc4635	9e81032d	2701f50c	99847ab4	a0e3df79	ba6cf38c
10843094	2537a95e	f46f6ffe	a1ff3b1f	208cfb6a	8f458c74	d9e0a277	4ec73a34
fc884f69	3e4de8df	ef0e0088	3559648d	8a45388c	1d804366	721d9bfd	a58684bb
e8256333	844e8212	128d8098	fed33fb4	ce280ae1	27e19ba5	d5a6c252	e49754bd
c5d655dd	eb667064	77840b4d	a1b6a801	84db26a9	e0b56714	21f043b7	e5d05860
54f03084	066ff472	a31aa153	dadc4755	b5625dbf	68561be6	83ca6b94	2d6ed23b
eccf01db	a6d3d0ba	b6803d5c	af77a709	33b4a34c	397bc8d6	5ee22b95	5f0e5304
81ed6f61	20e74364	b45e1378	de18639b	881ca122	b96726d1	8049a7e8	22b7da7b
5e552d25	5272d237	79d2951c	c60d894c	488cb402	1ba4fe5b	a4b09f6b	1ca815cf
a20c3005	8871df63	b9de2fcb	0cc6c9e9	0beeff53	e3214517	b4542835	9f63293c

ee41e729	6e1d2d7c	50045286	1e6685f3	f33401c6	30a22c95	31a70850	60930f13
73f98417	a1269859	ec645c44	52c877a9	cdff33a6	a02b1741	7cbad9a2	2180036f
50d99c08	cb3f4861	c26bd765	64a3f6ab	80342676	25a75e7b	c4e6d1fc	20c710e6
cdf0b680	17844d3b	31eef84d	7e0824e4	2ccb49eb	846a3bae	8ff77888	ee5d60f6
7af75673	2fdd5cdb	a11631c1	30f66f43	b3faec54	157fd7fa	ef8579cc	d152de58
db2ffd5e	8f32ce19	306af97a	02f03ef8	99319ad5	c242fa0f	a7e3ebb0	c68e4906
b8da230c	80823028	dcdcf3c8	d35fb171	088a1bc8	bec0c560	61a3c9c8	bca8f54d
c72feffa	22833e99	82c570b4	d8d94e89	8b1c34bc	301e16e6	273be979	b0ffeaa6
61d9b8c6	00b24869	b7ffce3f	08dc283b	43daf65a	f7e19798	7619b72f	8f1c9ba4
dc8637a0	16a7d3b1	9fc393b7	a7136eeb	c6bcc63e	1a513742	ef6828bc	520365d6
2d6a77ab	3527ed4b	821fd216	095c6e2e	db92f2fb	5eea29cb	145892f5	91584f7f
5483697b	2667a8cc	85196048	8c4bacea	822860d4	0d23e0f9	6c387e8a	0ae6d249
b284600c	d835731d	dcb1c647	ac4c56ea	3ebd81b3	230eabb0	6438bc87	f0b5b1fa
8f5ea2b3	fc184642	0a036b7a	4fb089bd	649da589	a345415e	5c038323	3e5d3bb9
43d79572	7e6dd07c	06dfdf1e	6c6cc4ef	7160a539	73bfbe70	83877605	4523ecf1

S_5:

8defc240	25fa5d9f	eb903dbf	e810c907	47607fff	369fe44b	8c1fc644	aececa90
beb1f9bf	eefbcaea	e8cf1950	51df07ae	920e8806	f0ad0548	e13c8d83	927010d5
11107d9f	07647db9	b2e3e4d4	3d4f285e	b9afa820	fade82e0	a067268b	8272792e
553fb2c0	489ae22b	d4ef9794	125e3fbc	21fffcee	825b1bfd	9255c5ed	1257a240
4e1a8302	bae07fff	528246e7	8e57140e	3373f7bf	8c9f8188	a6fc4ee8	c982b5a5
a8c01db7	579fc264	67094f31	f2bd3f5f	40fff7c1	1fb78dfc	8e6bd2c1	437be59b
99b03dbf	b5dbc64b	638dc0e6	55819d99	a197c81c	4a012d6e	c5884a28	ccc36f71
b843c213	6c0743f1	8309893c	0feddd5f	2f7fe850	d7c07f7e	02507fbf	5afb9a04
a747d2d0	1651192e	af70bf3e	58c31380	5f98302e	727cc3c4	0a0fb402	0f7fef82
8c96fdad	5d2c2aae	8ee99a49	50da88b8	8427f4a0	1eac5790	796fb449	8252dc15
efbd7d9b	a672597d	ada840d8	45f54504	fa5d7403	e83ec305	4f91751a	925669c2
23efe941	a903f12e	60270df2	0276e4b6	94fd6574	927985b2	8276dbcb	02778176
f8af918d	4e48f79e	8f616ddf	e29d840e	842f7d83	340ce5c8	96bbb682	93b4b148
ef303cab	984faf28	779faf9b	92dc560d	224d1e20	8437aa88	7d29dc96	2756d3dc
8b907cee	b51fd240	e7c07ce3	e566b4a1	c3e9615e	3cf8209d	9094d1e3	cd9ca341
5c76460e	00ea983b	d4d67881	fd47572c	f76cedd9	bda8229c	127dadaa	438a074e
1f97c090	081bdb8a	93a07ebe	b938ca15	97b03cff	3dc2c0f8	8d1ab2ec	64380e51
68cc7bfb	d90f2788	12490181	5de5ffd4	dd7ef86a	76a2e214	b9a40368	925d958f
4b39fffa	ba39aee9	a4ffd30b	faf7933b	6d498623	193cbcfa	27627545	825cf47a
61bd8ba0	d11e42d1	cead04f4	127ea392	10428db7	8272a972	9270c4a8	127de50b
285ba1c8	3c624ff4	35c0eaa5	e805d231	428929fb	b4fc5f82	4fb66a53	0e7dc15b
1f081fab	108618ae	fcfd086d	f9ff2889	694bcc11	236a5cae	12deca4d	2c3f8cc5
d2d02dfe	f8ef5896	e4ef52da	95155b67	494a488c	b9b6a80c	5c8f82bc	89d36b45
3a609437	ec00c9a9	44715253	0a874b49	d773bc40	7c34671c	02717ef6	4feb5536
a2d02fff	d2bf60c4	d43f03c0	50b4ef6d	07478cd1	006e1888	a2e53f55	b9e6d4bc
a2048016	97573833	d7207d67	de0f8f3d	72f87b33	abcc4f33	7688c55d	7b00a6b0
947b0001	570075d2	f9bb88f8	8942019e	4264a5ff	856302e0	72dbd92b	eee971b69

6ea22fdc 5f08ae2b af7a616d e5c98767 cf1febd2 61efc8c2 f1ac2571 cc8239c2
67214cb8 b1e583d1 b7dc3e62 7f10bdce f90a5c38 0ff0443d 606e6dc6 60543a49
5727c148 2be98a1d 8ab41738 20e1be24 af96da0f 68458425 99833be5 600d457d
282f9350 8334b362 d91d1120 2b6d8da0 642b1e31 9c305a00 52bce688 1b03588a
f7baefd5 4142ed9c a4315c11 83323ec5 dfef4636 a133c501 e9d3531c ee353783

S_4:

9db30420 1fb6e9de a7be7bef d273a298 4a4f7bdb 64ad8c57 85510443 fa020ed1
7e287a55 e60fb663 095f35a1 79ebf120 fd059d43 6497b7b1 f3641f63 241e4adf
28147f5f 4fa2b8cd c9430040 0cc32220 fdd30b30 c0a5374f 1d2d00d9 24147b15
ee4d111a 0fca5167 71ff904c 2d195ffe 1a05645f 0c13fefe 086cf3d6 05170121
80530100 e83e5efe ac9af4f8 7fe72701 d2b8ee5f 06df4261 bb9e9b8a 7293ea25
ce84ffdf f5718801 3dd64b04 a26f263b 7ed48400 547eebe6 446d4ca0 6cf356f5
2649abdf aca0c7f5 36338cc1 503f7e93 d3772061 11b638e1 72500e03 f80eb2bb
ade0502e ec8d77de 57971e81 e14f6746 c9335400 6920318f 081dbb99 ffc304a5
4d351805 7f3d5ce3 a6c866c6 5d5bcca9 daec6fea 9f926f91 9f46222 3991467d
a5bf6d8e 1143c44f 43958302 d0214eeb 022083b8 6fb6180c 18f8931e 281658e6
26486e3e 8bd78a70 7477e4c1 b506e07c f32d0a25 79098b02 e4eabb81 28123b23
69dead38 1574ca16 df871b62 211c40b7 a51a9ef9 0014377b 0418eac8 09114003
bd59e4d2 e3d156d5 4fe876d5 2f91a340 557be8de 00eae4a7 0ce5c2ec 4db4bba6
e756bdff dd3369ac ec17b035 06572327 99afc8b0 56c8c391 6b65811c 5e146119
6e85cb75 be07c002 c2325577 893ff4ec 5bbfc92d d0ec3b25 b7801ab7 8d6d3b24
20c763ef c366a5fc 9c382880 0ace3205 aac9548a eca1d7c7 041afa32 1d16625a
6701902c 9b757a54 31d477f7 9126b031 36cc6fdb c70b8b46 d9e66a48 56e55a79
026a4ceb 52437eff 2f8f76b4 0df980a5 8674cde3 edda04eb 17a9be04 2c18f4df
b7747f9d ab2af7b4 efc34d20 2e096b7c 1741a254 e5b6a035 213d42f6 2c1c7c26
61c2f50f 6552daf9 d2c231f8 25130f69 d8167fa2 0418f2c8 001a96a6 0d1526ab
63315c21 5e0a72ec 49bafefd 187908d9 8d0dbd86 311170a7 3e9b640c cc3e10d7
d5cad3b6 0caec388 f73001e1 6c729aff 71eae2a1 1f9af36e cfcbd12f c1de8417
ac07be6b cb44a1d8 8b9b0f56 013988c3 b1c52fca b4be31ce d8782806 12a3a4e2
6f7de532 58fd7eb6 d01ee900 24adffc2 f4990fc5 9711aace 001d4b95 82e5e7d2
109873f6 00613096 c32d9521 ada121ff 29908415 7fbb977f af9eb3db 29c9ed2a
5ce2a465 a730f32c d0aa3fe8 8a5cc091 d49e2ce7 0ce454a9 d60acd86 015f1919
77079103 dea03af6 78a8565e dee356df 21f05cbe 8b75e387 b3c50651 b8a5c3ef
d8eeb6d2 e523be77 c2154529 2f69efdf afe67afb f470c4b2 f3e0eb5b d6cc9876
39e4460c 1fda8538 1987832f ca007367 a99144f8 296b299e 492fc295 9266beab
b5676e69 9bd3ddda df7e052f db25701c 1b5e51ee f65324e6 6afce36c 0316cc04
8644213e b7dc59d0 7965291f ccd6fd43 41823979 932bcdf6 b657c34d 4edfd282
7ae5290c 3cb9536b 851e20fe 9833557e 13ecf0b0 d3ffb372 3f85c5c1 0aef7ed2

3.9 SAFER$^+$ 密 码

SAFER$^+$加密算法是在SAFER系列算法基础上提出来的。它的安全性是有保证的。SAFER$^+$算法中仅使用字节运算，比较简单，需要的内存比较少，是一个比较好的算法。

3.9.1 算法说明

SAFER$^+$的明文块为16字节，如图3.32(a)所示。与DES类似，明文的第一字节在最左端。第16字节在最右端。经r轮迭代给出密文。r和密钥k的长度关系如下：

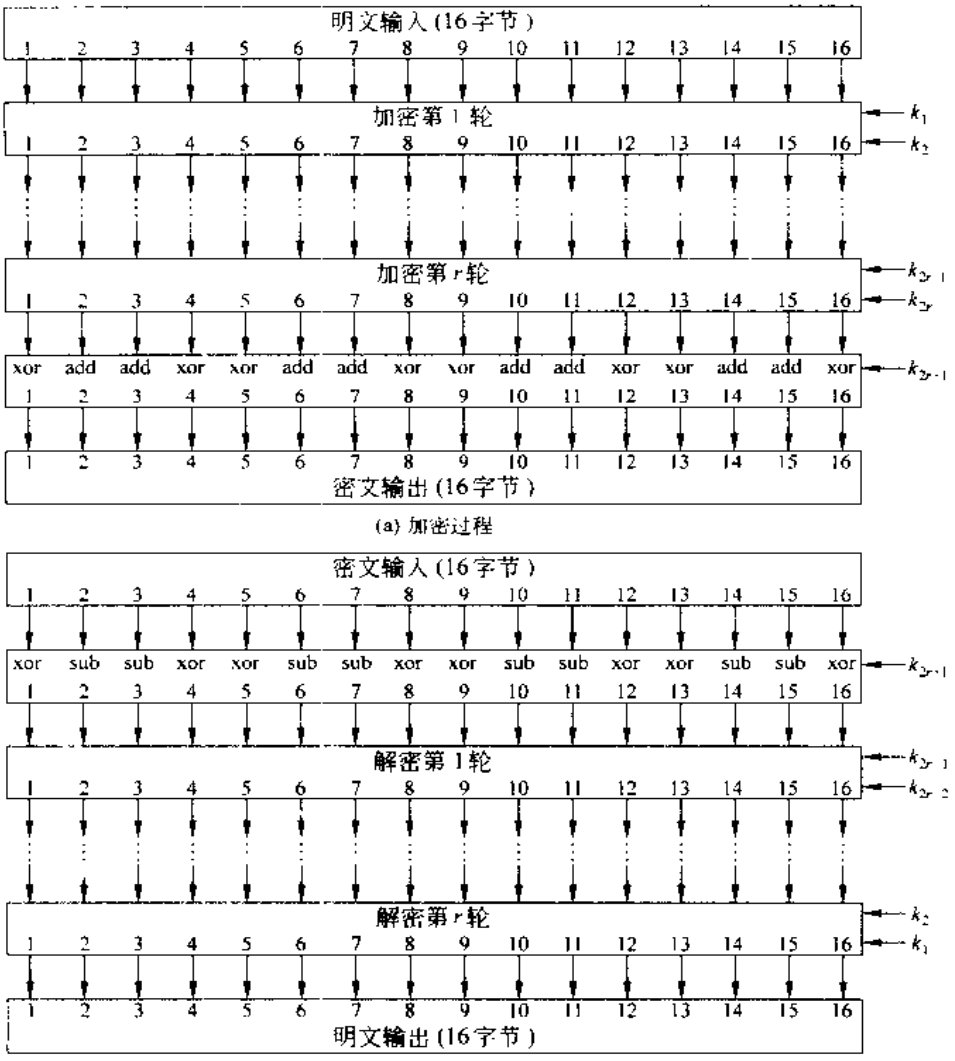


图 3.32

若密钥 k 长为 128 比特，则 $r=8$；

若密钥 k 长为 192 比特，则 $r=12$；

若密钥 k 长为 256 比特，则 $r=16$。

每轮用 k 的子密钥为 16 字节，各轮用的子密钥：

$$k_1, k_2, \cdots, k_{2r}$$

由 k 确定，k 是用户选用的密钥，子密钥的产生在后面叙述。最后一轮的子密钥 k_{2r+1} 作用到第 r 轮产生的块上，在第 1,4,5,8,9,12,13 和 16 字节作按位的异或运算；而当第 2,3,6,7,10,11,14 和 15 字节，每字节作 mod 256 的加法运算。k_{2r+1} 与之作运算的结果为 SAFER$^+$ 加密的结果 —— 密文 c，密文 c 也是 16 字节长。

关于解密：如图 3.32(b)所示，开始的变换为使加密算法最后一轮的输出的变换得以还原。在加密时第 1,4,5,8,9,12,13 和第 16 字节作与子密钥 k_{2r+1} 的逐位异或运算，在解密时这些字节也对 k_{2r-1} 作逐位异或运算；类似对第 2,3,6,7,10,11,14 和 15 作 mod 256 的减法。为此"减"的结果使恢复到 r 轮加密前的状态。其他 r 轮解密都是在对 r 轮加密的依次恢复。解密用的子密钥正好是加密子密钥的逆序。

3.9.2 SAFER$^+$ 的各轮加密算法

SAFER$^+$ 的各轮加密算法如图 3.33 所示。在 $1\leqslant i\leqslant r$ 时，第 i 轮作 k_{2i-1} 子密钥与该轮的 16 字节作所谓的"加"法运算。即其中 1,4,5,8,9,12,13 作按位 mod 2 加法。其余各字节作 mod 256 的加法，将 16 字节结果，对第 1,4,5,8,9,12,13,16 字节的值 x，作非线性变换

$$45^x \bmod 257$$

对第 $j=2,3,6,7,10,11,14$ 和 15 字节的值 x，作转换

$$\log_{45}(x)$$

按以惯例 $x=128$ 时

$$45^{128}\equiv 256 \bmod 257$$

$45^{128} \bmod 256$ 表以 0，$x=0$ 时

$$\log_{45}(0)=128$$

i 轮子密钥 k_{2i}"加"到非线性层输出是这样进行的，第 2,3,6,7,10,11,14 和 15 字节做按位 mod 2 加；而第 1,4,5,8,9,12,13 和 16 字节做 mod 256 的加。所得的 16 字节结果

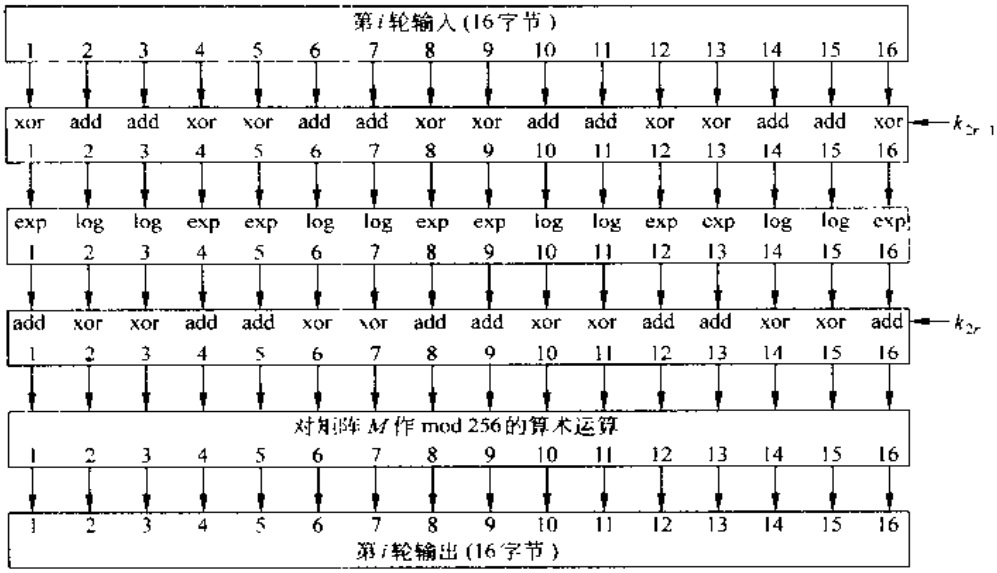
$$X=(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15}, x_{16})$$

作

$$y=xM$$

$$Y=(y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9, y_{10}, y_{11}, y_{12}, y_{13}, y_{14}, y_{15}, y_{16})$$

便是输出结果，其中 M 是 16×16 的矩阵。加法是在 mod 256 下进行的。



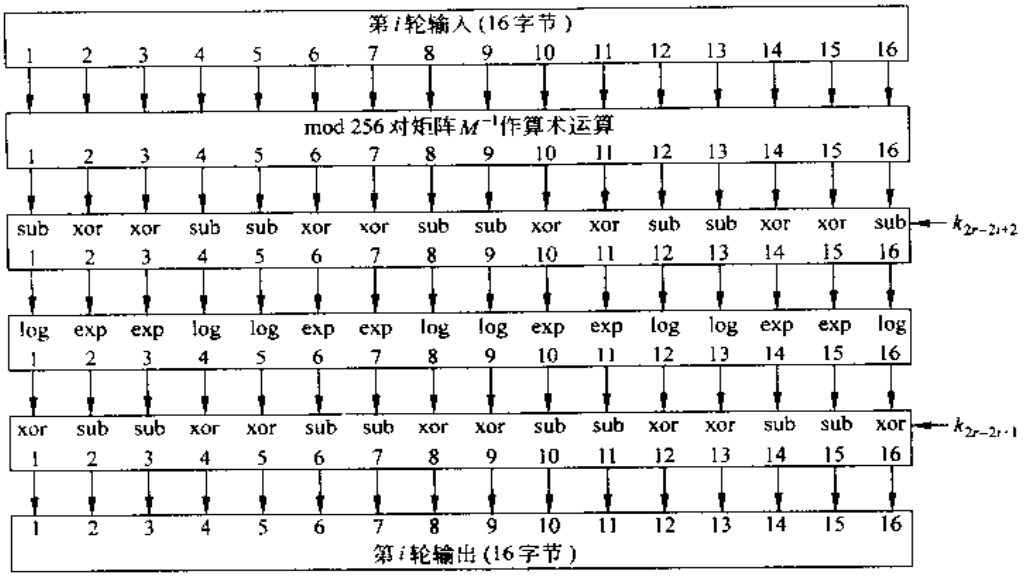
xor：字节每位作异或运算
add：mod 256作和运算
exp：mod 257求 45^x，其中 x 为输入字节
$45^{128} = 256 \mod 257$
log：表示 $\log_{45}(x)$，x 是输入字节，$\log_{45}(0)=128$

图 3.33　第 i 轮加密运算

$$\boldsymbol{M} = \begin{pmatrix}
2 & 2 & 1 & 1 & 16 & 8 & 2 & 1 & 4 & 2 & 4 & 2 & 1 & 1 & 4 & 4 \\
1 & 1 & 1 & 1 & 8 & 4 & 2 & 1 & 2 & 1 & 4 & 2 & 1 & 1 & 2 & 2 \\
1 & 1 & 4 & 4 & 2 & 1 & 4 & 2 & 4 & 2 & 16 & 8 & 2 & 2 & 1 & 1 \\
1 & 1 & 2 & 2 & 2 & 1 & 2 & 1 & 4 & 2 & 8 & 4 & 1 & 1 & 1 & 1 \\
4 & 4 & 2 & 1 & 4 & 2 & 4 & 2 & 16 & 8 & 1 & 1 & 1 & 1 & 2 & 2 \\
2 & 2 & 2 & 1 & 2 & 1 & 4 & 2 & 8 & 4 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 4 & 2 & 4 & 2 & 16 & 8 & 2 & 1 & 2 & 2 & 4 & 4 & 1 & 1 \\
1 & 1 & 2 & 1 & 4 & 2 & 8 & 4 & 2 & 1 & 1 & 1 & 2 & 2 & 1 & 1 \\
2 & 1 & 16 & 8 & 1 & 1 & 2 & 2 & 1 & 1 & 4 & 4 & 4 & 2 & 4 & 2 \\
2 & 1 & 8 & 4 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 4 & 2 & 2 & 1 \\
4 & 2 & 4 & 2 & 4 & 4 & 1 & 1 & 2 & 2 & 1 & 1 & 16 & 8 & 2 & 1 \\
2 & 1 & 4 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 8 & 4 & 2 & 1 \\
4 & 2 & 2 & 2 & 1 & 1 & 4 & 4 & 1 & 1 & 4 & 2 & 2 & 1 & 16 & 8 \\
4 & 2 & 1 & 1 & 1 & 1 & 2 & 2 & 1 & 1 & 2 & 1 & 2 & 1 & 8 & 4 \\
16 & 8 & 1 & 1 & 2 & 2 & 1 & 1 & 4 & 4 & 2 & 1 & 4 & 2 & 4 & 2 \\
8 & 4 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 1 & 2 & 1 & 4 & 2
\end{pmatrix}$$

3.9.3 解密的各轮算法

SAFER⁺解密的流程图如图 3.34 所示，其算法只是加密算法的顺序的逆。



第i轮解密运算

图 3.34

第一个运算是对输入 16 字节的 y：

$$Y=(y_1,y_2,y_3,y_4,y_5,y_6,y_7,y_8,y_9,y_{10},y_{11},y_{12},y_{13},y_{14},y_{15},y_{16})$$

作

$$x=y\boldsymbol{M}^{-1}$$

运算得 16 字节的 X：

$$X=(x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9,x_{10},x_{11},x_{12},x_{13},x_{14},x_{15},x_{16})$$

$\boldsymbol{M}^{-1}$是 mod 256 $\boldsymbol{M}$ 的逆(见后面内容)。

至于子密钥 $k_{2r\ 2r-2}$"减"X，对于第 1,4,5,8,9,12,13 和 16 字节是 mod 256 作减法；对于第 2,3,6,7,10,11,14 和 15 字节则作按位的异或运算。即按位 mod 2 作加法运算。其结果再对第 1,4,5,8,9,12,13 和 16 字节的值 x 作 $\log_{45}(x)$的非线性变换。对于第 2,3,6,7,10,11,14 和 15 字节的值 x 作

$$45^x \bmod 257$$

的非线性变换。

上述结果接着对子密钥 $k_{2r\ 2i+1}$进行如下的减法：对第 1,4,5,8,9,12,13,16 字节作按位 mod 2 加；对第 2,3,6,7,10,11,14,15 字节作 mod 257 的减法运算。

其中 $\boldsymbol{M}^{-1}$是下列 16×16 的方阵

$$\begin{bmatrix}
2 & -2 & 1 & -2 & 1 & -1 & 4 & -8 & 2 & -4 & 1 & -1 & 1 & -2 & 1 & -1 \\
-4 & 4 & -2 & 4 & -2 & 2 & -8 & 16 & -2 & 4 & -1 & 1 & -1 & 2 & -1 & 1 \\
1 & -2 & 1 & -1 & 2 & -4 & 1 & -1 & 1 & -1 & 1 & -2 & 2 & 2 & 4 & -8 \\
-2 & 4 & -2 & 2 & -2 & 4 & -1 & 1 & -1 & 1 & -1 & 2 & -4 & 4 & -8 & 16 \\
1 & -1 & 2 & -4 & 1 & -1 & 1 & -2 & 1 & -2 & 1 & -1 & 4 & -8 & 2 & -2 \\
-1 & 1 & -2 & 4 & -1 & 1 & -1 & 2 & -2 & 4 & -2 & 2 & -8 & 16 & -4 & 4 \\
2 & -4 & 1 & -1 & 1 & -2 & 1 & -1 & 2 & -2 & 4 & -8 & 1 & -1 & 1 & 2 \\
-2 & 4 & -1 & 1 & -1 & 2 & -1 & 1 & -4 & 4 & -8 & 16 & -2 & 2 & -2 & 4 \\
1 & 1 & 1 & -2 & 1 & -1 & 2 & -4 & 4 & -8 & 2 & -2 & 1 & -2 & 1 & -1 \\
-1 & 1 & -1 & 2 & -1 & 1 & -2 & 4 & -8 & 16 & -4 & 4 & -2 & 4 & -2 & 2 \\
1 & -2 & 1 & -1 & 4 & -8 & 2 & -2 & 1 & -1 & 1 & -2 & 1 & -1 & 2 & -4 \\
-1 & 2 & -1 & 1 & -8 & 16 & -4 & 4 & -2 & 2 & -2 & 4 & -1 & 1 & -2 & 4 \\
4 & -8 & 2 & -2 & 1 & -2 & 1 & -1 & 1 & -2 & 1 & -1 & 2 & -4 & 1 & -1 \\
-8 & 16 & -4 & 4 & -2 & 4 & -2 & 2 & -1 & 2 & -1 & 1 & -2 & 4 & -1 & 1 \\
1 & -1 & 4 & -8 & 2 & -2 & 1 & -2 & 1 & -1 & 2 & -4 & 1 & -1 & 1 & -2 \\
-2 & 2 & -8 & 16 & -4 & 4 & -2 & 4 & -1 & 1 & -2 & 4 & -1 & 1 & -1 & 2
\end{bmatrix}$$

其中 -1 即为 255，—2 即为 254，即 mod 256

$$-i \equiv 256 - i \bmod 256$$

即

$$\boldsymbol{M}^{-1} = \begin{bmatrix}
2 & 254 & 1 & 254 & 1 & 255 & 4 & 248 & 2 & 252 & 1 & 255 & 1 & 254 & 1 & 255 \\
252 & 4 & 254 & 4 & 254 & 2 & 248 & 16 & 254 & 4 & 255 & 1 & 255 & 2 & 255 & 1 \\
1 & 254 & 1 & 255 & 2 & 252 & 1 & 255 & 1 & 255 & 1 & 254 & 2 & 254 & 48 & 248 \\
254 & 4 & 254 & 2 & 254 & 4 & 255 & 1 & 255 & 1 & 255 & 2 & 252 & 4 & 248 & 16 \\
1 & 255 & 2 & 252 & 1 & 255 & 1 & 254 & 1 & 254 & 1 & 255 & 4 & 248 & 2 & 254 \\
255 & 1 & 254 & 4 & 255 & 1 & 255 & 2 & 254 & 4 & 254 & 2 & 248 & 16 & 252 & 4 \\
2 & 252 & 1 & 255 & 1 & 254 & 1 & 255 & 2 & 254 & 4 & 248 & 1 & 255 & 1 & 254 \\
254 & 2 & 255 & 1 & 255 & 2 & 255 & 1 & 252 & 4 & 248 & 16 & 254 & 2 & 254 & 4 \\
1 & 255 & 1 & 254 & 1 & 255 & 2 & 252 & 4 & 248 & 2 & 254 & 1 & 254 & 1 & 255 \\
255 & 1 & 255 & 2 & 255 & 1 & 254 & 4 & 248 & 16 & 252 & 4 & 254 & 4 & 254 & 2 \\
1 & 254 & 1 & 255 & 4 & 248 & 2 & 254 & 1 & 255 & 1 & 254 & 1 & 255 & 2 & 252 \\
255 & 2 & 255 & 1 & 248 & 16 & 252 & 4 & 254 & 2 & 254 & 4 & 255 & 1 & 254 & 4 \\
4 & 248 & 2 & 254 & 1 & 254 & 1 & 255 & 1 & 254 & 1 & 255 & 2 & 252 & 1 & 255 \\
248 & 16 & 252 & 4 & 254 & 4 & 254 & 2 & 255 & 2 & 255 & 1 & 254 & 4 & 255 & 1 \\
1 & 255 & 4 & 248 & 2 & 254 & 1 & 254 & 1 & 255 & 2 & 252 & 1 & 255 & 1 & 254 \\
254 & 2 & 248 & 16 & 252 & 4 & 254 & 4 & 255 & 1 & 254 & 4 & 255 & 1 & 255 & 2
\end{bmatrix}$$

3.9.4 子密钥的生成

SAFER$^+$ 每轮需要两个子密钥，包括最后输出，共 $2r+1$ 个，每个子密钥 16 字节。

解密过程也是一样。

SAFER$^+$利用16字节的B字来随机产生子密钥，B字的数目和密钥字的数目，都是一样的$2r+1$个。当用户的密钥长度为128,192,256比特时，B字个数或轮数依次为17,25,33。第一个B字为空字，它不用，它的存在仅仅是为了使算法描述上方便起见。

设B_i表第i个B字，$B_{i,j}$为B_i的第j个字节。$B_2,B_3,\cdots,B_{17}$有

$$B_{i,j} \equiv 45^{(45^{17i+j} \bmod 257)} \bmod 257$$
$$i = 2,3,\cdots,17; \quad j = 1,2,\cdots,16$$

适用于用户密钥长度为128比特。

$B_{18},B_{19},\cdots,B_{33}$有

$$B_{i,j} \equiv 45^{17i+j} \bmod 257$$
$$i = 18,19,\cdots,33, \quad j = 1,2,\cdots,16$$

前面8个：$B_{18},B_{19},\cdots,B_{25}$为密钥长192比特时所需要，全部16个为密钥长256比特的需要。

$B_2,B_3,\cdots,B_{33}$如下所示。

70	151	177	186	163	183	16	10	197	55	179	201	90	40	172	100
236	171	170	198	103	149	88	13	248	154	246	110	102	220	5	61
138	195	216	137	106	233	54	73	67	191	235	212	150	155	104	160
93	87	146	31	213	113	92	187	34	193	190	123	188	153	99	148
42	97	184	52	50	25	253	251	23	64	230	81	29	65	68	143
221	4	128	222	231	49	214	127	1	162	247	57	218	111	35	202
58	208	28	209	48	62	18	161	205	15	224	168	175	130	89	44
125	173	178	239	194	138	206	117	6	19	2	144	79	46	114	51
192	141	207	169	129	226	196	39	47	108	122	159	82	225	21	56
252	32	66	199	8	228	9	85	94	140	20	118	96	255	223	215
251	11	33	0	26	249	166	185	232	158	98	76	217	145	80	210
24	180	7	132	234	91	164	200	14	203	72	105	75	78	156	53
69	77	84	229	37	60	12	74	139	63	204	167	219	107	174	244
45	243	124	109	157	181	38	116	242	147	83	176	240	17	237	131
182	3	22	115	59	30	142	112	189	134	27	71	126	36	86	241
136	70	151	177	186	163	183	16	10	197	55	179	201	90	40	172
220	134	119	215	166	17	251	244	186	146	145	100	131	241	51	239
44	181	178	43	136	209	153	203	140	132	29	20	129	151	113	202
163	139	87	60	130	196	82	92	28	232	160	4	180	133	74	246
84	182	223	12	26	142	222	224	57	252	32	155	36	78	169	152
171	242	96	208	108	234	250	199	217	0	212	31	110	67	188	236
137	254	122	93	73	201	50	194	249	154	248	109	22	219	89	150
233	205	230	70	66	143	10	193	204	185	101	176	210	198	172	30
98	41	46	14	116	80	2	90	195	37	123	138	42	91	240	6
71	111	112	157	126	16	206	18	39	213	76	79	214	121	48	104
117	125	228	237	128	106	144	55	162	94	118	170	197	127	61	175

229	25	97	253	77	124	183	11	238	173	75	34	245	231	115	35
200	5	225	102	221	179	88	105	99	86	15	161	49	149	23	7
40	1	45	226	147	190	69	21	174	120	3	135	164	184	56	207
8	103	9	148	235	38	168	107	189	24	52	27	187	191	114	247
53	72	156	81	47	59	85	227	192	159	216	211	243	141	177	255
62	220	134	119	215	166	17	250	244	186	146	145	100	131	241	51

3.9.5 密钥长 128 比特的子密钥生成

密钥长 128 比特的密钥生成见图 3.35，轮 17 个子密钥的生成步骤如下：

用户自己选的密钥作为第 1 轮子密钥 k_1，将它装入一个有 17 个字节的密钥寄存器中的 16 字节。最后一字节是用来装载这密钥 16 字节的按位作 mod 2 加的结果。密钥寄存器左旋 3 比特。第 2 个子密钥 k_2 由 16 字节的 B_2 和密钥寄存器第 2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17 字节的数分别作 mod 256 的加法运算。密钥寄存器再向左旋转 3 比特位。第 3 个子密钥 k_3 是 16 字节的 B_3 分别和密钥寄存器中第 3,4,5,6,7,8,9,10,11,12,13,14,15,16 和 1 构成的数作 mod 256 的加法运算而得到。

这个过程一直延续，即密钥寄存器向左旋转 3 比特后和适当的 B' 字的 16 字节相加，但密钥寄存器右移一字节，而且字节 1 看作是在字节 17 的右边，直到第 17 个子密钥 k_{17} 产生为止。k_{17} 由 B_{17} 的 16 字节分别和密钥寄存器中第 17,1,2,3,4,5,6,7,8,9,10,11,12,13,14 和 15 分别作 mod 256 的加法。

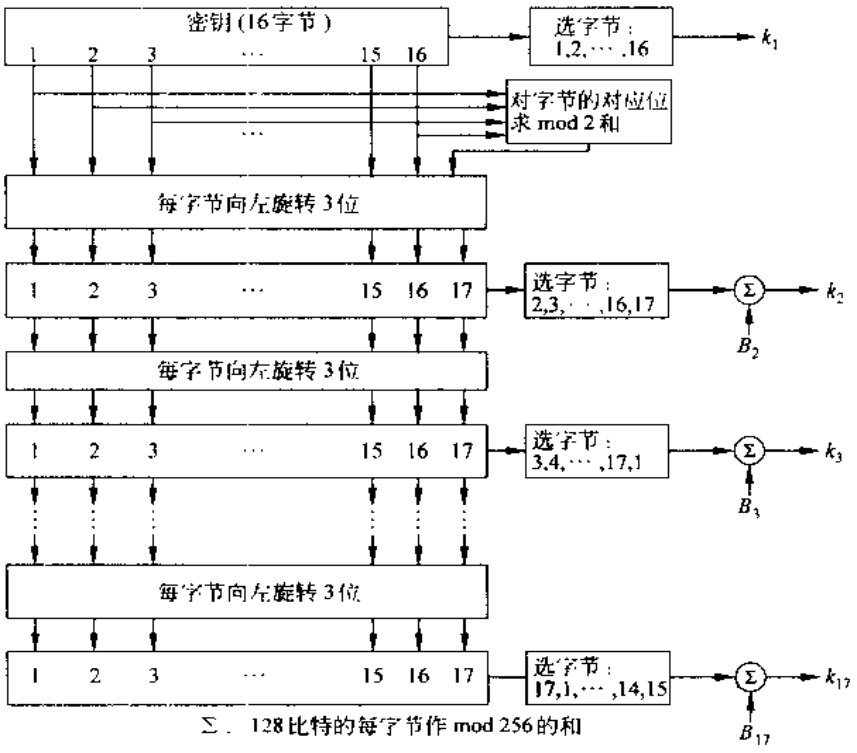


图 3.35

3.9.6 密钥长 192 比特的子密钥生成

192 比特(24 字节)的密钥的子密钥安排如图 3.36 所示。

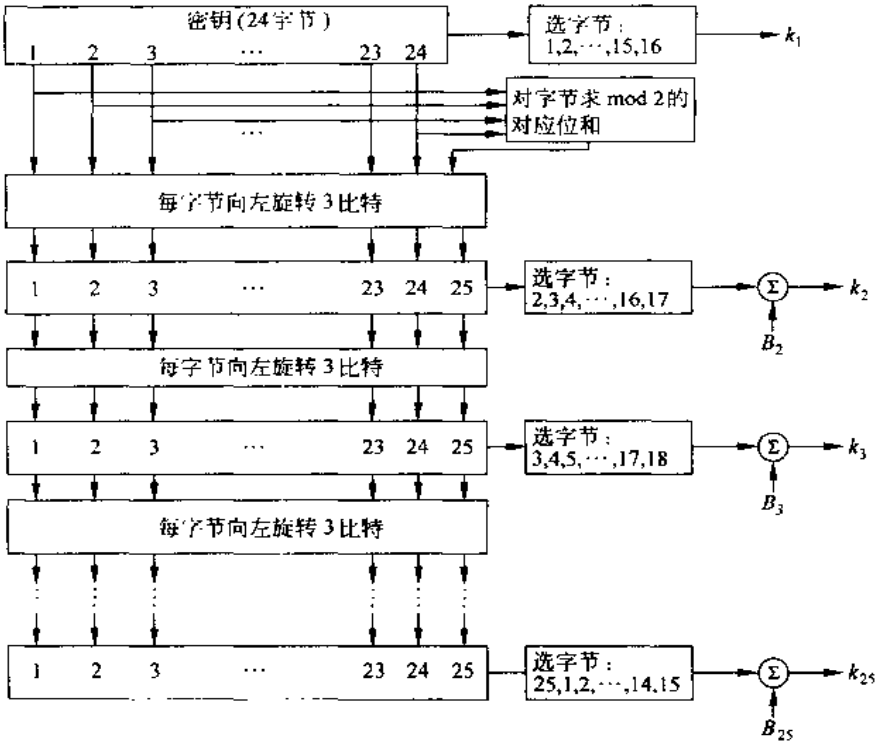


图 3.36

密钥长 192 比特的 12 轮 25 个子密钥是如下面所述的方式产生的。

密钥的前 16 字节便是第 1 个子密钥 k_1,将密钥装进 25 字节密钥寄存器中的前 24 个。密钥寄存器最后一字节是密钥 24 字节的按位 mod 2 和。密钥寄存器每一字节左旋 3 比特,第 2 个子密钥 k_2 是 16 字节的 B^* 字 B_2 按字节分别与密钥寄存器的第 2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17 作 mod 256 的和。密钥寄存器再向左旋转 3 比特,第 3 个子密钥 k_3 是 B_3 的 16 字节分别和密钥寄存器的 3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18 字节作 mod 256 的和。

这个步骤一直继续下去,即密钥寄存器向左移 3 比特,再和适当的 B 字作按字节求和,密钥寄存器每次向右移 1 字节,将位置 1 看作是第 25 字节的右边。一直到 k_{25} 子密钥产生后为止。k_{25} 是 16 字节的 B_{25} 分别和密钥寄存器中的第 25,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 字节作 mod 256 的和。

3.9.7 密钥长 256 比特的子密钥生成

密钥长 256(32 字节)需要 16 轮,33 个子密钥,产生子密钥的过程如图 3.37 所示。

第 1 个子密钥 k_1 即为密钥的前 16 字节,将 32 字节的密钥装进 33 字节的密钥寄存

器的前 32 字节位置。密钥寄存器的最后一字节是密钥的 32 字节作按位求 mod 2 加的结果。密钥寄存器中的每一个字节都向左旋转移位 3 比特，第 2 个子密钥 k_2 是这样产生

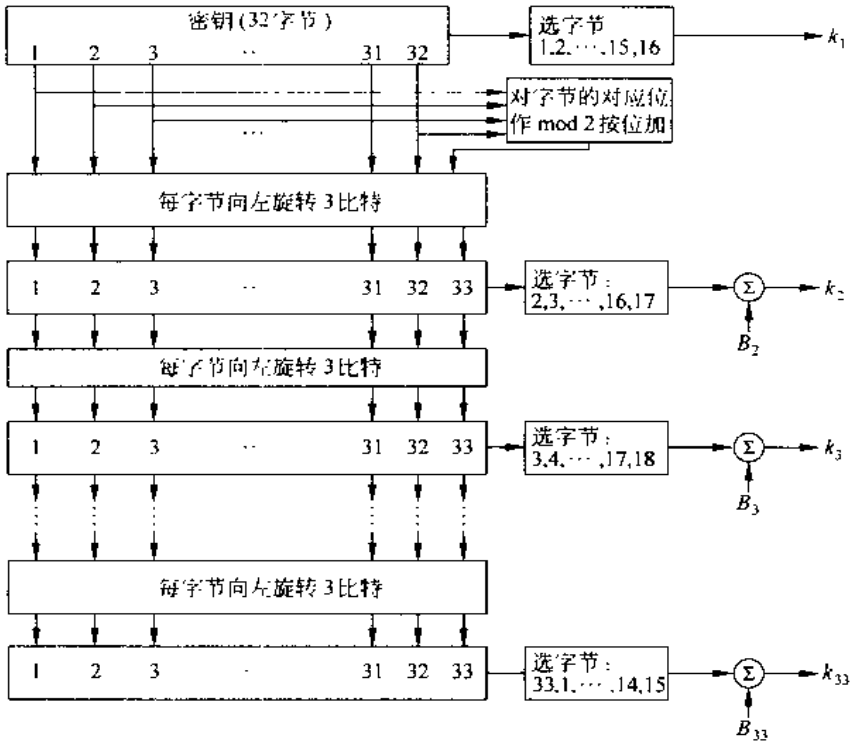


图 3.37

的，由 16 字节的 B_2 和密钥寄存器的第 2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17，此后密钥寄存器的各字节再向左旋转 3 比特。k_3 是 B_3 的 16 字节依次和密钥寄存器里的第 3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18 比特位置作按字节作 mod 256 的加法。依此类推，直到 k_{33} 产生为止，k_{33} 是 B_{33} 的 16 字节依次和密钥寄存的第 33,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 作 mod 256 按字节相加而得到的。

3.10 MARS 密 码

3.10.1 算法的描述

MARS 是由 IBM 公司提出的，也是利用 Feistel 网络的结构，但不是均衡的，算法由 6 个部分组成，其中 3 个阶段如图 3.38 所示。

图 3.38 中的 3 阶段：

(1) 向前混合(forward mixing)：包含“密钥加”和 8 轮“无密钥向前混合”两部分；

(2) 加密核心(cryptographic core)：包含 8 轮“有密钥向前变换”和 8 轮“有密钥向后变换”两部分；

(3) 第 3 阶段包含“8 轮无密钥的向后混合”和“密钥减”两部分，分别在后面内容中进

行叙述。

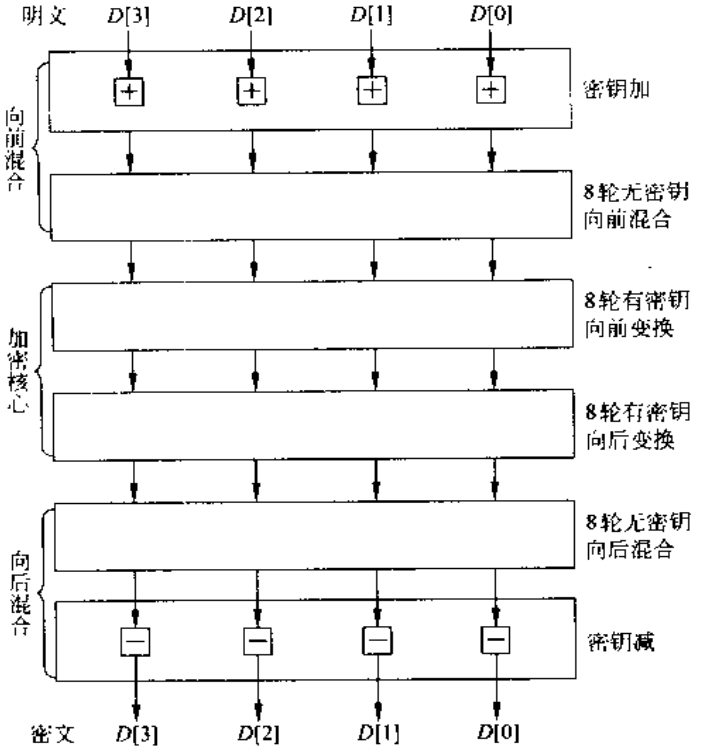


图 3.38

MARS是面向字运算，内部运算都是以32比特的字作为单位。

如在以后的叙述中用$D[\ ]$表示4个32比特的字，$D[\ ]$初始是明文，加密的终了为密文。$K[\ ]$表示40个32比特的扩展密钥；$S[\ ]$512个32比特字的S盒内容。前面256个内容用S_0，后256个用S_1表示。

3.10.2 第1阶段——向前混合

在这阶段首先将密钥字(32比特)和每一个数值字作⊞运算，接着作8轮无密钥Feistel混合，这一轮一个数值字(称为源字)去修改其他3个数值字(称之为目标字)，办法如下：

令源字$D[0]$的4个字节作为S_0和S_1盒的输入，将S盒的结果与其他3个数值字作"异或"或"加"运算。令源字的4个字节为b_0,b_1,b_2,b_3；b_0是最低的字节，b_3是最高字节，b_0,b_2作为S_0盒的输入，b_1,b_3作为S_1盒的输入。

首先$S_0[b_0]$对第1个目标字$D[1]$作"异或"运算，再和$S_1[b_1]$的结果作加法⊞。

同样$S_0[b_2]$与第2个目标字$D[2]$作"加"法，$S_1[b_3]$对第3个目标字$D[3]$作"异或"，最后源字$D[0]$右旋转移位24比特。

第2轮将源字的4个字节作旋转，使第1个目标字$D[1]$变成当前的源字。第2个目标字$D[3]$成为当前的第2个目标字，第3个目标字成为第2轮的第3个目标字。当前源

$D[1]$成为第 2 轮的第 3 个目标字。

在第 1 轮和第 5 轮结束时，第 3 个目标字加到源字，第 2 轮第 6 轮结束时，第 1 个目标字加到源字，这一阶段(向前混合)的流程如图 3.39 所示。

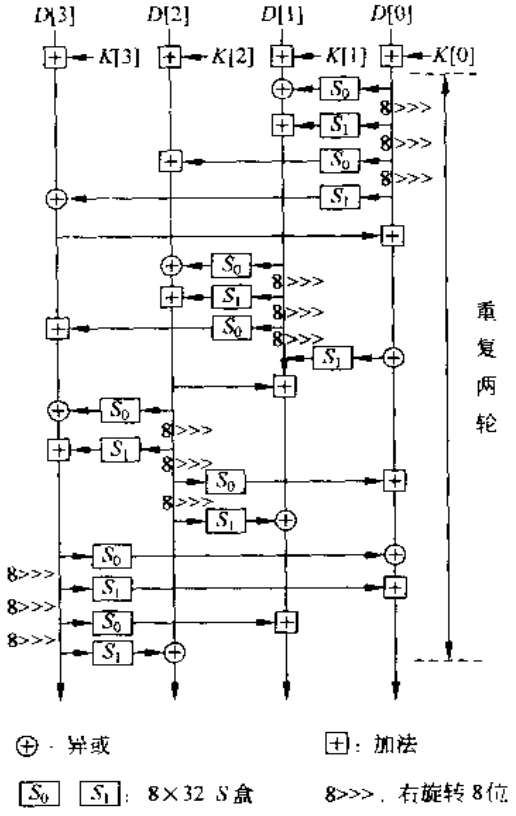


图 3.39

下面给出形式化的算法描述：

// 子密钥与数值作⊞运算

S1. i 从 0 到 3 作

$D[i]\leftarrow D[i]+K[i]$;

// 8 轮的向前混合

S2. i 从 0 到 7 作//利用 $D[0]$去修改 $D[1]$

始 //$D[2]$,$D[3]$

$D[1]\leftarrow D[1]\oplus S_0[D[0]$的最低一字节$]$;

$D[1]\leftarrow D[1]+S_1[D[0]$的第 2 字节$]$;

$D[2]\leftarrow D[2]+S_0[D[0]$的第 3 字节$]$;

$D[3]\leftarrow D[3]\oplus S_1[D[0]$的最高字节$]$;

// 源字向右旋转

$D[0]\leftarrow D[0]\gg 24$;

// 附加的混合运算

若 i 为 0 或 4 作

$D[0]\leftarrow D[0]+D[3]$;

若 i 为 1 或 5 作

$D[0]\leftarrow D[0]+D[1]$;

// $D[\ ]$向右旋转 32 比特;

$(D[3],D[2],D[1],D[0])\leftarrow(D[0],D[3],D[2],D[1])$。

终。

3.10.3 第 2 阶段——密钥控制的变换

MARS 的加密核心也是 Feistel 网络，它包含有 16 轮。每一轮都由密钥参与，其功能由膨胀的函数完成，有乘的运算，依赖于数据的旋转以及 S 盒查找等组成。E 函数以一个数值字的 32 比特为输入、它的 3 个输出分别与其他 3 个数值字相加或作异或运算。其流程图如图 3.40 所示。

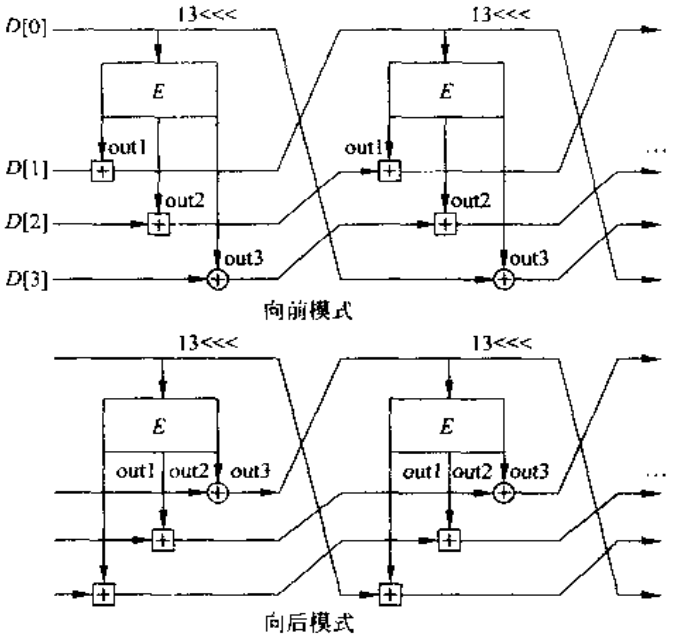


图 3.40

其中 E 函数的流程图如图 3.41 所示，它的输入是 1 个数据字，2 个密钥字，产生 3 个输出字 L,M,R。

1. 算法的叙述

E-function(in,k_1,k_2)

// 临时采用 L,M,R 3 个变量表示左、中、右 3 部分。

始

$M\leftarrow in+k_1$;　　// 加第 1 个密钥字

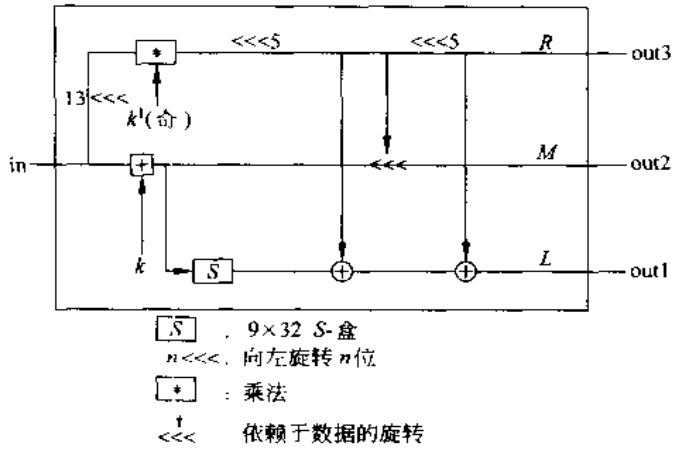


图 3.41

R←(in≪13)×k_2；　　// k_2 必须是奇数

i←M 的最低的 9 比特位；

L←$S[i]$；

R←R≪5；

r←R 的最低 5 比特位；

M←M≪r；　　// 与数有关的旋转

L←L⊕R；

R←R≪5；

L←L⊕R；

r←R 的最低 5 比特位；

L←L≪r；　　// 第 2 次作数值有关的旋转

输出(L,M,R)

终。

2. 利用 E-function 对第 2 阶段有密钥的变换算法叙述如下：

// 16 轮有密钥的变换

i 从 0 到 15 作

始

(out1,out2,out3)←E-function($D[0]$,$k[2i+4]$,$k[2i+5]$)

$D[0]$←$D[0]$≪13；

$D[2]$←$D[2]$+out2；

若 i<8 则作

始

$D[1]$←$D[1]$+out1；

$D[3]$←$D[3]$⊕out3；

终。

否则作

始

$D[3]\leftarrow D[3]$; out1;

$D[1]\leftarrow D[1]\oplus$out3;

终。

// $D[\ ]$向右旋转移位 32 比特

$(D[3],D[2],D[1],D[0])\leftarrow(D[0],D[3],D[2],D[1])$;

终。

3.10.4 第 3 阶段——向后混合

向后混合和解密时向前混合类似，向后混合流程图如图 3.42 所示，只是数值字的顺序不同。

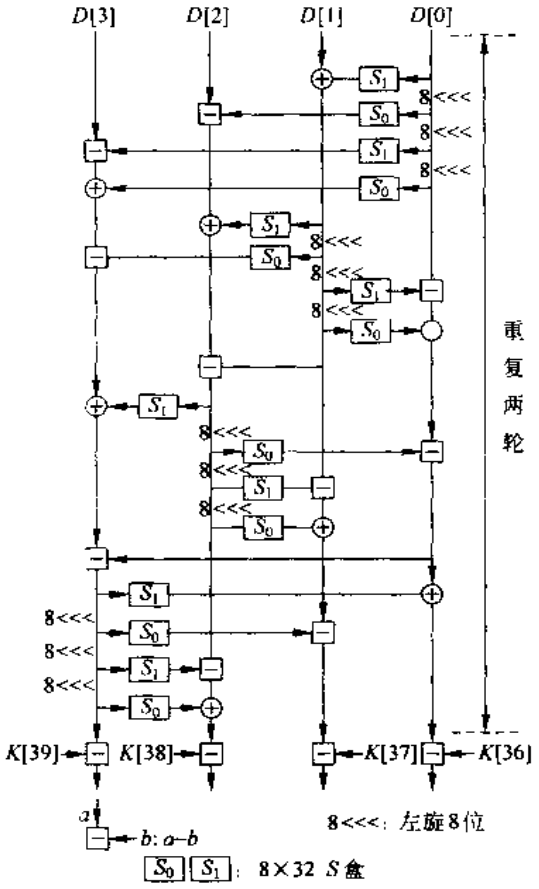


图 3.42

算法的叙述：

// 8 轮的向后混合

i 从 0 到 7 作

始

若 $i=2$ 或 6 则作

$D[0]\leftarrow D[0]-D[3]$;

若 $i=3$ 或 7 则作

$D[0]\leftarrow D[0]+D[1]$;

$D[1]\leftarrow D[1]\oplus S_1[D[1]$的最低一字节$]$;

$D[2]\leftarrow D[2]+S_0[D[0]$的最高一字节$]$;

$D[3]\leftarrow D[3]+S_1[D[0]$的第 3 字节$]$;

$D[3]\leftarrow D[3]\oplus S_0[D[0]$的第 2 字节$]$;

$D[0]\leftarrow D[0]\ll 24$;

$(D[3],D[2],D[1],D[0])\leftarrow(D[0],D[3],D[2],D[1])$;

终。

i 从 0 到 3 作

$D[i]=D[i]-K[36+i]$;

加密算法就是依次由这 3 阶段另加函数组成的。

3.10.5 解密

MARS 密码的解密算法和加密算法类似,但不完全相同,现叙述如下:

MARS-decrypt(input; $D[\ ],K[\ ]$)

第 1 阶段:向前混合

for $i=0$ 到 3 作

$D[i]\leftarrow D[i]+K[36+i]$

for $i=7$ 到 0 作

始

$(D[3],D[2],D[1],D[0])\leftarrow(D[2],D[1],D[0],D[3])$;

$D[0]\leftarrow D[0]\gg 24$;

$D[3]\leftarrow D[3]\oplus S_1[D[0]$的第 2 字节$]$;

$D[3]\leftarrow D[3]+S_1[D[0]$的第 3 字节$]$;

$D[2]\leftarrow D[2]+S_0[D[0]$的最高字节$]$;

$D[1]\leftarrow D[1]\oplus S_1[D[0]$的最低字节$]$;

若 $i=2$ 或 6 则作

$D[0]=D[0]+D[3]$;

若 $i=3$ 或 7 则作

$D[0]\leftarrow D[0]-D[1]$;

终。//第 1 阶段终

//第 2 阶段:带密钥的变换

i 从 15 到 0 作

始

$(D[3],D[2],D[1],D[0])\leftarrow(D[2],D[1],D[0],D[3])$;

$D[0]\leftarrow D[0]\gg 13$;

(out1,out2,out3)←E-function($D[0],K[2i+4],K[2i+5]$);

$D[2]\leftarrow D[2]-$out2;

若 $i<8$ 则作

始

$D[1]\leftarrow D[1]-$out1;

$D[3]\leftarrow D[3]\oplus$out3;

终。

否则作

始

$D[3]\leftarrow D[3]-$out1;

$D[1]\leftarrow D[1]\oplus$out3;

终。

终。//第 2 阶段终

第 3 阶段：向后混合

若 i 从 7 到 0 作

始

$(D[3],D[2],D[1],D[0])\leftarrow(D[2],D[1],D[0],D[3])$;

若 $i=0$ 或 4 则作

$D[0]\leftarrow D[0]-D[3]$;

$i=1$ 或 5 则作

$D[0]\leftarrow D[0]-D[1]$;

$D[0]\leftarrow D[0]\ll 24$;

$D[3]\leftarrow D[3]\oplus S_1[D[0]$的高字节$]$;

$D[2]\leftarrow D[2]-S_0[D[0]$的第 3 字节$]$;

$D[1]\leftarrow D[1]-S_1[D[0]$的第 2 字节$]$;

$D[0]\leftarrow D[1]\oplus S_0[D[0]$的低字节$]$;

终。

i 从 0 到 3 作

$D[i]\leftarrow D[i]-K[i]$;

//第 3 阶段终

当然 E-function 也是解密运算的一部分。

3.10.6 子密钥生成

子密钥生成如图 3.43 所示。

子密钥生成的过程是将 $k[\ ]$扩展为 $K[\ ]$，$k[\ ]$有 n 个 32 比特字，$K[\ ]$有 40 个 32 比特字，其中 $4\leqslant n\leqslant 39$。

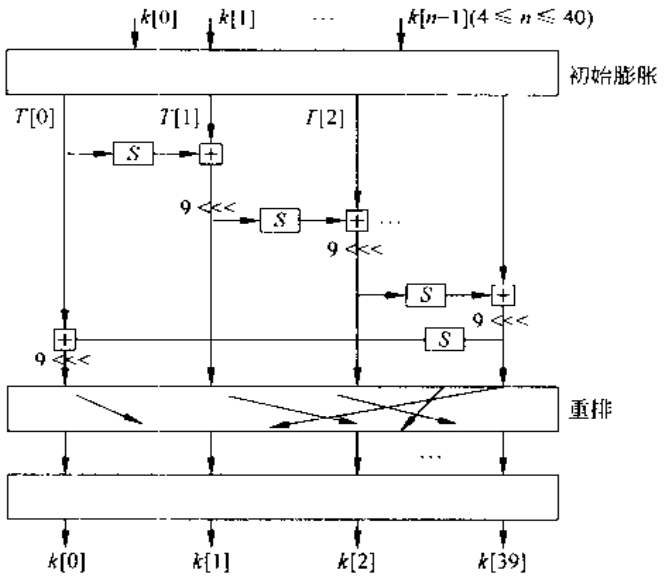


图 3.43

$k[]$不要求具有任何性质，只要不包含全等位就可以了。$K[\ ]$具有：

(1) 最低两位用于乘法时，令其为 0；

(2) 不存在 10 个相连的 0 或 1。

子密钥生成的过程有 4 步。

(1) 初始膨胀：构造临时的数组($T[]$)

$T[i]\leftarrow S[i+7],\quad i=-7,-6,\cdots,-1$

其中 S 是 S 盒。

$T[i]\leftarrow(T[i-7]\oplus T[i-2]\lll 3)\oplus k[i \bmod n]\oplus i$；

$i=0,1,2,\cdots,38$。

$T[39]\leftarrow n$；

(2) 搅和 7 轮

$T[i]\leftarrow(T[i]+S[T[i-1]$的低位 9 比特$])\lll 9,\quad i=1,2,\cdots,39$

$T[0]\leftarrow(T[0]+S[T[39]$的低位 9 比特$])\lll 9$

(3) 重新排序

$K[7i \bmod 40]\leftarrow T[i],\quad i=0,1,2,\cdots,39$

其中

n 是 $k[]$的字数。

$K[]$膨胀密钥数组，共 40 个字。

$T[]$临时数组，从 $T[-7],T[-6],\cdots,T[39]$，共 47 个字。

(4) 检测弱密钥

所谓字 W 是弱的，指 W 的最低 2 比特全为零后有相邻 10 位是零，或相邻 10 位是 1。讨论从略。

$B[\ ]=\{$A4A8D57B,5B5D193B,C8A8309B,73F9A978$\}$,

S1. $T[-7,\cdots,-1]\leftarrow S[0,\cdots,6]$

S2. $i\leftarrow 1$,

S3. 若 $i\leqslant 39$ 则作

始

$T[i]\leftarrow((T[i-7]\oplus T[i-2])\lll 3)\oplus k[i \bmod n]\oplus i$

$i\leftarrow i+1$,转 S3。

终。

S4. $T[39]\leftarrow n$

S5. $j\leftarrow 1$

S6. 若 $j\leqslant 7$ 则作

始

i 从 1～39 作

$T[i]\leftarrow(T[i]+S[T[i-1]$的低位 9 比特$])\lll 9$

$T[0]\leftarrow(T[0]+S[T[39]$的低位 9 比特$])\lll 9$

$j\leftarrow j+1$,转 S6,

终,否则转 S7。

S7. i 从 0 到 39 作

$K[7i \bmod 40]\leftarrow T[i]$;

S8. $i=5,7,\cdots,35$ 作

始

$j\leftarrow K[i]$的最低两位;

$W\leftarrow K[i]$的最低两位置 1 后的 $K[i]$;

//若 $K[i]$非弱密钥,则 $M\leftarrow 0$。

$M_i\leftarrow 1$ 当且仅当 w_i 属于 W 存在 10 个 0 或 1 相连的序列,$i\geqslant 2$ 且

$$w_{i-1}=w_i=w_{i+1}$$

$r\leftarrow K[i+3]$的最小 5 比特。

$p\leftarrow B[j]\lll r$,

$K[i]\leftarrow w\oplus(p\wedge m)$;

终。

3.10.7 *S* 盒

MARS 的 S 盒生成过程是

$$S[5i+j]\leftarrow\text{SHA-1}(i,C_1,C_2,C_3),$$
$$i=0,1,\cdots,102;\quad j=0,1,2,3,4$$

其中 SHA-1$(\cdot)_j$ 是 SHA-1$(\cdot)$ 输出的第 j 个字。$C_1=$0xb7e15162,$C_2=$0x243f6a88,C_3 使之浮动直到找到使 S 盒的性质达到满意为止。

```
sbox[ ] = {
0x09d0c479, 0x28c8ffe0, 0x84aa6c39, 0x9dad7287, 0x7dff9be3, 0xd4268361,
0xc96da31d4, 0x7974cc93, 0x85d0582e, 0x2a4b5705, 0x1ca16a62, 0xc3bd279d,
0x0f1f25e5, 0x5160372f, 0xc695c1fb, 2x4d7ff1e4, 0xae5f6bf4, 0x0d72ee46,
0xff23de8a, 0xb1cf8e83, 0xf14902e2, 0x3e981e42, 0x8bf53eb6, 0x7f4bf8ac,
0x83631f83, 0x25970205, 0x76afe784, 0x3a7931d4, 0x4f846450, 0x5c64c3f6,
0x210a5f18, 0xc6986a26, 0x28f4c826, 0x3a60a81c, 0xd340a664, 0x7ea820c4,
0x526687c5, 0x7edddd12b, 0x32a11d1d, 0x9c9ef086, 0x80f6e831, 0xab6f04ad,
0x56fb9b53, 0x8b2e095c, 0xb68556ae, 0xd2250b0d, 0x294a7721, 0xe21fb253,
0xae136749, 0xe82aae86, 0x93365104, 0x99404a66, 0x78a784dc, 0xb69ba84b,
0x04046793, 0x23db5c1e, 0x46cae1d6, 0x2fe28134, 0x5a223942, 0x1863cd5b,
0xc199c6e3, 0x07dfb846, 0x6eb88816, 0x2d0dcc4a, 0xa4ccae59, 0x3798670d,
0xcbfa9493, 0x4f481d45, 0xeafc8ca8, 0xdb1129d6, 0xb0449e20, 0x0f5407fb,
0x6167d9a8, 0xd1f45763, 0x4daa96c3, 0x3bec5958, 0xababa014, 0xb6ccd201,
0x38d6279f, 0x02682215, 0x8f376cd5, 0x092c237e, 0xbfc56593, 0x32889d2c,
0x854b3e95, 0x05bb9b43, 0x7dcd5dcd, 0xa02e926c, 0xfae527e5, 0x36a1c330,
0x3412e1ae, 0xf257f462, 0x3c4f1d71, 0x30a2e809, 0x68e5f551, 0x9c61ba44,
0x5ded0ab8, 0x75ce09c8, 0x9654f93e, 0x698c0cca, 0x243cb3e4, 0x2b062b97,
0x0f3b8d9e, 0x00e050df, 0xfc5d6166, 0xe35f9288, 0xc079550d, 0x0591aee8,
0x8e531e74, 0x75fe3578, 0x2f6d829a, 0xf60b21ae, 0x95e8eb8d, 0x6699486b,
0x901d7d9b, 0xfd6d6e31, 0x1090acef, 0xe0670dd8, 0xdab2e692, 0xcd6d4365,
0xe5393514, 0x3af345f0, 0x6241fc4d, 0x460da3a3, 0x7bcf3729, 0x8bf1d1e0,
0x14aac070, 0x1587ed55, 0x3afd7d3e, 0xd2f29e01, 0x29a9d1f6, 0xefb10c53,
0xcf3b870f, 0xb414935c, 0x664465ed, 0x024acac7, 0x59a744c1, 0x1d2936a7,
0xdc580aa6, 0xcf574ca8, 0x040a7a10, 0x6cd81807, 0x8a98be4c, 0xaccea063,

0xc33e92b5, 0xd1e0e03d, 0xb322517e, 0x2092bd13, 0x386b2c4a, 0x52e8dd58,
0x58656dfb, 0x50820371, 0x41811896, 0xe337ef7e, 0xd39fb119, 0xc97f0df6,
0x68fea01b, 0xa150a6e5, 0x55258962, 0xeb6ff41b, 0xd7c9cd7a, 0xa619cd9e,
0xbcf09576, 0x2672c073, 0xf003fb3c, 0x4ab7a50b, 0x1484126a, 0x487ba9b1,
0xa64fc9c6, 0xf6957d49, 0x38b06a75, 0xdd805fcd, 0x63d094cf, 0xf51c999e,
0x1aa4d343, 0xb8495294, 0xce9f8e99, 0xbffcd770, 0xc7c275cc, 0x378453a7,
0x7b21be33, 0x397f41bd, 0x4e94d131, 0x92cc1f98, 0x5915ea51, 0x99f861b7,
0xc9980a88, 0x1d74fd5f, 0xb0a495f8, 0x614deed0, 0xb5778eea, 0x5941792d,
```

```
0xfa90c1f8, 0x33f824b4, 0xc4965372, 0x3ff6d550, 0x4ca5fec0, 0x8630e964,
0x5b3fbbd6, 0x7da26a48, 0xb203231a, 0x04297514, 0x2d639306, 0x2eb13149,
0x16a45272, 0x523459a0, 0x8e5f4872, 0xf966c7d9, 0x07128dc0, 0x0d44db62,
0xafc8d52d, 0x06316131, 0xd838e7ce, 0x1bc41d00, 0x3a2e8c0f, 0xea86837e,
0xb984737d, 0x13ba4891, 0xc4f8b949, 0xa6d6acb3, 0xa215cdce, 0x8359838b,
0x6bd1aa31, 0xf579dd52, 0x21b93f93, 0xf5176781, 0x187dfdde, 0xe94aeb76,
0x2b38fd54, 0x431de1da, 0xab394825, 0x9ad3048f, 0xdfea32aa, 0x659473e3,
0x623f7863, 0xf3346c59, 0xab3ab685, 0x3346a90b, 0x6b56443e, 0xc6de01f8,
0x8d421fc0, 0x9b0ed10c, 0x88f1a1e9, 0x54c1f029, 0x7dead57b, 0x8d7ba426,
0x4cf5178a, 0x551a7cca, 0x1a9a5f08, 0xfcd651b9, 0x25605182, 0xe11fc6c3,
0xb6fd9676, 0x337b3027, 0xb7c8eb14, 0x9e5fd030,

0x6b57e354, 0xad913cf7, 0x7e16688d, 0x58872a69, 0x2c2fc7df, 0xe389ccc6,
0x30738df1, 0x0824a734, 0xe1797a8b, 0xa4a8d57b, 0x5b5d193b, 0xc8a8309b,
0x73f9a978, 0x73398d32, 0x0f59573e, 0xe9df2b03, 0xe8a5b6c8, 0x848d0704,
0x98df93c2, 0x720a1dc3, 0x684f259a, 0x943ba848, 0xa6370152, 0x863b5ea3,
0xd17b978b, 0x6d9b58ef, 0x0a700dd4, 0xa73d36bf, 0x8e6a0829, 0x8695bc14,
0xe35b3447, 0x933ac568, 0x8894b022, 0x2f511c27, 0xddfbcc3c, 0x006662b6,
0x117c83fe, 0x4e12b414, 0xc2bca766, 0x3a2fec10, 0xf4562420, 0x55792e2a,
0x48f5d857, 0xceda25ce, 0xc3601d3b, 0x6c00ab46, 0xefac9c28, 0xb3c35047,
0x611dfee3, 0x257c3207, 0xfdd58482, 0x3b14d84f, 0x23becb64, 0xa075f3a3,
0x088f8ead, 0x07adf158, 0x7796943c, 0xfacabf3d, 0xc09730cd, 0xf7679969,
0xda44e9ed, 0x2c854c12, 0x35935fa3, 0x2f057d9f, 0x690624f8, 0x1cb0bafd,
0x7b0dbdc6, 0x810f23bb, 0xfa929a1a, 0x6d969a17, 0x6742979b, 0x74ac7d05,
0x010e65c4, 0x86a3d963, 0xf907b5a0, 0xd0042bd3, 0x158d7d03, 0x287a8255,
0xbba8366f, 0x096edc33, 0x21916a7b, 0x77b56b86, 0x951622f9, 0xa6c5e650,
0x8cea17d1, 0xcd8c62bc, 0xa3d63433, 0x358a68fd, 0x0f9b9d3c, 0xd6aa295b,
0xfe33384a, 0xc000738e, 0xcd67eb2f, 0xe2eb6dc2, 0x97338b02, 0x06c9f246,
0x419cf1ad, 0x2b83c045, 0x3723f18a, 0xcb5b3089, 0x160bead7, 0x5d494656,
0x35f8a74b, 0x1e4e6c9e, 0x000399bd, 0x67466880, 0xb4174831, 0xacf423b2,
0xca815ab3, 0x5a6395e7, 0x302a67c5, 0x8bdb446b, 0x108f8fa4, 0x10223eda,
0x92b8b48b, 0x7f38d0ee, 0xab2701d4, 0x0262d415, 0xaf224a30, 0xb3d88aba,
0xf8b2c3af, 0xdaf7ef70, 0xcc97d3b7, 0xe9614b6c, 0x2baebff4, 0x70f687cf,
```

```
0x386c9156, 0xce092ee5, 0x01e87da6, 0x6ce91e6a, 0xbb7bcc84, 0xc7922c20,
0x9d3b71fd, 0x060e41c6, 0xd7590f15, 0x4e03bb47, 0x183c198e, 0x63eeb240,
0x2ddbf49a, 0x6d5cba54, 0x923750af, 0xf9e14236, 0x7838162b, 0x59726c72,
0x81b66760, 0xbb2926c1, 0x48a0ce0d, 0xa6c0496d, 0xad43507b, 0x718d496a,
0x9df057af, 0x44b1bde6, 0x054356dc, 0xde7ced35, 0xd51a138b, 0x62088cc9,
0x35830311, 0xc96efca2, 0x686f86ec, 0x8e77cb68, 0x63e1d6b8, 0xc80f9778,
0x79c491fd, 0x1b4c67f2, 0x72698d7d, 0x5e368c31, 0xf7d95e2e, 0xa1d3493f,

0xdcd9433e, 0x896f1552, 0x4bc4ca7a, 0xa6d1baf4, 0xa5a96dcc, 0x0bef8b46,
0xa169fda7, 0x74df40b7, 0x4e208804, 0x9a756607, 0x038e87c8, 0x20211e44,
0x8b7ad4bf, 0xc6403f35, 0x1848e36d, 0x80bdb038, 0x1e62891c, 0x643d2107,
0xbf04d6f8, 0x21092c8c, 0xf644f389, 0x0778404e, 0x7b78adb8, 0xa2c52d53,
0x42157abe, 0xa2253e2e, 0x7bf3f4ae, 0x80f594f9, 0x953194e7, 0x77eb92ed,
0xb3816930, 0xda8d9336, 0xbf447469, 0xf26d9483, 0xee6faed5, 0x71371235,
0xde425f73, 0xb4e59f43, 0x7dbe2d4e, 0x2d37b185, 0x49dc9a63, 0x98c39d98,
0x1301c9a2, 0x389b1bbf, 0x0c18588d, 0xa421c1ba, 0x7aa3865c, 0x71e08558,
0x3c5cfcaa, 0x7d239ca4, 0x0297d9dd, 0xd7dc2830, 0x4b37802b, 0x7428ab54,
0xaeee0347, 0x4b3fbb85, 0x692f2f08, 0x134e578e, 0x36d9e0bf, 0xae8b5fcf,
0xedb93ecf, 0x2b27248e, 0x170eb1ef, 0x7dc57fd6, 0x1e760f16, 0xb1136601,
0x864e1b9b, 0xd7ea7319, 0x3ab871bd, 0xcfa4d76f, 0xe31bd782, 0x0dbeb469,
0xabb96061, 0x5370f85d, 0xffb07e37, 0xda30d0fb, 0xebc977b6, 0x0b98b40f,
0x3a4d0fe6, 0xdf4fc26b, 0x159cf22a, 0xc298d6e2, 0x2b78ef6a, 0x61a94ac0,
0xab561187, 0x14eea0f0, 0xdf0d4164, 0x19af70ee};
```

3.11 TEA 密码

下面介绍叫做 TEA 的加密算法。TEA 是 Tiny Encryption Algorithm 的缩写，即极小型的加密算法。由 David Wheeler 和 Roger Needham 在剑桥大学计算机实验室联合研究的。特点是加密速度极快，高速高效，抗差分攻击能力强。明文密文块 64 比特，但密钥长度为 128 比特。算法的描述十分简单：

S1.（初始化过程）。

加密数据分成两部分 $v(0)$ 和 $v(1)$ 各 32 比特，$y \leftarrow v(0)$，$z \leftarrow v(1)$，$Sum \leftarrow 0$。

$Delta \leftarrow$0x9E3779B9（十六进制）

密钥 128 比特分成 4 部分 $k(0)$，$k(1)$，$k(2)$，$k(3)$，各 32 比特。

$a \leftarrow k(0)$，$b \leftarrow k(1)$，$c \leftarrow k(2)$，$d \leftarrow k(3)$，$n \leftarrow 32$

S2. 若 $n>0$ 则转 S3，否则转 S4。

S3. $Sum \leftarrow Sum + Delta$；

$y \leftarrow y + (z \ll 4) + a \oplus z + Sum \oplus (z \gg 5) + b$；

$z \leftarrow z + (y \ll 4) + c \oplus y + Sum \oplus (y \gg 5) + d$；

$n \leftarrow n - 1$，转 S2。

S4. $v(0) \leftarrow y$，$v(1) \leftarrow z$，结束。

TEA 加密算法迭代的次数可以改变，32 次很充分，16 次足够了，8 次也可以。明文改变 1 比特，迭代 6 次便可影响使之改变 32 比特位。

S3 中有作≪、≫运算，有作⊕运算，还有作＋运算。其顺序是先作≪或≫，再作⊕运算，最后才是＋运算。

⊕ 是按位作异或运算。

≪ 是按位左移，高位舍弃，低位补零。

≫ 是按位右移，低位舍弃，高位补零。

解密算法和加密类似：

S1.（初始化过程）。

解密数据分两部分 $v(0)$ 和 $v(1)$，各 32 比特。

$y \leftarrow v(0)$，$z \leftarrow v(1)$。

$Sum \leftarrow$ 0xC6EF3720(十六进制)

$Delta \leftarrow$ 0x9E3779B9(十六进制)

$a \leftarrow k(0)$，$b \leftarrow k(1)$，$c \leftarrow k(2)$，$d \leftarrow k(3)$，$n \leftarrow 32$。

S2. 若 $n>0$ 则转 S3，否则转 S4。

S3. $z \leftarrow z(y \ll 4) + c \oplus y + Sum \oplus (y \gg 5) + d$；

$y \leftarrow y(z \ll 4) + a \oplus z + Sum \oplus (z \gg 5) + b$；

$Sum \leftarrow Sum - delta$；

$n \leftarrow n - 1$，转 S2。

S4. $v(0) \leftarrow y$，$v(1) \leftarrow z$，结束。

第4章 公钥密码

4.1 引 言

保密通信进入计算机网络时代，传统密码便逐渐暴露出它的固有弱点。传统密码要求通信双方用的密钥应通过秘密信道私下约定。互联网上若有 n 个用户，则需要

$$\binom{n}{2}=\frac{n}{2}(n-1)$$

个密钥，若 $n=1000$，则 $C(1000,2)\approx 500\,000$，如此庞大的密钥如何管理？它的定期更换也是十分复杂的工程。更有甚者，每一个用户与其他 $n-1$ 个用户的保密通信，需保存 $n-1$ 个密钥，记在本上或存储在计算机内部，这方法的本身就是十分不安全的。

前面已提到 Diffie-Hellman 在他们的一篇《密码学新方向》的文章中，提出一种“公钥密码”的新思想：每个用户有一加密用的密钥 k_e，还有一个解密用的密钥 k_d，将 k_e 公开，k_d 保密，而且 k_e 的公开不至于妨碍到 k_d 的安全。可将各用户的 k_e 集中成公钥文件，像电话本一样公开发给所有用户。若 B 欲与 A 保密通信，查公钥文件得 A 的加密用公钥 $k_e^{(A)}$，用之加密得密文

$$c=E_{k_e^{(A)}}(m)$$

将 c 寄给 A，A 用只有他自己才掌握的解密密钥 $k_d^{(A)}$ 解密恢复明文 m：

$$m=D_{k_d^{(A)}}(c)$$

任何第三者截获密文 c 后，由于不掌握解密密钥，也无法得知明文 m。

由于 $k_e^{(A)}\neq k_d^{(A)}$，故公钥密码也就称为“非对称密码”，相对于公钥密钥，以往的传统密码通信双方用的密钥是一样的，因此也就称之为“对称密码”。

当 Diffie-Hellman 提出公钥密码的新思想时，还没有公钥密码的范例。但在它的思想影响下，各种公钥密码开始涌现，有的成功了，有的失败了，有的还待实践。这一章里将介绍它们。可以毫不夸张地说“没有公钥密码，就没有近代密码学”，公钥密码的出现在密码学的发展史上是一件值得大书特书的事件。

Diffie-Hellman 基于求离散对数的困难性提出一种代替办法。

办法是这样的：对于保密通信系统有一大素数 p，通过 p 建立一 $GF(p)$ 域，若 g 是域 $GF(p)$ 上的本原元素，系统将 p 和 g 向用户公开。

每一用户（设为 A）任选一数 x_A

$$x_A\in[0,1,2,\cdots,p-1]$$

计算

$$y_A\equiv g^{x_A}\bmod p$$

将 y_A 公开称为 A 的公钥，x_A 由 A 秘密保存称 x_A 为 A 的私钥。

若用户 B 欲与 A 保密通信，可用自己的密钥 x_B 及 A 的公钥计算

$$k_{AB} \equiv (y_A)^{x_B} \bmod p \equiv (g^{x_A} \bmod p)^{x_B} \bmod p$$

$$\equiv (g^{x_A x_B}) \bmod p \equiv (g^{x_B})^{x_A} \bmod p$$

由于

$$y_B = g^{x_B} \bmod p$$

所以

$$k_{AB} \equiv (y_B)^{x_A} \bmod p \equiv (y_A)^{x_B} \bmod p$$

故这个 A 与 B 间通信用的密钥 k_{AB} 不论 A 或 B 都能从对方的公钥及自己的密钥通过计算得到。k_{AB} 便是 AB 的会话密钥，保密通信还是通过传统密码进行。

它本身不是公钥密码意义下的公钥密码。

例

$$p=11,\ g=7,\ x_A=8,\ y_A=7^8 \bmod 11$$

所以

$$y_A=7^8=5764001\equiv 9 \bmod 11, \qquad x_B=4$$

$$y_B=7^4=2401\equiv 3 \bmod 11$$

$$k_{AB}=(y_A)^{x_B} \bmod 11\equiv(9)^4 \bmod 11=6561 \bmod 11=5 \bmod 11$$

同样

$$k_{AB}\equiv(y_B)^{x_A} \bmod 11\equiv(3)^8 \bmod 11\equiv(4)^{-2} \bmod 11\equiv 5$$

4.2 背包公钥密码系统

4.2.1 背包问题

当 Diffie 和 Hellman 提出公钥密码的设想时，还没有一个这样的实例。两年后首先由 Merkle 和 Hellman 提出一基于组合数学中背包问题的公钥密码系统。这个背包系统称之为 MH 背包公钥。

所谓背包问题是这样：已知有 n 个物品，它的重量分别为 $a_1, a_2, \cdots, a_n$。今有装有其中某些物品，重量为 b 的背包，问装的是哪些物品？这问题导致求 $x_i=0$ 或 $1, i=1, 2, \cdots, n$ 使满足

$$\sum_{i=1}^{n} a_i x_i = b$$

背包问题是著名的 NP 完备类困难问题，至今还没有好的求解方法。若对 2^n 种所有可能进行穷举搜索，实际上是不可能的。以 $n=100$ 为例：

$$2^{100} = 1.27 \times 10^{30}$$

以每秒搜索 10^7 种方案的超高速电子计算机进行穷举，一年只能完成 3.1536×10^{14} 次，所以共要

$$1.27 \times 10^{30}/(3.1536 \times 10^{14}) = 4.02 \times 10^{15}(\text{年})$$

算法复杂性理论已经证明：背包问题属于 NP 完备类，也就是说它是 NP 类问题中难度最大的一类。这一类问题还没有有效算法。对 2^n 种可能穷举搜索不是好算法，因为它不可能在实际允许的时间内完成。

必须指出的是，并非所有的背包问题都没有有效算法求解。如若序列 $a_1, a_2, \cdots, a_n$ 满足条件：

$$a_i > \sum_{j=1}^{i-1} a_j \quad i = 2,3,\cdots,n$$

成立时，有多项式解法。这样的序列称之为超递增序列，例如：$1,2,4,8,\cdots,2^n$ 就是超递增序列。

$$x_1 + 2x_2 + 4x_3 + 8x_4 + 16x_5 + 32x_6 = 43$$

则因 $43>32$，故 $x_6=1$，以之代入得：

$$x_1 + 2x_2 + 4x_3 + 8x_4 + 16x_5 = 11$$

$11<16$，只能有 $x_5=0$

$$x_1 + 2x_2 + 4x_3 + 8x_4 = 11$$

所以 $$x_1 = x_2 = x_4 = x_6 = 1,\ x_3 = x_5 = 0$$

是问题的解。

4.2.2 MH 背包公钥密码

MH 背包公钥密码是由 Merkle 和 Hellman 联合提出的，是第一批涌现的“公钥密码”。

背包公钥密码系统是选取一组正整数 $a_1, a_2, \cdots, a_n$ 作为公钥予以公布，$m = m_1 m_2 \cdots m_n$ 是 n 位 0,1 明文符号串。利用公钥加密如下：

$$c = a_1 m_1 + a_2 m_2 + \cdots + a_n m_n$$

反过来从已知密文 c 求解明文 $m = m_1 m_2 \cdots m_n$ 等价于解背包问题。例如，已知：

$$a_1 = 28,\ a_2 = 32,\ a_3 = 11,\ a_4 = 08$$

$$a_5 = 71,\ a_6 = 51,\ a_7 = 43,\ a_8 = 67$$

明文

$$m_1 = 10110110,\ m_2 = 11001001$$

分别加密得密文：

$$c_1 = 28 + 11 + 8 + 51 + 43 = 141$$

$$c_2 = 28 + 32 + 71 + 67 = 198$$

但是，MH 背包公钥系统选的背包序列 $a_1, a_2, \cdots, a_n$ 是由超递增序列进行以下变换得到的。

设 $b_1, b_2, \cdots, b_n$ 是超递增序列，即

$$b_i > \sum_{j=1}^{i-1} b_j, \quad i = 2,3,\cdots,n$$

选取 $m > 2b_n$，w 和 m 互素，$\overline{w}$ 满足 $w\overline{w} \equiv 1 \pmod{m}$。作变换（称为墨科-赫尔曼(Merkle Hellman)变换）：

$$a_k \equiv w b_k \pmod{m}, \quad k = 1,2,\cdots,n$$

于是从超递增序列：$b_1,b_2,\cdots,b_n$ 得序列：$a_1,a_2,\cdots,a_n$。一般说来，$\{a_i\}$表面上再不具有超递增的特性，MH 背包公钥密码系统便是以这样的序列$\{a_i\}$作为公钥的。

假定已知

$$a_1m_1+a_2m_2+\cdots+a_nm_n=c$$

c 为 $m=m_1m_2\cdots m_n$ 的密文，其中 $m_1,m_2,\cdots,m_n$ 是 n 位的 0,1 符号串，则：

$$\begin{aligned}\bar{w}c&=\bar{w}a_1m_1+\bar{w}a_2m_2+\cdots+\bar{w}a_nm_n\\&\equiv b_1m_1+b_2m_2+\cdots+b_nm_n(\mathrm{mod}\ m)\end{aligned}$$

这是超递增的背包问题。

例 4.1 已知 1,3,7,13,26,65,119,267 是超递增序列。

$$1+3+7+13+26+65+119+267=501$$

取

$$m=523,\ w=467,\ \bar{w}=28$$

$$w\bar{w}=13076\equiv 1 \bmod 523$$

$$a_1\equiv 467\times 1\equiv 467\ (\mathrm{mod}\ 523)$$

$$a_2\equiv 467\times 3\equiv 355\ (\mathrm{mod}\ 523)$$

$$a_3\equiv 467\times 7\equiv 131\ (\mathrm{mod}\ 523)$$

$$a_4\equiv 467\times 13\equiv 318\ (\mathrm{mod}\ 523)$$

$$a_5\equiv 467\times 26\equiv 113\ (\mathrm{mod}\ 523)$$

$$a_6\equiv 467\times 65\equiv 21\ (\mathrm{mod}\ 523)$$

$$a_7\equiv 467\times 119\equiv 135\ (\mathrm{mod}\ 523)$$

$$a_8\equiv 467\times 267\equiv 215\ (\mathrm{mod}\ 523)$$

(467,355,131,318,113,21,135,215)作为公钥予以公布。对于明文 $m=10101100$ 加密得密文：

$$a_1+a_3+a_5+a_6=467+131+113+21=732$$

接收方收到密文 732 后，乘以 $\bar{w}=28$，mod 523 得

$$732\times 28=20496\equiv 99\ (\mathrm{mod}\ 523)$$

解超递增序列背包问题：

$$m_1+3m_2+7m_3+13m_4+26m_5+65m_6+119m_7+267m_8\equiv 99$$

所以

$$m_1=m_3=m_5=m_6=1,\ m_2=m_4=m_7=m_8=0$$

即得明文 10101100

Merkle 和 Hellman 的背包公钥密码提出后有一段有趣的佳话，他们提出：谁能破译它的背包公钥系统，可获奖励 50 美金。他们利用默科-赫尔曼变换将超递增序列的特性隐蔽起来，但得到的序列毕竟不是“正宗”的超递增序列，终究会露出“尾巴”。两年后被 Shamir 抓住并将它破译。Merkle 和 Hellman 又企图通过多次的默科-赫尔曼变换将它变得彻底些，即从超递增序列 $(a_0^{(0)},a_1^{(0)},\cdots,a_n^{(0)})$ 经过第 1 次 MH 变换得序列 $(a_0^{(1)},a_1^{(1)},\cdots,a_n^{(1)})$，再通过 MH 变换得序列 $(a_0^{(2)},a_1^{(2)},\cdots,a_n^{(2)})$，依此反复 m 次得序列 $(a_0^{(m)},a_1^{(m)},\cdots,a_n^{(m)})$，把它作为公钥公开。再次悬赏 100 美金。再过两年，新的破译方法解决了

低密度的背包公钥密码问题，使 MH 型背包公钥密码系统受到致命的打击。

4.3 Galois 域上的背包公钥密码

前面讲到的背包公钥密码系统由 Cooper 和 Patterson 将它推广到 $GF(p^n)$域。设

$$\begin{aligned}
b_1(x) &= b_{10} + b_{11}x + b_{12}x^2 + \cdots + b_{1k}x^k \\
b_2(x) &= b_{20} + b_{21}x + b_{22}x^2 + \cdots + b_{2k}x^k \\
&\vdots \\
b_n(x) &= b_{n0} + b_{n1}x + b_{n2}x^2 + \cdots + b_{nk}x^k
\end{aligned}$$

使得矩阵

$$\boldsymbol{B} = \begin{vmatrix} b_{10} & b_{11} & b_{12} & \cdots & b_{1k} \\ b_{20} & b_{21} & b_{22} & \cdots & b_{2k} \\ b_{30} & b_{31} & b_{32} & \cdots & b_{3k} \\ \vdots & & & & \\ b_{n0} & b_{n1} & b_{n2} & \cdots & b_{nk} \end{vmatrix} = (\boldsymbol{b}_0 \boldsymbol{b}_1 \cdots \boldsymbol{b}_k)$$

的每一列构成一超增序列。即 $\boldsymbol{b}_i = (b_{1i}, b_{2i}, \cdots, b_{ni})^T$ 而 $b_{1i}, b_{2i}, \cdots, b_{ni}$ 是一超递增序列，$i = 0, 1, \cdots, k$。令

$$M = \max_{i,j}\{b_{ij}\}$$

p 为大于 $2M$ 的素数。$m(x)$是 $GF(p)$域上的 $k+1$ 次不可化约的多项式，$w(x)$是方次不超过 k 的多项式。在 $GF(p)$域上 mod $m(x)$构成 $GF(p^{k+1})$域。在 $GF(p^{k+1})$域上令

$$a_j(x) \equiv w(x)b_j(x) \pmod{m(x)}$$

$$j = 1, 2, \cdots, n$$

将 $a_1(x), a_2(x), \cdots, a_n(x)$公开。对于 n 位 0,1 字符串的明文

$$m = m_1 m_2 \cdots m_n$$

加密步骤如下：

$$c(x) = m_1 a_1(x) + m_2 a_2(x) + m_3 a_3(x) + \cdots + m_n a_n(x)$$

解密办法是对于密文 $c(x) \pmod{m(x)}$乘以 $\bar{w}(x)$，$\bar{w}(x)$满足

$$w(x)\bar{w}(x) \equiv 1 \pmod{m(x)}$$

$$\begin{aligned}
w(x)c(x) &= m_1\bar{w}(x)a_1(x) + m_2\bar{w}(x)a_2(x) + \cdots + m_n\bar{w}(x)a_n(x) \\
&\equiv m_1 b_1(x) + m_2 b_2(x) + \cdots + m_n b_n(x) \pmod{m(x)}
\end{aligned}$$

比较等式两端某项系数便可求得 $m_1, m_2, \cdots, m_n$，这相当于解一超递增序列的背包问题。

也可以考虑分组解决。设 $n = rs$，将 $m_1, m_2, \cdots, m_n$ 分成 r 组，每组 s 个，分别由比较 r 项 $x^{i_1}, x^{i_2}, \cdots, x^{i_r}$ 的系数来决定，其中 $i_1, i_2, \cdots, i_r$ 可以随机选择。这只要矩阵 B 在第 $i_1, i_2, \cdots, i_r$ 列具有如下性质便可。首先每一列有且仅有 s 个非零元素，其余的 $n-s$ 个元素均为零；其次，若任一列中第 j 个元素非零，则其余 $r-1$ 列的第 j 个元素均为零。即 n 个非零元素覆盖了所有 n 行，在这第 $i_1, i_2, \cdots, i_r$ 列中每行有一个且仅有一个非零元素。除这 r 列外，矩阵中其余 $k+1-r$ 列的元素实际上是多余的，可以任意选取，只要不超过规定

的界即可。

如何选择素数 p 特别是不可化约的 $k+1$ 次多项式 $m(x)$，可以参考近世代数，这里不赘述。读者可构造一个 $n=20, r=4, s=5$ 的例子。例如取 $m(x)=x^{20}-2$，$w(x)=x^{10}+1$，$\bar{w}(x)=x^{10}-1$，并利用它进行加密和解密。

显然，这样的变形并没有改变 MH 背包公钥的实质，但第 $i_1, i_2, \cdots, i_r$ 列的任意选择，以及各列中 s 个非零超递增元素的排列的任意性，都给破译者增加了不少的麻烦。然而付出的代价也是巨大的，即公钥的量大大地膨胀了。

例 4.2 取 $p=101, k=19$，4 个超递增序列为

$$\{1,3,7,12,26\}, \quad \{1,4,7,14,27\}$$
$$\{2,5,9,18,36\}, \quad \{1,2,4,8,16\}$$

这 4 个超递增序列位于 x^2, x^7, x^{11}, x^{14}，见 B 表(表 4.1)。其他所有系数均为 0～100 间的随机数。

取 $GF(p)$域上不可化约的多项式

$$p(x) = x^{20} - 2$$

又取

$$m(x)=x^{10}+1, \quad m^{-1}(x)=x^{10}-1$$
$$\tilde{b}_i(x) \equiv b_i(x)\ m(x) \mod (p, p(x))$$

见 $\tilde{B}$ 表(表 4.2)。可见公钥量大大地膨胀了。

表 4.1

B 表

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
b_1	12	77	0	70	12	92	93	0	92	4	1	0	19	92	1	10	6	70	50	97
b_2	21	5	0	71	80	15	55	16	45	51	58	0	97	41	0	38	1	60	85	4
b_3	43	46	0	22	78	99	69	0	68	64	15	0	9	11	3	87	18	80	54	34
b_4	45	67	27	76	93	26	92	0	33	0	94	0	86	9	0	1	91	33	32	81
b_5	0	99	1	41	17	0	45	0	29	15	61	0	40	14	0	92	15	38	14	13
b_6	7	66	0	7	22	44	70	0	54	30	16	2	98	24	0	28	30	34	55	57
b_7	61	42	0	2	8	37	50	2	18	38	5	0	74	80	0	58	42	20	15	43
b_8	68	69	4	68	92	62	87	0	29	68	20	0	13	66	0	20	75	34	77	53
b_9	22	99	0	0	63	75	38	0	27	2	22	0	72	33	12	31	60	13	94	14
b_{10}	98	12	0	30	17	73	42	0	0	41	95	18	3	62	0	96	91	74	37	94
b_{11}	47	21	0	26	67	87	26	1	9	50	40	0	41	57	0	13	58	56	84	6
b_{12}	13	7	0	89	69	44	69	0	6	29	94	5	22	97	0	89	51	7	9	68
b_{13}	8	30	7	13	56	37	3	0	90	10	76	0	92	22	0	93	70	84	83	18
b_{14}	48	38	0	99	12	99	91	0	52	77	56	0	81	93	7	22	19	30	15	16
b_{15}	86	78	0	25	93	82	9	4	86	41	35	0	13	15	0	14	28	89	92	81
b_{16}	28	26	14	37	69	13	11	0	43	44	14	0	68	29	0	50	92	21	36	76
b_{17}	80	14	0	90	81	59	48	0	50	0	37	0	81	57	26	99	64	43	40	41
b_{18}	79	98	0	10	26	26	59	8	52	76	91	0	3	69	0	71	64	3	91	93
b_{19}	55	33	0	14	92	19	22	0	23	40	15	36	32	91	0	28	94	64	69	64
b_{20}	48	46	0	38	30	72	66	0	59	75	94	9	3	75	0	85	56	36	91	95

表 4.2

B 表	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$\widetilde{b_1}$	14	77	38	52	40	12	4	39	91	97	13	77	19	61	13	1	49	70	11	0
$\widetilde{b_2}$	36	5	93	52	80	91	57	35	14	59	79	5	97	11	80	53	56	76	29	55
$\widetilde{b_3}$	73	46	18	44	84	71	4	59	75	31	58	46	9	33	81	85	87	80	21	98
$\widetilde{b_4}$	31	67	98	94	93	28	72	66	97	61	38	67	12	85	93	27	82	33	65	81
$\widetilde{b_5}$	21	99	81	69	17	83	75	76	57	41	61	99	41	55	17	92	60	38	43	28
$\widetilde{b_6}$	39	70	97	55	22	100	29	68	63	43	23	68	98	31	22	72	100	34	8	87
$\widetilde{b_7}$	71	42	47	61	8	72	33	42	48	23	66	42	74	82	8	14	92	22	33	81
$\widetilde{b_8}$	7	69	30	99	92	1	35	68	82	73	88	69	17	33	92	82	61	34	5	20
$\widetilde{b_9}$	66	99	43	66	87	26	57	26	13	30	44	99	72	33	75	5	98	13	20	16
$\widetilde{b_{10}}$	86	48	6	53	17	63	22	47	71	27	92	30	3	92	17	68	32	74	37	34
$\widetilde{b_{11}}$	26	21	82	39	67	12	41	12	76	62	87	21	41	83	67	100	84	57	93	56
$\widetilde{b_{12}}$	100	17	44	81	69	20	70	14	24	64	6	12	22	87	69	32	19	7	15	97
$\widetilde{b_{13}}$	59	30	90	57	56	21	42	67	54	46	84	30	99	85	56	29	73	84	72	28
$\widetilde{b_{14}}$	59	38	61	83	26	42	28	60	82	8	3	38	81	91	19	20	9	30	67	93
$\widetilde{b_{15}}$	55	78	26	55	93	9	65	81	68	1	20	78	13	40	93	96	37	93	77	21
$\widetilde{b_{16}}$	56	26	49	95	69	12	94	42	14	95	42	26	82	66	69	63	2	21	79	19
$\widetilde{b_{17}}$	53	14	61	2	32	55	75	86	29	82	16	14	81	46	6	57	11	43	90	41
$\widetilde{b_{18}}$	59	98	6	46	26	67	86	14	32	60	69	98	3	79	26	97	22	11	42	68
$\widetilde{b_{19}}$	85	4	64	95	92	75	8	27	60	67	70	69	32	4	92	47	15	64	92	3
$\widetilde{b_{20}}$	34	64	6	87	30	40	77	72	39	63	41	55	31	13	30	56	21	36	49	69

若 $m=10111\quad 01100\quad 10010\quad 01100$

$$\begin{aligned} c(x) &= \widetilde{b_1}(x)+\widetilde{b_2}(x)+\widetilde{b_4}(x)+\widetilde{b_5}(x)+\widetilde{b_7}(x)+\widetilde{b_8}(x)+\widetilde{b_{11}}(x)+\widetilde{b_{14}}(x)\\ &\quad+\widetilde{b_{17}}(x)+\widetilde{b_{18}}(x)\\ &=10+66x+17x^2+85x^3+55x^4+38x^5+49x^6+17x^7+63x^8\\ &\quad+33x^9+95x^{10}+66x^{11}+75x^{12}+42x^{13}+18x^{14}+70x^{15}\\ &\quad+x^{16}+14x^{17}+96x^{18}+61x^{19} \end{aligned}$$

解密过程是对 $c(x)\bmod(p,p(x))$ 乘以 $m^{-1}(x)=x^{10}-1$ 得

$$\begin{aligned} E(x) &= 79+66x+32x^2+100x^3+82x^4+x^5+54x^6+11x^7+28x^8\\ &\quad+90x^9+16x^{10}+0x^{11}+42x^{12}+43x^{13}+37x^{14}+69x^{15}\\ &\quad+48x^{16}+3x^{17}+68x^{18}+73x^{19} \end{aligned}$$

从 B 表可知，由 x^2 项系数有

$$m_3+4m_8+7m_{13}+14m_{16}+27m_4=32$$

故 $$m_4=m_8=m_5=1,\ m_{16}=m_{13}=0$$

由 x^7 系数有 $$m_{11}+2m_1+4m_{15}+8m_{18}+16m_2=11$$

故 $$m_1=m_{18}=m_{11}=1,\ m_{15}=m_{18}=m_2=0$$

由 x^{11} 系数有

$$2m_6+5m_{12}+9m_{20}+18m_{10}+36m_{19}=0$$

故 $$m_6=m_{10}=m_{12}=m_{19}=m_{20}=0$$

由 x^{14} 系数有

$$m_1 + 3m_3 + 7m_{14} + 12m_9 + 26m_{17} = 37$$

$$m_1 = m_3 = m_{14} = m_{17} = 1,\ m_9 = m_{14} = 0$$

故

$$m = 10111\quad 01100\quad 10010\quad 01100$$

4.4 RSA 公钥密码

在 Diffie 和 Hellman 提出公钥的设想后两年，先后由 Merkle 和 Hellman 提出了 MH 背包公钥密码，Rivest、Shamir 和 Adleman 联合提出的简称为 RSA 公钥密码系统。在时间上，RSA 和 MH 背包公钥系统基本上同时提出，前面已说明 MH 背包公钥已寿终正寝，但 RSA 到目前为止仍不失为最有希望的一种公钥密码，目前已成为公钥密码的国际标准。RSA 的基础是数论的欧拉定理，它的安全性依赖于大数的因数分解的困难性。

4.4.1 欧拉定理

定理 4.1 若整数 a 和 m 互素，则

$$a^{\varphi(m)} \equiv 1 \pmod{m}$$

其中 $\varphi(m)$ 是比 m 小但与 m 互素的正整数个数。

证明 设 $\varphi(m)=k$。又设 $r_1, r_2, \cdots, r_k$ 是小于 m 并与 m 互素的数，且由于 a 是与 m 互素的数，则 $ar_1, ar_2, \cdots, ar_k$ 也和 m 互素且两两不同。若

$$ar_i \equiv ar_j \pmod{m}$$

则根据数论定理，因 a 和 m 互素，所以存在 $\bar{a}$ 满足

$$\bar{a}a \equiv 1 \pmod{m}$$

所以

$$\bar{a}ar_i \equiv \bar{a}ar_j \pmod{m}$$

即

$$r_i \equiv r_j \pmod{m}$$

与假定矛盾，所以

$$a^k r_1 r_2 \cdots r_k \equiv r_1 r_2 \cdots r_k \pmod{m}$$

但 $r_1 r_2 \cdots r_k$ 和 m 互素，故

$$a^k \equiv 1 \pmod{m}$$

4.4.2 RSA 加密算法

RSA 加密算法的过程是

(1) 取两个素数 p 和 q(保密)。

(2) 计算 $n=pq$(公开)，$\varphi(n)=(p-1)(q-1)$(保密)。

(3) 随机选取整数 e，满足 $\gcd(e,\varphi(n))=1$(公开)。

(4) 计算 d，满足 $de \equiv 1 \pmod{\varphi(n)}$(保密)。

利用 RSA 加密第一步需将明文数字化，并取长度小于 $\log_2 n$ 位的数字作明文块。

加密算法　　$c=E(m)\equiv m^e \pmod n$

解密算法　　$D(c)\equiv c^d \pmod n$

下面证明解密过程是正确的。

证明　对于任何 k 及任何 $m(<n)$，恒有

$$m^{k\varphi(n)+1}\equiv m \pmod n$$

若 $(m,n)=1$，则由欧拉定理可知

$$m^{\varphi(n)}\equiv 1 \pmod n$$

若 $(m,n)>1$，由于 $n=pq$，故 (m,n) 必含 p,q 之一。设 $(m,n)=p$，或 $m=cp$，$1\leqslant c<q$，由欧拉定理

$$m^{\varphi(q)}\equiv 1 \pmod q$$

因此，对任何 k，总有

$$m^{k(q-1)}\equiv 1 \pmod q$$

$$m^{k(p-1)(q-1)}\equiv (1)^{k(p-1)}\equiv 1 \pmod q$$

即　　$m^{k\varphi(n)}\equiv 1 \pmod q$

或　　$1=m^{k\varphi(n)}+hq$

其中 h 是某个整数。由假定 $m=cp$。

所以　　$m=m^{k\varphi(n)+1}+hcpq$

这就证明了

$$m\equiv m^{k\varphi(n)+1} \pmod n$$

例 4.3　设

$$p=43,\ q=59,\ n=pq=43\times 59=2539$$

$$\varphi(n)=42\times 58=2436,\ 取\ e=13$$

解方程　$de\equiv 1 \pmod{2436}$

$$2436=13\times 187+5,\quad 13=2\times 5+3$$

$$5=3+2,\quad 3=2+1$$

所以

$$1=3-2,\quad 2=5-3,\quad 3=13-2\times 5$$

$$5=2436-13\times 187$$

所以

$$\begin{aligned}1&=3-2=3-(5-3)=2\times 3-5\\&=2(13-2\times 5)-5\\&=2\times 13-5\times 5\\&=2\times 13-5(2436-13\times 187)\\&=937\times 13-5\times 2436\end{aligned}$$

即　　$937\times 13\equiv 1 \pmod{2436}$

取　　$e=13,\ d=937$

若有明文：

public key encryptions

先将明文分块为：

pu　bl　ic　ke　ye　nc　ry　pt　io　ns

如利用英文字母表的顺序：即 a 为 00，b 为 01，c=2，…，y 为 24，z 为 25，将明文数字化得：

1520 0111 0802 1004 2404

1302 1724 1519 0814 1418

利用加密得密文：

0095 1648 1410 1299 1365

1379 2333 2132 1751 1289

4.4.3 RSA 的安全性讨论

若 $n=pq$ 被因子分解，则 RSA 便被击破。

因为若 p,q 已知，则 $\varphi(n)=(p-1)(q-1)$便可算出。解密密钥 d 关于 e 满足：

$$de \equiv 1 \pmod{\varphi(n)}$$

故 d 便也不难求得。因此 RSA 的安全依赖于因子分解的困难性。目前因子分解速度最快的方法，其时间复杂性为

$$\exp(\mathrm{sqrt}(\ln(n)\ln\ln(n)))$$

其中 $\exp(x)$表示 e^x，sqrt 表示$\sqrt{x}$，故 $\exp(\mathrm{sqrt}(\ln(n)\ln\ln(n)))$表示 $e^{\sqrt{\ln(n)\ln\ln(n)}}$

Rivest、Shamir 和 Adleman 建议取 p 和 q 为 100 位十进制数($\approx 2^{332}$)，这样 n 为 200 位十进制数。要分解 200 位的十进制数，按每秒 10^7 次运算的超高速电子计算机，也要 10^8 年。近来对大数分解算法的研究引起了数学工作者的重视。1990 年有 150 位的特殊类型的数(第 9 个费尔玛(Fermat)数)已被成功分解。最新记录是 129 位十进制数在网络上通过分布计算被分解成功。估计对 200 位十进制数的因数分解，在亿次机上要进行 55 万年。

若 n 被分解成功，则 RSA 便被攻破。虽然还不能证明对 RSA 攻击的难度和分解 n 相当，故对 RSA 的攻击的困难程度不比大数分解更难。当然，若从求 $\varphi(n)$入手对 RSA 进行攻击，它的难度和分解 n 相当，但还没有找到比因数分解 n 更好的攻击方法。已知 n，若求得 $\varphi(n)$，则 p 和 q 可以求得。因为

$$\varphi(n)=(p-1)(q-1)=pq-(p+q)+1$$

及

$$(p-q)^2=(p+q)^2-4pq$$

所以

$$\begin{cases} p+q=n-\varphi(n)+1 \\ p-q=\sqrt{(p+q)^2-4n} \end{cases}$$

这是关于 p 和 q 的联立方程。

为了使 $pq=n$ 因数分解不容易实现，后面将提到强素数概念。也就是说 p,q 不仅仅是素数还要求是强素数。同样 e 和 d 也要有些条件，这里先提出一个要求 q 和 p 相差要比较大。

如若不然，$q-p=k$，$q=p+k$。则

$$(q+p)^2-(q-p)^2=4pq=n$$

$$(q+p)^2=n+k^2$$

$$(q+p)=\sqrt{n+k^2}$$

和 $$q-p=k$$

可联立解出 p 和 q。

例如 $n=5007958289$，$k=200$。则

$$(q+p)^2-4n+(200)^2-20031833166+40000$$

$$=20031873166=(141534)^2$$

$$q\quad p=200$$

所以 $$q=70867$$

$$p=70667$$

e 和 d 也要有些要求，比如 e 不可太小，否则不安全。比如 C 欲向 A、B 二人送去信息 m，A、B 二人的 $e_A=e_B=e$，n_A、n_B 无公因数，因为若 n_A，n_B 有公因数，则求 n_A，n_B 的公因数便可将 n_A 和 n_B 因数分解。又若 $e_A=e_B=e$，则有密文

$$c_1\equiv m^e \bmod n_A,\quad c_2\equiv m^e \bmod n_B$$

利用中国剩余定理可求得解

$$m^e\equiv c \bmod (n_An_B)$$

比如 $e=3$，只要对 c 开立方便可求得 m。

4.4.4 数字签名

RSA 公钥密码还有一个十分重要的性质，即可作数字签名，其重要性不亚于公钥本身。

RSA 有两个参数 e 和 d，$ed\equiv 1 \bmod \varphi(n)$。若将 e 公开，则 d 保密。

密码能保证信息 m 不被非法窃取，但不能防止发信方抵赖，也不能防止收信方作假。因为

$$c\equiv m \bmod n$$

其中 e 和 n 是公开的。若发生双方纠纷，是发信方抵赖？还是收信方伪造？很难判断。

先假定信息 m 无需保密，但必需保证发信方不能否认或抵赖。由 RSA 系统可如下执行：

S1. $S\equiv m^{d_A} \bmod n_A$

S2. A 将 (m,S) 同时寄给 B。

B 收到后作

$$\bar{m}=S^{e_A} \bmod n_A$$

B 将 $\bar{m}$ 与 m 比较，若一致则确认是 A 寄来的，并将 (m,s) 放到数据库里。若发生纠纷，B 取出 (m,s) 作为凭证，由于 S 是用到 A 的密钥 d_A，是 A 的签名，他人无法伪造，A 无法抵赖。

$$S^{e_A} \bmod n_A\equiv (m^{d_A})^{e_A} \bmod n_A\equiv m^{e_Ad_A} \bmod n_A$$

但 $$e^{e_Ad_A}\equiv 1 \bmod \varphi(n_A)$$

$$e_Ad_A=l\varphi(n_A)+1$$

$$m^{\varphi(n_A)}\equiv 1 \bmod n_A$$

所以 $$S^{e_A} \bmod n_A\equiv m$$

其实 A 向 B 直接寄去亦可，不必附上 m，但这样做将使信息 m 不保密。还可以将保密与签名毕其功于一役，办法如下：

S1. A 计算

$$S \equiv m^{d_A} \bmod n_A$$

S2. A 将 S 用 B 的公钥加密得密文

$$C \equiv S^{e_B} \bmod n_B$$

S3. A 将 C 寄给 B

B 收到后解密如下：

S1. 对 C 用 B 的密钥解密得

$$C^{d_B} \bmod n_B \equiv S^{e_B d_B} \bmod n_B \equiv S$$

S2. 对用 A 的公钥再解密

$$S^{e_A} \bmod n_A \equiv m^{d_A e_A} \bmod n_A \equiv m$$

4.4.5 数字签名的注意事项

上一节讲到将保密与签名置于一役的方法即作

$$C \equiv (m^{d_A} \bmod n_A)^{e_B} \bmod n_B$$

必须考虑到 $n_A > n_B$ 吗？如若 $n_A > n_B$，先作

$$S \equiv m^{d_A} \bmod n_A$$

其结果有可能 $> n_B$。这时候应该将 S 分开几段加密。否则 $\bar{S} \equiv S \bmod n_B$，再 $\bar{S}$ 用 B 的公开密钥加密便会出错。还有一种补救的办法若 $n_A > n_B$ 时，A 可先用 B 的公钥加密，后再用自己的私钥签名，即

$$C \equiv (m^{e_B} \bmod n_B)^{d_A} \bmod n_A$$

B 收到后顺序也应该颠倒过来，即

$$m \equiv ((C)^{e_A} \bmod n_A)^{d_B} \bmod n_B$$

当然，这样做的结果可能隐藏着不安全的因素，尽管这样的可能性不是太大。

4.4.6 强素数

为了使 $n = pq$ 不易于被因数分解，也就是说躲开若干因子分解的有效算法，比如 $p-1$ 因数分解法。要求 p 和 q 不仅是素数，而且是强素数，强素数的概念如下：

素数 p 称为是强素数，若满足如下的条件：

(1) $p-1$ 和 $p+1$ 分别有大素数因子 p_1 和 p_1'；

(2) $p_1 - 1$ 有大素数因子 q_1。

4.5 Rabin 公钥密码

Rabin 公钥从表面上似乎是 RSA 公钥 $e=2$ 的一种特例，其实不然。RSA 要求 $(e, \varphi(n)) = 1$，但 $e=2$ 时，由 $\varphi(n) = (p-1)(q-1)$ 必是偶数，则条件 $(e, \varphi(n)) = 1$ 便不满足。

前面讨论 RSA 时，已分析，若将 n 因数分解为 $n=pq$，则 RSA 便被破译。也就是说 RSA 的抗攻击的能力不超过大数的因子分解，也没找到对 RSA 的其他攻击方法。猜想 RSA 的抗攻击能力与 n 的因子分解相当，但又不能证明。Rabin 提出 $e=2$。即

$$C \equiv m^2 \bmod n$$

由于 $n=pq$，故

$$C \equiv m^2 \bmod p, \quad C \equiv m^2 \bmod q$$

令 QR_n 表示模 n 的平方剩余集，即

$$QR_n \triangleq \{a \quad \exists x \in Z, \text{使 } x^2 \equiv a \bmod n\}$$

即存在 $x \in Z$ 满足

$$x^2 \equiv a \bmod n$$

称 a 属于 QR_n，否则 $a \notin QR_n$，或称 Q 为非平方剩余。非平方剩余集合用 NQR_n 表示它。

引理 4.1 若 p 为奇素数，且 p 除不尽 a，则

$$x^2 \equiv a \pmod p$$

或无解或有两个模 p 不同余的解。

证明 若 $x^2 \equiv a \pmod p$ 有解 $x_1(0<x_1<p)$，不难验证 $-x_1$ 也满足这个同余方程，而且 x_1 和 $-x_1$ 不是模 p 同余，若不然，$x_1 \equiv -x_1 \pmod p$，则

$$2x_1 \equiv 0 \pmod p$$

这与 $(0<x_1<p)$，且 p 是奇素数的假定相矛盾。

下证只有这两个、别无其他解。若

$$x_2^2 \equiv a \pmod p$$

则

$$x_1^2 \equiv x_2^2 \pmod p$$

所以

$$x_1^2 - x_2^2 \equiv (x_1 - x_2)(x_1 + x_2) \equiv 0 \pmod p$$

所以

$$p \mid (x_1 - x_2) \text{ 或 } p \mid (x_1 + x_2)$$

这就证明了

$$x_1 \equiv x_2 \pmod p \text{ 或 } x_1 \equiv -x_2 \pmod p$$

定理 4.2 若 p 是奇素数，则整数 $1,2,\cdots,p-1$ 中正好有 $(p-1)/2$ 个属于 QR_p，其余 $(p-1)/2$ 个属于非平方剩余集合 NQR_p。

证明 计算 $1,2,\cdots,p-1$ 的平方模 p 的最小正剩余。因为只有 $p-1$ 个非零的剩余，且 $x^2 \equiv a \pmod p$ 的解的数目或为零或有不同余的两个。事实上正好有 $(p-1)/2$ 个模 p 的平方剩余。即存在 $(p-1)/2$ 个 a，满足 $1 \leqslant a \leqslant p-1$，使得 $a \in QR_p$，其余的 $(p-1)/2$ 个属于非平方剩余 NQR_p。

例 4.4 $p=5$，$1^2 \equiv 1 \pmod 5$，$2^2 \equiv 4 \pmod 5$，$3^2 \equiv 4 \pmod 5$，$4^2 \equiv 1 \pmod 5$，所以 1 和 4 属于 QR_5，2 和 3 则属于非平方剩余 NQR_5。

总之，求 n 和 x_1+x_2（或 x_1-x_2）的最大公因子便是 p（或 q）。

若 p 和 q 都 mod 4 与 3 同余，则

$$x^e \equiv c \bmod p \text{ 和 } x^e \equiv c \bmod q$$

求平方根计算比较容易。例如

$p\equiv 3 \bmod 4$，则 $p+1=4k$，或$\frac{1}{4}(p+1)$是一整数。由于 c 是 mod p 平方剩余，故

$$c^{\varphi(p)/2} \equiv 1 \bmod p$$

$$c^{\frac{p-1}{2}} \equiv 1 \bmod p$$

但

$$\left(c^{\frac{(p-1)^2}{4}}\right)^2 \equiv c^{\frac{p+1}{2}} \equiv c^{\frac{p-1}{2}} \cdot c \equiv c \bmod p$$

新 x 可以是

$$c^{\frac{p+1}{4}} \quad 或 \quad p-c^{\frac{p+1}{4}}$$

同理 x 可以是

$$c^{\frac{q+1}{4}} \quad 或 \quad q-c^{\frac{q+1}{4}}$$

$n=pq$，所以这些只能是平方根。

因而 Rabin 的建议是取两个秘密素数 p 和 q，p 和 q mod 4 都和 3 同余。公布 $n=pq$，对于明文 $m\in(0,n-1)$

加密算法：$c\equiv m^2 \bmod n$

解密算法：mod n 计算

$$(c)^{\frac{p+1}{4}},\quad p-(c)^{(p+1)/4},\quad (c)^{\frac{q+1}{4}},\quad q-(c)^{\frac{q+1}{4}}$$

当 m 是某种语言的文章字段时，只能利用语言本身来完成挑选其中有意义的一个。

4.6 ElGamal 公钥密码

4.6.1 加密算法

ElGamal 基于求离散对数的困难性，是 RSA 以后比较有希望的一个公钥密码。系统提供一大素数 p 和 $GF(p)$上的本原元素 g。对每一个用户 A 可选择

$$x_A \in [0,1,2,\cdots,p-1]$$

计算

$$y_A \equiv g^{x_A} \bmod p$$

将 y_A 公开，x_A 保密，由 A 自己掌握。

若 A 欲与 B 保密通信，设明文是 m，$m\in[0,1,2,\cdots,p-1]$

S1. A 找出 B 的公钥

$$y_B = g^{x_B} \bmod p$$

S2. A 任意选随机数 $x\in[0,1,2,\cdots,p-1]$，A 计算

$$c_1 \equiv (g)^x \bmod p$$

S3. A 计算：$K\equiv(y_B)^x \bmod p\equiv(g)^{x_B x} \bmod p$

求

$$c_2 \equiv Km \bmod p$$

S4. A 将(c_1,c_2)作为密文寄给 B。

B 收到密文(c_1,c_2)后，解密如下：

S1. B 用自己的密钥 x_B 计算

$$K\equiv (y_B)^x \bmod p \equiv g^{xx_B} \bmod p$$
$$\equiv (c_1)^{x_B} \bmod p$$

S2. B 计算：$K^{-1} \bmod p$，

S3. 求 $m\equiv K^{-1}c_2 \bmod p$。

4.6.2 举例

设 $p=11,g=7$，在 $GF(11)$ 上有

$7^1=1,7^1\equiv 7,7^2=5,7^3=2,7^4=3,7^5=10,7^6=4,7^7=6,7^8=9,7^9=8,7^{10}=1$

g 是 $GF(11)$ 上的本原元素。

设 A 的私钥 $x_A=3$，公钥 $y_A\equiv 2$；B 的私钥 $x_B=5$，公钥 $y_B=10$。

假定 A 欲将信息 $m=6$ 保密寄给 B。A 取 $x=7$，A 计算：

$$c_1 \equiv g^7 \bmod 11 \equiv 6$$
$$K \equiv (10)^7 \bmod 11 \equiv 10$$
$$c_2 \equiv Km \bmod 11 \equiv 60 \equiv 5 \bmod 11$$

A 将(6,5)作为密文寄给 B。B 收到后计算

$$K \equiv (g)^5 \bmod 11 \equiv 10$$
$$K^{-1} \equiv g^{10-5} \equiv g^5 \equiv 10$$
$$m \equiv K^{-1}c_2 \equiv 50 \bmod 11 \equiv 6$$

故恢复 $m\equiv 6$。

4.7 Chor-Rivest 背包公钥密码

4.7.1 理论基础

Benny Chor 和 Ronald L. Rivest 于 1985 年发表一篇名为《A Knapsack Type Public Key Cryptosystem Based on Arithmetic in Finite Fields》的文章，其中提出一种新型的背包公钥系统，以前的低密度攻击法对于该系统失去了作用。

背包问题的解不仅困难，还可能不惟一。但超递增序列的背包问题没有这个问题。(Chor-Rivest)背包公钥系统与之相应的依据是布斯-邹拉(Bose-Chowla)定理，定理如下：

定理 4.3(布斯-邹拉) 设 p 是素数，n 是不低于 2 的整数，则存在序列

$$A=\{a_i \mid 0\leqslant i\leqslant p-1,\quad a_i\in Z\}$$

使得

(a) $1 \leqslant a_i \leqslant p^n - 1$，　$i = 0,1,2,\cdots,p-1$

(b) $\xi = (x_0, x_1, \cdots, x_{p-1})$，$y = (y_0, y_1, \cdots, y_{p-1})$是两个坐标为非负整数的不同向量，且若

$$\sum_{i=0}^{p-1} x_i = \sum_{j=0}^{p-1} y_j \leqslant n$$

则

$$\sum_{i=0}^{p-1} a_i x_i \neq \sum_{j=0}^{p-1} a_j y_j$$

成立。

证明　证明是构造出序列 A。

通过选择素数 p 及在 $GF(p)$域上不可化约多项式 $I(x)$，构造域 $GF(p^n)$。设 $t \in GF(p^n)$，且是 $GF(p)$域上 n 次最小多项式的根。又设 $g \in GF(p^n)$是 $GF(p^n)$上乘法群的本原元素。

$$t + GF(p) = \{t + i \in GF(p^n) \mid i = 0,1,2,\cdots,p-1\}$$

显然

$$t + GF(p) \subseteq GF(p^n)$$

设

$$a_i = \log_g(t+i), \quad i = 0,1,2,\cdots,p-1$$

则$\{a_i\}_{i=0}^{p-1}$便是所求的序列。因为 g 是 $GF(p^n)$的乘法群的母元素，所以 g 有 $p^n - 1$ 个不同指数，故 a_i 的全体都在$[1, p^n - 1]$区间上。

由于$(x_0, x_1, \cdots, x_{p-1})$和$(y_0, y_1, \cdots, y_{p-1})$是两个坐标为非负整数的不同向量，而且 $\sum_i x_i = \sum_j y_j \leqslant n$。

若

$$\sum_{i=0}^{p-1} a_i x_i = \sum_{j=1}^{p-1} a_j y_j$$

则

$$g^{\sum_i a_i x_i} = g^{\sum_j a_j y_j}$$

所以

$$\prod_i g^{a_i x_i} = \prod_j g^{a_j x_j}$$

但

$$g^{a_i} = t + i$$

所以

$$(t+i_1)^{x_1}(t+i_2)^{x_2}\cdots(t+i_h)^{x_h} = (t+j_1)^{y_1}(t+j_2)^{y_2}\cdots(t+j_k)^{y_k}$$

其中$(i_1, i_2, \cdots, i_h)$和$(j_1, j_2, \cdots, j_k)$是来自$\{0,1,\cdots,p-1\}$的非空集合。由于

$$\sum_i x_i = \sum_j y_j \leqslant n$$

故每个集合至多 n 个元素。

由于向量 $\xi \neq \eta$，所以

$$\prod_{p=1}^{h}(t+i_p)^{x_p} - \prod_{q=1}^{k}(t+j_q)^{y_q}$$

是系数在 $GF(p)$上的非零多项式，而且方次小于等于 $n-1$。首项系数为 1 相消，故至少

降 1 次方。这与 t 是 $GF(p)$ 上 n 次最小多项式的根的假定相矛盾。

4.7.2 Chor-Rivest 密码

建立 Chor-Rivest 密码系统的第一步是挑选 p 和 n，进而在 $GF(p^n)$ 上选取方次为 n 的 $t\in GF(p^n)$ 和 $g\in GF(p^n)$。t 和 g 的选择有较大的自由度。根据布斯-邹拉(Bose-Chowla)定理计算 $GF(p)+t$ 中 p 个元素以 g 为底的对数。现将密码系统产生的过程叙述如下：

(1) 选择素数 p 及整数 $n\leqslant p$，使得在 $GF(p^n)$ 中的离散对数能有效地计算。

(2) 通过选择在 $GF(p)$ 上不可化约的 n 次多项式 $I(x)$ 构造 $GF(p^n)$，也就是由 mod $I(x)$ 来表示 $GF(p^n)$。

(3) 随机选择 $t\in GF(p^n)$ 使得 t 是 $GF(p)$ 上 n 次极小多项式的根。

(4) 随机选择 $g\in GF(p^n)$，要求它是 $GF(p^n)$ 的乘法群的母元素。

(5) 计算 $a_i=\log_g(t+i)$，　$i=0,1,2,\cdots,p-1$。

(6) 随机选择 p 个元素的置换 π，使得 $\pi:a_i\to b_i=a_{\pi(i)}$，　$i=0,1,2,\cdots,p-1$。

(7) 选一随机数 d，$0\leqslant d\leqslant p^n-2$，令

$$c_i=b_i+d,\quad i=0,1,2,\cdots,p-1$$

(8) 将 $c_0,c_1,c_2,\cdots,c_{p-1}$ 及 p,n 公开。

(9) t,g,π,d 保密。

加密算法：

Chor-Rivest 系统适合于对权正好为 n 的 p 位 0,1 字符串明文块进行加密。因此，首先要建立表达[0,$C(p,n)$]中整数的位串到长度 p，权 n 的变换。

对权 n 的位串 m 加密如下：

$$E(m)=c_{i_1}+c_{i_2}+\cdots+c_{i_n}(\mathrm{mod}\ (p^n-1))$$

其中 $i_1,i_2,\cdots,i_n$ 是 n 个 1 所在的位。

解密算法：

(1) 令 $r(t)\equiv t^n(\mathrm{mod}\ I(t))$，$r(t)$ 是方次 $\leqslant n-1$ 的多项式。

(2) 设 $c=E(m)$，计算 $c'\equiv c-nd\ (\mathrm{mod}\ (p^n-1))$。

(3) 计算 $p(t)\equiv g^{c'}(\mathrm{mod}\ I(t))$，$p(t)$ 为 t 的 $n-1$ 次方的多项式。

(4) 计算 $d(t)=t^n+p(t)-r(t)$，$d(t)$ 是系数在 $GF(p)$ 上 t 的 n 次多项式。

(5) 将 $d(t)$ 因子分解得

$$d(t)=(t+i_1)(t+i_2)\cdots(t+i_n)$$

这是因为

$$\begin{aligned}g^{E(m)-nd}&=g^{c_{i_1}}g^{c_{i_2}}\cdots g^{c_{i_n}}g^{-nd}\\&=g^{(c_{i_1}-d)}g^{(c_{i_2}-d)}\cdots g^{(c_{i_n}-d)}\\&=g^{b_{i_1}}g^{b_{i_2}}\cdots g^{b_{i_n}}\end{aligned}$$

其中 b_{i_j} 是 $t+GF(p)$ 中某些元素的离散对数，所以考虑 $d(t)$ 是这样的一些线性因子的积。

(6) 通过置换找到 n 个根 i_j，由 π^{-1} 找到原来为 1 的坐标。

4.7.3 举例

取 $p=5, n=3$，不可化约多项式 $I(x)=x^3+x+1$，在 $GF(5^3)$ 域上有

$$x^3 = -x-1$$
$$x^4 = -x^2-x$$
$$x^5 = -x^3-x^2 = -x^2+x+1$$
$$x^{10} = (-x^2+x+1)^2 = x^4-2x^3-x^2+2x+1$$
$$= -x^2-x+2x+2-x^2+2x+1 = -2x^2+3x+3$$
$$x^{15} = (-2x^2+3x+3)(-x^2+x+1) = 2x^4-x^2+x+3$$
$$= -2x^2-2x-2x^2+x+3 = x^2-x+3$$
$$x^{20} = (-2x^2+3x+3)^2 = 4x^4-2x^3-3x^2+3x+4$$
$$= -4x^2-4x+2x+2-3x^2+3x+4$$
$$= -2x^2-4x+1$$
$$x^{30} = (-2x^2+3x+3)(-2x^2-4x+1)$$
$$= 4x^4+2x^3-2x^2-9x+3$$
$$= -4x^2-4x-2x-2-4x+3$$
$$= x^2+1$$
$$x^{34} = (x^2+1)(-x^2-x) = -x^4-x^3-x^2-x$$
$$= x^2+x+x+1-x^2-x = x+1$$

因为 $x^{31} = x^3+x = -x-1+x = -1$

所以 $x^{62} = 1$

令 $g = 2x^2+x+2$

$$g^2 = (2x^2+x+2)^2 = 4x^4+4x^3+4x^2+4x+4$$
$$= -4x^2-4x-4x-4+4x^2+4x+4$$
$$= -4x = x$$

故有 $x^{10} = g^{60} = x^2+1$

$$g^{62} = -1$$
$$g^{124} = 1$$
$$g^{68} = g^{62} \cdot g^6 = -x^3 = x+1$$
$$g^{98} = g^{68} \cdot g^{30} = (x+1)(x^2-x+3)$$
$$= x^3+2x+3 = -x-1+2x+3 = x+2$$
$$g^{86} = g^{62} \cdot g^{24} = -(x^{12}) = -(-2x^2+3x+3)x^2$$
$$= 2x^4-3x^3-3x^2 = -2x^2-2x+3x+3-3x^2$$
$$= x+3$$

$$g^{109} = g^{98} \cdot g^{11} = (x+2)x^5(2x^2+x+2)$$
$$= (x+2)(-x^2+x+1)(2x^2+x+2)$$
$$= (-x^3-x^2+3x+2)(2x^2+x+2)$$
$$= (-x^2+4x+3)(2x^2+x+2)$$
$$= -2x^4+7x^3+3x^2+x+1$$
$$= 2x^2+2x-2x-2+3x^2+x+1 = x-1 = x+4$$

即 $$g^2 = x, \quad g^{68} = x+1, \quad g^{98} = x+2$$
$$g^{86} = x+3, \quad g^{109} = x+4$$

令 $$a_0 = 2, \quad a_1 = 68, \quad a_2 = 98, \quad a_3 = 86, \quad a_4 = 109$$

令 $$b_0 = 98, \quad b_1 = 109, \quad b_2 = 2, \quad b_3 = 68, \quad b_4 = 86$$
$$c_0 = 109, \quad c_1 = 120, \quad c_2 = 13, \quad c_3 = 79, \quad c_4 = 97$$

对于 $$m = 1\ 0\ 1\ 0\ 1$$
$$E(m) = 109+13+97+219 = c$$
$$c' = c-33 = 186$$
$$g^{c'} = g^{186} = g^{124} \cdot g^{62} = g^{62} = g^{60} \cdot g^2$$
$$= (x^2-1)x = x^3+x = -x-1+x = -1$$
$$x^3+x+1+(-1) = x^3+x = x(x^2+1) = x(x-2)(x-3)$$
$$= x(x+3)(x+2)$$

故 $$x(x+2)(x+3) = g^2 g^{98} g^{86} = g^{b_2} g^{b_0} g^{b_4}$$

故解得明文 $$m = 1\ 0\ 1\ 0\ 1$$

又如 $$b_0 = 86, \quad b_1 = 109, \quad b_2 = 68, \quad b_3 = 2, \quad b_4 = 98$$
$$c_0 = 98, \quad c_1 = 121, \quad c_2 = 80, \quad c_3 = 14, \quad c_4 = 110$$

已知 $$m = 1\ 1\ 0\ 0\ 1$$
$$c = E(m) = 98+121+110 = 329$$
$$c' = 329-36 = 293$$
$$g^{c'} = g^{45} = g^{44}g = x^{22}g = (-2x^2-4x+1)x^2 g$$
$$= (-2x^4-4x^3+x^2)(2x^2+x+2)$$
$$= (2x^2+2x+4x+4+x^2)(2x^2+x+2)$$
$$= (3x^2+x+4)(2x^2+x+2)$$
$$= x^4+x+3 = -x^2-x+x+3 = x^2+3$$
$$x^3+x^2+x+4 = (x-1)(x-2)(x-3)$$
$$= (x+2)(x+3)(x+4) = g^{a_2} g^{a_3} g^{a_4}$$
$$= g^{b_4} g^{b_0} g^{b_1}$$

故 $$m = 1\ 1\ 0\ 0\ 1$$

4.8 McEliece 公钥密码

4.8.1 编码理论准备

在近代密码学研究中有一种努力，这就是企图将纠错码用在密码上。由于纠错码有许多独特的功能，所以它具有潜在的优势。首先可一次编码完成对信息的加密和纠错两种功能，在保密与纠错间找到一种优美的折衷。因为一般说来，两者是有矛盾的。

1. 基本概念

为了讨论方便起见，有必要介绍纠错码理论。已掌握这部分内容的读者可不读这一节。

通常，(n,l)组码表示由 l 位 0,1 字符串到 n 位 0,1 字符串的编码过程，即编码的过程将明文

$$m = m_1 m_2 \cdots m_l \quad m_i \in \{0,1\}, \quad i = 1,2,\cdots,l$$

变换为码文

$$E(m) = w = w_1 w_2 \cdots w_n \quad w_i \in \{0,1\} \quad i = 1,2,\cdots,n$$

码文在信道中传输由于受到干扰难免出错，编码的目的在于发现错误，这是检错。进而将出错处纠正过来，这是纠错。所以检错和纠错是编码理论所要研究的内容。当然密码也是一种编码，它是以保密和安全为目的的。最简单的编码如下面两个例子：

① $(n+1,n)$检错码　即在 n 位明文后面附加一位奇偶校验位，即

$$E(m) = m_1 m_2 \cdots m_n m_{n+1}$$

如果传输过程中出一个错，即将某一位的 0 错为 1，或将 1 错为 0，则可立即被发现。当然在 n 位中若出两个错，则反而发现不了。不过在 n 位中出现两个错的可能性比出现一个错的机率要小一些。这样的码并不能判定是哪位出了差错，所以只是检错码。

② $(3n,n)$码　即

$$E(m) = m_1 m_2 \cdots m_n m_1 m_2 \cdots m_n m_1 m_2 \cdots m_n$$

即对每组 n 位的码连续传输 3 遍，若某 1 位出现不同，则必有两组是一样的，便以出现两次一样的为准，从而达到纠错的目的。当然，同时在同一位上出两次或 3 次错的可能性虽存在，但终究是比在这一位出现一个错的机率要少一些。某一位 3 次传输相同，由于错的一样而被认为正确的，而出错的概率是很小的。例如出错率为 0.001，传输正确的概率为 0.999，3 次都正确的概率为$(0.999)^3 = 0.997\,003$，出现 1 个错的概率为

$$\binom{3}{1}(0.001)(0.999)^2 = 0.002\,994$$

出现一个错误的传输可以通过上述$(3n,n)$码得到纠正，所以$(3n,n)$码传输正确的概率包含 3 次都正确及最多出一次错的概率之和，故为

$$0.997\,003 + 0.002\,994 = 0.999\,997$$

因此，传输正确的概率从每位为 0.999，提高到 0.999 997，但码文比明文扩大 3 倍，这很难为人们所接受。

2. 有关纠错码的几个定理

令 $m=m_1m_2\cdots m_n$，$m'=m_1'm_2'\cdots m_n'$ 都是长度为 n 的 0,1 符号串，$w(m)$ 为 m 的 n 位中 1 的数目，称为 m 的权。

$d(m,m')=w(m\oplus m')$ 称为 m 和 m' 间的距离。

由于 $1\oplus1=0$，$0\oplus0=0$，$1\oplus0=0\oplus1=1$，所以 $d(m,m')$ 实际上是 m 和 m' 不同的位的数目，以后⊕简记为＋，不必特别声明。

引理 4.2 若 $\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}$ 都是长度为 n 的 0,1 符号串，即

$$\boldsymbol{a}=a_1a_2\cdots a_n$$
$$\boldsymbol{b}=b_1b_2\cdots b_n$$
$$\boldsymbol{c}=c_1c_2\cdots c_n$$

$a_i,b_i,c_i\in\{0,1\}$，$i=1,2,\cdots,n$。则

(a) $d(\boldsymbol{a},\boldsymbol{b})=d(\boldsymbol{b},\boldsymbol{a})$

(b) $d(\boldsymbol{a},\boldsymbol{c})\leqslant d(\boldsymbol{a},\boldsymbol{b})+d(\boldsymbol{b},\boldsymbol{c})$

证明

(a) $d(\boldsymbol{a},\boldsymbol{b})=w(\boldsymbol{a}+\boldsymbol{b})=w(\boldsymbol{b}+\boldsymbol{a})=d(\boldsymbol{b},\boldsymbol{a})$

(b) 若 $a_i\neq c_i$，则有两种可能，即

$$a_i=b_i \text{ 但 } b_i\neq c_i \quad \text{或} \quad a_i\neq b_i \text{ 但 } b_i=c_i$$

当然，反过来若 $a_i=c_i$ 时，也有两种可能，一种是 $a_i=b_i=c_i$，另一种则是 $a_i\neq b_i$，$b_i\neq c_i$。基于以上讨论，令

$$d(a_i,b_i)=\begin{cases}0 & a_i=b_i\\ 1 & a_i\neq b_i\end{cases}$$

$$\begin{aligned}d(\boldsymbol{a},\boldsymbol{c})&=\sum_{i=1}^{n}d(a_i,c_i)\\&\leqslant\sum_{i=1}^{n}d(a_i,b_i)+\sum_{i=1}^{n}d(b_i,c_i)\\&\leqslant d(\boldsymbol{a},\boldsymbol{b})+d(\boldsymbol{b},\boldsymbol{c})\end{aligned}$$

定理 4.4 一组码可以检出 k 个错误的充要条件是码间最短距离至少为 $k+1$。

证明 设 $\boldsymbol{w}=w_1w_2\cdots w_n$ 是码字，$\boldsymbol{w}$ 传输中有误差 $\boldsymbol{e}=e_1e_2\cdots e_n$，指收到的是 $\boldsymbol{r}=\boldsymbol{w}+\boldsymbol{e}$。假定 $w(\boldsymbol{e})=k$，$\boldsymbol{e}$ 能被检出的充要条件是 $\boldsymbol{w}+\boldsymbol{e}$ 不是码字。因此，能检出 k 个错误的充要条件是码字之间的距离至少为 $k+1$。

定理 4.5 若两个码字之间的最短距离为 $2k+1$，则可以纠正权不超过 k 的误差。

证明 设码字 $\boldsymbol{a}$ 传输过程中发生误差，得到的是 $\boldsymbol{r}$，且 $d(\boldsymbol{a},\boldsymbol{r})\leqslant k$，则不存在别的码字与 $\boldsymbol{r}$ 的距离不超过 k。否则设为 $\boldsymbol{b}$，满足 $d(\boldsymbol{r},\boldsymbol{b})\leqslant k$，将有

$$\begin{aligned}d(\boldsymbol{a},\boldsymbol{b})&\leqslant d(\boldsymbol{a},\boldsymbol{r})+d(\boldsymbol{r},\boldsymbol{b})\\&\leqslant k+k<2k+1\end{aligned}$$

与假设矛盾，故定理得证。

定理 4.5 是极大似然译码方法的依据，例如有码字

10010　01001　10101　01110

两两做⊕运算：

	10010	01001	10101	01110
10010	—	11011	00111	11100
01001	11011	—	11100	00111
10101	00111	11100	—	11011
01110	11100	00111	11011	—

$\min\limits_{i\neq j}\{d(a_i,a_j)\}=3$，故能纠正一个错误。

w	10010	01001	10101	01110
	00010	11001	00101	11110
	11010	00001	11101	00110
r	10110	01101	10001	01010
	10000	01011	10111	01100
	10011	01000	10100	01111

上面给出距离为1的译码表，比如$r=01011$，它与01001距离为1，故译为01001。认为第4位出错。码字数量大时靠译码表译码工作量和存储量都太大。

3. 生成矩阵

令

$$\boldsymbol{G}=(g_{ij})_{l\times n}=\begin{pmatrix}\boldsymbol{G}_1\\ \boldsymbol{G}_2\\ \vdots\\ \boldsymbol{G}_l\end{pmatrix}=\begin{pmatrix}g_{11} & g_{12} & \cdots & g_{1n}\\ g_{21} & g_{22} & \cdots & g_{2n}\\ \cdots & \cdots & & \cdots\\ g_{l1} & g_{l2} & \cdots & g_{ln}\end{pmatrix}$$

其中$\boldsymbol{G}_i$为$\boldsymbol{G}$矩阵的第i行向量，即

$$\boldsymbol{G}_i=(g_{i1}g_{i2}\cdots g_{in})$$

$$g_{ij}\in\{0,1\},\quad i=1,2,\cdots,l;\ j=1,2,\cdots,n$$

设$m=m_1m_2\cdots m_l$为信息块，或长度为l的明文块，编码过程为

$$\boldsymbol{E}(m)=(m_1m_2\cdots m_l)\begin{pmatrix}g_{11} & g_{12} & \cdots & g_{1n}\\ g_{21} & g_{22} & \cdots & g_{2n}\\ \cdots & \cdots & & \cdots\\ g_{l1} & g_{l2} & \cdots & g_{ln}\end{pmatrix}=m\boldsymbol{G}$$

$$=\sum_{i=1}^{l}m_i\boldsymbol{G}_i=\boldsymbol{W}$$

例如

$$G=\begin{pmatrix}0&1&0&1&1&0\\0&1&0&0&1&1\\0&0&1&1&0&1\end{pmatrix}$$ 对于信息 1 0 1，对应的码字为

$$\begin{aligned}W&=(1\ 0\ 1)\begin{pmatrix}1&0&0&1&1&0\\0&1&0&0&1&1\\0&0&1&1&0&1\end{pmatrix}\\&=(1\ 0\ 0\ 1\ 1\ 0)+(0\ 0\ 1\ 1\ 0\ 1)\\&=(1\ 0\ 1\ 0\ 1\ 1)\end{aligned}$$

现将长度为 3 的信息和对应的码字列表于下：

信　息	码　字
0 0 0	0 0 0 0 0 0
0 0 1	0 0 1 1 0 1
0 1 0	0 1 0 0 1 1
0 1 1	0 1 1 1 1 0
1 0 0	1 0 0 1 1 0
1 0 1	1 0 1 0 1 1
1 1 0	1 1 0 1 0 1
1 1 1	1 1 1 0 0 0

显然，码字(6 位)前 3 位为信息位，余下的 3 位是校验位。可将这种情况推广到 (n,l)码上，即

$$G=\begin{pmatrix}1&0&0&\cdots&0&g_{11}&g_{12}&\cdots&g_{1n-l}\\0&1&0&\cdots&0&g_{21}&g_{22}&\cdots&g_{2n-l}\\\vdots&&&&&&&&\\0&0&0&\cdots&1&g_{l1}&g_{l2}&\cdots&g_{ln-l}\end{pmatrix}_{l\times n}$$

上，信息 $m=m_1m_2\cdots m_l$ 经编码过程得 n 位码字

信息位	校验位
l位	$n-l$位

为了方便起见，假定明文 $m=m_1m_2\cdots m_l$ 对应的码文是

$$\begin{aligned}W&=(m_1m_2\cdots m_l)\begin{pmatrix}1&0&0&\cdots&0&g_{11}&g_{12}&\cdots&g_{1n-l}\\0&1&0&\cdots&0&g_{21}&g_{22}&\cdots&g_{2n-l}\\\vdots&&&&&&&&\\0&0&0&\cdots&1&g_{l1}&g_{l2}&\cdots&g_{ln-l}\end{pmatrix}_{l\times n}\\&=(m_1m_2m_3\cdots m_lm_{l+1}\cdots m_n)=r\end{aligned}$$

显然有

$$m_{l+j}=\sum_{i=1}^{l}g_{ij}m_i,\quad j=1,2,\cdots,n-l \tag{4-1}$$

或

$$g_{1j}m_1 + g_{2j}m_2 + \cdots + g_{lj}m_l + m_{l+j} = 0 \qquad j = 1,2,\cdots,n-l$$

也就是说 $m_{l+1}, m_{l+2}, \cdots, m_n$ 即附加的多余部分是用来检验传输过程是否出错及哪些出错的，所以叫做校验位。$n-l$ 个方程(4-1)可用矩阵形式表示为

$$\begin{pmatrix} g_{11} & g_{21} & \cdots & g_{l1} & 1 & 0 & \cdots & 0 \\ g_{12} & g_{22} & \cdots & g_{l2} & 0 & 1 & \cdots & 0 \\ \vdots & & & & & & & \\ g_{1,n-l} & g_{2,n-l} & \cdots & g_{l,n-l} & 0 & 0 & \cdots & 1 \end{pmatrix} \begin{pmatrix} m_1 \\ m_2 \\ \vdots \\ m_n \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

或写作

$$\boldsymbol{H}\boldsymbol{r}^T = \boldsymbol{0} \tag{4-2}$$

称之为校验方程，$\boldsymbol{H} = (h_{ij})_{(n-l)\times n}$ 称为校验矩阵。显然生成矩阵 $\boldsymbol{G}$ 和校验矩阵 $\boldsymbol{H}$ 有如下关系。若

$$\boldsymbol{G} = (E_{(l)} \mid \boldsymbol{A})_{l\times n}$$

其中 $\boldsymbol{A} = (a_{ij})_{l\times(n-l)}$，$\boldsymbol{E}_{(l)}$ 为 l 阶单位矩阵，则

$$\boldsymbol{H} = (A^T \mid \boldsymbol{E}_{(n-l)})_{(n-l)\times n}$$

校验矩阵可用来纠正错误。(4-2)式说明正确传输应满足的等式。如若 $\boldsymbol{r} = \boldsymbol{w} + \boldsymbol{e}$，$\boldsymbol{e}$ 是误差。

$$\boldsymbol{H}\boldsymbol{r}^T = \boldsymbol{H}(\boldsymbol{w}+\boldsymbol{e})^T = \boldsymbol{H}\boldsymbol{w}^T + \boldsymbol{H}\boldsymbol{e}^T = \boldsymbol{H}\boldsymbol{e}^T$$

若 $\boldsymbol{H}\boldsymbol{e}^T \neq \boldsymbol{0}$，可由 $\boldsymbol{H}\boldsymbol{e}^T$ 看出究竟第几位出了错，并给予纠正。以

$$\boldsymbol{H} = \begin{pmatrix} 1&0&1&1&0&0 \\ 1&1&0&0&1&0 \\ 0&1&1&0&0&1 \end{pmatrix}, \quad \boldsymbol{r} = 100101$$

为例

$$\boldsymbol{H}\boldsymbol{r}^T = \begin{pmatrix} 1&0&1&1&0&0 \\ 1&1&0&0&1&0 \\ 0&1&1&0&0&1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

$(011)^T$ 正好是矩阵 $\boldsymbol{H}$ 的第 2 列，故可知第 2 位出错。将 $\boldsymbol{r}=100101$ 的第 2 位的 0 改为 1，故得

$$\boldsymbol{W}=110101$$

信息位为 110，译码便结束。

当然，若出现两个以上的错误，这种方法便失败，也就是说它不能获得正确的纠错。现在将译码步骤列下。设收到的信息为

$$\boldsymbol{r} = r_1 r_2 \cdots r_n$$

S1. 计算 $\boldsymbol{S} = \boldsymbol{H}\boldsymbol{r}^T$；

S2. 若 $\boldsymbol{S}=\boldsymbol{0}$，则可认为传输过程是正确的，则明文 $\boldsymbol{m} = r_1 r_2 \cdots r_l$，若 $\boldsymbol{S}\neq\boldsymbol{0}$ 转 S3；

S3. 若 $\boldsymbol{S}$ 是矩阵 $\boldsymbol{H}$ 的第 i 列，则认为 r_i 有错误，予以改正。然后取前面的 l 位作为明

文；若 S 不是 H 的列向量(且不为零)，则认为传输过程至少出现两个以上的错误，无法正确纠错。以上的 S 通常叫校正子。

定理 4.6 $(n-l)\times n$ 的校验矩阵能正确纠正 1 个错误的充要条件是 H 的各列为不相同的非零列向量。

这个定理是很显然的，证明由读者自己来完成。

4. 充要条件

根据 $k\times n$ 阶矩阵 H 能纠一个错的充要条件是 H 矩阵的各列为两两不相同的非零列向量，当 k 确定后，n 最大可取 2^k-1，又因 $k=n-l$，故 $l=n-k$。

例如，$k=3$ 时，$n=2^3-1=7$，$l=7-3=4$

$$\boldsymbol{H}=\begin{bmatrix}0&1&1&1&1&0&0\\1&0&1&1&0&1&0\\1&1&0&1&0&0&1\end{bmatrix}$$

$\boldsymbol{H}$ 的各列包含了除 $\begin{bmatrix}0\\0\\0\end{bmatrix}$ 以外的所有状态，对应的生成矩阵为

$$\boldsymbol{G}=\begin{bmatrix}1&0&0&0&0&1&1\\0&1&0&0&1&0&1\\0&0&1&0&1&1&0\\0&0&0&1&1&1&1\end{bmatrix}$$

用这种办法构成的可纠一个错的码称为(7,4)汉明(Hamming)码。又如 $k=4$ 时，

$$n=2^4-1=15,\ l=n-k=15-4=11$$

也可以构造汉明码，0,1 字符串长度为 11 的明文经编码得长度为 15 的码字。

下面便是(15,11)的汉明码的校验矩阵

$$\boldsymbol{H}=\begin{bmatrix}1&1&1&0&0&0&1&1&1&0&1&1&0&0&0\\1&0&0&1&1&0&1&1&0&1&1&0&1&0&0\\0&1&0&1&0&1&1&0&1&1&1&0&0&1&0\\0&0&1&0&1&1&0&1&1&1&1&0&0&0&1\end{bmatrix}$$

从校验矩阵不难求得对应的生成矩阵 $\boldsymbol{G}$，这里从略。

4.8.2 BCH 码和 Goppa 码

1. BCH 码

BCH 是 Bose，Chaudhari，Hocquenghem 的缩写。BCH 码是他们三人于 1959 年同时发明的。现在假定读者对有限域 $GF(p^n)$ 有基本的了解。

假定 α 是域 $GF(2^4)$ 的本原元素，满足 $\alpha^4+\alpha+1=0$，即 $\mathrm{mod}(\alpha^4+\alpha+1)$ 构成一个 $GF(2^4)$ 域，令

$$\boldsymbol{H}=\begin{bmatrix}1 & \alpha & \alpha^2 & \alpha^3 & \cdots & \alpha^{14}\\ 1 & \alpha^3 & (\alpha^2)^3 & (\alpha^3)^3 & \cdots & (\alpha^{14})^3\end{bmatrix} \tag{4-3}$$

由于 $\alpha^0=1$，$\alpha^1=\alpha$，α^2，α^3，$\alpha^4=1+\alpha$，$\alpha^5=\alpha+\alpha^2$，$\alpha^6=\alpha^2+\alpha^3$，$\alpha^7=\alpha^3+\alpha^4=1+\alpha+\alpha^3$，$\alpha^8=\alpha+\alpha^2+\alpha^4=1+\alpha^2$，$\alpha^9=\alpha+\alpha^3$，$\alpha^{10}=\alpha^2+\alpha^4=1+\alpha+\alpha^2$，$\alpha^{11}=\alpha+\alpha^2+\alpha^3$，$\alpha^{12}=\alpha^2+\alpha^3+\alpha^4=1+\alpha^2+\alpha^3+\alpha$，$\alpha^{13}=\alpha+\alpha^2+\alpha^3+\alpha^4=1+\alpha^2+\alpha^3$，$\alpha^{14}=\alpha+\alpha^3+\alpha^4=1+\alpha^3$，$\alpha^{15}=\alpha+\alpha^4=1$。

所以
$$\boldsymbol{H}=\begin{bmatrix}1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \cdots & \alpha^{14}\\ 1 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} & 1 & \cdots & \alpha^{12}\end{bmatrix}$$

用下列矩阵来表示

$$\boldsymbol{H}=\begin{bmatrix}1&0&0&0&1&0&0&1&1&0&1&0&1&1&1\\ 0&1&0&0&1&1&0&1&0&1&1&1&1&0&0\\ 0&0&1&0&0&1&1&0&1&0&1&1&1&1&0\\ 0&0&0&1&0&0&1&1&0&1&0&1&1&1&1\\ 1&0&0&0&1&1&0&0&0&1&1&0&0&0&1\\ 0&0&0&1&1&0&0&0&1&1&0&0&0&1&1\\ 0&0&1&0&1&0&0&1&0&1&0&0&1&0&1\\ 0&1&1&1&1&0&1&1&1&1&0&1&1&1&1\end{bmatrix} \tag{4-4}$$

这里用

$$\begin{bmatrix}0\\1\\0\\1\end{bmatrix},\quad\begin{bmatrix}1\\0\\1\\1\end{bmatrix}$$

分别表示 $\alpha+\alpha^3=\alpha^9$，$1+\alpha^2+\alpha^3=\alpha^{13}$。其他依此类推。

若在第 h，k 两位发生错误，则校验子

$$\boldsymbol{S}=\begin{bmatrix}\alpha^h+\alpha^k\\ \alpha^{3h}+\alpha^{3k}\end{bmatrix}=\begin{bmatrix}z_1\\ z_2\end{bmatrix}$$

$$z_1=\alpha^h+\alpha^k,\ z_2=\alpha^{3h}+\alpha^{3k}$$

所以
$$\begin{aligned}z_2&=(\alpha^h+\alpha^k)(\alpha^{2h}+\alpha^{h+k}+\alpha^{2k})\\ &=z_1(z_1^2+\alpha^{h+k})\end{aligned}$$

$$\alpha^{h+k}=\frac{z_2}{z_1}+z_1^2$$

故 α^h 和 α^k 是方程

$$z^2+z_1z+\left(\frac{z_2}{z_1}+z_1^2\right)=0 \tag{4-5}$$

的两个根。

若 $z_1=z_2=0$，则认为无差错，若 $z_2=z_1^3=\alpha^{3h}$，则认为有一个错误发生在第 h 位。

若在第 7，9 位发生错误，则

$$z_1=\begin{bmatrix}0\\0\\1\\1\end{bmatrix}+\begin{bmatrix}1\\0\\1\\0\end{bmatrix}=\begin{bmatrix}1\\0\\0\\1\end{bmatrix}=\alpha^{14}$$

$$z_2=\begin{bmatrix}0\\0\\0\\1\end{bmatrix}+\begin{bmatrix}0\\1\\0\\1\end{bmatrix}=\begin{bmatrix}0\\1\\0\\0\end{bmatrix}=\alpha=\alpha^{16}$$

$$\frac{z_2}{z_1}+z_1^2=\alpha^7+\alpha^{13}=\alpha^2+1+\alpha^2+\alpha^3=1+\alpha^3=\alpha^{14}$$

所以
$$z^2+\alpha^{14}z+\alpha^{14}=(z+\alpha^6)(z+\alpha^8)=0$$

这就验证了 BCH 纠错是正确的。

2. Goppa 码

长为 n 的 Goppa 码要用到两个概念：一是系数在 $GF(p^m)$ 的 Goppa 多项式 $G(z)$；另一个是 $GF(p^m)$ 满足条件 $G(\beta_i)\neq 0$，$i=1,2,\cdots,n$ 的子集 $L=\{\beta_1,\beta_1,\cdots,\beta_n\}$。

关于 $GF(p)$ 的任一向量 $\boldsymbol{\alpha}=\{\alpha_1,\alpha_2,\cdots,\alpha_n\}$，有一有理函数

$$R_\alpha(z)=\sum_{i=1}^{n}\frac{\alpha_i}{z-\beta_i} \tag{4-6}$$

由满足

$$R_\alpha(z)\equiv 0\ (\mathrm{mod}\ G(z))$$

的所有向量 $\boldsymbol{\alpha}$ 构成的码称为 Goppa 码，用 $\boldsymbol{\Gamma}(\boldsymbol{L},\boldsymbol{G})$ 表示。Goppa 码的校验矩阵不难求得。

因为
$$\begin{aligned}&-(z-\beta_i)\frac{(G(z)-G(\beta_i))}{z-\beta_i}G(\beta_i)^{-1}\\&=-G(z)G(\beta_i)^{-1}+G(\beta_i)G(\beta_i)^{-1}\\&\equiv 1(\mathrm{mod}\ G(z))\end{aligned}$$

由此可得 $\mathrm{mod}\ G(z)$，$(z-\beta_i)$ 的逆，即

$$(z-\beta_i)^{-1}=-\frac{G(z)-G(\beta_i)}{z-\beta_i}G(\beta_i)^{-1}$$

所以 $\alpha\in\boldsymbol{\Gamma}(\boldsymbol{L},\boldsymbol{G})$ 的充要条件是

$$\sum_{i=1}^{n}\alpha_i\frac{G(z)-G(\beta_i)}{z-\beta_i}G(\beta_i)^{-1}=0 \tag{4-7}$$

若 $G(z)=\sum_{i=0}^{r}g_iz^i$，$g_i\in GF(p^m)$，$g_r\neq 0$，则由于

$$(z-\beta_i)(z^{k-1}+z^{k-2}\beta_i+\cdots+\beta_i^{k-1})=z^k-\beta_i^k,\quad k=2,3,\cdots,r$$

$$
\begin{aligned}
&(z-\beta_i)[g_r(z^{r-1}+z^{r-2}\beta_i+\cdots+\beta_i^{r-1})\\
&+g_{r-1}(z^{r-2}+z^{r-3}\beta_i+\cdots+\beta_i^{r-2})+\cdots\\
&+g_2(z+\beta_i)+g_1]\\
=&g_r(z-\beta_i)(z^{r-1}+z^{r-2}\beta_i+\cdots+\beta_i^{r-1})\\
&+g_{r-1}(z-\beta_i)(z^{r-2}+z^{r-3}\beta_i+\cdots+\beta_i^{r-2})+\cdots\\
&+g_2(z-\beta_i)(z+\beta_i)+g_1(z-\beta_i)\\
=&g_r(z^r-\beta_i^r)+g_{r-1}(z^{r-1}-\beta_i^{r-1})+\cdots\\
&+g_2(z^2-\beta_i^2)+g_1(z-\beta_i)\\
=&G(z)-G(\beta_i)
\end{aligned}
$$

这就证明了

$$
\begin{aligned}
\frac{G(z)-G(\beta_i)}{z-\beta_i}=&g_r(z^{r-1}+z^{r-2}\beta_i+\cdots+\beta_i^{r-1})\\
&+g_{r-1}(z^{r-2}+z^{r-3}\beta_i+\cdots+\beta_i^{r-2})+\cdots\\
&+g_2(z+\beta_i)+g_1
\end{aligned}
$$

根据 $a\in\Gamma(L,G)$ 的充要条件(7)，所以

$$
\begin{aligned}
&\sum_{i=1}^{n}a_i\frac{G(z)-G(\beta_i)}{z-\beta_i}G(\beta_i)^{-1}\\
=&\sum_{i=1}^{n}a_i[g_r(z^{r-1}+z^{r-2}\beta_i+\cdots+\beta_i^{r-1})\\
&+g_{r-1}(z^{r-2}+z^{r-3}\beta_i+\cdots+\beta_i^{r-2})+\cdots\\
&+g_2(z+\beta_i)+g_1]G(\beta_i)^{-1}\\
=&z^{r-1}g_r\Big[\sum_{i=1}^{n}a_iG(\beta_i)^{-1}\Big]\\
&+z^{r-2}\Big[\sum_{i=1}^{n}a_i(g_r\beta_i+g_{r-1})G(\beta_i)^{-1}\Big]\\
&+z^{r-3}\Big[\sum_{i=1}^{n}a_i(g_r\beta_i^2+g_{r-1}\beta_i+g_{r-2})G(\beta_i)^{-1}\Big]+\cdots\\
&+z\sum_{i=1}^{n}a_i(g_r\beta_i^{r-2}+g_{r-1}\beta_i^{r-3}+\cdots+g_2)G(\beta_i)^{-1}\\
&+\sum_{i=1}^{n}a_i(g_r\beta_i^{r-1}+g_{r-1}\beta_i^{r-2}+\cdots+g_1)G(\beta_i)^{-1}=0
\end{aligned}
$$

令 $z^{r-1},z^{r-2},\cdots,r,1$ 的系数分别等于0，故有

$$
\begin{aligned}
&a_1g_rG(\beta_1)^{-1}+a_2g_rG(\beta_2)^{-1}+\cdots+a_ng_rG(\beta_n)^{-1}=0\\
&a_1(\beta_1g_r+g_{r-1})G(\beta_1)^{-1}+a_2(\beta_2g_r+g_{r-1})G(\beta_2)^{-1}+\cdots
\end{aligned}
$$

$$
\begin{aligned}
&+a_n(\beta_n g_r+g_{r-1})G(\beta_n)^{-1}=0\\
&a_1(\beta_1^2 g_r+\beta_1 g_{r-1}+g_{r-2})G(\beta_1)^{-1}+a_2(\beta_2^2 g_r+\beta_2 g_{r-1}\\
&+g_{r-2})G(\beta_2)^{-1}+\cdots+a_n(\beta_n^2 g_r+\beta_n g_{r-1}+g_{r-2})G(\beta_n)^{-1}=0\\
&\vdots\\
&a_1(\beta_1^{r-1} g_r+\beta_1^{r-2} g_{r-1}+\cdots+g_1)G(\beta_1)^{-1}\\
&+a_2(\beta_2^{r-1} g_r+\beta_2^{r-2} g_{r-1}+\cdots+g_1)G(\beta_2)^{-1}+\cdots\\
&+a_n(\beta_n^{r-1} g_r+\beta_n^{r-2} g_{r-1}+\cdots+g_1)G(\beta_n)^{-1}=0
\end{aligned}
$$

写成矩阵形式：

$$
\begin{pmatrix}
g_r G(\beta_1)^{-1} & \cdots & g_r G(\beta_n)^{-1}\\
(\beta_1 g_r+g_{r-1})G(\beta_1)^{-1} & \cdots & (\beta_n g_r+g_{r-1})G(\beta_n)^{-1}\\
(\beta_1^2 g_r+\beta_1 g_{r-1}+g_{r-2})G(\beta_1)^{-1} & \cdots & (\beta_n^2 g_r+\beta_n g_{r-1}+g_{r-2})G(\beta_n)^{-1}\\
\vdots & & \\
(\beta_1^{r-1} g_r+\beta_1^{r-2} g_{r-1}+\cdots+g_1)G(\beta_1)^{-1} & \cdots & (\beta_n^{r-1} g_r+\beta_n^{r-2} g_{r-1}+\cdots+g_1)G(\beta_n)^{-1}
\end{pmatrix}
\times\begin{pmatrix}a_1\\a_2\\a_3\\\vdots\\a_n\end{pmatrix}=\mathbf{0}
$$

或写作

$$
\boldsymbol{H}\boldsymbol{a}^{T}=\mathbf{0}
$$

称 H 为校验矩阵，而且有

$$
\begin{pmatrix}
g_r G(\beta_1)^{-1} & \cdots & g_r G(\beta_n)^{-1}\\
(\beta_1 g_r+g_{r-1})G(\beta_1)^{-1} & \cdots & (\beta_n g_r+g_{r-1})G(\beta_n)^{-1}\\
(\beta_1^2 g_r+\beta_1 g_{r-1}+g_{r-2})G(\beta_1)^{-1} & \cdots & (\beta_n^2 g_r+\beta_n g_{r-1}+g_{r-2})G(\beta_n)^{-1}\\
\vdots & & \\
(\beta_1^{r-1} g_r+\beta_1^{r-2} g_{r-1}+\cdots+g_1)G(\beta_1)^{-1} & \cdots & (\beta_n^{r-1} g_r+\beta_n^{r-2} g_{r-1}+\cdots+g_1)G(\beta_n)^{-1}
\end{pmatrix}
=\begin{pmatrix}
g_r & 0 & 0 & \cdots & 0\\
g_{r-1} & g_r & 0 & \cdots & 0\\
g_{r-2} & g_{r-1} & g_r & \cdots & 0\\
 & & \vdots & & \\
g_1 & g_2 & g_3 & & g_r
\end{pmatrix}
\begin{pmatrix}
1 & 1 & \cdots & 1\\
\beta_1 & \beta_2 & \cdots & \beta_n\\
\beta_1^2 & \beta_2^2 & \cdots & \beta_n^2\\
 & & \vdots & \\
\beta_1^{r-1} & \beta_2^{r-1} & \cdots & \beta_n^{r-1}
\end{pmatrix}
$$

$$\times\begin{pmatrix} G(\beta_1)^{-1} & & & & \\ & G(\beta_2)^{-1} & & & \mathbf{0} \\ & & G(\beta_3)^{-1} & & \\ & \mathbf{0} & & \ddots & \\ & & & & G(\beta_n)^{-1} \end{pmatrix} = \boldsymbol{LAR}$$

其中

$$\boldsymbol{L} = \begin{pmatrix} g_r & 0 & 0 & \cdots & 0 \\ g_{r-1} & g_r & 0 & \cdots & 0 \\ g_{r-2} & g_{r-1} & g_r & \cdots & 0 \\ \vdots & & & & \\ g_1 & g_2 & g_3 & \cdots & g_r \end{pmatrix}$$

$$\boldsymbol{A} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ \beta_1 & \beta_2 & \beta_3 & \cdots & \beta_n \\ \beta_1^2 & \beta_2^2 & \beta_3^2 & \cdots & \beta_n^2 \\ \vdots & & & & \\ \beta_1^{r-1} & \beta_2^{r-1} & \beta_3^{r-1} & \cdots & \beta_n^{r-1} \end{pmatrix}$$

$$\boldsymbol{R} = \begin{pmatrix} G(\beta_1)^{-1} & & & & \\ & G(\beta_2)^{-1} & & & \mathbf{0} \\ & & \ddots & & \\ & \mathbf{0} & & & \\ & & & & G(\beta_n)^{-1} \end{pmatrix}$$

由于 $\boldsymbol{L}$ 是可逆矩阵，故可取 $\boldsymbol{AR}$ 作为校验矩阵。

$$H' = \boldsymbol{AR}\begin{pmatrix} G(\beta_1)^{-1} & G(\beta_2)^{-1} & \cdots & G(\beta_n)^{-1} \\ \beta_1 G(\beta_1)^{-1} & \beta_2 G(\beta_2)^{-1} & \cdots & \beta_n G(\beta_n)^{-1} \\ \vdots & & & \\ \beta_1^{r-1} G(\beta_1)^{-1} & \beta_2^{r-1} G(\beta_2)^{-1} & \cdots & \beta_n^{r-1} G(\beta_n)^{-1} \end{pmatrix}$$

作为校验矩阵。

Goppa 码是线性码。码字长 n，码间最短距离 $d \geqslant r+1$，r 是 $G(z)$ 的次方，明文长度 $l \geqslant n - mr$。

例如，取

$$G(z) = z^2 + z + 1$$

$$\boldsymbol{L} = GF(2^3) = \{0, 1, \beta, \beta^2, \cdots, \beta^6\}$$

$p=2$，$p^m = 2^3 = 8$。取不可化约的多项式 $p(z) = z^3 + z + 1$，β 是 $GF(2^3)$ 的生成元素，可以证明 $\boldsymbol{L}$ 中的元素均满足 $G(\beta_i) \neq 0$。

因为

$$\beta^3 = \beta + 1, \quad \beta^4 = \beta + \beta^2$$

$$\beta^5 = \beta^2 + \beta^4 = 1 + \beta + \beta^2$$

$$\beta^6 = \beta + \beta + \beta^3 = 1 + \beta^2, \quad \beta^7 = \beta + \beta^3 = 1$$

$$\beta^8 = \beta$$

β 的二进制表示和多项式表示如表 4.3 所示。

表 4.3

β表示	二进制表示	多项式表示
0	0 0 0	0
1	1 0 0	1
β	0 1 0	x
β^2	0 0 1	x^2
β^3	1 1 0	$1+x$
β^4	0 1 1	$x+x^2$
β^5	1 1 1	$1+x+x^2$
β^6	1 0 1	$1+x^2$

$$G(x)=x^3+x+1$$

$$G(0)=1$$

$$G(1)=1$$

$$G(\beta)=\beta^3+\beta+1=\beta^5$$

$$G(\beta^2)=\beta^6+\beta^2+1=1+\beta-\beta^3$$

$$G(\beta^3)=\beta^6+\beta^3+1=1+\beta^2+1+\beta+1=1+\beta+\beta^2=\beta^7$$

$$G(\beta^4)=\beta^8+\beta^4+1=\beta+\beta+\beta^2+1=1+\beta^2=\beta^6$$

$$G(\beta^5)=\beta^{10}+\beta^5+1=\beta^3+1+\beta+\beta^2+1$$

$$=1+\beta+1+\beta+\beta^2+1=\beta^5$$

$$G(\beta^6)=\beta^{12}+\beta^6+1=\beta^5+\beta^6+1$$

$$=1+\beta+\beta^2+1+\beta^2+1$$

$$=1+\beta=\beta^3$$

因为

$$\beta^7=1$$

所以

$$\boldsymbol{H}=\begin{bmatrix}\frac{1}{1} & \frac{1}{1} & \frac{1}{\beta^5} & \frac{1}{\beta^3} & \frac{1}{\beta^5} & \frac{1}{\beta^6} & \frac{1}{\beta^6} & \frac{1}{\beta^3}\\ \frac{0}{1} & \frac{1}{1} & \frac{\beta}{\beta^5} & \frac{\beta^2}{\beta^3} & \frac{\beta^3}{\beta^5} & \frac{\beta^4}{\beta^6} & \frac{\beta^5}{\beta^6} & \frac{\beta^6}{\beta^3}\end{bmatrix}$$

$$=\begin{bmatrix}1 & 1 & \beta^2 & \beta^4 & \beta^2 & \beta & \beta & \beta^4\\ 0 & 1 & \beta^5 & \beta^6 & \beta^7 & \beta^5 & \beta^6 & \beta^5\end{bmatrix}$$

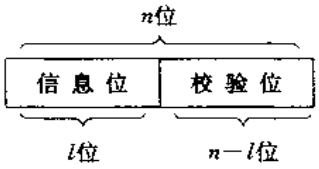
$$=\begin{bmatrix}1&1&0&0&0&0&0&0\\0&0&0&1&0&1&1&1\\0&0&1&1&1&0&0&1\\ &&&\vdots&&&&\\0&1&1&1&1&1&1&1\\0&0&1&0&1&1&0&1\\0&0&0&1&1&1&1&0\end{bmatrix}$$

4.8.3 McEliece 码

McEliece 于 1978 年提出一种利用 Goppa 码构造公钥密码的方案。设 Goppa 码能纠

正 t 个错误。明文为 l 位的 0,1 符号串。设 $m=m_1m_2\cdots m_l$ 生成矩阵 $G=(\boldsymbol{E}_{(l)}\ \vdots\ \boldsymbol{A})_{l\times n}$

其中 $\boldsymbol{E}_{(l)}$ 为 l 阶单位矩阵，所以郭帕码将 l 位明文变换为 n 位的码字，码字前 l 位是明文，后面 $n-l$ 位是校验位。



其中 $n=2^t$。用户 A 构造一 Goppa 码的 $l_A\times n_A$ 阶生成矩阵 G_A。用户 A 随机选一个 $l_A\times l_A$ 阶非奇异矩阵 S_A 及 $n_A\times n_A$ 的排列矩阵 P_A，记

$$G_A^* = S_AG_AP_A$$

用户 A 将 G_A^* 公开，而将 S_A，G_A，P_A 保密。

若用户 B 要向 A 送去信息 m。设 m 为 l_A 位的 0,1 符号串，加密算法是：

$$c = mG_A^* + e$$

McEliece 公钥密码虽然对码长为 1024 的 Goppa 码而设计的，但它可以归纳为：设信息 $m=m_1m_2\cdots m_k$ 为 k 比特，生成矩阵

$$G = [A_{k\times(n-k)}\ I_k]_{k\times n}$$

码字 $C=c_1c_2\cdots c_n$ 为

$$C = m\cdot G$$

或 $C=(c_1c_2\cdots c_{n-k},m_1m_2\cdots m_k)$称为$(n,k)$码字，接收到的字为

$$R = C+E$$

或

$$C = R+E$$

McEliece 系统即在于寻找一 $k\times k$ 的非奇异的方阵 S，及 $n\times n$ 的排列矩 P，使

$$G^* = SGP$$

G^* 称为 $k\times n$ 的公开生成矩阵，作为公钥公开。若对 m 加密码成密文 C

$$\begin{aligned} C &= mG^* + E \\ &= mSGP + E \\ C^* &= Cp^{-1} = (mS)G + E^* \end{aligned}$$

其中

$$E^* = EP^{-1}$$

$$m = (mS)S^{-1}$$

McEliece 建议分组长度 $n=2^{10}=1024$ 比特的 Goppa 码。

例 4.5 设已知(7,4)汉明码，码字的最短距离 $d_{\min}=3$，能纠一个错，码的生成矩阵

$$\boldsymbol{G}=\begin{bmatrix}1&1&0&1&0&0&0\\0&1&1&0&1&0&0\\1&1&1&0&0&1&0\\1&0&1&0&0&0&1\end{bmatrix}$$

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

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

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

$$G'=\begin{bmatrix}0&1&0&1&1&0&1&0&0&1&1&1&1&1&1&1\\1&0&1&1&0&0&1&0&0&1&0&1&1&1&1&1\\0&1&1&0&0&1&1&0&1&1&0&1&1&1&1&0\\1&1&1&1&1&0&1&1&0&1&0&1&1&1&0&0\\1&0&1&0&0&1&1&&0&1&0&1&1&0&0&0\\1&1&0&0&1&0&1&0&0&1&0&1&0&0&0&0\\1&0&1&0&1&0&1&1&1&1&0&0&0&0&0&0\\1&1&0&1&0&0&0&1&0&1&0&0&0&0&0&0\end{bmatrix}$$

设 $m=1100\quad 1001$

传输误差 $e=0000\quad 1000\quad 1000\quad 0000$

$mG'=1001\quad 1110\quad 0011\quad 1000$

$$C' = mG' + e = 1001 \quad 0110 \quad 1011 \quad 1000$$

$$C'P = 1000 \quad 1101 \quad 0111 \quad 0001$$

利用 Goppa 译码方法对 $C'P^{-1}$ 译码得

$$1\ 1\ 0\ 0\ 1\ 0\ 0\ 1$$

McEliece 建议 Goppa 码取长的 $n=2^{10}=1024$ 位能纠正 50 个错明文长 $k=524$ 位，故公钥规模太大，因而失去实用价值。

$$\boldsymbol{SG} = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

$$\boldsymbol{G'} = \boldsymbol{SGP} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$

假如

$$\boldsymbol{m} = (0100)$$

$$\boldsymbol{C} = (0100)\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$

$$= (0100110)$$

若有误差

$$\boldsymbol{E} = (0000100)$$

$$\boldsymbol{R} = \boldsymbol{C} + \boldsymbol{E} = (0100010)$$

$$\boldsymbol{C'} = \boldsymbol{RP}^{-1}$$

Jordan 法求 $\boldsymbol{P}^{-1}$ 如下，作加边矩阵

$$[\boldsymbol{P} \mid \boldsymbol{I}_{(7)}] = \left(\begin{array}{ccccccc:ccccccc} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right)$$

对上面矩阵作初等变换便得

$$\left(\begin{array}{ccccccc:ccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{array}\right)$$

所以
$$\boldsymbol{P}^{-1}=\begin{bmatrix}0&1&0&0&0&0&0\\0&0&0&1&0&0&0\\1&0&0&0&0&0&0\\0&0&0&0&0&0&1\\0&0&1&0&0&0&0\\0&0&0&0&0&1&0\\0&0&0&0&1&0&0\end{bmatrix}$$

所以
$$\boldsymbol{C}^{*}=\boldsymbol{C}\boldsymbol{P}^{-1}=(0001010)$$

类似方法可求
$$\boldsymbol{S}^{T}=\begin{bmatrix}1&0&1&0\\1&1&0&0\\1&1&1&0\\0&1&0&1\end{bmatrix}$$

校正子
$$\boldsymbol{S}=\boldsymbol{C}^{*}\boldsymbol{H}^{T}$$

校验矩阵 $\boldsymbol{H}$ 由 $\boldsymbol{G}$ 产生。$\boldsymbol{G}=(A\mid I_{(4)})$，$\boldsymbol{H}=(I_{(3)}\mid A^{T})$
$$\boldsymbol{H}=\begin{bmatrix}1&0&0&1&0&1&1\\0&1&0&1&1&1&0\\0&0&1&0&1&1&1\end{bmatrix}$$

$$\boldsymbol{S}=(0001010)\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\\1&1&0\\0&1&1\\1&1&1\\1&0&1\end{bmatrix}=(001)$$

$$\boldsymbol{E}^{*}=\boldsymbol{E}\boldsymbol{P}^{-1}=(0000100)\begin{bmatrix}0&1&0&0&0&0&0\\0&0&0&1&0&0&0\\1&0&0&0&0&0&0\\0&0&0&0&0&0&1\\0&0&1&0&0&0&0\\0&0&0&0&0&1&0\\0&0&0&0&1&0&0\end{bmatrix}$$

$$=(0010000)$$

$$\boldsymbol{C}^{*}+\boldsymbol{E}^{*}=(0011010)$$

后面 4 位即为：
$$m^{*}=m\boldsymbol{S}=(1010),\quad(m\boldsymbol{S})\boldsymbol{S}^{-1}=(0100)$$

4.9 MH 背包公钥的 Shamir 攻击

MH 背包公钥密码中的公钥序列 $a_1,a_2,\cdots,a_n$ 是由超递增序列 $b_1,b_2,\cdots,b_n$ 通过默科-赫尔曼变换得到的。即

$$a_k = wb_k(\mathrm{mod}\ m),\quad k=1,2,\cdots,n$$

假定所得的 a_k 它的十进制数表示的位数是相同的，Shamir 便抓住它作为突破口将 MH 背包公钥密码攻破了。了解他是怎样进行分析的对我们颇有启发。

4.9.1 算法的非形式化叙述

破译的问题在于寻找一对陷门(m,u)使得

$$b_k \equiv ua_k(\mathrm{mod}\ m)\quad k=1,2,\cdots,\mathrm{n}$$

是超递增序列。若所得的超递增序列的每一项大致是前面一项的两倍，最后一项比 $m/2$ 小，或超递增序列的和小于 m，这样有

$$b_n < 2^{-1}m,\quad b_{n-1} < 2^{-2}m,\quad \cdots,\quad b_1 < 2^{-n}m$$

以 $b_1\equiv ua_1(\mathrm{mod}\ m)$为例，作图(如图 4.1 所示)

$$y = a_1x(\mathrm{mod}\ m),\quad 0\leqslant x\leqslant m$$

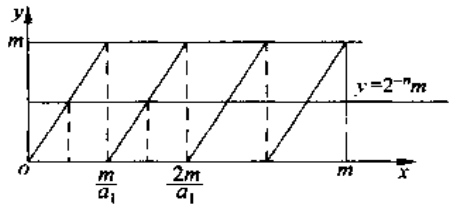


图 4.1

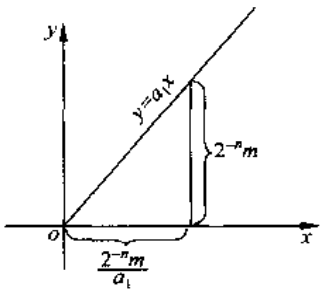


图 4.2

这些斜率为 a_1 的锯齿曲线的极小点分别为$\frac{m}{a_1},\frac{2m}{a_1},\cdots$，间距为$\frac{m}{a_1}$。由于

$$b_1 < 2^{-n}m \tag{4-8}$$

所以，u 必须在区间

$$\left(\frac{km}{a_1},\frac{km}{a_1}+\frac{2^{-n}m}{a_1}\right) \tag{4-9}$$

上，$k=0,1,2,\cdots,a_1-1$。这可从图 4.1 中看出。

但 u 和 m 都是要求的整数。为方便起见，将 m 作为 1 个单位，问题引出求 u,m 使比值 u/m 必须位于下列区间上

$$\left(\frac{k}{a_1},\frac{k}{a_1}+d_1\right),\quad k=0,1,2,\cdots,a_1-1 \tag{4-10}$$

其中 $d_1=2^{-n}/a_1$，d_1 的几何意义如图 4.2 所示。

用类似方法讨论

$$b_2 \equiv u a_2 (\bmod m)$$

做 $y \equiv a_2 x (\bmod 1)$的图像。由于

$$b_2 < 2^{n-1}$$

所以，$\frac{u}{m}$必须位于下列区间上

$$\left(\frac{k}{a_2}, \frac{k}{a_2} + d_2\right), \quad k = 0,1,2,\cdots,a_2 - 1 \tag{4-11}$$

其中 $d_2 = 2^{n-1}/a_2$，$\frac{u}{m}$必须同时在区间(4-10)和(4-11)上，故必须在区间(4-10)和(4-11)的交集上。因此求$\frac{u}{m}$的所在范围缩小了，而且$\frac{u}{m}$出现在(4-10)和(4-11)的极小点十分接近的地方。

对 $a_3, a_4, \cdots, a_n$ 用类似的算法进行讨论，可以得出所求的$\frac{u}{m}$是图像 $y = a_i x (\bmod 1)$，$i = 1,2,\cdots,n$ 极小点的聚点。

4.9.2 举例

在进一步讨论沙米尔(Shamir)攻击方法的其他细节之前，先看一个例子有助于理解。

设已知公钥序列{467,355,131,318,113,21,135,215}，$a_1 = 467$，故$\frac{u}{m}$应落入下列区间上

$$A_n = \left(\frac{n}{467}, \frac{n}{467} + \frac{1}{467 \times 256}\right), \quad n = 1,2,\cdots,466$$

其中 $256 = 2^8$。

$a_2 = 355$，同理$\frac{n}{m}$应落入下面的区间上

$$B_k = \left(\frac{k}{355}, \frac{k}{355} + \frac{1}{355 \times 128}\right), \quad k = 1,2,\cdots,354$$

因此，$\frac{u}{m}$必须在 $\bigcup_{k=1}^{466} A_k$ 和 $\bigcup_{k=1}^{354} B_k$ 的交集上。通过计算得出它们是下面 4 个区间

$$\begin{aligned}
&(0.053533190578158, \quad 0.053541555139186)\\
&(0.47323943661919718, \quad 0.473241769271949)\\
&(0.526766595289079, \quad 0.526774959850107)\\
&(0.580299785867238, \quad 0.580303697183099)
\end{aligned} \tag{4-12}$$

若考虑 $a_3 = 131$，$\frac{u}{m}$还必须位于以下区间上

$$C_j = \left(\frac{j}{131}, \frac{j}{131} + \frac{1}{131 \times 32}\right), \quad j = 1,2,\cdots,130$$

则$\frac{u}{m}$必须位于(4-12)的 4 个区间和 $\bigcup_{j=1}^{130} C_j$ 的交集上，于是再将$\frac{u}{m}$搜索区间缩至以下的

2 个

$$(0.053533190578158,\quad 0.053541555139186)$$
$$(0.526766595289079,\quad 0.526774959850107)$$

进而考虑 a_i-318，$\frac{u}{m}$还必须落入如下的区间上

$$D_l=\left(\frac{l}{318},\frac{l}{318}+\frac{1}{318\times 16}\right),\quad l=1,2,\cdots,317$$

将$\frac{u}{m}$的搜索区间进一步缩小为

$$(0.0535331905178158,\quad 0.053541555139186) \tag{4-13}$$

如何确定整数 u 和 m 使得$\frac{u}{m}$落入(4-13)区间呢？在后面将给出一般的方法。对于本例至少有

(i) $u=53,m=990$

通过变换 $b_i=53a_i(\mathrm{mod}\ 990),i=1,2,\cdots,8$，得超递增序列

$$1,3,13,24,49,123,225,505$$

例如，$53\times 467=24751\equiv 1(\mathrm{mod}\ 990)$，$53\times 355=18815\equiv 5(\mathrm{mod}\ 990)$等。

(ii) $u=28,m=523$ 得

$$1,3,7,13,26,65,119,269$$

4.10 LLL 算法

LLL 算法是指由 A. K. Lenstra，H. W. Lenstra 和 L. Lovasz 提出的一种基归约算法，对于背包公钥密码系统的沙米尔攻击以及后面的拉格尼阿斯 粤得尼兹科(Lagarias Odlyzko)攻击、勃里克尔(Brickell)攻击都用到它。LLL 算法有的简称为 L^3 算法。

4.10.1 格 L

设 $b_1,b_2,\cdots,b_n\in R^n$，$L$ 为 n 维空间 R^n 的子集。

$$L\triangleq\left\{\sum_{i=1}^{n}z_ib_i\mid z_i\in Z\right\}=\sum_{i=1}^{n}Zb_i \tag{4-14}$$

其中 R 为实数集合，Z 为整数集合，称 L 为格。$b_1,b_2,\cdots,b_n$ 为格 L 的一组基，n 是格 L 的秩。格 L 的行列式定义为

$$d(L)=|\det(b_1,b_2,\cdots,b_n)| \tag{4-15}$$

任给 n 个线性无关的向量 $b_1,b_2,\cdots,b_n\in R^n$ 可使用下述的 Gram Schmidt 正交化过程获得一组两两正交的向量

$$b_2^*,b_3^*,\cdots,b_n^*$$
$$b_1^*=b_1$$
$$b_j^*=b_j-\sum_{i=1}^{j-1}\mu_{ij}b_i^*\quad j>1 \tag{4-16}$$

其中 $$\mu_{ij}=(b_i\cdot b_j^*)/(b_j^*\cdot b_j^*)\quad i>j \tag{4-17}$$

上式中的"·"是通常定义下的内积。

这里 b_i^* 是 b_i 在子空间 $\sum_{k=1}^{i}Rb_k$ 的正交补空间上的投影，显然有 $\sum_{k=1}^{i-1}Rb_k=\sum_{k=1}^{i-1}Rb_k^*$。以后为了方便起见，用 V_i 表示由 $b_1,b_2,\cdots,b_{i-1}$ 构成的子空间 $\sum_{k=1}^{i-1}Rb_k$，$\sum_{k=1}^{i-1}Rb_k$ 的正交补空间用 V_i 表示，故 b_i^* 是 b_i 在 V_i 上的投影。

如果格 L 的一组基 $b_1,b_2,\cdots,b_n$ 同时满足以下两个条件

$$|\mu_{ij}|\leqslant\frac{1}{2}\quad i>j \tag{4-18}$$

$$|b_i^*+\mu_{i,i-1}b_{i-1}^*|^2\geqslant\frac{3}{4}|b_{i-1}^*|^2,\quad i>1 \tag{4-19}$$

则称 $b_1,b_2,\cdots,b_n$ 为格 L 的一组归约基。"$|\ \ |$"表示通常的欧几里得长度。

注意：这里 $b_i^*+\mu_{i,i-1}b_{i-1}^*$ 和 b_{i-1}^* 实际上分别是 b_i 和 b_{i-1} 在 V_{i-1} 上的投影。

例 4.6 $b_1=(1,1,1),b_2=(1,1,-1),b_3=(-1,1,1)$

令

$$b_1^*=b_1=(1,1,1)$$

$$\mu_{21}=(b_2\cdot b_1^*)/(b_1^*\cdot b_1^*)=1/3$$

$$b_2^*=b_2-\mu_{21}b_1^*=(1,1,-1)-\frac{1}{3}(1,1,1)$$

$$=\frac{1}{3}(2,2,-4)$$

$$\mu_{31}=(b_3\cdot b_1^*)/(b_1^*\cdot b_1^*)=\frac{1}{3}$$

$$\mu_{32}=(b_3\cdot b_2^*)/(b_2^*\cdot b_2^*)=\left(-\frac{4}{3}\right)\Big/\left(\frac{24}{9}\right)=-\frac{1}{2}$$

$$b_3^*=b_3-\mu_{31}b_1^*-\mu_{32}b_2^*$$

$$=(-1,1,1)-\frac{1}{3}(1,1,1)+\frac{1}{2}\left(\frac{2}{3},\frac{2}{3},\frac{4}{3}\right)$$

$$=(-1,1,0)$$

不难发现 b_1,b_2,b_3 还是一组归约基。

4.10.2 相关定理

下面假定 $b_1,b_2,\cdots,b_n$ 是格 $L\in R^n$ 的一组归约基，$b_1^*,b_2^*,\cdots,b_n^*$ 是由格朗姆-斯密特正交化过程定义的。

定理 4.7

(i) $$|b_j|^2\leqslant 2^{i-1}|b_i^*|^2,\quad 1\leqslant j\leqslant i\leqslant n \tag{4-20}$$

(ii) $$d(L)\leqslant\prod_{i=1}^{n}|b_i|\leqslant 2^{n(n-1)/4}d(L) \tag{4-21}$$

(iii) $$|b_1| \leqslant 2^{(n-1)/4} d(L)^{1/n} \tag{4-22}$$

证明 (i) 由(4-19)有

$$\begin{aligned}
&|b_i^* + \mu_{i,i-1} b_{i-1}^*|^2 \\
&= (b_i^* + \mu_{i,i-1} b_{i-1}^*) \cdot (b_i^* + \mu_{i,i-1} b_{i-1}^*) \\
&= (b_i^* \cdot b_i^*) + \mu_{i,i-1}(b_{i-1}^* \cdot b_i^*) + \mu_{i,i-1}(b_i^* \cdot b_{i-1}^*) + \mu_{i,i-1}^2 (b_{i-1}^* \cdot b_{i-1}^*) \\
&= |b_i^*|^2 + \mu_{i,i-1}^2 |b_{i-1}^*|^2 \\
&\geqslant \frac{3}{4} |b_{i-1}^*|
\end{aligned}$$

由于 $$|\mu_{i,i-1}| \leqslant \frac{1}{2}$$

所以 $$|b_i^*|^2 \geqslant \left(\frac{3}{4} - \mu_{i,i-1}^2\right) |b_{i-1}^*|^2 \geqslant \frac{1}{2} |b_{i-1}^*|^2$$

对上式进行递归可得

$$|b_i^*|^2 \geqslant \frac{1}{2} |b_{i-1}^*|^2 \geqslant \frac{1}{2^2} |b_{i-2}^*|^2 \geqslant \cdots \geqslant \frac{1}{2^i} |b_{i-i}^*|^2$$

令 $$i - l = j, \quad l = i - j$$

则 $$|b_j^*|^2 \leqslant 2^{i-j} |b_i^*|^2, \quad 1 \leqslant j \leqslant i \leqslant n \tag{4-23}$$

由(4-10)得

$$\begin{aligned}
|b_j|^2 &= \left| b_j^* + \sum_{k=1}^{j-1} \mu_{j,k} b_k^* \right|^2 \\
&= |b_j^*|^2 + \sum_{k=1}^{j-1} \mu_{j,k}^2 |b_k^*|^2 \\
&\leqslant |b_j^*|^2 + \sum_{k=1}^{j-1} \frac{1}{4} |b_j^*|^2 \\
&\leqslant |b_j^*|^2 + \frac{1}{4} \sum_{k=1}^{j-1} 2^{j-k} |b_j^*|^2 \\
&= \left[1 + \frac{1}{4}(2^j - 2)\right] \cdot |b_j^*|^2 \\
&\leqslant 2^{j-1} |b_j^*|^2 \\
&\leqslant 2^{j-1} \cdot 2^{i-j} |b_i^*|^2 \\
&= 2^{i-1} |b_i^*|^2 \qquad (1 \leqslant j \leqslant i \leqslant n)
\end{aligned}$$

(ii) $$\begin{aligned}
d(L) &= |\det(b_1, b_2, \cdots, b_n)| \\
&= |\det(b_1^*, b_2, \cdots, b_n)| \\
&= |\det(b_1^*, b_2^* + \mu_2 b_1^*, \cdots, b_n)| \\
&= |\det(b_1^*, b_2^*, \cdots, b_n)| \\
&= \cdots \\
&= |\det(b_1^*, b_2^*, \cdots, b_n^*)|
\end{aligned}$$

由于 $b_1^*, b_2^*, \cdots, b_n^*$ 两两正交，故

$$d(L) = \prod_{i=1}^{n} | b_i^* |$$

由于 b_i^* 是在 V_i 上的投影，所以 $|b_i^*| \leqslant |b_i|$。故

$$d(L) = \prod_{j=1}^{n} | b_j^* | \leqslant \prod_{j=1}^{n} | b_j |$$

由(4-20)可得

$$| b_j |^2 \leqslant 2^{j-1} | b_j^* |^2$$

故

$$\prod_{j=1}^{n} | b_j |^2 \leqslant \prod_{j=1}^{n} 2^{j-1} | b_j^* |^2$$

$$= 2 \sum_{j=1}^{n} (i-1)^n \cdot \prod_{j=1}^{n} | b_j^* |^2$$

$$= 2^{\frac{n(n-1)}{2}} d(L)^2$$

故

$$\prod_{j=1}^{n} | b_j | \leqslant 2^{n(n-1)/4} d(L)$$

(iii) 在(4-20)中令 $j=1$ 得

$$| b_1 |^2 \leqslant 2^{i-1} | b_i^* |^2$$

$$\prod_{i=1}^{n} | b_1 |^2 = | b_1 |^{2n} \leqslant \prod_{i=1}^{n} 2^{i-1} | b_i^* |^2 = 2^{n(n-1)/2} d(L)^2$$

故

$$|b_1| \leqslant 2^{n(n-1)/4} d(L)^{1/n}$$

定理 4.8 $\forall x \in L, x \neq 0$，则

$$| b_1 |^2 \leqslant 2^{n-1} | x |^2 \tag{4-24}$$

证明 因为 $x \in L$，所以

$$x = \sum_{i=1}^{n} z_i b_i = \sum_{i=1}^{n} z_i^* b_i^*, z_i \in Z, z_i^* \in R$$

令 i 是使 $z_i \neq 0$ 的最大下标，可证 $z_i = z_i^*$。因为

$$x = \sum_{j=1}^{i} z_j b_j$$

$$= z_i b_i + \sum_{j=1}^{i-1} z_j b_j$$

$$= z_i \left(b_i^* + \sum_{j=1}^{i-1} \mu_{i,j} b_i^* \right) + \sum_{j=1}^{i-1} z_i^* b_i^*$$

$$= z_i b_i^* + \sum_{j=1}^{i-1} (\mu_{i,j} z_i + z_j^*) b_j^*$$

所以

$$z_i = z_i^*$$

$$| x |^2 \geqslant z_i^2 | b_i^* |^2 \geqslant | b_i^* |^2 \tag{4-25}$$

由(4-20)得

$$\| b_1 \|^2 \leqslant 2^{i-1} \| b_i^* \|^2 \leqslant 2^{n-1} \| b_i^* \|^2 \leqslant 2^{n-1} \| x \|^2$$

定理 4.9 $x_1, x_2, \cdots, x_t \in L$ 是 t 个线性无关的向量，$1 \leqslant t \leqslant n$，则

$$\| b_j \|^2 \leqslant 2^{n-1} \max\{ \| x_1 \|^2, \| x_2 \|^2, \cdots, \| x_t \|^2 \} \quad 1 \leqslant j \leqslant t$$

证明 因为 $x_j \in L$，所以

$$x_j = \sum_{i=1}^{n} z_{ij} b_i \qquad j = 1, 2, \cdots, t$$

对确定的 j，令 $i(j)$ 表示使 $z_{ij} \neq 0$ 的最大下标。由(4-25)得

$$\| x_j \|^2 \geqslant \| b_{i(j)} \|^2$$

将 x_j 重新排列使得

$$i(1) \leqslant i(2) \leqslant \cdots \leqslant i(t)$$

则有

$$j \leqslant i(j), \quad j = 1, 2, \cdots, t$$

否则若存在 k，$k > i(k)$，则 $x_1, x_2, \cdots, x_k$ 都在 $\sum_{k=1}^{i(k)} R b_k$ 中，这和假定 $x_1, x_2, \cdots, x_k$ 线性无关相矛盾。由(4-20)和(4-25)得

$$\| b_j \|^2 \leqslant 2^{n-1} \| b_{i(j)} \|^2 \leqslant 2^{n-1} \| b_{i(j)} \|^2 \leqslant 2^{n-1} \| x_j \|^2, \quad 1 \leqslant j \leqslant t$$

4.10.3 LLL 算法详细介绍

下面介绍一种从格 L 的一组基 $b_1, b_2, \cdots, b_n$ 开始，构造一组归约基的算法，也就是有的也叫 L^3 算法。因三位发明者的第一个字母都是 L 而得名。

首先，利用格朗姆-斯密特正交化方法计算 b_i^* 和 μ_{ij}，$1 \leqslant i \leqslant n$，$1 \leqslant j < i$。然后开始一个归约过程。在这个归约过程中，$b_1, b_2, \cdots, b_n$ 可能改变，然而始终保持 L 的一组基。如果某个 b_i 发生变化，则相应地 b_i^* 和 μ_{ij} 也改变，但(4-10)和(4-11)依然成立，每次归约的结果总要使得一部分基满足下面两个条件

$$| \mu_{ij} | \leqslant \frac{1}{2}, \quad 1 < j < i \leqslant k \tag{4-26}$$

$$\| b_i^* + \mu_{i,i-1} b_{i-1}^* \|^2 \geqslant \frac{3}{4} \| b_{i-1}^* \|^2 \quad 1 < i < k$$

从 $k=2$ 开始，上述条件自然成立。若 $k=n+1$，则归约过程终止，便获得一组归约的基 $b_1, b_2, \cdots, b_n$。

$k>1$ 时，若 $|\mu_{k,k-1}| \leqslant \frac{1}{2}$ 不成立，令 r 是与 $\mu_{k,k-1}$ 最接近的整数，用 $b_k - r b_{k-l}$ 来取代 b_k，对 $j < k-l$ 用 $\mu_{kj} - r\mu_{k-l,j}$ 代替 μ_{kj}，用 $\mu_{k,k-r} - r$ 代替 $\mu_{k,k-1}$，其他的 μ_{ij} 和 b_i^* 不变，使 $|\mu_{k,k-1}| \leqslant \frac{1}{2}$ 在归约过程中，每一步都可能遇上两种情况：

(i) $k \geqslant 2$ 且 $|b_k^* + \mu_{k,k-1} b_{k-1}^*|^2 < \frac{3}{4} |b_{k-1}^*|^2$

这时令 b_k 和 b_{k-1} 互换，其他的 b_i 保持不变，重新计算 b_{k-1}^*，b_k^*，$\mu_{k,k-1}$，$\mu_{k-1,j}$，μ_{kj}，$\mu_{i,k-1}$，μ_{ik}，其中 $j > k-1$，$i > k$。

注意：$b_k^*{}_{-1}$和$b_k^*-\mu_{k,k-1}b_{k-1}^*$也互换，则有$b_k^*{}_{-1}$小于原来的$b_{k-1}^*$的$\frac{3}{4}$，然后将$k$减1，继续进行迭代。

(ii) $k=1$或$|b_k^*+\mu_{k,k-1}b_{k-1}^*|^2\geqslant\frac{3}{4}|b_{k-1}^*|^2$

这时先查出有$\mu_{kl}>\frac{1}{2}$，令b_k用b_k-rb_l取代之，r是最接近μ_{kl}的整数。反复进行，直到所有的μ_{kj}都满足条件为止，然后k加1。

下面给出全部算法：其中$B_i\equiv(b_i^*\cdot b_i^*)$

$$\left.\begin{aligned}&b_i^*:=b_i\\&\mu_{ij}:=(b_i\cdot b_j^*)/B_j\\&b_i^*:=b_i-\mu_{ij}b_j^*\end{aligned}\right\}\text{for } j:=1 \text{ to } i-1\Bigg\}\text{for } i:=1 \text{ to } n$$

$$B_i:=b_i^*\cdot b_i^*$$

$k:=2$

(i) for $l:=k-1$ 执行(*)

if $B_k<\left(\frac{3}{4}-\mu_{k,k-1}^2\right)B_{k-1}$, go to (ii)

for $l:=k-2$ down to 1 执行(*)

if $k=n$ 终止

$k:=k+1$; go to (I)

(ii) $\mu:=\mu_{k,k-1}$; $B:=B_k+\mu^2B_{k-1}$; $\mu_{k,k-1}:=\mu B_{k-1}/B$;

$B_k:=B_{k-1}B_k/B$; $B_{k-1}:=B$

$$\begin{bmatrix}b_{k-1}\\b_k\end{bmatrix}:=\begin{bmatrix}b_k\\b_{k-1}\end{bmatrix}$$

$$\begin{bmatrix}\mu_{k-1,j}\\\mu_{kj}\end{bmatrix}:=\begin{bmatrix}\mu_{kj}\\\mu_{k-1,j}\end{bmatrix}\text{ for } j:=1 \text{ to } k-2$$

$$\begin{bmatrix}\mu_{i,k-1}\\\mu_{ik}\end{bmatrix}:=\begin{bmatrix}1&\mu_{k,k-1}\\0&1\end{bmatrix}\begin{bmatrix}0&1\\1&-\mu\end{bmatrix}\begin{bmatrix}\mu_{i,k-1}\\\mu_{ik}\end{bmatrix}$$

if $k\geqslant 2$ then $k:=k-1$ goto (i)

(*) if $|\mu_{kl}|>\frac{1}{2}$ then

$$\begin{cases}r:=\text{离}\ \mu_{kl}\ \text{最近的整数}\\b_k:=b_k-rb_{k-1}\\\mu_{kj}:=\mu_{kj}-r\mu_{lj}\text{, for } j:=1 \text{ to } l-1\\\mu_{kl}:=\mu_{kl}-r\end{cases}$$

在上节中已将对MH背包公钥系统的攻击归结为解一组线性不等式。

$$-\varepsilon_0\leqslant\frac{p}{a_1}-\frac{q}{a_2}\leqslant\varepsilon_0'\qquad\begin{matrix}1\leqslant p\leqslant a_1-1\\1\leqslant q\leqslant a_2-1\end{matrix}$$

$$-\epsilon_2 \leqslant \frac{p}{a_1} - \frac{r}{a_3} \leqslant \epsilon_3' \quad 1 \leqslant r \leqslant a_3 - 1$$

$$\cdots \qquad\qquad \cdots$$

可将上述不等式转化为

$$|p_i - qw_i| \leqslant \epsilon, \qquad 1 \leqslant i \leqslant n$$

$$1 \leqslant q \leqslant 2^{n(n+1)/4}\epsilon^{-n}$$

其中 ω_i 是有理数，$p_1, p_2, \cdots, p_n, q$ 是整数，$0<\epsilon<1$。可令

$$b_1 = (1,0,\cdots,0,-w_1)$$

$$b_2 = (0,1,\cdots,0,-w_2)$$

$$\vdots$$

$$b_n = (0,0,\cdots,1,-w_n)$$

$$b_{n+1} = (0,0,\cdots,0,2^{-n(n+1)/4}\epsilon^{n+1})$$

通过 LLL 算法可得一归约基 $b_1^*, b_2^*, \cdots, b_{n+1}^*$。由定理 4.7 可知

$$|b_1^*|^2 \leqslant 2^{n/4} d(L)^{1/(n+1)}$$

$$d(L) = \begin{vmatrix} 1 & 0 & \cdots & 0 & -w_1 \\ 0 & 1 & \cdots & 0 & -w_2 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -w_n \\ 0 & 0 & \cdots & 0 & 2^{-(n+1)n/4}\xi^{n+1} \end{vmatrix}$$

$$= 2^{n(n+1)/4}\epsilon^{n+1}$$

则

$$2^{n/4} d(L)^{1/(n+1)} = 2^{n/4}(2^{n(n+1)/4}\epsilon^{n+1})^{1/(n+1)}$$

$$= 2^{n/4} 2^{-n/4} \cdot \epsilon = \epsilon$$

由于 $b_1^* \in L$，故

$$b_1^* = \sum_{i=1}^{n} p_i b_i + q b_{n+1}, \quad p_i, q \in Z$$

故

$$b_1^* = (p_1 - qw_1, p_2 - qw_2, \cdots, p_n - qw_n, q2^{-n(n+1)/4}\epsilon^{n+1})$$

故

$$|p_i - qw_i| \leqslant \epsilon, \quad i = 1,2,\cdots,n$$

$$|q2^{-n(n+1)/4}\epsilon^{n+1}| < \epsilon$$

$$|q| \leqslant 2^{n(n+1)/4}\epsilon^{-n}$$

最后有两种可能：q 或是正整数，或是负的。若属后者，b_1^* 用 $-b_1^*$ 代之。

4.11 Lagarias-Odlyzko-Brickell 攻击

Shamir 攻击方法发表后，Merkle 和 Hellman 试图多次重复利用它们构造公开的背包密钥的 MH 算法，企图使超递增序列的“原形”隐蔽得更深一些。使 Shamir 的攻击失败，但过了两年之后，Lagarias-Odlyzko 和 Brickell 几乎同时提出攻击方法，虽然出发点各异，但结论非常接近，可谓殊途同归。

首先定义背包序列 $a_1,a_2,\cdots,a_n$ 的背包密度

$$D=n/\log_2(\max a_i)$$

这里 n 实际是明文 $m=m_1m_2\cdots m_n$ 的比特数。

Lagarias Odlyzko 和 Brickell 都证明了对 $D<0.645$ 的背包公钥问题可以破译。MH 背包公钥一般都是 D 很小，密文膨胀得比较大。

背包密度可以近似地看作是

$$D\approx\frac{\text{明文的比特数}}{\text{密文的平均比特数}}$$

例如对于背包 $\{1,2,2^2,\cdots,2^{n-1}\}$ 有

$$D=n/\log_2(\max a_k)=n/(n-1)\approx 1$$

又如背包 $\{1979,2726,1914,3016,3879\}$ 的

$$D=5/\log_2\{3879\}=5/11.924\approx 0.419$$

一般说来，信息率越高，隐藏信息越难隐蔽，所谓信息率就是明文长度和码文长度之比，所以攻破 $D<0.645$ 的背包公钥系统的方法较之 Shamir 方法更具一般性。

下面仅叙述 Lagarias-Odlyzko 方法。假设已知公开的背包序列 $a_1,a_2,\cdots,a_n$ 及 b，$a_i\in Z,i=1,2,\cdots,n$，求解

$$\sum_{i=1}^{n}a_ix_i=b,x_i=0,1,\quad i=1,2,\cdots,n$$

S_1. 取格 L 的一组基：

$$b_1=(1,0,0,\cdots,0,\quad a_1)$$
$$b_2=(0,1,0,\cdots,0,-a_2)$$
$$b_n=(0,0,0,\cdots,0,-a_n)$$
$$b_{n+1}=(0,0,0,\cdots,b)$$

S_2. 利用 L^3 算法求 L 的归约基

$$b_1^*,b_2^*,\cdots,b_n^*,b_{n+1}^*$$

S_3. 若任意 $b_i^*=(b_{i1}^*,b_{i2}^*,\cdots,b_{i,n+1}^*)$ 有所有 $b_{ij}^*=0$ 或常数 $\lambda,1\leqslant j\leqslant n$

$$x_j=\frac{1}{\lambda}b_{ij}^*\quad i\leqslant j\leqslant\lambda$$

给出的解，则停止．否则转 S4.

S_4. $b-\sum_{j=1}^{n}a_j=b$，重复 S1 到 S3，得解 $(x_1,x_2,\cdots,x_n)$．原来背包问题的解为

$$(1-x_1,1-x_2,\cdots,1-x_n)$$

例 4.7 已知超递增序列

$b_1=128,b_2=143,b_3=275,b_4=562,b_5=1136,b_6=2531,b_7=6305,$

$b_8=11590,b_9=23145,b_{10}=50308,b_{11}=100697,b_{12}=239213$

取 $m=978426,w=4865$，将 $\{b_i\}$ 转换为

$$a_1=00622720,a_2=00695695,a_3=00359449$$

$$a_4 = 00777278, a_5 = 00731810, a_6 = 00572203$$
$$a_7 = 00342619, a_8 = 00517768, a_9 = 00081435$$
$$a_{10} = 00141920, a_{11} = 00677905, a_{12} = 00422731$$

取格 L 的一组基如下：

$$b_1 = (1,0,0,0,0,0,0,0,0,0,0,0,-622720)$$
$$b_2 = (0,1,0,0,0,0,0,0,0,0,0,0,-695695)$$
$$b_3 = (0,0,1,0,0,0,0,0,0,0,0,0,-359449)$$
$$b_4 = (0,0,0,1,0,0,0,0,0,0,0,0,-777278)$$
$$b_5 = (0,0,0,0,1,0,0,0,0,0,0,0,-731810)$$
$$b_6 = (0,0,0,0,0,1,0,0,0,0,0,0,-572203)$$
$$b_7 = (0,0,0,0,0,0,1,0,0,0,0,0,-342619)$$
$$b_8 = (0,0,0,0,0,0,0,1,0,0,0,0,-517768)$$
$$b_9 = (0,0,0,0,0,0,0,0,1,0,0,0,-81435)$$
$$b_{10} = (0,0,0,0,0,0,0,0,0,1,0,0,-141920)$$
$$b_{11} = (0,0,0,0,0,0,0,0,0,0,1,0,-677905)$$
$$b_{12} = (0,0,0,0,0,0,0,0,0,0,0,1,-422731)$$
$$b_{13} = (0,0,0,0,0,0,0,0,0,0,0,0,2764551)$$

其中 $b=2764551$。

用 L^* 算法得归约基

$$b_1^* = (-1, 0, -2, 1, 1, 0, -2, 1, 0, 0, 0, 0, 0)$$
$$b_2^* = (1, 0, 3, 0, -1, -2, -1, 1, 0, 0, 0, 0, 0)$$
$$b_3^* = (-1, 1, -2, 3, -1, 0, 0, -2, 1, 0, 0, 0, 0)$$
$$b_4^* = (-1, -2, 1, 0, 0, 0, -1, 2, 0, 2, 1, 0, -1)$$
$$b_5^* = (-1, -1, 1, 1, -1, 1, 0, 0, -1, 0, 0, 1, -1)$$
$$b_6^* = (1, -1, 0, -1, -1, 2, 1, 1, 0, 0, 0, -1, 1)$$
$$b_7^* = (1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0)$$
$$b_8^* = (0, 3, -2, 1, 0, 1, 0, -1, 0, 1, 0, 1, 0)$$
$$b_9^* = (2, 0, 2, -1, 1, 1, -2, 0, 0, -1, 1, 1, 0)$$
$$b_{10}^* = (0, -2, -2, 1, 1, 1, 1, -2, -1, 2, -1, -1, 0)$$
$$b_{11}^* = (1, -2, -1, -1, 0, 0, 0, 1, -1, -1, 3, -1, 0)$$
$$b_{12}^* = (1, 0, -1, -1, -2, 1, 0, 0, 2, 1, 1, 1, -2)$$
$$b_{13}^* = (0, -1, -1, 1, 3, 1, 1, 1, -2, 0, 0, -1, -2)$$

从 b_7^* 可知，其分量或等于 0 或等于常数 1。

令
$$x_1 = x_4 = x_7 = x_8 = x_9 = x_{12} = 1$$
$$x_2 = x_3 = x_5 = x_6 = x_{10} = x_{11} = x_{12} = 0$$

代入验证可得

$$a_1+a_4+a_7+a_8+a_9+a_{12}=2764551$$

满足要求。所以 100100111001 是对应于密文 2764551 的明文。

4.12 利用传统密码建立网络保密通信的若干办法

(1) 公钥密码的思想无疑是先进的，但时间不长，还没有一个足够可靠的公钥系统可供使用。RSA 虽然很有希望，但处理 200 位十进制数的幂运算效率较低，公钥密码由于要将相当大一部分关于密码的信息予以公开，它势必对系统产生影响，为此要付出代价来补偿。

公钥一出现便显示出它的优势，有一种咄咄逼人取而代之的架势。实际上公钥还在发展中，它和传统密码学构成近代密码学的不可分割的组成部分。下面介绍将两者结合起来取长补短的方法。

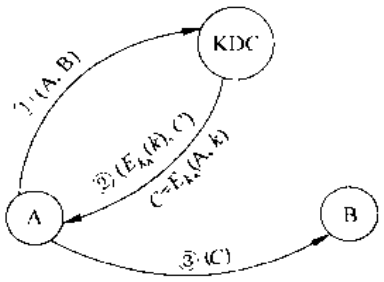


图 4.3

假定存在一个通信网络系统，这个系统有一密钥分配中心 KDC(Key Distribution Center)，它专门负责管理通信密钥，每一用户都在 KDC 存有各自的密钥。这些用户密钥 k_A，k_B…，都用只有 KDC 掌握的密钥予以加密后保存。若用户 A 欲和用户 B 秘密通信，则 A 先向 KDC 提出申请，为此送去信息(A,B)。KDC 便随机产生一通信密钥 k，分别用 A 和 B 的密钥 k_A 和 k_B 加密得 $E_{k_A}(k)$，$E_{k_B}(k)$ 并送给 A，A 收到后，从 k_A(A)解密得通信密钥 k，再利用 k 加密信息 m，得密文 $C=E_k(m)$，连同 $E_{k_B}(k)$ 送给 B，B 收到后先利用自己掌握的密钥，从 $E_{k_B}(k)$ 获得 k，再从密文 C 恢复明文 m。

$$m=D_k(C)$$

这个过程如图 4.3 所示。

在这个系统中 KDC 是核心，它必须保证绝对安全，通过它利用传统密码建立起密码通信，一切通信都要通过 KDC，但当系统较大时不胜其烦，也是一弊。

这样建立起来的保密通信系统开销是很大的，效率也很低，还可以改为由密钥管理中心定时地更换通信密钥分发到各终端作为规定时间内保密通信用，虽然不要求如上所述的分三步进行，但问题仍不少。

(2) 混合密码。公钥密码的长处在于密钥管理方便，但效率低，公钥的短处正好是传统密码的长处，反之亦然。如何取长补短，建立起有效的公钥系统，正是混合密码研究的核心，即通过公钥传送通信密钥，然后通过传统密码加密，以达到保密通信的目的。

公钥密码还有一个弱点，即若公钥文件被篡改，即 A 的公钥文件中，B 的公钥被篡改为 C 的公钥，A 不易发觉，则 A 对 B 的通信，C 可获悉，即公钥被攻破，Shamir 建议以用户的身份(如姓名、性别等)做公钥来克服这个缺点。

4.13 公钥密码系统的密钥分配

公钥密码系统以 RSA 为例，每个用户的公钥和私钥显然很难能由各用户自己来产生确定。应有一个公正的机构，假定它仍叫 KDC。KDC 产生一组“密钥源”，它包含“k_e，k_d，$n=pq$”，由用户随机地选择其中一个作为自己的密钥。KDC 保证只生产这些“密钥源”，而不去了解谁选的是哪一组。还可以先由 A 选若干密钥，并用 A 的密钥加密，再送 B，B 用自己的密钥第二次加密，再送 A，A 用自己的密钥解密，再退给 B，B 从中再取一个解密作为自己的“密钥”。当然和扑克游戏一样，要求

$$E_i(E_j(m)) = E_j(E_i(m))。$$

第 5 章　线性反馈移位寄存器(LFSR)和序列密码

5.1　序列的随机性概念

Shannon 证明了一次一密密码体制是不可破的。这一结果给密码学研究以很大的刺激。若能以一种方式产生一随机序列，这一序列由密钥所确定，则利用这样的序列就可进行加密。这个体制的运行如图 5.1 所示。其中⊕是模 2 的加法运算。这样的密码体制称为序列密码体制，有时也称流码，是密码学中最重要的加密方式之一。密钥流产生器实际上是一给定的算法，产生的密钥流通常是 0、1 数据流。流码的称呼由此产生。序列密码可以看成是多表密码的一种，后面将看到这些密钥流有其周期。如果它的周期不大，它将非常类似于 Vigenere 密码。所以希望密钥流有尽可能大的周期，至少应和明文的长度相等，并且具有随机性，使得密码分析者对它无法预测。也就是说，即便截获其中一段，也无法推测后面是什么。如果密钥流是周期的，要完全做到随机这一点是有困难的。严格地说，这样的序列不可能做到随机，只能要求截获比周期短的一段时不会泄露更多信息。这样的序列称为伪随机序列。

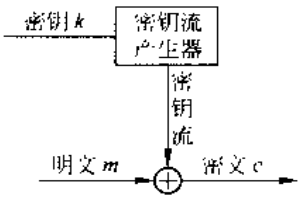


图　5.1

密钥流是 0　1 序列，例如 00110111。这序列前两个数字是 00，称为 0 的 2 游程；接着是 11，是 1 的 2 游程；继之是 0 的 1 游程和 1 的 3 游程。

假定 $s_1 s_2 s_3 \cdots$ 是 0、1 序列，用 $\{s_i\}$ 表示。r 是对于所有的正整数 m，满足 $s_{m+r}=s_m$ 的最小正整数。若存在这样的 r，则称序列 $\{s_i\}$ 为以 r 为周期。若有下列两个子序列

$$s_1, s_2, \cdots, s_r$$

$$s_{1+\tau}, s_{2+\tau}, \cdots, s_{r+\tau}$$

从前一序列后移 τ 位便得到后一序列。若 $s_i=s_{i+\tau}$，则称对应于第 i 位是相同的。这两个子序列中相同的位的数目用 n_τ 表示，不同的位的数目用 $d_\tau=r-n_\tau$ 表示。定义

$$R(\tau)=\frac{n_\tau-d_\tau}{r}$$

为自相关数。

$\tau=0$ 时，显然有 $n_\tau=r$，$d_\tau=0$，$R(0)=1$。

$\tau\neq 0$ 时，$R(\tau)$ 为异相自相关函数。

Golomb 提出 0、1 序列的随机性公设如下：

(1) 若 r 是奇数，则 0—1 序列 $\{s_i\}$ 的一个周期内 0 的个数比 1 的个数多一个或少一个；若 r 是偶数，则 0 的个数与 1 的个数相等。

(2) 在长度为 r 的周期内，1 游程的个数为游程总数的 $\frac{1}{2}$，2 游程的个数占游程总数

的$\frac{1}{2^2}$,…,c 游程的个数占游程总数的$\frac{1}{2^c}$。而且对于任意长度,0 的游程个数和 1 的游程个数相等。

(3) 异相自相关函数是一个常数。

公设(1)和(2)的意义都很明显,(3)意味着通过对序列与其平移后的序列作比较,不能给出其他任何信息。在前面讨论 Vigenere 密码的分析时,曾利用序列的平移得到的信息帮助破译。

5.2 有限状态机

所谓有限状态机由下面 3 部分组成:

(1) 有限状态集 $S=\{s_i,\ i=1,2,\cdots,l\}$;

(2) 有限输入字符集 $A_1=\{A_j^{(1)}\mid j=1,2,\cdots,m\}$和有限输出字符集 $A_2=\{A_k^{(2)}\mid k=1,2,\cdots,n\}$;

(3) 转移函数

$$A_k^{(2)}=f_1(s_i,A_j^{(1)}),\quad s_h=f_2(s_i,A_j^{(1)})$$

即当处于 s_i 状态时,接收输入 $A_j^{(1)}$ 后,变为 s_h 状态,并且输出 $A_k^{(2)}$。

例 5.1 $S=\{s_1,s_2,s_3\}$, $A=\{A_1^{(1)},A_2^{(1)},A_3^{(1)}\}$, $A_2=\{A_1^{(2)},A_2^{(2)},A_3^{(2)}\}$。转移函数见表 5.1。

表 5.1

f_1	$A_1^{(1)}$	$A_2^{(1)}$	$A_3^{(1)}$
s_1	$A_1^{(2)}$	$A_3^{(2)}$	$A_2^{(2)}$
s_2	$A_2^{(2)}$	$A_1^{(2)}$	$A_3^{(2)}$
s_3	$A_3^{(2)}$	$A_2^{(2)}$	$A_1^{(2)}$

f_2	$A_1^{(1)}$	$A_2^{(1)}$	$A_3^{(1)}$
s_1	s_2	s_1	s_3
s_2	s_3	s_2	s_1
s_3	s_1	s_3	s_2

上面这一有限状态机也可以用图 5.2 表示。图 5.2 中每一顶点表示一种状态。若状态 s_i 在输入 $A_j^{(1)}$ 时转为状态 s_j,且输出一字符 $A_j^{(2)}$,则从 s_i 引一弧指向 s_j,并在弧上标以 $(A_j^{(1)},A_j^{(2)})$,括号里的第 1 个符号为输入字符,第 2 个字符为输出字符。

对于上面给出的状态机,若输入序列为 $A_1^{(1)}A_2^{(1)}A_1^{(1)}A_3^{(1)}A_3^{(1)}A_1^{(1)}$,初始状态为 s_1,将依次得到状态序列

$$s_1\,s_2\,s_2\,s_3\,s_2\,s_2\,s_3$$

依次输出字符序列

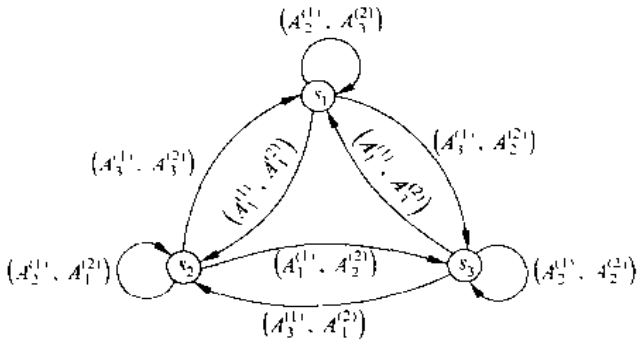


图 5.2

$$A_3^{(2)}A_1^{(2)}A_2^{(2)}A_1^{(2)}A_1^{(2)}A_2^{(2)}$$

上述过程列于表 5.2。

表 5.2

时间	状态	输入	输出
1	s_1	$A_3^{(1)}$	$A_1^{(2)}$
2	s_2	$A_2^{(1)}$	$A_1^{(2)}$
3	s_2	$A_3^{(1)}$	$A_2^{(2)}$
4	s_3	$A_3^{(1)}$	$A_1^{(2)}$
5	s_4	$A_2^{(1)}$	$A_1^{(2)}$
6	s_4	$A_1^{(1)}$	$A_2^{(2)}$

即根据有限状态机的转移函数，从状态 s_1 输入 $A_3^{(1)}$，输出 $A_1^{(2)}$，变为状态 s_2；从状态 s_2 输入 $A_2^{(1)}$，输出 $A_1^{(2)}$，状态不变仍然保持为状态 s_2；依此类推。

一个无限序列$\{s_i\}$由一有限序列不断重复而成时，则这个序列是周期的。即存在一正整数 r，使得对所有的正整数 t，有

$$s_{t+r} = s_t$$

也即无限序列$\{s_i\}$是如下构成的

$$s_0 s_1 s_2 \cdots s_{r-1} s_0 s_1 s_2 \cdots s_{r-1} s_0 s_1 s_2 \cdots$$

具有此性质的最小正整数 r，称作这个序列的周期。$s_i s_1 \cdots s_{r-1}$ 称作这个周期序列的生成序列。

若序列$\{s_i\}$除开始若干项后的其余部分是周期序列，则此序列称为准周期序列。

定理 5.1 若有限状态机的输入是准周期序列，则输出也是准周期序列。

证明 假定输入序列$\{a_i\}$去掉前 m 项 $a_0 a_1 a_2 \cdots a_{m-1}$ 后是以 r 为周期的序列；S 是状态集，$|S| = l$，对应的状态序列是$\{s_i\}$；对应的输出序列是$\{b_i\}$。若整数 $n \geqslant m$，$\{s_i\}$的子序列

$$s_n s_{n+r} s_{n+2r} \cdots s_{n+lr}$$

中的元素不可能都不相同。设 $0 \leqslant h < k \leqslant l$，则

$$s_{n+kr} = s_{n+kr}$$

令 $p=n+hr$，$c=k-h$，于是，$n+kr=p+cr$，$s_p=s_{p+c}$。由假设 $a_i=a_{i-jr}$，i,j 是正整数 $i\geqslant m$。显然 $a_p=a_{p+c}$。以下证明 $\{b_i\}$ 是周期性的，$i\geqslant p$。只要证明 $b_{p+j}=b_{p+c+j}$，$s_{p+j}=s_{p+cr+j}$ 对一切非负整数 j 成立即可。对 j 进行数学归纳法，$j=0$ 时，

$$b_p=f_1(s_p,a_p)=f_1(s_{p+c},a_{p+c})=b_{p+c}$$

假设对 $j-1$ 有 $s_{p+j-1}=s_{p+cr+j-1}$，则

$$s_{p+j}=f_2(s_{p+j-1},a_{p+j-1})=f_2(s_{p+cr+j-1},a_{p+cr+j-1})=s_{p+cr+j}$$

$$b_{p+j}=f_1(s_{p+j},a_{p+j})=f_1(s_{p+cr+j},a_{p+cr+j})=b_{p+cr+j}$$

故去掉前 p 项后 $\{b_i\}$ 是周期的，即 $\{b_i\}$ 是准周期的。

称周期为 1 的序列为常数序列。

推论 5.1 若有限状态机的输入为准常数序列，则输出是准周期序列。

5.3 反馈移位寄存器

下面介绍的移位寄存器便是有限状态机的典型例子。

设 $\{x_i\}$ 是 0 1 序列。0，1 两个数对于模 2 加和普通数乘构成一个域，这个域用 $GF(2)$ 表示。图 5.3 是反馈移位寄存器的流程图，信号流从左向右。图中标有 $a_1,a_2,\cdots,a_{n-1},a_n$ 的小方框表示二值(0,1)存储单元，可以是一个双稳触发器。这 n 个二值存储单元称为该反馈移位寄存器的级。在任一时刻，这些级的内容构成该反馈移位寄存器的状态。这个反馈移位寄存器的状态对应于一个 $GF(2)$ 上的 n 维向量，共有 2^n 种可能的状态。每个时刻的状态可用 n 长序列

$$a_1,a_2,\cdots,a_n$$

或 n 维行向量

$$(a_1,a_2,\cdots,a_n)$$

表示。其中 a_i 为当时第 i 级存储器中的内容。

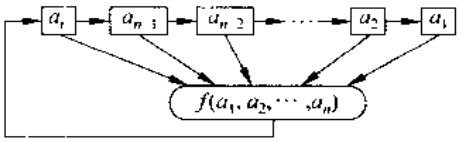


图 5.3

在主时钟确定的周期区间上，每一级存储器 a_i 都将其内容向下一级 a_{i-1} 传递，并根据寄存器当时的状态计算 $f(a_1,a_2,\cdots,a_n)$ 作为 a_n 的下一时间的内容。其中反馈函数 $f(a_1,a_2,\cdots,a_n)$ 是 n 元布尔函数。所以在时钟的每一脉冲下，总是从一个状态转移到下一状态，见表 5.3 与图 5.4，或以图 5.5 所示。

表 5.3

状态			下一状态		
a_3	a_2	a_1	a_3	a_2	a_1
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	1
0	1	1	0	0	1
1	0	0	0	1	0
1	0	1	0	1	0
1	1	0	1	1	1
1	1	1	1	1	1

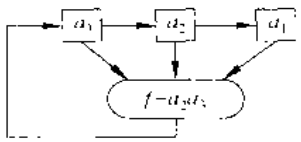


图 5.4

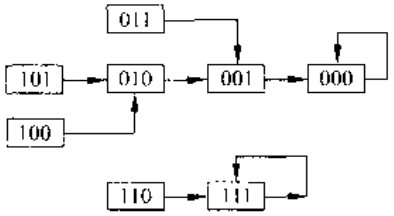


图 5.5

图 5.6 是更一般的反馈移位寄存器。这个反馈移位寄存器有输入和输出。

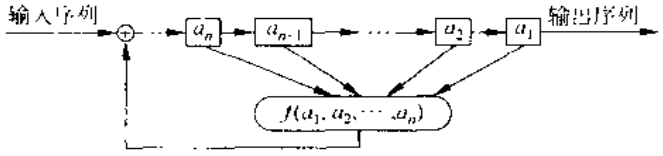


图 5.6

若反馈函数

$$f(a_1,a_2,\cdots,a_n)=c_na_1\oplus c_{n-1}a_2\oplus\cdots\oplus c_1a_n$$

其中常数 $c_i=0,1$，$\oplus$ 是模 2 加法。这个反馈函数是 $a_1,a_2,\cdots,a_n$ 的线性函数。线性的反馈移位寄存器有时用 LFSR 记它，LFSR 是 Linear Feedback Shift Register 的缩写。这样的线性函数有 2^n 种，而 n 元布尔函数有 2^{2^n} 种。

在图 5.7 中分别用相应开关的闭合或断开来实现常数 $c_i=1$ 或 0。令 $a_i(t)$ 表示 t 时刻第 i 级的内容。对于 $i=1,2,\cdots,n-1$ 有以下关系成立。

$$a_i(t+1)=a_{i+1}(t)$$

$$a_n(t+1)=c_na_1(t)\oplus c_{n-1}a_2(t)\oplus\cdots\oplus c_1a_n(t)$$

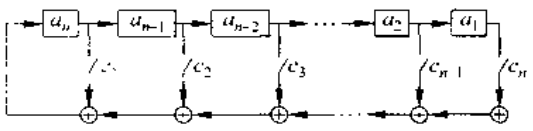


图 5.7

例 5.2 图 5.8 所示的线性反馈移位寄存器有

$$a_i(t-1) = a_{i+1}(t), \quad i = 1,2,3,4$$

$$a_5(t+1) = a_1(t) \oplus a_4(t)$$

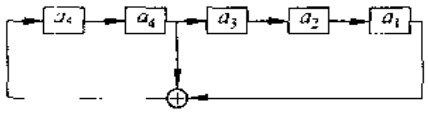


图 5.8

如果 $a_5a_4a_3a_2a_1$ 的初态(即 $t=0$ 时的状态)为 10101,则 $t=1$ 时的状态为

$$f(10101)1010$$

而

$$f(10101) = 1 \oplus 0 = 1$$

故 $t=1$ 时的状态是 11010

同理 $t=2$ 时的状态是

$$f(01011)1101 = 11101$$

现将该线性反馈移位寄存器在各时间的状态列于表 5.4。

表 5.4

t	a_5	a_4	a_3	a_2	a_1	t	a_5	a_4	a_3	a_2	a_1
0	1	0	1	0	1	20	0	1	0	1	1
1	1	1	0	1	0	21	0	0	1	0	1
2	1	1	1	0	1	22	1	0	0	1	0
3	0	1	1	1	0	23	0	1	0	0	1
4	1	0	1	1	1	24	0	0	1	0	0
5	1	1	0	1	1	25	0	0	0	1	0
6	0	1	1	0	1	26	0	0	0	0	1
7	0	0	1	1	0	27	1	0	0	0	0
8	0	0	0	1	1	28	0	1	0	0	0
9	1	0	0	0	1	29	1	0	1	0	0
10	1	1	0	0	0	30	0	1	0	1	0
11	1	1	1	0	0	31	1	0	1	0	1
12	1	1	1	1	0	32	1	1	0	1	0
13	1	1	1	1	1	33	1	1	1	0	1
14	0	1	1	1	1	34	0	1	1	1	0
15	0	0	1	1	1	35	1	0	1	1	1
16	1	0	0	1	1	36	1	1	0	1	1
17	1	1	0	0	1	37	0	1	1	0	1
18	0	1	1	0	0	38	0	0	1	1	0
19	1	0	1	1	0	39	0	0	0	1	1

可见 $t=31$ 时状态恢复到 $t=0$ 时的情况,以后的状态重复前面 $t=1$ 到 $t=30$ 的全过程。该线性反馈移位寄存器的周期是 31。

在线性反馈移位寄存器中总假定 $c_1,c_2,\cdots,c_n$ 中至少有一个系数不为 0。若不然,就相当于 $f(a_1,a_2,\cdots,a_n)\equiv 0$,无论初始状态如何,在 n 个脉冲后状态必然是 00…0,以后一

直维持这种状态。

若只有一个系数不为0，设仅有 $a_j(t)$ 项系数非0，则有

$$a_n(t+1)=a_j(t)$$

$$a_i(t+1)=a_{i-1}(t),\quad i=1,2,\cdots,n-1$$

实际上是一种延迟装置。一般对于 n 级线性反馈移位寄存器，总假定 $c_n=1$。

引理 5.1 $c_n=1$ 的 n 级线性移位寄存器对于零输入其输出序列是周期序列。

证明 零输入 n 级线性移位寄存器如图 5.9 所示。

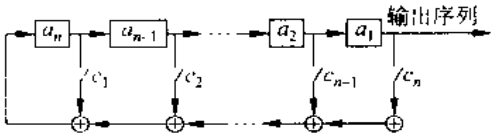


图 5.9

由前所述，零输入线性移位寄存器可看作有限状态机，其输出序列 $\{a_i\}$ 是准周期的。设 r 是周期，m 是满足 $a_{m+t}=a_{m+r+t}$ 的最小正整数，t 是一切正整数，即 $a_m\neq a_{m-r}$。第 $m,m+1,m-r,m+r-1$ 时间的状态分别是

$$a_m a_{m+1}\cdots a_{m+n-1}$$

$$a_{m-1}a_{m-2}\cdots a_{m-n}$$

$$a_{m-r}a_{m-r+1}\cdots a_{m-r+n-1}$$

$$a_{m+r-1}a_{m+r-2}\cdots a_{m+r-n}$$

由假设

$$a_{m-n}=c_n a_m+c_{n-1}a_{m+1}+\cdots+c_1 a_{m+n-1}$$

$$a_{m-r-n}=c_n a_{m+r}+c_{n-1}a_{m+r+1}+\cdots+c_1 a_{m+r+n-1}$$

但

$$a_{m+n}=a_{m+r+n}$$

从而得到

$$c_n a_m=c_n a_{m+r}$$

由 $c_n=1$ 推出 $a_m=a_{m-r}$，与假设矛盾。从而有 $m=0$，即 $\{a_i\}$ 是周期序列。

引理 5.2 n 级线性反馈移位寄存器对应于零输入的输出是周期性的，其周期 $r\leqslant 2^n-1$。

证明 根据引理 5.1，n 级线性反馈移位寄存器对应于零输入的输出是周期性的。若初始状态为 $00\cdots0$，则输出是常数0。否则其状态序列中就不会出现 $00\cdots0$，而状态最多有 2^n-1 种，后一状态及前一状态都由当前状态惟一确定，故周期 $r\leqslant 2^n-1$。

若 n 级线性反馈移位寄存器的输出序列的周期达到最大 (2^n-1)，则称为 m 序列。

5.4 特征多项式

设 n 级线性移位寄存器的零输入对应的输出序列 $\{a_i\}$ 满足递推关系

$$a_{n+1}=c_1 a_n+c_2 a_{n-1}+\cdots+c_n a_1$$

$$a_1 = b_1,\ a_2 = b_2,\ \cdots,\ a_n = b_n \tag{5-1}$$

称 $p(x)=1+c_1x+c_2x^2+\cdots+c_{n-1}x^{n-1}+c_nx^n$ 为递推关系(5-1)或该线性移位寄存器的特征多项式。

设 n 级线性移位寄存器对应于递推关系(5-1)。由于 $a_i\in GF(2)$，$i=1,2,\cdots,n$，故共有 2^n 组初始状态，即有 2^n 个递推序列，其中非恒为零的序列有 2^n-1 个。令这 2^n-1 个非零序列的全体为 $S(p_n(x))$。即 $S(p_n(x))$ 表示由满足递推关系

$$a_{n+t}=c_1a_{n+t-1}+c_2a_{n+t-2}+\cdots+c_na_t$$

的序列 $\{a_j\}$ 构成的集合，t 为任意正整数。对于 $S(p_n(x))$ 中任一序列 $\{a_j\}$，有母函数

$$G(x)=\sum_{i=1}^{\infty}a_ix^{i-1}$$

定理 5.2 设 $p_n(x)=1+c_1x+c_2x^2+\cdots+c_nx^n$，是 $GF(2)$ 上的多项式，且递推序列 $\{a_i\}\in S(p_n(x))$。

令

$$G(x)=\sum_{i=1}^{\infty}a_ix^{i-1}$$

则

$$G(x)=\frac{\phi(x)}{p_n(x)}$$

其中

$$\phi(x)=\sum_{i=1}^{n}c_{n-i}x^{n-i}\sum_{j=1}^{i}a_jx^{j-1}$$

证明 在等式

$$\begin{aligned}a_{n+1}&=c_1a_n+c_2a_{n-1}+\cdots+c_na_1\\a_{n+2}&=c_1a_{n+1}+c_2a_n+\cdots+c_na_2\\&\vdots\end{aligned}$$

的两边分别乘以 x^n，x^{n+1}，$\cdots$，再求和得

$$\begin{aligned}&G(x)-(a_1+a_2x+\cdots+a_nx^{n-1})\\&\quad=c_1x[G(x)-(a_1+a_2x+\cdots+a_{n-1}x^{n-2})]\\&\qquad+c_2x^2[G(x)-(a_1+a_2x+\cdots+a_{n-2}x^{n-3})]+\cdots+c_nx^nG(x)\end{aligned}$$

移项整理得

$$\begin{aligned}&(1+c_1x+c_2x^2+\cdots+c_nx^n)G(x)\\&\quad=(a_1+a_2x+\cdots+a_nx^{n-1})+c_1x(a_1+a_2x+\cdots+a_{n-1}x^{n-2})\\&\qquad+c_2x^2(a_1+a_2x+\cdots+a_{n-2}x^{n-3})+\cdots+c_{n-1}x^{n-1}a_1\end{aligned}$$

即

$$p_n(x)G(x)=\sum_{i=1}^{n}c_{n-i}x^{n-i}\sum_{j=1}^{i}a_jx^{j-1}=\phi(x)$$

注意对于 $GF(2)$，有 $a+a=0$。

定理 5.3 若 $p_n(x)$ 是 $GF(2)$ 上的 n 次多项式，是序列 $\{a_j\}$ 的特征多项式，

$p_n(x)|(x^m+1)$，则$\{a_i\}$的周期$r|m$。

证明 设
$$G(x)=\sum_{i=1}^{\infty}a_ix^{-i}$$
由于$p_n(x)|(x^m-1)$，故有$q(x)$，便得
$$p_n(x)q(x)=x^m+1$$

证明 设$\{a_i\}$的周期为r。由定理2，$r|m$，故$r\leqslant m$。设$G(x)$为$\{a_i\}$的母函数，由定理1，$G(x)p_n(x)=\phi(x)\neq 0$，$\phi(x)$的次数不超过$n-1$。设$x^m+1=p_n(x)q(x)$。则
$$G(x)=\frac{\phi(x)}{p(x)}=\frac{a_1+a_2x+\cdots+a_rx^{r-1}}{x^r+1}$$
$$\phi(x)(x^r+1)=p_n(x)(a_1+a_2x+\cdots+a_rx^{r-1})$$
因$p_n(x)$不可化约，故$(p_n(x),\phi(x))=1$，从而$p_n(x)|(x^r-1)$，所以$m\leqslant r$。总起来$r=m$。

设$m=\min\{l\,|\,p_n(x)$整除$(x^l+1)\}$，则称m为n次多项式$p_n(x)$的阶。阶为2^n-1的不可化约多项式称为本原多项式。

定理5.4 设$\{a_i\}\in S(p_n(x))$，则$\{a_i\}$为m序列的充要条件是$p_n(x)$为本原多项式。

证明 充分性。设$p_n(x)$是本原多项式，其阶为2^n-1，由定理5.3，$\{a_i\}$的周期等于$p_n(x)$的阶，即2^n-1，故根据m序列的定义，$\{a_i\}$是m序列。

必要性。若$\{a_i\}\in S(p_n(x))$的周期为2^n-1，由定理5.3，2^n-1整除$p_n(x)$的阶，而$p_n(x)$的阶不超过2^n-1，故$p_n(x)$的阶为2^n-1。若$p_n(x)$可化约，设$p_n(x)=q_1(x)q_2(x)$，$q_1(x)$不可化约，$q_1(x)$的次数为$k<n$。显然，$S(q_1(x))\subset S(p_n(x))$，这样，$\{a_i\}\in S(q_1(x))$一方面周期应为$2^n-1$，另一方面又不会超过$2^k-1$，产生矛盾。这就证明了$p_n(x)$是不可化约的。

$\{a_i\}$为m序列的关键在于$p_n(x)$为本原多项式，n次本原多项式的个数
$$\lambda(n)=\frac{\phi(2^n-1)}{n}\tag{5-2}$$
其中ϕ为欧拉函数。若有素数幂分解$n=p_1^{a_1}p_2^{a_2}\cdots p_k^{a_k}$，则
$$\phi(n)=n\left(1-\frac{1}{p_1}\right)\left(1-\frac{1}{p_2}\right)\cdots\left(1-\frac{1}{p_k}\right)$$
即$\phi(n)$等于不大于n且与n互素的正整数的数目。

(5-2)的证明从略。

从表5.5中可见，随着n的增加，$\lambda(n)$从大趋势上看也在增加，但$\lambda(n)$不是单调增的。通常当p是素数时，虽然$n>>p$，仍可能有$\phi(p)>\phi(n)$。因而当2^n-1是素数时可望$\lambda(n)$的值相对大些。形如2^n-1的素数称为麦尔逊(Mersenne)素数。

定理5.4告诉我们要生成m序列的充要条件是$p_n(x)$为本原多项式。但如何寻找本原多项式呢？表5.6给出本原多项式的系数表，n的本原多项式有$\lambda(n)$个，这里只给出其中的一个，比如$n=20$时$\lambda(n)=24000$，表5.6只给出其中之一1C6873，即
$$00011100011010000111001 1$$
即
$$x^{20}+x^{19}+x^{18}+x^{14}+x^{13}+x^{11}+x^6+x^5+x^4+x+1$$

表 5.5

n	2^n-1	$\lambda(n)$	n	2^n-1	$\lambda(n)$
1	1	1	13	8191	630
2	3	1	14	16383	756
3	7	2	15	32767	1800
4	15	2	16	65535	2048
5	31	6	17	131071	7710
6	63	6	18	262143	7776
7	127	18	19	524287	27594
8	255	16	20	1048575	24000
9	511	48	21	2097151	84672
10	1023	60	22	4194303	120032
11	2047	176	23	8388607	356960
12	4095	144	24	16777215	276480

表 5.6

n	本原多项式系数	n	本原多项式系数
20	1C6873	43	9CE3D73DF51
21	243B09	44	1EDA4E04DE51
22	47254B	45	27D99F781D27
23	81CD05	46	722603941CA3
24	197F913	47	D78A0EBC2281
25	3FADF21	48	13C9C7F617F31
26	701EE0D	49	2CCD7DBDA8D43
27	C479D11	50	73E876C9DCD97
28	18A377A7	51	9FC955B71B96D
29	2398C9F1	52	1185C69B225EB1
30	74B5F959	53	2F75A2F8E0B2A3
31	C067CE4F	54	4050FD2DC69A57
32	1A050AB3B	55	BEC81BD6CE836B
33	3F8F68B0F	56	1433C75494130E9
34	54584D319	57	316BA15C8C21349
35	892EDAC73	58	6AE04A7531D745F
36	1E4661C7F3	59	E870BA43725E085
37	3DAD5CC579	60	1FB7B391826A7E39
38	6707346E51	61	2244CBBC1ADE11DB
39	887E35E387	62	76A8FDFFAE87120B
40	15A02FC7587	63	9EB1CBFB9FB26729
41	3FCA4062BC9	64	15D2121DC0F0DCF1F
42	56222F3CFC9	65	21A762CC0F6015D57

续表

n	本原多项式系数	n	本原多项式系数
66	71AF2097B2A8302E9	76	139317F776067674E4AD
67	8AC09585A28856E0B	77	25E62F5808AF38F15F1D
68	13BD59899C9B5CEBF5	78	76D6B7D3623856ECD4E5
69	2302F2E11695C6CF43	79	F5364748CA839BEBA06D
70	4E055031A9E62660FF	80	16428CF31D321B4DB6B1B
71	A4761FA9854B6C0907	81	39AE11EE0800262FACA0D
72	1F583161C48D68C14CD	82	5511B2DD49B9BC6E23897
73	3C8ED8701935BBBCD6D	83	D8EE478DAA28940752CB7
74	53E2E269A884E576473	84	122C0BC8B4593A0C905E5D
75	824EC073FB02B82A49F	85	3189E29C9F12E75B82CDD5

若 2^n-1 是素数，则 n 必须是素数。因为若 n 是合数，2^n-1 也是合数。反之，n 是素数，2^n-1 未必是素数。

$2^7-1=127$ 是素数，$2^{11}-1=2047=23\times 87$ 是合数。

定理 5.5 若 p 是素数，2^p-1 的素因子必为 $2kp+1$ 型的数。其中 k 是正整数。

证明 设 q 是 2^p-1 的素因子。q 必是奇数。根据费尔玛定理，$q\mid(2^{q-1}-1)$，因而

$$q\mid(2^p-1,2^{q-1}-1)=2^{(p,q-1)}-1$$

故

$$2^{(p,q-1)}-1\geqslant q>1,(p,q-1)>1,(p,q-1)=p,p\mid q-1$$

因 $q-1$ 是偶数，所以有正整数 k，使得 $q-1=2kp$，即

$$q=2kp+1$$

这个定理可用来判断 2^n-1 是不是素数。

例 5.3 $2^{13}-1=8191$，$2^{23}-1=8388607$。$\sqrt{8191}\approx 90.5$，若 8191 是合数，必有一素因子比 90 小，比 90 小的 $2k\cdot 13+1$ 型的素数只有 53 和 79。显然，53 和 79 都不是 8191 的因子，所以 8191 是素数。$\sqrt{8388607}\approx 2896.3$，$2\cdot 1\cdot 23+1=47$，而 $47\mid 8388607$，故 8388607 是合数。

根据定理 5.2，若序列 $\{a_i\}\in S(p_n(x))$，其中 $p_n(x)$ 是 n 级线性移位寄存器的特征多项式，则 $\{a_i\}$ 的母函数 $G(x)=\phi(x)/p_n(x)$，其中 $\phi(x)$ 的次数低于 n。可以证明 $p(x)\mid q(x)$ 是 $S(p(x))\subset S(q(x))$ 的充要条件。

若 $p(x),q(x)$，设 $q(x)=p(x)r(x)$，则

$$G(x)=\frac{\phi(x)}{p(x)}=\frac{\phi(x)r(x)}{p(x)r(x)}=\frac{\phi(x)r(x)}{q(x)}$$

这就证明了若 $p(x)\mid q(x)$，序列 $\{a_i\}\in S(p(x))$，则 $\{a_i\}\in S(q(x))$。即

$$S(p(x))\subset S(q(x))$$

反之若

$$S(p(x))\subset S(q(x))$$

则对应于多项式 $\phi(x)$，有 $\{a_i\}\in S(p(x))$ 以 $G(x)=\phi(x)\mid p(x)$ 为母函数。特别是，以

$1/p(x)$为母函数的序列$\{a_i\}\in S(p(x))$，根据假定，$\{a_i\}\in S(q(x))$，故有 $r(x)$，$\{a_i\}$的母函数 $G(x)$也等于 $r(x)/q(x)$。从而

$$\frac{1}{p(x)}=\frac{r(x)}{q(x)}$$

即
$$q(x)=r(x)p(x)$$

或
$$p(x)\mid q(x)$$

这说明可用 n 级线性反馈移位寄存器产生的序列，也可以用级数更多的线性移位寄存器来实现。

令 $S_k,S_{k-1},\cdots,S_{k-k}$ 表示线性递推序列$\{a_i\}$的任意连续的 k 个状态向量。其中

$$S_k=\begin{pmatrix}a_k\\a_{k-1}\\\vdots\\a_{k-n-1}\end{pmatrix},\quad S_{k-1}=\begin{pmatrix}a_{k+1}\\a_{k-2}\\\vdots\\a_{k-n}\end{pmatrix},\cdots$$

假定序列$\{a_i\}$满足线性递推关系：

$$a_{k-n}=c_1a_{k-n-1}+c_2a_{k-n-2}+\cdots+c_na_k$$

这可以表示成为

$$\begin{pmatrix}a_{k-1}\\a_{k-2}\\\vdots\\a_{k-n}\end{pmatrix}=\begin{pmatrix}0&1&0&0&\cdots&0\\0&0&1&0&\cdots&0\\\vdots&&&&&\\0&0&0&0&\cdots&1\\c_n&c_{n-1}&&\cdots&&c_1\end{pmatrix}\begin{pmatrix}a_k\\a_{k-1}\\\vdots\\a_{k-n-1}\end{pmatrix}$$

或

$$S_{k-1}=MS_k$$

其中

$$M=\begin{pmatrix}0&1&0&0&\cdots&0\\0&0&1&0&\cdots&0\\\vdots&&&&&\\0&0&0&0&\cdots&1\\c_n&c_{n-1}&&\cdots&&c_1\end{pmatrix}\begin{pmatrix}a_k\\a_{k-1}\\\vdots\\a_{k-n-1}\end{pmatrix}$$

于是更一般地有

$$S_{k-l}=M^lS_k$$

由递推关系

$$a_{k-n}=c_1a_{k-n-1}+c_2a_{k-n-2}+\cdots+c_na_k$$

可推出向量的递推关系

$$S_{k-k}=c_1S_{k-n-1}+c_2S_{k-n-2}+\cdots+c_nS_k$$

如果存在最小整数 m，使 $S_1,S_2,\cdots,S_m$ 线性相关，即存在系数 $l_1,l_2,\cdots,l_m$，其中不妨设 $l_1=1$，使得

$$S_m+l_2S_{m-1}+l_3S_{m-2}+\cdots+l_mS_1=0$$

或

$$S_m = l_m S_1 + l_{m-1} S_2 + \cdots + l_2 S_{m-1}$$

对于任一整数 i 有

$$\begin{aligned} S_{m-i} &= M^i S_m \\ &= M^i (l_m S_1 + l_{m-1} S_2 + \cdots + l_2 S_{m-1}) \\ &= l_m M^i S_1 + l_{m-1} M^i S_2 + \cdots + l_2 M^i S_{m-1} \\ &= l_m S_{i+1} + l_{m-1} S_{i+2} + \cdots + l_2 S_{i+m-1} \end{aligned}$$

不难证明，m 至少和 n 一样大。实际上已经证明了 $S_1, S_2, \cdots, S_{m-1}$ 线性无关。

5.5 Golomb 随机性概念

前面讨论了移位寄存器的周期性，保证有足够的周期长度。这一节将讨论它的随机性。即证明它满足 Golomb 的随机性三公设。

(1) 由于移位寄存器状态相同时，输出的序列也相同，所以产生 m 序列的过程中必须遍历 2^n-1 个非 0 状态的每一个，然后才出现重复。这 2^n-1 个状态在 a_1 位有 2^{n-1} 个是 1，其余的 $2^{n-1}-1$ 个是 0。这就满足了第一公设。

(2) 由于寄存器中不会出现全 0 状态，所以不会出现 0 的 n 游程。而且必然有一个 1 的 n 游程。但也不可能有长度更大的 1 的游程。因为若出现 1 的 $n+1$ 游程，必然有两个全 1 状态相邻，这是不可能的。这就证明了 1 的 n 游程必然出现在如下的串中：

$$0\underbrace{1\cdots1}_{n\text{位}}0$$

当这 $n-2$ 位通过移位寄存器时，便依次产生以下状态：

$$0\underbrace{1\cdots1}_{n-1\text{位}}\quad\underbrace{1\cdots1}_{n\text{位}}\quad\underbrace{1\cdots1}_{n-1\text{位}}0$$

由于 $0\,1\cdots1$，$1\cdots1\,0$ 这两个状态只能各出现一次，所以不会有 1 的 $n-1$ 游程。

会出现 0 的 $n-1$ 游程：

$$1\underbrace{0\cdots0}_{n-1\text{位}}1$$

它产生 $1\,0\cdots0$ 和 $0\cdots0\,1$ 两个状态。

如果 $n=2$，则已考察了所有的情况。

如果 $n>2$，r 为不超过 $n-2$ 的任一正整数，则任何 1 的 r 游程意味着存在串：$0\underbrace{1\cdots1}_{r\text{位}}0$ 为了计算 1 的 r 游程的数目，只要计算左边是这样的 $r+2$ 位的状态数目就可以了。因为任何一个 1 的 r 游程总会在通过移位寄存器时处在这样的位置。其余的 $n-r-2$ 位可以是由 0，1 构成的任何状态。所以 1 的 r 游程的数目为 2^{n-r-2}。于是在每一循环中出现的 1 游程的数目为

$$1+\sum_{r=1}^{n-2} 2^{n-r-2} = 2^{n-2}$$

0 游程的数目也是 2^{n-2}。这就证明了第二公设。再小结如下：

① 任一循环均含 2^{n-1} 个 1，$2^{n-1}-1$ 个 0；

② 1 的 n 游程有 1 个，没有 0 的 n 游程；

③ 没有 1 的 $n-1$ 游程，有 1 个 0 的 $n-1$ 游程；

④ 若 $1\leqslant r\leqslant n-2$，则 1 和 0 的 r 游程各有 2^{n-r-2} 个；

⑤ 每一循环有 2^{n-2} 个 1 的游程和 2^{n-2} 个 0 的游程。

(3) **换行定理** 周期为 2^n-1 的 m 序列，其异相关自相关函数等于 $\frac{-1}{2^n-1}$。

证明 设 $\{a_t\}$ 是周期为 2^n-1 的 m 序列，对于任一正整数 τ，$0<\tau<2^n-1$，$\{a_t\}+\{a_{t+\tau}\}$ 在一个周期内为 0 的位的数目正好是序列 $\{a_t\}$ 和 $\{a_{t+\tau}\}$ 对应位的相同位的数目。其中 $\{a_{t+\tau}\}$ 是将序列平移 τ 位所得序列。

设序列 $\{a_t\}$ 满足递推关系：

$$a_{h+n}=c_1a_{h+n-1}+c_2a_{h+n-2}+\cdots+c_na_h$$

故

$$a_{h+n+\tau}=c_1a_{h+n-1+\tau}+c_2a_{h+n-2+\tau}+\cdots+c_na_{h+\tau}$$

$$a_{h+n}+a_{h+n+\tau}=c_1(a_{h+n-1}+a_{h+n-1+\tau})+c_2(a_{h+n-2}+a_{h+n-2+\tau})+\cdots+c_n(a_h+a_{h+\tau})$$

令 $b_j=a_j+a_{j+\tau}$，由递推序列 $\{a_j\}$ 可推得递推序列 $\{b_j\}$。且 $\{b_j\}$ 满足

$$b_{h+n}=c_1b_{h+n-1}+c_2b_{h+n-2}+\cdots+c_nb_h$$

即若 $\{a_j\}\in S(p_n(x))$，则 $\{b_j\}\in S(p_n(x))$。为了计算异相自相关函数 $R(\tau)$，只有将递推序列 $\{b_j\}$ 在某一循环中 0 的个数减去 1 的个数，再除以 2^n-1。注意到 $\{b_j\}$ 实际上也是 m 序列。所以

$$R(\tau)=\frac{2^{n-1}-1-2^{n-1}}{2^n-1}=\frac{-1}{2^n-1}$$

(4) 可利用线性反馈移位寄存器设计加密算法。这种加密算法的构思如图 5.10 所示。设所用线性反馈移位寄存器产生的是 m 序列，算法的密钥取决于初始状态和 c_1，c_2，…，c_n 的值。由前所述，这样的 n 级线性反馈移位寄存器对应的特征多项式是本原多项式。因 n 次本原多项式有 $\lambda(n)$ 个，非 0 初始状态有 2^n-1 个，故共有 $\lambda(n)(2^n-1)$ 个不同的密钥或密钥流。

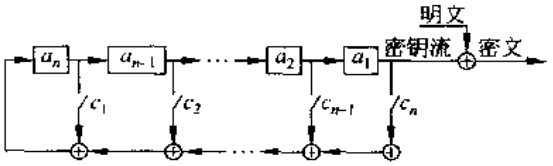


图 5.10

5.6 非线性反馈移位寄存器

5.6.1 n 级线性反馈移位寄存器

n 级移位寄存器是这样一种装置，当输入向量为 $(a_1,a_2,\cdots,a_n)$ 时，输出向量为

$(a_2,a_3,\cdots,a_n,a_{n+1})$。所以移位寄存器实际上是由对一切可能的$(a_1,a_2,\cdots,a_n)$给出相应的$a_{n+1}$所完全确定。

例 5.4 给出如表 5.7 所示的真值表。

表 5.7

a_1	a_2	a_3	a_4
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

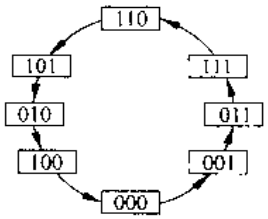


图 5.11

图 5.11 给出表 5.7 所确定的状态(a_1,a_2,a_3)的迁移图。实际上这个反馈移位寄存器是 n 维空间的变换

$$T:(a_1,a_2,\cdots,a_n)\mapsto(a_2,a_3,\cdots,a_{n+1})$$

例 5.5 若有$a_{n+1}=a_{n-1}\cdot a_{n-2}$，则相应的真值表及状态迁移图分别如表 5.8 及图 5.12 所示。

表 5.8

a_1	a_2	a_3	a_4
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

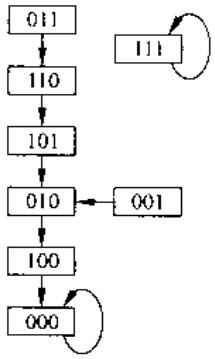


图 5.12

移位寄存器也可以看作 n 元布尔函数

$$a_{n+1}=f(a_1,a_2,\cdots,a_n)$$

不同的 n 个变元的 n 个变元的布尔函数有2^{2^n}个。线性 n 元布尔函数只是2^{2^n}个布尔函数中的2^n个。

5.6.2 由 LFSR1 与 LFSR2 构造非线性序列

下面介绍一种由两个线性反馈移位寄存器产生的 m 序列“相加”得到的序列。设第一个线性反馈移位寄存器 LFSR1 产生序列$\{a_i\}$，第二个线性反馈移位寄存器 LFSR2 产

生序列$\{b_t\}$，$\{a_t\}$与$\{b_t\}$"相乘"产生$\{c_t\}$，即$c_t=a_t \cdot b_t$。

设LFSR1为4级移位寄存器，$a_4=a_1+a_2$，LFSR2为3级移位寄存器，$b_4=b_1+b_2$。LFSR1的初态为$(a_4a_3a_2a_1)=(1111)$，LFSR2的初态为$(b_3b_2b_1)=(111)$。由LFSR1和LFSR2产生的序列如表5.9所示。

表 5.9

a_4	a_3	a_2	a_1	b_3	b_2	b_1	c	a_4	a_3	a_2	a_1	b_3	b_2	b_1	c
1	1	1	1	1	1	1	0	0	1	1	1	1	0	0	1
0	1	1	1	0	1	1	0	0	0	1	1	0	1	0	1
0	0	1	1	0	0	1	0	0	0	0	1	1	0	1	0
0	0	0	1	1	0	0	1	1	0	0	0	1	1	0	0
1	0	0	0	0	1	0	0	0	1	0	0	1	1	1	1
0	1	0	0	1	0	1	1	0	0	1	0	0	1	1	1
0	0	1	0	1	1	0	0	1	0	0	1	0	0	1	0
1	0	0	1	1	1	1	0	1	1	0	0	1	0	0	0
1	1	0	0	0	1	1	1	0	1	1	0	0	1	0	0
0	1	1	0	0	0	1	1	1	0	1	1	1	0	1	0
1	0	1	1	1	0	0	1	0	1	0	1	1	1	0	1
0	1	0	1	0	1	0	1	1	0	1	0	1	1	1	0
1	0	1	0	1	0	1	1	1	1	0	1	0	1	1	0
1	1	0	1	1	1	0	1	1	1	1	0	0	0	1	1
1	1	1	0	1	1	1	1	1	1	1	1	1	0	0	1
1	1	1	1	0	1	1	0	0	1	1	1	0	1	0	1
0	1	1	1	0	0	1	0	0	0	1	1	1	0	1	0
0	0	1	1	1	0	0	1	0	0	0	1	1	1	0	1
0	0	0	1	0	1	0	1	1	0	0	0	1	1	1	1
1	0	0	0	1	0	1	1	0	1	0	0	0	1	1	1
0	1	0	0	1	1	0	0	0	0	1	0	0	0	1	1
0	0	1	0	1	1	1	1	1	0	0	1	1	0	0	1
1	0	0	1	0	1	1	0	1	1	0	0	0	1	0	0
1	1	0	0	0	0	1	1	0	1	1	0	1	0	1	1
0	1	1	0	1	0	0	0	1	0	1	1	1	1	0	1
1	0	1	1	0	1	0	1	0	1	0	1	1	1	1	0
0	1	0	1	1	0	1	0	1	0	1	0	0	1	1	1
1	0	1	0	1	1	0	0	1	1	0	1	0	0	1	0
1	1	0	1	1	1	1	0	1	1	1	0	1	0	0	0
1	1	1	0	0	1	1	0	1	1	1	1	0	1	0	1
1	1	1	1	0	0	1	0	0	1	1	1	1	0	1	0

从表5.8可看出两序列相加的序列$\{c_t\}$。

由此可见序列$\{c\}$的周期较$\{a\}$及$\{b\}$扩大了。

5.6.3 J-K 触发器

如图 5.13 所示，LFSR1 是 m 级线性反馈移位寄存器，LFSR2 是 n 级反馈移位寄存器，这样的装置称为 J-K 触发器，它的工作用表 5.10 表示。其中 c_n 表示 $\{c_i\}$ 的第 n 个元素。

假定 LFSR1 输出 $\{a_i\}$，周期为 m，LFSR2 输出 $\{b_i\}$，周期为 n，$(m,n)=1$，J-K 触发器输出 $\{c_i\}$。由于 J-K 触发器的输出有时和前一项有关，不给出 c_{-1} 就不能给出 c_1。通常令 $c_{-1}=0$。于是有

$$c_n=(a_n+1)(b_n+1)c_{n-1}+(a_n+1)b_n\cdot 0+a_n(b_n+1)+a_nb_n(c_{n-1}+1)$$
$$-(a_n+b_n+1)c_{n-1}-a_n$$

当 $c_{-1}=0$ 时，有

$$c_1=a_0$$
$$c_1=(a_1+b_1+1)a_1+a_1$$
$$c_2=(a_2+b_2+1)[(a_1+b_1+1)a_1+a_1]+a_2$$
$$\vdots$$

显然，这些方程都是非线性的。

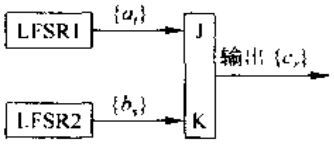


图 5.13

表 5.10

J	K	c_n
0	0	c_{n-1}
0	1	0
1	0	1
1	1	c_{n-1}

这种体制虽然在随机性方面比较好，然而只要知道它的序列的一部分，就能给出这些方程的解。例如，知道 c_n 和 c_{n-1}，由递推关系可看出

① 若 $c_n=c_{n+1}=0$，则 $a_{n-1}=0$

② 若 $c_n=0$，$c_{n+1}=1$，则 $a_{n-1}=1$

③ 若 $c_n=1$，$c_{n+1}=0$，则 $b_{n-1}=1$

④ 若 $c_n=c_{n+1}=1$，则 $b_{n-1}=0$

即从 c_n 和 c_{n+1} 便能对 a_{n-1}，b_{n-1} 中的一个作出判断，所以这个体制并非安全的。

5.6.4 Pless 体制

下面介绍一种 Pless 体制，可以用来生成伪随机序列。它由 8 个线性移位寄存器组成 4 组 J-K 触发器，另加一个循环计数器联接而成，如图 5.14 所示。循环计数器的作用是决定在每一个时间脉冲作用下输出的单元。这个体制的密钥是 8 个移位寄存器和它们的初态，

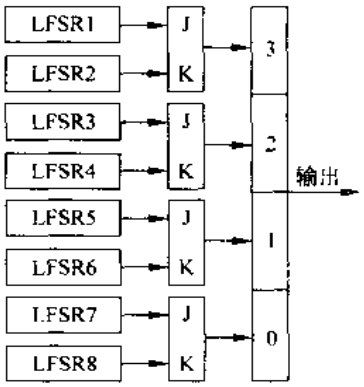


图 5.14

J-K 触发器的初态以及输出单元的顺序。

Pless 提出 8 个线性反馈移位寄存器的级数，不仅使得各对之间达到级数互素，而且达到各 J-K 触发器的输出周期元素，从而达到最后输出的周期为各个周期的乘积。

5.6.5 复合

如图 5.15 所示，输入部分 A 给出的是二进制地址，输出的正是依照这个地址从 B 取出的元素，如表 5.11 所示。

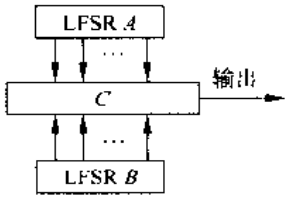


图 5.15

表 5.11

a_0	a_1	a	对应 b_i
0	0	0	b_0
0	0	1	b_1
0	1	0	b_2
0	1	1	b_3
1	0	0	b_4
1	0	1	b_5
1	1	0	b_6
1	1	1	b_7

根据 A 确定地址，按此地址从 B 取出相应元素，从而构成输出 $\{c_n\}$ 的具体方法是多样的。

假定 LFSR A 和 LFSR B 分别是产生 m 序列的 m 级和 n 级线性反馈移位寄存器，$A_0, A_1, \cdots, A_{m-1}$ 表示 LFSR A 的各级，$B_0, B_1, \cdots, B_{n-1}$ 表示 LFSR B 的各级，LFSR A 输出 $\{a_t\}$，LFSR B 输出 $\{b_t\}$。复合器 C 输出 $\{c_r\}$，$\{c_r\}$ 是按我们所说的复合方法构成。

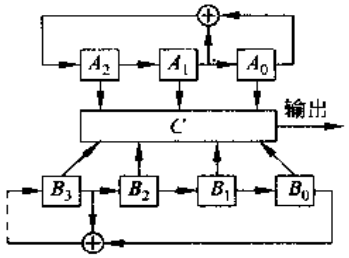


图 5.16

下面以一个前面的例子为基础，加上具体的取址方案，说明复合方法是如何工作的。图 5.16 是复合示意图，表 5.12 给出了具体数据。

在本例中，当 $t=2$ 时，$A_2A_1A_0=011$，$A_2A_1=01$，$N=1$，$B_3B_2B_1=1100$，$B_1=0$。

本例中 $m=3$，$n=4$，所以地址取两位，一般的讨论可类推。

复合序列可推广。即将 B_3, B_2, B_1, B_0 重新编序，使之分别和 0,1,2,3 一一对应。本例是一种特例，输出就是 B_N，$N=(A_2A_1)_2$。

复合序列和 LFSR A 及 LFSR B 的状态有关。一旦两个移位寄存器的状态同时重复，复合序列也就开始重复。所以有

表 5.12

A_2	A_1	A_0	B_3	B_2	B_1	B_0	$N(=A_2A_1)$	B_N
1	1	1	1	0	0	0	1 1	1
0	1	1	1	1	0	0	0 1	0
0	0	1	1	1	1	0	0 0	0

续表

A_2	A_1	A_0	B_3	B_2	B_1	B_0	$V(-A_2A_1)$		B_s
1	0	0	1	1	1	1	1	0	1
0	1	0	0	1	1	1	0	1	1
1	0	1	1	0	1	1	1	0	0
1	1	0	0	1	0	1	1	1	0
1	1	1	1	0	1	0	1	1	1
0	1	1	1	1	0	1	0	1	0
0	0	1	0	1	1	0	0	0	0
1	0	0	0	0	1	1	1	0	0
0	1	0	1	0	0	1	0	1	0
1	0	1	0	1	0	0	1	0	1
1	1	0	0	0	1	0	1	1	0
1	1	1	0	0	0	1	1	1	0
0	1	1	1	0	0	0	0	1	0

引理 5.3 复合序列是周期的，其周期 r 满足

$$r \leqslant (2^m - 1)(2^n - 1)$$

实际上当$(m,n)=1$时，$r=(2^m-1)(2^n-1)$。一般有 $r=\mathrm{lcm}\{2^m-1,2^n-1\}$。

5.7 利用线性反馈移位寄存器的密码反馈

密码反馈有多种形式。分组密码可以用来反馈，序列密码也可以。反馈的基本特性在于把密文反馈给系统，以影响后面的加密过程。先举一简单例子。明文是"cryptography is the science of data security"，用维吉利亚密码加密，密钥是 redstar，详见如下。

m:	c r y p t o g r a p h y i s t h e s c i e n c e o f d a t a s e c u r i t y
k:	r e d s t a r r e d s t a r r e d s t a r r e d s t a r r e d s t a r r e d
C:	T V B H M O X I E S Z R I J K L H K U I V E G H G Y D R K E V W V U I Z X B
m:	c r y p t o g r a p h y i s t h e s c i e n c e o f d a t a s e c u r i t y
k:	r e d s t a r T V B H M O X K V Q O K W P D O U G M G T Q E Y U R J T J E Q
C:	T V B H M O X K V Q O K W P D C U G M G T Q E Y U R J T J E Q Y T D K R Z O

为了利用密文反馈，在密钥字 redstar 结束后，紧接着用密文 TVBHMOX 进行加密，依此类推得：

m：c r y p t o g		r	a	p	h
k：r e d s t a r		(r+T)	(e+V)	(d+B)	(s+H)
C_i：T V B H M O X		B	Z	T	G
y	i	s	t	h	e
(t+M)	(a+O)	(r+X)	(r+B)	(e+Z)	(d+T)
D	W	G	L	K	A
s	c	i	e	n	c
(s+G)	(t+D)	(a+w)	(r+G)	(r+L)	(e+K)
Q	Y	E	B	P	Q
e	o	f	d	a	t
(d+A)	(s+Q)	(t+Y)	(a+E)	(r+B)	(r+P)
H	W	W	H	S	Z
a	s	e	c	u	r
(e+Q)	(d+H)	(s+W)	(t+W)	(a+H)	(r+S)
U	C	S	R	B	A
t	i	y			
(r+Z)	(e+O)	(d+C)			
Y	R	D			

不用说以上利用密文反馈的措施是没有价值的。因为只要知道密钥长度 h，则明文便可破译，前面 h 位除外。然而稍加改造便可获得有效的加密体制，保持了密文反馈所带来的优势，克服了上述的弱点，优势在于消除了密钥的周期性，因而使卡西斯基和重合指数分析方法失效。改造的方法是利用密钥字和密文两者结合形成密钥。

图 5.17 是下面介绍的利用线性反馈移位寄存器进行序列密码反馈加密的体制的示意图。

密钥为系数 $c_1, c_2, \cdots, c_n$ 及 $a_1, a_2, \cdots, a_n$ 的初态 $s_1, s_2, \cdots, s_n$，其中 $c_n = 1$。

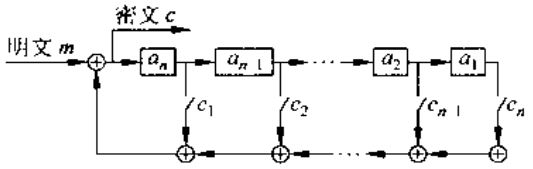


图 5.17

设明文 $m = m_1 m_2 m_3 \cdots$，密文 $c = e_1 e_2 e_3 \cdots$，则有

$$e_1 = m_1 + \sum_{i=1}^{n} c_i s_{n-i+1}$$

$$e_2 = m_2 + \sum_{i=2}^{n} c_i s_{n-i+2} + c_1 e_1$$

$$e_3 = m_3 + \sum_{i=3}^{n} c_i s_{n-i+3} + c_1 e_2 + c_2 e_1$$

$$\vdots$$

$$e_k = m_k - \sum_{i=k}^{n} c_i s_{n+k} + \sum_{i=1}^{k-1} c_i e_{k-i} \qquad 1 \leqslant k \leqslant n$$

$$e_{n-1} = m_{n-1} + \sum_{i=1}^{n} c_i e_{n+1-i}$$

$$\vdots$$

$$e_{n-h} = m_{n-h} + \sum_{i=1}^{n} c_i e_{n+h-i} \qquad h \geqslant 1$$

图 5.18 所示的解密移位寄存器的 $a_1, a_2, \cdots, a_n$ 的初态是 $s_1, s_2, \cdots, s_n$，系数是 c_1，$c_2, \cdots, c_n$，都与加密线性反馈移位寄存器相同，即解密密钥与加密密钥相同。这样当输入密文 $c = e_1 e_2 e_3 \cdots$ 时，便输出明文 $m = m_1 m_2 m_3 \cdots$，实现了脱密。

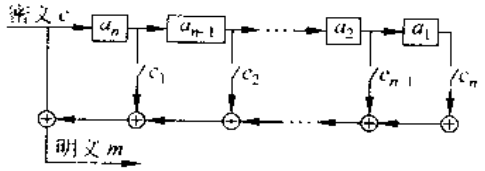


图 5.18

第 6 章　大数的快速计算

不论是 RSA 还是 ElGamal 公钥密码都要作大数的模幂运算，所以如何操作以提高效率是至关重要的。

6.1　数的 m 进制表示

一个整数 n 的 m 进制表示法：

$$n=(n_k n_{k-1}\cdots n_2 n_1 n_0)_m$$
$$=n_k\cdot m^k+n_{k-1}m^{k-1}+\cdots+n_2m^2+n_1m+n_0$$

$m=10$，即十进制表示，例如 3725

$$3725=3\times10^3+7\times10^2+2\times10+5$$

但计算机科学常用的是二进制和十六进制。设一个整数 n 要用 m 进制表示，即

$$n=n_k\cdot m^k+n_{k-1}m^{k-1}+\cdots+n_2m^2+n_1m+n_0$$

令 $n_0=n$，则

$$n_1=\left\lfloor\frac{n}{m}\right\rfloor=n_km^{k-1}+n_{k-1}m^{k-2}+\cdots+n_2m+n_1$$

余数 $r_1=n_0$

$$n_2=\left\lfloor\frac{n_1}{m}\right\rfloor=n_km^{k-2}+n_{k-1}m^{k-3}+\cdots+n_2$$

余数 $r_2=n_1$

$$n_{i+1}=\left\lfloor\frac{n_i}{m}\right\rfloor=n_km^{k-i-1}+n_{k-1}m^{k-i-2}+\cdots+n_{i+1}$$

余数 $r_{i+1}=n_i$，　$i=0,1,2,\cdots,k$

一直进行到 $n_i<m$ 为止，或归结为算法

S1. $k\leftarrow0$，　$x\leftarrow n$，　$q\leftarrow\left\lfloor\frac{x}{m}\right\rfloor$，　$n_i\leftarrow x-qm$；

S2. 若 $q>0$，则作

始　$k\leftarrow k+1$，　$x\leftarrow q$，　$q\leftarrow\left\lfloor\frac{x}{m}\right\rfloor$

$n_k\leftarrow x-qm$，转 S2；

终。

否则转 S3；

S3. 输出 $n=(n_k n_{k-1}\cdots n_2 n_1 n_0)_m$。

6.2 数的 m^l 进制表示

设 $n=(n_k n_{k-1}\cdots n_2 n_1 n_0)_m$，将 n 改为 m^l 进制表示，其中 l 是一正整数，令

$$n=(n_h n_{h-1}\cdots \bar{n}_2\ \bar{n}_1\ \bar{n}_0)_{m^l}$$

即 n 用 m^l 进制表示，则

$$h=\left\lceil\frac{k+1}{l}\right\rceil-1$$

其中

$$\bar{n}_i=\sum_{j=0}^{l-1}n_{il+j}m^j \quad 0\leqslant i\leqslant h-1$$

$$n_h=\sum_{j=0}^{k-hl}n_{hl+j}m^j$$

例如，$n=1323$，$m=2$，$l=4$。即将 n 化为十六进制。

先将 $n=1323$ 化为二进制数，

$$n=1323,\quad \left\lfloor\frac{1323}{2}\right\rfloor=661,\quad n_0=1$$

$$\left\lfloor\frac{661}{2}\right\rfloor=330,\quad n_1=1$$

$$\left\lfloor\frac{330}{2}\right\rfloor=165,\quad n_2=0$$

$$\left\lfloor\frac{165}{2}\right\rfloor=82,\quad n_3=1$$

$$\left\lfloor\frac{82}{2}\right\rfloor=41,\quad n_4=0$$

$$\left\lfloor\frac{41}{2}\right\rfloor=20,\quad n_5=1$$

$$\left\lfloor\frac{20}{2}\right\rfloor=10,\quad n_6=0$$

$$\left\lfloor\frac{10}{2}\right\rfloor=5,\quad n_7=0$$

$$\left\lfloor\frac{5}{2}\right\rfloor=2,\quad n_8=1$$

$$\left\lfloor\frac{2}{2}\right\rfloor=1,\quad n_9=0$$

$$\left\lfloor\frac{1}{2}\right\rfloor=0,\quad n_{10}=1$$

故

$$\begin{aligned}1323&=(10100101011)_2\\&=((101)(0010)(1011))_{2^4}\\&=(52B)_{16}=0x52B\end{aligned}$$

十六进制数通常在前面冠以 0x，例如 0x52B 即 $5\times16^2+2\times16+11=1323$。十六进制与二进制的转换比较方便，如上例所示。

6.3 加法和减法

已知 m 进制的两个数，不妨假定都是 h 位数

$$A=(a_ha_{h-1}\cdots a_2a_1a_0)_m$$

$$B=(b_hb_{h-1}\cdots b_2b_1b_0)_m$$

求 $A+B=D=(d_{h+1}d_h\cdots d_2d_1d_0)_m$。

加法步骤：

S1. $c\leftarrow 0$（c 是进位数字），$i\leftarrow 0$；

S2. $d_i\leftarrow(a_i+b_i+c) \bmod m$；

S3. 若 $d_i<m$ 则作

始 $c\leftarrow 0$

否则作 $c\leftarrow 1$，

终 $i\leftarrow i+1$；

S4. 若 $i\leqslant h$ 则转 S2；

S5. $d_{h+1}\leftarrow c$，输出 $D=(d_{h+1}\ d_h\cdots d_2d_1d_0)_m$。

减法步骤：

设 $E=A-B=(e_he_{h-1}\cdots e_2e_1e_0)_m$

S1. $c\leftarrow 0$，$i\leftarrow 0$；

S2. $e_i\leftarrow(a_i-b_i+c_i) \bmod m$；

S3. 若 $e_i\geqslant 0$ 则作

始 $c\leftarrow 0$，否则作

$c\leftarrow -1$

终 $i\leftarrow i+1$；

S4. 若 $i\leqslant h$ 则转 S2；

S5. 输出 $E=(e_he_{h-1}\cdots e_2e_1e_0)_m$。

6.4 多位数乘法

设 $A=(a_ha_{h-1}\cdots a_2a_1a_0)_m$，$B=(b_kb_{k-1}\cdots b_2b_1b_0)_m$，求 $D=A\cdot B=(d_{h+k+1}\ d_{h+k}\cdots d_2\ d_1\ d_0)_m$。

步骤：

S1. i 从 0 到 $h+k+1$ 作 $d_i\leftarrow 0$，$i\leftarrow 0$；

S2. 若 $i\leqslant k$ 则作

始 $j\leftarrow 0$，$c\leftarrow 0$，转 S3

终 否则转 S6；

S3. $(a\ b)_m=d_{i+j}+a_jb_i+c$

$$d_{i+j}\leftarrow b,\quad c\leftarrow a,\quad j\leftarrow j+1;$$

S4. 若 $j \leqslant h$，则转 S3。否则作 $d_{i-h+1} \leftarrow a$；

S5. $i \leftarrow i+1$；

S6. 输出$(d_{h+k-1}\ d_{h+k}\cdots\ d_2\ d_1\ d_0)_m$。

注意：S3 中$(ab)_m$ 表示 m 进制的两位数，它可能右端结果是一位数，则 $a=0$。若右端结果是两位数，则 a 为高位，b 为低位。

例如，$A=9274$，$B=847$。$h=3$，$k=2$。$a_3=9$，$a_2=2$，$a_1=7$，$a_0=4$；$b_2=8$，$b_1=4$，$b_0=7$，则

i	j	b_i	a_j	$d_{i+j}+a_jb_i+c$	c	a	b	d_6	d_5	d_4	d_3	d_2	d_1	d_0
0	0	7	4	0+28+0	2	2	8	0	0	0	0	0	0	8
	1		7	0+49+2	5	5	1	0	0	0	0	0	1	8
	2		2	0+14+5	1	1	9	0	0	0	0	9	1	8
	3		9	0+63+1		6	4	0	0	6	4	9	1	8
1	0	4	4	1+16+0	1	1	7	0	0	6	4	9	7	8
	1		7	9+28+1	3	3	8	0	0	6	4	8	7	8
	2		2	4+8+3	1	1	5	0	0	6	5	8	7	8
	3		9	6+36+1		4	3	0	4	3	5	8	7	8
2	0	8	4	8+32+0	4	4	0	0	4	3	5	0	7	8
	1		7	5+56+4	6	6	5	0	4	3	5	0	7	8
	2		2	3+16+6	2	2	5	0	4	5	5	0	7	8
	3		9	4+72+2		7	8	7	8	5	5	0	7	8

该题的笔算

$$
\begin{array}{r}
9\ 2\ 7\ 4 \\
\times\quad 8\ 4\ 7 \\
\hline
6\ 4\ 9\ 1\ 8 \\
3\ 7\ 0\ 9\ 6 \\
7\ 4\ 1\ 9\ 2 \\
\hline
7\ 8\ 5\ 5\ 0\ 7\ 8
\end{array}
$$

上表中 $i=0,1,2$ 三段的最后结果分别是

64918

64918+370960=435878

64918+370960+7419200=7855078

算法的复杂性分析：算法中计算称之为内积的部分

$$d_{i+j}+a_j \cdot b_i+c$$

共需

$$(m-1)+(m-1)^2+m-1=m^2-2m+1+2m-2=m^2-1$$

次位运算。整个算法共需$(h+1)(k+1)$次单位数乘积。

6.5 数的平方运算

已知 $A=(a_{k-1}a_{k-2}\cdots a_2\ a_1\ a_0)_m$，求 A^2。

设 $A^2=x\cdot x=(b_{2k-1}\ b_{2k-2}\cdots b_2\ b_1\ b_0)_m$。

步骤：

S1. i 从 0 到 $2k-1$ 作 $b_i\leftarrow 0$，　$i\leftarrow 0$；

S2. 若 $i\leqslant(t-1)$则转 S3，否则转 S5；

S3. $(xy)_m\leftarrow d_{2i}+a_i\cdot a_i$

$d_{2i}\leftarrow y$，　$c\leftarrow x$，　$j\leftarrow i+1$；

S4. 若 $j\leqslant k-1$，则作

始　$(xy)_m\leftarrow d_{i+j}+2x_j\cdot x_i+c$，

$d_{i+j}\leftarrow y$，$c\leftarrow x$。

$j\leftarrow j+1$，转 S4。

终，否则作

$d_{i+k}\leftarrow x$，$i\leftarrow i+1$，转 S2；

S5. 输出$(d_{2k-1}\ d_{2k-2}\cdots d_2\ d_1\ d_0)_m$。

请注意求 A^2 比求不同的两个整数之积的两倍可能还要来得快，所以可利用

$$AB=[(A+B)^2-(A-B)^2]/4$$

求积。

例如，设 $A=989$，$m=10$，$k=3$，$a_0=9$，$a_1=8$，$a_2=9$，则

i	j	$d_{2i}+a_i^2$	$d_{i+j}+2a_ja_i+c$	x	y	d_5	d_4	d_3	d_2	d_1	d_0
0	--	0+81	—	8	1	0	0	0	0	0	1
	1	—	0+2·8·9+8	15	2	0	0	0	0	2	1
	2	—	0+2·9·9+15	17	7	0	0	0	7	2	1
				17	7	0	0	17	7	2	1
1	—	7+64	—	7	1	0	0	17	1	2	1
	2	—	17+2·9·8+7	16	8	0	0	8	1	2	1
				16	8	0	16	8	1	2	1
2	—	16+8	—	9	7	0	7	8	1	2	1
				9	7	9	7	8	1	2	1

6.6 除 法 运 算

已知 $A=(a_{h}a_{h-1}\cdots a_2\ a_1\ a_0)_m$，$B=(b_k\ b_{k-1}\cdots b_2\ b_1\ b_0)_m$，$h\geqslant k\geqslant 1$，$b_k\neq 0$。求 $A=qB+r$，其中 $0\leqslant r<B$。

设 $q=(q_{h-k}q_{h-k-1}\cdots q_1\ q_0)_m$，$r=(r_k\ r_{k-1}\cdots r_1\ r_0)_m$。

步骤：

S1. j 从 0 到 $h-k$ 作 $q_j \leftarrow 0$；

S2. 若 $A \geqslant m^{h-k}B$，则作

始 $q_{h-k} \leftarrow q_{h-k}+1$，$A \leftarrow A-m^{h-k}B$，

转 S2；

终。否则作

始 $i \leftarrow h$，转 S3

终。

S3. 若 $i>t$ 则作转 S4，否则转 S9；

S4. 若 $a_i=b_k$ 则作

$q_{i-k-1} \leftarrow m-1$，

否则作

$q_{i-k-1} \leftarrow \lfloor (a_i m+a_{i-1})/b_k \rfloor$；

S5. 若 $(q_{i-k-1}(b_k m+b_{k-1}))>a_i m^2+a_{i-1}m+a_{i-2}$ 则作

始 $q_{i-k-1} \leftarrow q_{i-k-1}-1$，转 S5；

终。

S6. $A \leftarrow A-q_{i-k-1}Bm^{i-k-1}$；

S7. 若 $A<0$ 则作

始 $A \leftarrow A+Bm^{i-k-1}$，

$q_{i-k-1} \leftarrow q_{i-k-1}-1$；

终。

S8. $i \leftarrow i-1$；转 S3。

S9. $r \leftarrow A$，输出 (q,r)。

例如，$A=721948327$，$B=84461$，$h=8$，$k=4$，$b_4=8$，$b_3=4$，$b_2=4$，$b_1=6$，$b_0=1$，$m=10$，则

i	q_4	q_3	q_2	q_1	q_0	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
—	0	0	0	0	0	7	2	1	9	4	8	3	2	7
8	0	9	0	0	0	7	2	1	9	4	8	3	2	7
							4	6	2	6	0	3	2	7
7		8	5	0	0			4	0	2	9	8	2	7
6		8	5	5	0			4	0	2	9	8	2	7
		8	5	4	0				6	5	1	3	8	7
5		8	5	4	8				6	5	1	3	8	7
		8	5	4	7					6	0	1	6	0

所以 $q=8547$， $r=60160$

现将过程叙述如下：

步骤 S2 由于 $721948327<84461\times10^4$，故转 S3；

步骤 S3 由 $i=8>5=k+1$，故转 S4；

步骤 S4 中 $a_8\neq b_4$ 即 $7\neq8$，故

$$q_{5-4-1}=\lfloor(a_8\cdot10+a_7)/b_4\rfloor$$

即

$$q_3=\lfloor72/8\rfloor=9$$

步骤 S5 条件满足，故 $q_3\leftarrow9-1=8$；

步骤 S6 中 $721948321-8(84461)\cdot10^3=721948321-675688000=46260327$；

步骤 S8 中 $i\leftarrow7$；

步骤 S3 中 $q_2=\lfloor(a_7\cdot10+a_6)\rfloor/b_4=\lfloor4b/8\rfloor=5$；

其余同此不再详述。

注意：上述的算法如若 $b_k\geqslant\left\lfloor\frac{m}{2}\right\rfloor$ 则 S2 执行不必超过一次。而且 S5 执行也不会超过两次。将 S4 的 $q_{i-k-1}\leftarrow\lfloor(a_im+a_{i-1})/b_k\rfloor$ 改为 $q_{i-k-1}\leftarrow\lfloor(a_im+a_{i-1}m+a_{i-2})/(b_km+b_{k-1})\rfloor$ 也可改善算法。

为了保证 $b_k\geqslant\left\lfloor\frac{m}{2}\right\rfloor$，可将 A,B 代以 lA 和 lB，其中 l 是适当选择的数，lA 除以 lB，q 不变，但 r 增加了 l 倍。l 最好选择 2 的幂，因为这样 lA 只要对 A 的二进制表示左移 8 位。

例如 $A=73418$，$B=267$，为了保证 $b_k\geqslant\left\lfloor\frac{10}{2}\right\rfloor=5$，可选 $l=3$。$A'=3A=220254$，$B'=3B=801$，则

i	q_3	q_2	q_1	q_0	a_5	a_4	a_3	a_2	a_1	a_0
—	0	0	0	0	2	2	0	2	5	4
5	0	2	0	0		6	0	0	5	4
4		2	7	0			3	9	8	4
3		2	7	4				7	8	0

$$q=274,\quad r=780/3=260$$

算法的复杂性分析，其结果使 A 的位数增加一位，每一回迭代需要 $1+(k+2)=k+3$ 次一位数乘法，计算需要 $(h-k)(k+3)$ 次一位数乘法，至于除法需要 $h-k$ 次一位数的除。

6.7 模幂算法

RSA 加、解密变换都要进行 mod n 的幂运算

求 $x\pmod p$ 可通过对指数 r 的二进制数化来实现。例如求 $11^7\pmod{17}$，$7=$

(111)，即

$$7 = 2^2 + 2^1 + 2^0 = 2^2 + 2 + 1$$

故

$$11^7 \pmod{17} = (11)^{2^2} \cdot 11^2 \cdot 11 \pmod{17}$$

下面给出一种传统的方法。在此前，先通过实例观察幂运算的计算步骤。

$$p = 1520^{13} \pmod{2537} = (1520)^{2\times 6+1} \pmod{2537}$$

$$= (1520^2)^6 1520 \pmod{2537}$$

$$(1520)^2 \equiv 1730 \pmod{2537}$$

$$p \equiv (1730)^6 1520 \pmod{2537}$$

$$= (1730^2)^3 1520 \pmod{2537}$$

$$1730^2 \equiv 1777 \pmod{2537}$$

$$p \equiv (1777)^3 \cdot 1520 \pmod{2537}$$

$$= 1777^2 \cdot (1777 - 1520) \pmod{2537}$$

$$1777^2 \pmod{2537} \equiv 1701$$

$$1777 \cdot 1520 \pmod{2537} \equiv 1672$$

$$p \equiv 1701 \cdot 1672 \pmod{2537} \equiv 95$$

1. 求 $x^r \bmod p$ 的算法之一

S1. $a \leftarrow x, b \leftarrow r, c \leftarrow 1$；

S2. 若 $b=0$ 则输出 c；结束。

S3. 若 b 是正的偶整数则作

始 $b \leftarrow b/2, a \leftarrow a^2 \bmod p$，转 S3；

终。否则转 S4。

S4. $b \leftarrow b-1, c \leftarrow ac \bmod p$，转 S2。

例 6.1 求 $(16)^{13} \bmod 4731$。

(1) $a \leftarrow 16, b \leftarrow 15, c \leftarrow 1$。

(2) $b \leftarrow 14, c \leftarrow 16$。

(3) $b \leftarrow 7, (16)^2 \bmod 4731 \equiv 256, a \leftarrow 256$。

(4) $b \leftarrow 6, 256 \times 16 \equiv 4096 \bmod 4731, c \leftarrow 4096$。

(5) $b \leftarrow 3, (256)^2 \equiv 4033 \bmod 4731, a \leftarrow 4033$。

(6) $b \leftarrow 2, 4033 \times 4096 \equiv 3247 \bmod 4796, c \leftarrow 3247$。

(7) $b \leftarrow 1, (4033)^2 \equiv 4642 \bmod 4731, a \leftarrow 4642$。

(8) $b \leftarrow 0, 4642 \times 3247 \equiv 4339 \bmod 4731, c \leftarrow 4339$。

(9) $b=0$，所以 $(16)^{13} \equiv 4359 \bmod 4731$。

2. 求 $x^r \bmod p$ 算法之二

若将 r 化为二进制数：

$$r=(r_l r_{l-1}\cdots r_1 r_0)_2=\sum_{i=0}^{l} r_i 2^i=\sum_{r_j\neq 0} 2^j$$

则
$$x^r \bmod p \equiv x^{\sum_{r_j\neq 0} 2^j} \bmod p = \prod_{r_j\neq 0} x^{2^j} \bmod p$$

例 6.2 求 $16^7 \bmod 4731$。

由于 $7=(111)_2$

$$j=0,16^{2^0}=16;\quad j=1,16^{2^1}=16^2=256$$

$$j=2,16^4\equiv(256)^2 \bmod 4731\equiv 65536 \bmod 4731\equiv 3033$$

所以
$$16^7 \bmod 4731\equiv 16\times 256\times 3033 \bmod 4731$$
$$\equiv 4096\times 3033 \bmod 4731=12423168 \bmod 4731$$
$$\equiv 2846$$

3. 求 $x^r \bmod n$ 算法之三

还可以将步骤归纳为，若

$$r=r_{l-1}r_{l-2}\cdots r_1 r_0,\quad r_i\in(0,1),i=0,1,\cdots,l-1$$

S1. $A\leftarrow 1,B\leftarrow x\quad i\leftarrow l-1$；

S2. 若 $i\geqslant 0$ 则作

$$A\leftarrow A*A \bmod n;$$

S3. 若 $r_i=1$，则作 $A\leftarrow A*B \bmod n, i\leftarrow i-1$，转 S2。

以 $m^{23} \bmod n$ 为例，$23=(10111)_2$，则

i	r_i	$A(\leftarrow 1)$	$B(\leftarrow x)$
0	1	x	x^2
1	1	$x^3=x\cdot x^2$	x^4
2	1	$x^7=x^4\cdot x^3$	x^8
3	0	x^7	$x^{16}=(x^8)^2$
4	1	$x^{23}=x^{16}\cdot x^7$	

由上读者仔细分析可见若 $r_i=0$ 只要作一次平方的运算，而 $r_i=1$ 时则除了平方以外还得作一次乘法的运算。所以可以将 $r=r_{l-1}r_{l-2}\cdots r_1 r_0$ 每逢 $r_i=0$ 代以 S 表示作平方运算的意思。每当 $r_i=1$，则代以 SM 表示作平方及乘法的意思。结果得一由 S 和 M 构成的序列，比如 $23=(10111)_2$，得 $R=SMS\ SM\ SMSM$。除去最左端的第一个 SM 结果执行

S1. $A\leftarrow 1$；

S2. 从左向右扫描 R 若遇到字符 S 则作

$$A\leftarrow A^2$$

若遇到 M 则执行

$$A\leftarrow A\cdot x$$

直到最后。

以 x^{23} 为例，$23=(10111)$ 为例

	初始	S	M	S	S	M	S	M	S	M
A	1	1	X	X^2	X^4	X^5	X^{10}	X^{11}	X^{22}	X^{23}

还可以从 $S\ SM\ SMSM$ 的过程看运算状态变化

$$\underset{(x)^2}{S} \longrightarrow \underset{(x^2)^2}{S} \longrightarrow \underset{(x^2)^2\cdot x}{M} \longrightarrow \underset{((x^2)^2\cdot x)^2}{S} \longrightarrow \underset{((x^2)^2\cdot x)^2\cdot x}{M} \longrightarrow$$

$$\underset{(((x^2)^2\cdot x)^2\cdot x)^2}{S} \longrightarrow \underset{(((x^2)^2\cdot x)^2\cdot x)^2\cdot x}{M}$$

即
$$x^{23}=(((x^2)^2\cdot x)^2\cdot x)^2\cdot x$$

而且 r 的二进制表示第一个字符必然为 1，即 $r_{l-1}=1$，R 的前两个字符必为 SM，可再进一步简化，将第一个 SM 去掉，余 $R\doteq SSMSMSM$，改为

S1. $A\leftarrow X$；

S2. 对 R' 从左向右扫描，若遇到 S 则作

$A\leftarrow A^2$

若遇到 M 则作

$A\leftarrow AX$

可得

	初始	S	S	M	S	M	S	M
A	X	X^2	X^4	X^5	X^{10}	X^{11}	X^{22}	X^{23}

顺便谈一个有意义的值得重视的事实。正因为由于 $r=r_{l-1}r_{l-2}\cdots r_1r_0$ 遇到 0 时只相应作一次平方运算，遇到 1 时除了作一次平方运算外，还要作一次乘的运算，即遇到的运算时间较遇到 0 的运行时间为长。计算机运行时间相应地有微波辐射，如若测得机器运行时的微波辐射的强弱可推测出 r 的二进制表示，给 RSA 的破解会带来威胁。

4. 算法之四

在这里顺便提一提，将指数 r 写成二进制数以后，有时会发现存在某一整数 k 使得 $X^k \bmod n$ 出现次数较多，也可以预处理 $x^k \bmod n$。举一个极端的例子，求

$$x^{27} \bmod n$$

将 27 写成二进制数为$(11011)_2$，$(11)_2=3$，则

$$x^{27} \bmod n=(x^3 \bmod n)^{2^3}\cdot x^3 \bmod n$$

可预定理 $x^3 \bmod n$。指数往右移一位，相当于作一次平方的运算。

若预先计算 $x^3 \bmod n$ 作两次乘法，计算 x^{27} 还要作三次平方，最后再作一次乘法。共四次平方、二次乘法运算。若将 27 化为$(11011)_2$，再写成

$$SMSSMSM$$

需作 3 次乘法，4 次平方运算。又如

因为
$$55=(110111)_2$$

所以 $$x^{55}=((x^3)^{2^3}\cdot x^3)^2\cdot x$$

共用 3 次乘法，5 次平方，若通过

$$SM\ S\ SMSMSM$$

则需要 5 次平方，4 次乘法。

上面讲了几种求 x^r，也就是 $x^r \bmod n$ 的算法，实质上是相同的道理，只不过不同的理解而已。下面转而讨论加速求模幂运算的方法。

6.8 Barrett 求模算法

Barrett 归约法用于计算 $x \bmod m$，它先得计算 $\left\lfloor \frac{b^{2k}}{m} \right\rfloor$，对于模同一个模数有效，比如 RSA 的加密。

已知 $x=(x_{2k-1}\ x_{2k-2}\cdots x_2\ x_1\ x_0)_b$，$m=(m_{k-1}\ m_{k-2}\cdots m_1\ m_0)_b$，$m_{k-1}\neq 0$，$\mu=\lfloor b^{2k}/m \rfloor$，求 $x \bmod m$。

S1. $q_1 \leftarrow \lfloor x/b^{k-1} \rfloor$，$q_2 \leftarrow q_1\mu$，$q_3 \leftarrow \lfloor q_2/b^{k+1} \rfloor$；

S2. $r_1 \leftarrow x \bmod b^{k+1}$，$r_2 \leftarrow q_3 m \bmod b^{k+1}$；$r \leftarrow r_1 - r_2$；

S3. 若 $r<0$ 则 $r \leftarrow r+b^{k+1}$；

S4. 若 $r \geqslant m$ 则作

始 $r \leftarrow r-m$，转 S4；

终。否则转 S5。

S5. 输出 r。

由于 $x=qm+r$，$0 \leqslant r<m$，故 $q-2 \leqslant q_3 \leqslant q$。

又由于 $q=\lfloor (x/b^{k-1})(b^{2k}/m)(1/b^{k+1}) \rfloor$，因为

$$\frac{x}{b^{k-1}}\frac{b^{2k}}{m}\frac{1}{b^{k+1}}=\left\lfloor \frac{x}{m} \right\rfloor$$

故 $$q_3=\lfloor \lfloor x/b^{k-1} \rfloor M/b^{k+1} \rfloor$$

故 q_3 不超过 q。请注意

$$-b^{k+1}<r_1-r_2<b^{k+1}$$

$$r_1-r_2 \equiv (q-q_3)m+r \bmod b^{k+1}$$

设 $x=3561$，写成二进制数表示，$b=4$，$k=3$。

$$\begin{aligned} x &= (110111101001)_2 \\ &= ((11)(01)(11)(10)(10)(01)) \\ &= (313221)_4 \end{aligned}$$

$$m=47=(101111)_2=((10)(11)(11))_2=(233)_4$$

$$\begin{aligned} \mu &= \lfloor b^{2k}/m \rfloor = \lfloor 4^6/47 \rfloor = \lfloor (1000000)_4/(233)_4 \rfloor \\ &= (1113)_4 = 87 \end{aligned}$$

通过移位求

$$q_1 = \left\lfloor \frac{x}{b^{k-1}} \right\rfloor = \left\lfloor \frac{(313221)_4}{4^2} \right\rfloor = (3132)_4$$

通过四进制数的乘法得：

$$q_2 = q_1 \cdot \mu = (3132)_4(1113)_4 = (10231302)_4$$

通过移位求

$$q_3 = \left\lfloor \frac{q_2}{b^{k+1}} \right\rfloor = \left\lfloor \frac{10231302}{4^4} \right\rfloor = (1023)_4$$

$$r_1 = x \bmod b^{k+1}$$

$$= (313221)_4 \bmod 4^4 = (3221)_4$$

$$r_2 = q_3 \cdot m \bmod b^{k+1}$$

$$= (1023)_4(233)_4 \bmod 4^4$$

$$= (313011)_4 \bmod 4^4 = 3011$$

$$r_1 - r_2 = (210)_4 = 36$$

$$x \bmod m = 3561 \bmod 47 = 36$$

算法是基于观察 $\lfloor x/m \rfloor = q = \lfloor (x/b^{k-1})(b^{2k}/m)(1/b^{k+1}) \rfloor$ 因 $\frac{x}{b^{k-1}} \frac{b^{2k}}{m} \frac{1}{b^{k+1}} = \frac{x}{m}$。$q$ 和 $q_3 = \lfloor \lfloor x/b^{k-1} \rfloor \mu/b^{k+1} \rfloor$ 很近似，因

$$\frac{x}{b^{k-1}}\mu/b^{k+1} = \frac{x}{b^{k-1}} \lfloor b^{2k}/m \rfloor \frac{1}{b^{k+1}}$$

故用 q_3 近似 q，q_3 不会超过 q。而且就差最多也不超过 2。

同时 $-b^{k+1} < r_1 - r_2 < b^{k+1}$，$r_1 - r_2 \equiv (q - q_3)m + R \bmod b^{k+1}$，$0 \leqslant (q - q_3)m + R < 3m < b^{k+1}$，因为 $m < b^k$，$b > 3$。

若 $r_1 - r_2 \geqslant 0$，则

$$r_1 - r_2 = (q - q_3)m + R;$$

若 $r_1 - r_2 < 0$，则

$$r_1 - r_2 + b^{k+1} = (q - q_3)m + R。$$

因 $0 \leqslant r < 3m$ 故算法中 S4 最多重复两次。

Barrett 求模算法的除法运算都只要通过移位来实现。

q_2 只是为了计算 q_3 而设的，q_2 的后面 $k+1$ 位数字对 q_3 的决定并不需要。

假定基 b 关于 k 是足够大，对 q_2 来说最小的 $k-1$ 有效位可以不计算。因 μ 和 q_1 最多有 $k+1$ 位数。

确定 q_3 最多只需要 $(k+1)^2 - \binom{k}{2} = (k^2 + 2k + 1) + \frac{k(k-1)}{2} = \frac{1}{2}(k^2 + 5k + 2)$ 一位数的乘法。

同样的理由 $r_2 = q_3 m \bmod b^{k+1}$ 也只要计算后面最小的 $k+1$ 位，最多只要作

$$\binom{k+1}{2} + k = \frac{k(k+1)}{2} + k = \frac{1}{2}(k^2 + 3k)$$

次一位数的乘法。

6.9 多位数的 Montgomery 求模算法

令 m 是一正整数，R 和 T 是整数，$R>m$，$(m,R)=1$，即 m 和 R 互素，$0\leqslant T<mR$。计算 $TR^{-1} \bmod m$ 有一种方便的方法，无需像传统方法一样先求 TR^{-1}，再求 $TR^{-1} \bmod m$。

令 A、B 是满足 $0\leqslant A,B<m$ 的整数，令

$$\bar{A} = AR \bmod m$$

$$\bar{B} = BR \bmod m$$

$\bar{A}\bar{B}$ 的 Montgomery 归约是

$$\bar{A}\bar{B}R^{-1} \bmod m = ABR \bmod m$$

简单地说，以计算

$$A^5 \bmod m \quad 1 \leqslant A < m$$

为例，首先 $\bar{A}=AR \bmod m$，再计算 $\bar{A}\bar{A}$ 的 Montgomery 的归约：

$$M = \bar{A}^2R^{-1} \bmod m$$

M^2 的归约为

$$M^2R^{-1} \bmod m = \bar{A}^4R^{-3} \bmod m$$

最后 $(M^2R^{-1} \bmod m)\bar{A}$ 的 Montgomery 归约为

$$(M^2R^{-1})\bar{A}R^{-1} \bmod m = \bar{A}^5R^{-4} \bmod m = A^5R \bmod m$$

乘以 $R^{-1} \bmod m$，得 $A^5 \bmod m$。

如果 m 是基 b 长度为 n 的数，选 $R=b^n$。$R>m$ 明显满足，但 $\gcd(R,m)=1$ 的充分条件是 $\gcd(b,m)=1$。

已知 m 和 R 两个整数，$\gcd(m,R)=1$，令

$$m' = -m^{-1} \bmod R$$

$$0 \leqslant T < mR$$

若 $U=Tm' \bmod R$，$T+Um \equiv T \bmod m$，则 $(T+Um)/R$ 是整数，且 $(T+Um)R^{-1} \equiv TR^{-1} \bmod m$，$(T+Um)R^{-1}$ 是一整数，$U=Tm'+kR$，$m'm=-1+lR$，k 和 l 是整数，则

$$\begin{aligned}(T+Um)/R &= [T+(Tm'+kR)m]/R \\ &= [T+T(-1+lR)+kRm]/R = lT+km\end{aligned}$$

则有：

(1) $(T+Um)/R$ 是对 $TR^{-1} \bmod m$ 的计算，因 $T<mR$，$U<R$ 则 $(T+Um)/R<(mR+mR)/R=2m$，故

$$(T+Um)/R = TR^{-1} \bmod m \quad 或 \quad (T+Um)/R = (TR^{-1} \bmod m)+m$$

(2) 若 $R=b^n$，则 $TR^{-1} \bmod m$ 可以通过计算两个多位的数的乘来实现，即 $U=T \cdot m'$ 和 $U \cdot m$，因为除以 R 可以简单地通过对 $T+Um$ 的向右移位来完成。

例 6.3 令 $m=187$，$R=190$。则有 $190^{-1} \bmod 187 \equiv 125$，$187^{-1} \bmod 190 = 63$。

现先计算如下：

$$190 = 187 + 3,\ 3 = 190 - 187,187 = 62 \times 3 + 1$$

所以 $$1 = 187 - 62 \times 3 = 187 - 62(190 - 187)$$

$$= 63 \times 187 - 62 \times 190$$

所以 $$190^{-1} \bmod 187 = -62 \bmod 187 = 125$$

$$187^{-1} \bmod 187 = 63$$

$$m' = -m \bmod R$$

即 $$m' = -187^{-1} \bmod 190$$

所以 $$m' = -63 \bmod 190 = 127$$

若 $T=563$，则 $U = Tm' \bmod R$，即

$$U \equiv 563 \times 127 \bmod 190$$

$$\equiv 183 \times 127 \bmod 190$$

$$\equiv 23241 \bmod 190 \equiv 61$$

$$(T + Um)/R \bmod m \equiv TR^{-1} \bmod m \equiv 563 \times 125 \bmod 187$$

$$\equiv 2 \times 125 \bmod 187 \equiv 250 \bmod 187 - 63$$

若 $T=1125$，则 $Tm' \bmod R \equiv U \equiv 1125 \times 127 \bmod 190$，即

$$U \equiv 175 \times 127 \bmod 190 \equiv 142875 \bmod 190 \equiv 185 \equiv 7$$

已知 $m = (m_{n-1} m_{n-2} \cdots m_1 m_0)_b$，$gcd(m, b) = 1$，$R = b^n$，$m' = -m^{-1} \bmod b$，$T = (t_{2n-1} t_{2n-2} \cdots t_1 t_0)_b < mR$，$A = (a_{2n-1}, a_{2n-2} \cdots a_1 a_0)_b$。

求 $TR^{-1} \bmod m$ 的 Montgomery 的归约步骤。

S1. $A \leftarrow T$；$i \leftarrow 0$；

S2. 若 $i \leqslant n-1$ 则转 S3，否则转 S4；

S3. $u_i \leftarrow a_i m' \bmod b$；

$A \leftarrow A + u_i m b^i$；

$i \leftarrow i+1$，转 S2；

S4. $A \leftarrow A/b^n$；

S5. 若 $A \geqslant m$ 则 $A \leftarrow A - m$；

S6. 输出 A。

关于算法的几点说明：

(1) $m' = -m^{-1} \bmod R$，由于 $R = b^n$，故改为

$$m' = -m^{-1} \bmod b$$

(2) 在步骤 S3，存在 l，A 有 $a_j = 0, 0 \leqslant j \leqslant l-1$ 时令 $a_l = 0$。使得 S3 中 A 可被 b^n 整除。

(3) 在步骤 S4，A 的值等于 T 加上某一 m 的倍数，$A = (T + km)/b^n$ 是一整数，且 $A \equiv TR^{-1} \bmod m$，只要证 $A < 2m$，S5 最多只要一次减法。

$$T = T + \sum_{i=0}^{n-1} u_i b^i m$$

但 $\sum_{i=0}^{n-1} u_i b^i m < b^n m = Rm$，$T<Rm$，所以

$$A < 2Rm$$

故 $A/R<2m$。

例如 $m=72639$，$b=10$，$R=10^5$，$T=7118368$，$n=5$，$m'=-m^{-1} \bmod 10=1$，$T \bmod m=72385$，计算 $TR^{-1} \bmod m$ 如下：

i	$u_i=a_i m' \bmod 10$	$u_i m b^i$	A
		-	7118368
0	8	581112	7699480
1	8	5811120	13510600
2	6	43583400	57094000
3	4	290556000	347650000
4	5	3631950000	3979600000

6.10 乘的 Montgomery 算法

已知 $m=(m_{n-1}m_{n-2}\cdots m_1 m_0)_b$，$x=(x_{n-1}x_{n-2}\cdots x_1 x_0)_b$，$y=(y_{n-1}y_{n-2}\cdots y_1 y_0)_b$，$0 \leqslant x, y \leqslant m$，$R=b^n$，$\gcd(m,b)=1$，$m'=-m^{-1} \bmod b$。求 $xyR^{-1} \bmod m$。设 $A=(a_n a_{n-1}\cdots a_1 a_0)_b$。

S1. $A \leftarrow 0$，$i \leftarrow 0$；

S2. 若 $i<n$ 则转 S3 否则转 S4；

S3. $u_i \leftarrow (a_0+x_i y_0)m' \bmod b$；

$A \leftarrow (A+x_i y+u_i m)/b$；

$i \leftarrow i+1$，转 S2；

S4. 若 $A \geqslant m$，则 $A \leftarrow A-m$；

S5. 输出 A。

S3 的第 i 轮迭代 $0 \leqslant A<2m-1$，A 代以 $(A+x_i y+u_i m)/b$，但 $(A+x_i y+u_i m)/b \leqslant (2m-2+(b-1)(m-1)+(b-1)m)/b=2m-1-(1/b)$ 故 $A<2m-1$。$A+x_i y+u_i m$ 是 b 的倍数，除以 b 可以向右移位来取代。S2～S3，执行 n 次，共需 $2n(n+1)=n(2n+2)$ 次一位数乘法运算。

特别要注意利用 Montgomery 乘法求 $xy \bmod m$，假定 x，y 和 m 都是 b 进制位数为 n 的数。且

$$0 \leqslant x, y < m$$

不考虑输入的预计算，算法计算 $xyR^{-1} \bmod m$ 共需 $2n(n+1)$ 次一位数乘。不考虑 $R^2 \bmod m$ 的计算，利用算法计算 $xyR^{-1} \bmod m$ 和 $R^2 \bmod m$，$xy \bmod n$ 共作

$$4n(n+1)$$

次一位数乘的运算。比起经典的乘法要不经济，但 Montgomery 在求模幂运算时比较好。

假如用 $M(u,v)$表利用 Montgomery 算法计算 $xyR^{-1} \bmod m$，$(m,R)=1$。

Montgomery 指数算法，已知 $m=(m_{l-1}m_{l-2}\cdots m_1 m_0)_b$，$R=b^l$，$m'=-m^{-1} \bmod b$。$l=(l_t l_{t-1}\cdots l_1 l_0)$，$l_t=1$，$1\leqslant x<m$，求 $x^l \bmod m$。

算法假定 $R \bmod m$ 和 $R^2 \bmod m$ 预先计算妥当作为输入。

S1. $X \leftarrow M(x,R^2 \bmod m)$，$A \leftarrow R \bmod m$，$i \leftarrow t$；

S2. 若 $i \geqslant 0$ 则转 S3，否则转 S4；

S3. $A \leftarrow M(A,A)$；

若 $e_i=1$ 则 $A \leftarrow M(A,x)$；

$i \leftarrow i-1$，转 S2；

S4. $A \leftarrow M(A,1)$；

S5. 输出 A。

例如，假定已知 x，m 和 R，$e=(1011)_2$，$t=3$。

i	3	2	1	0
$A \bmod m$	x	x^2R^{-1}	x^5R^{-4}	$x^{11}R^{-10}$

$$M(x^{11}R^{-11},1)=x^{11}R^{-11}=x^{11}$$

Montgomery 模幂运算的复杂度，即求 $x^l \bmod m$ 所需的一位数乘法的平均数为 $3l(l+1)(t+1)$。

S3 每一次迭代调用 Montgomery 乘法需作 $2l(l+1)$次一位数的乘法，但无需一位数的除。类似的模幂运算的古典算法同样需 $2l(l+1)$次一位数乘法，除此之外，l 次一位数的除。

6.11 加 法 链

加法链的目的是对作幂运算时减少乘法的数目。

定义 6.1 对于一个正整数 n 的一个长为 S 的序列称之为加法链：

$$V=\{c_0, c_1, \cdots, c_S\}$$

c_i 都是正整数，与 i 有关的数偶序列

$$b_1, b_2, \cdots, b_S$$

其中 $b_i=(i_1,i_2)$，$0\leqslant i_1,i_2<i$，使得$\{c_i\}$满足

(i) $c_0=1$，$c_1=n$

(ii) 每一个 c_i，$1\leqslant i\leqslant S$，$c_i=c_{i_1}+c_{i_2}$

加法链与幂：即通过加法链达到求 g^n 的运算

S1. $g_0 \leftarrow g$，$i \leftarrow 1$；

S2. 若 $i \leqslant s$ 作

始 $g_i \leftarrow g_{i_1} * g_{i_2}$，$i \leftarrow i+1$，转 S2，

终。

否则转 S3。

S3. 输出 g_S。

例如，$n=15$，$S=5$，使得 $c_0=1$，$c_1=2$，$c_2=3$，$c_3=6$，$c_4=12$，$c_5=15$，$b_1=(0,0)$，$b_2=(0,1)$，$b_3=(2,2)$，$b_4=(3,3)$，$b_5=(2,4)$。

当 $n=15$ 是指数时，求 g^n 便有

$i=0$　$g_0=g$；

$i=1$　$b_1=(0,0)$，$g_1=g_0\cdot g_0=g^2$，

$i=2$　$b_2=(0,1)$，$g_2=g_0\cdot g_1=g\cdot g^2=g^3$；

$i=3$　$b_3=(2,2)$，$g_3=g_2\cdot g_2=g^3\cdot g^3=g^6$；

$i=4$　$b_4=(3,3)$，$g_4=g_3\cdot g_3=g^6\cdot g^6=g^{12}$；

$i=5$　$b_5=(2,4)$，$g_5=g_4\cdot g_4=g^3\cdot g^{12}=g^{15}$。

这样求 g^{15} 只用到 5 次乘法。

若将 15 表示成二进制数：$15=(1111)_2$，根据求幂运算的法则

$$15=2^0+2^1+2^2+2^3$$

$$g^{15}=g\cdot g^2(g^2)^2\cdot((g^2)^2)^2$$

求 g^{15} 次方共需 6 次乘法，求平方也看作是一次乘法。

找最短的加法链问题属于 NP-困难类问题。也就是已知 n，求最短的加法链，使求 g^n 所需乘法数达到最小问题本身是困难的，没有有效的办法。

6.12　预处理算法

先举一例，求 x^{2731}。将指数 2731 化为二进制数，即

$$2731=(\underline{1010}\ \underline{1010}\ \underline{1011})_2$$

若取窗口宽度为 3，则(101)出现 3 次。$(101)_2=5$，预计算 $x^5=(x^2)^2\cdot x$，则

$$x^{2731}=(((x^5)^{2^4}\cdot x^5)^{2^4}\cdot x^5)^2x$$

共作 11 次平方，4 次乘法。

若取 $10=(1010)_2$，$x^{10}=((x^2)^2)^2\cdot x^2$ 预计算 x^{10}，则 $x^{2731}=((x^{10})^{2^4}\cdot x^{10})^{2^4}\cdot x^{10}\cdot x$，也只作 11 次平方和 4 次乘法。

若将 2731 写成对应的

$$R=SSMSSMSSMSSMSSMSM$$

则共作 11 次平方，6 次乘法，节省了$\frac{1}{8}$的计算量。

若要求做到最少的运算，必须对 r 的结构作分析，找出出现次数最多的项，然后再进行评估，特别在计算机自动进行，不由人进行干预。下面提出一种特定窗口长进行估计，特别是窗口长度允许不同的近优算法。比如下面例子对窗口长为 3 和 2 两种情况。

例 6.4　求 $r^{1420027}$。

$$1420027=(\underline{101}\ \underline{0110}\ \underline{1010}\ \underline{101}\ \underline{11}\ \underline{11}\ \underline{011})_2$$

窗口长为 3 位的有 101，窗口长为 2 的有 11，先预计算 x^2，$x^3=x^2\cdot x$，$x^5=x^4\cdot x^2$，需 2 次乘法，1 次平方。

$$x^{1420027}=(((((((x^2)^{2^3}\cdot x^5)^{2^4}\cdot x^5)^{2^4}\cdot x^3)^{2^4}\cdot x^3)^{2^4}\cdot x^3$$

共计 19 次平方，8 次乘法。若按常规计算，则需 20 次平方，15 次乘法。节省了近$\frac{1}{3}$的计算量。适当地预处理无疑可提高效率。

例 6.5 求 x^{493}

$$493=(111101101)_2$$

但

$$493=167+152+174$$

即将 493 分析成 167＋152＋174 的和，而

$$167=(10100111)_2$$

$$152=(10011000)_2$$

$$174=(10101110)_2$$

于是得一矩阵

$$\boldsymbol{R}=\begin{pmatrix}1&0&1&0&0&1&1&1\\1&0&0&1&1&0&0&0\\1&0&1&0&1&1&1&0\end{pmatrix}$$

对于 $\boldsymbol{R}$ 矩阵的每一列的列向量，其中 1 的个数 r，可对应一 x^r。例如

$$\begin{pmatrix}1\\0\\1\end{pmatrix}$$

的权 2，对应一 x^2。

从 $\boldsymbol{R}$ 矩阵各列，可预处理 x^2 和 x^5。则

$$x^{493}=(((((((x^3)^4\cdot x^2)^2\cdot x)^2\cdot x^2)^2\cdot x^2)^2\cdot x^2)^2\cdot x$$

共用 8 次平方，7 次乘法。

同时对于矩阵 $\boldsymbol{R}$，可以类似于窗口法，分割成窗口宽度等于 2 的窗口如下，原则上是

$$\begin{pmatrix}0&0\\0&0\\0&1\end{pmatrix},\ \begin{pmatrix}0&0\\0&1\\0&1\end{pmatrix},\ \begin{pmatrix}0&1\\0&1\\0&1\end{pmatrix},\ \begin{pmatrix}0&0\\0&0\\1&0\end{pmatrix},\ \begin{pmatrix}0&0\\0&0\\1&1\end{pmatrix},\ \cdots,\ \begin{pmatrix}1&1\\1&1\\1&1\end{pmatrix}$$

例如 $\begin{pmatrix}1&0\\1&0\\1&1\end{pmatrix}$ 应对应于 $x^7=(x^5)^2\cdot x$，实际上本题只需 x^6 和 x^5 两项，无需预处理从 x^2 到 x^7。

例如

$$\begin{pmatrix}1&0\\0&1\\1&0\end{pmatrix}$$

应有

$$(x^2)^2\cdot x=x^5$$

其他依此类推。

故预处理 x^5 和 x^6。则

$$x^{491}=(((x^6)^4\cdot x^5)^4\cdot x^6)^4x^5$$

如若将 493 分解为 3 个数

$$493=164+164+165$$
$$164=(10100100)_2$$
$$165=(10100101)_2$$

形成矩阵

$$\boldsymbol{R}=\begin{vmatrix}1&0&1&0&0&1&0&0\\1&0&1&0&0&1&0&0\\1&0&1&0&0&1&0&1\end{vmatrix}$$

$\boldsymbol{R}$ 矩阵除最后一列外，各列均为全 1 或全 0。

将 $\boldsymbol{R}$ 分拆成

$$\begin{pmatrix}1&0\\1&0\\1&0\end{pmatrix},\ \begin{pmatrix}1&0\\1&0\\1&0\end{pmatrix},\ \begin{pmatrix}0&1\\0&1\\0&1\end{pmatrix},\ \begin{pmatrix}0&0\\0&0\\0&1\end{pmatrix}$$

实际上只要预处理两项：

$$x^6,\ x^3$$
$$x^{493}=(((x^6)^4\cdot x^6)^4\cdot x^3)^4\cdot x$$

共用 8 个平方，4 个乘法。

本题所用的方法可推广到一般

$$\begin{pmatrix}0&1\\0&1\\0&1\end{pmatrix},\ \begin{pmatrix}1&0\\1&0\\1&0\end{pmatrix},\ \begin{pmatrix}1&1\\1&1\\1&1\end{pmatrix}$$

分别对应于

$$x^3,\ x^6,\ x^9$$

这样预处理量可大大地降低。只要预处理 3 次求 x^r，将 r 分成几个等分，形成的 R 矩阵的窗口取多宽与效率有关。比如说分成 4 等分，R 矩阵除了最后一列外各列都是全 1 或全 0。

6.13 大数模运算的预处理算法

预处理技术是对 $m \bmod n$ 进行计算之前，先对某些特定的数进行预计算，获得这些数的模运算的结果。在求 $m \bmod n$ 可利用预计算的结果以加快运算速度。

设 m 和 n 分别是 ξ,η 位二进制数，$r=\xi-\eta>0$，

$$m=m_{\xi-1}m_{\xi-2}\cdots m_1m_0,\ n=n_{\eta-1}n_{\eta-2}\cdots n_1n_0$$

为计算 $m \bmod n$，预计算

$$a_i\leftarrow i2^n \bmod n,\ i=1,2,\cdots,2^r-1$$

将这 2^r-1 个数存储起来，以备查用。

S1. i 从 1 到 2^r-1 作　$a_i \leftarrow 2^\eta i \bmod n$；　　/*预处理*/

S2. $v \leftarrow \left\lfloor \frac{m}{2^\eta} \right\rfloor$；　　/*模运算*/

$m' \leftarrow m \bmod 2^\eta + a_v$

S3. 若 $m' \geqslant 2n$ 则作

始　　$m \bmod n \leftarrow m'-2n$　　/*调整结果*/

终。

否则作

始　若 $m'>n$　则作

$$m \bmod n \leftarrow m'-n$$

否则作

$$m \bmod n \leftarrow m'$$

终。

其中 S1 是预处理形成一表。S2 则利用预处理作模运算，以避免低效的除法运算。S3 为对结果进行调整，由于 S2 的结果可能超过 n。可以证明 S2 的结果不会大于或等于 $3n$，证明从略。

预处理结果进行存储，需要 $(2^r-1)\eta$ 比特。显然随着 r 的增大而指数型增长。所以 r 的值不宜过大。

为了减少预处理存储，可选一整数 d，计算

$$a_i \leftarrow id2^\eta \bmod n \quad i=1,2,\cdots,\left\lfloor \frac{2^r-1}{d} \right\rfloor$$

再预计算

$$b_j \leftarrow j2^\eta \bmod n, \quad j=1,2,\cdots,d-1$$

存储量共为

$$\left(\left\lfloor \frac{2^r-1}{d} \right\rfloor + d-1\right)\eta \text{ 比特}$$

算法：

S1. i 从 1 到 $\lfloor (2^r-1)/d \rfloor$ 作　　/*预处理*/

$$a_i \leftarrow id2^\eta \bmod n$$

S2. j 从 1 到 $d-1$ 作

$$b_i \leftarrow j2^\eta \bmod n$$

S3. $u \leftarrow \left\lfloor \frac{m}{2^\eta} \right\rfloor$, $v \leftarrow \left\lfloor \frac{\mu}{d} \right\rfloor$。　　/*第 1 次查表*/

$$m' \leftarrow m \bmod 2^\eta + (u \bmod d)2^\eta + a_v$$

S4. $u \leftarrow \left\lfloor \frac{m'}{2^\eta} \right\rfloor$, $v \leftarrow \left\lfloor \frac{\mu}{d} \right\rfloor$。

$$w \leftarrow u \bmod d$$

$$m'' \leftarrow m' \bmod 2^\eta + a_v + b_w;$$　　/*第 2 次查表*/

S5. 若 $m''\geqslant 2n$ 则作

$$m \bmod n \leftarrow m''-2n \qquad /*\text{调整结果}*/$$

否则若 $m''\geqslant n$ 则作

$$m \bmod n \leftarrow m''-n$$

否则

$$m \bmod n \leftarrow m''$$

采用上面的算法降低存储器的容量，代价是在进行模运算的时候，要查两次表。第一次查表后，能保证结果小于$(d+1)2^{\eta}$，第二次查表后能保证结果小于 $3n$。在第二次查表中 u 的最大值为 1。所以当 $v=1$ 时，w 必为 0。在模幂乘的各次运算中，并不要保证结果小于 $3n$。通过运算小于$(d+1)2^{\eta}$，就可以在模乘运算时使用进行一次查表，保护模幂乘运算的效率不因采用基于表的预处理而降低。在运算都完成后，才利用表使结果小于 $3n$。最后结果再调整，以保证模幂乘结果小于 n。

例 6.6 $m=22230$，$n=299$

$m=(101\ 0110\ 1101\ 0110)_2$，$\xi=15$

$n=(1\ 0010\ 1011)_2$，$\eta=9$

$\gamma=15-9=6$

$d=2^3=8$

预处理主表：

$$\begin{aligned}
a_1 &= 8\times 2^9 \bmod 299 = 4096 \bmod 299\\
&= 209 \bmod 299\\
&= (11010001)_2\\
a_2 &= 16\times 2^9 \bmod 299 \equiv 418 \bmod 299\\
&\equiv 119 = (1110111)_2\\
a_3 &= 24\times 2^9 \bmod 299 = 29 = (1\ 1101)_2\\
a_4 &= 32\times 2^9 \bmod 299 = 238 = (1110\ 1110)_2\\
a_5 &= 40\times 2^9 \bmod 299 = 148 = (1001\ 0100)_2\\
a_6 &= 48\times 2^9 \bmod 299 = 58 = (11\ 1010)_2\\
a_7 &= 56 * 2^9 \bmod 299 = 267 = (1\ 0000\ 1011)_2
\end{aligned}$$

类似可得副表：

$$\begin{aligned}
b_1 &= 1 * 2^9 \bmod 299 = 213 = (0\ 1101\ 0101)_2\\
b_2 &= 2 * 2^9 \bmod 299 = 127 = (0\ 0111\ 1111)_2\\
b_3 &= 3 * 2^9 \bmod 299 = 41 = (0\ 0010\ 1001)_2\\
b_4 &= 4 * 2^9 \bmod 299 = 254 = (0\ 1111\ 1110)_2\\
b_5 &= 5 * 2^9 \bmod 299 = 168 = (0\ 1010\ 1000)_2\\
b_6 &= 6 * 2^9 \bmod 299 = 82 = (0\ 0101\ 0010)_2\\
b_7 &= 7 * 2^9 \bmod 299 = 295 = (1\ 0010\ 0111)_2
\end{aligned}$$

第 1 次查表：

$$u = \left\lfloor \frac{m}{2^9} \right\rfloor = \left\lfloor \frac{22230}{512} \right\rfloor = 43$$

$$v = \left\lfloor \frac{\mu}{d} \right\rfloor = \left\lfloor \frac{43}{8} \right\rfloor = 5$$

所以

$$\begin{aligned} m' &= m \bmod 2^8 + (U \bmod d)2^4 + a_v \\ &= 0\ 1101\ 0110 + 110\ 0000\ 0000 + 1001\ 0100 \\ &= 111\ 0110\ 1010 \end{aligned}$$

第 2 次查表：

$$\left\lfloor \frac{m'}{2^8} \right\rfloor = 3$$

$$\begin{aligned} m' \bmod n &= m' \bmod 2^9 + b_3 \\ &= 1\ 0110\ 1010 + 0\ 0010\ 1001 = 1\ 1001\ 0011 \end{aligned}$$

调整结果：

$$1\ 1001\ 0011 - n = 110\ 1000$$

所以

$$m \bmod n \equiv 110\ 1000$$

6.14 利用中国剩余定理加快 RSA 解密

RSA 的解密运算要作大数的模幂运算：

$$m \equiv c^d \bmod n$$

根据 Fermat 定理 $c^{p-1} \equiv 1 \bmod p$，故设计算 $m_1 \equiv c^d \bmod p \equiv (c \bmod p)^{d \bmod (p-1)} \bmod p$，$m_2 \equiv c^d \bmod q \equiv (c \bmod q)^{d \bmod (q-1)} \bmod q$。

根据中国剩余定理，必须先求得

$$q\,y_1 \equiv 1 \bmod p$$

$$p\,y_2 \equiv 1 \bmod q$$

的解 y_1，y_2。由于用户已知 p 和 q 可以计算得 y_1 和 y_2。$y_1 = p^{-1} \bmod q$，$y_2 = q^{-1} \bmod p$，则 $m \equiv qm_1 y_1 + m_2 y_2 \bmod n$。

问题导致解

$$p\,y_1 \equiv 1 \bmod q$$

$$q\,y_2 \equiv 1 \bmod p$$

及计算

$$m \equiv qm_1\ y_1 + pm_2 y_2 \bmod n$$

由经验可知，软件实现只有原来计算量的$\frac{1}{4}$，可将 qy_1 和 py_2 记录备用，令 $A = qy_1$，$B = py_2$。

第 7 章　大素数生成及其有关算法

公钥密码与许多数学问题有关。这一章除了讨论大素数的生成，随机数的生成外，还相对集中地讨论其他有关的数学算法。

7.1　素数的概率测试法

素数的研究有很长的一段历史，整个数论几乎都在讨论它，但近代密码学的兴起给它注入了新的活力，提出了新课题，其中最重要的是素数的判定，特别是大素数的生成，还有大数的因数分解。这些都是与公钥密码密切相关的，特别是强素数的提出，源自 RSA 对 p,q 两素数的要求。

先介绍比较实用的概率算法。

在进行素数测试之前，必先将合数过滤掉。特别是素数的分布比较稀疏，所以筛选过滤十分重要。

7.1.1　若干关于素数的判定定理

关于素数分布有一个重要定理。

定理 7.1　设 $\pi(x)$ 为小于或等于 x 的全部素数个数，则

$$\lim_{x\to\infty}\pi(x)\Big/\frac{x}{\ln x}=1$$

当 x 充分大时

$$\pi(x)\sim\frac{x}{\ln x}$$

利用这个公式可以估计在 $10^{99}\sim10^{100}-1$ 这个区间上素数的个数，即 100 位十进制数中素数的个数 N 为

$$N\sim\frac{10^{100}}{\ln 10^{100}}-\frac{10^{99}}{\ln 10^{99}}=\frac{10^{100}}{100\ln 10}-\frac{10^{99}}{99\ln 10}$$

$$=\frac{10^{99}}{\ln 10}\left(\frac{10}{100}-\frac{1}{99}\right)\sim\frac{9\cdot 10^{99}}{100\ln 10}$$

100 位十进制数的个数为

$$10^{100}-10^{99}-1\sim 9\cdot 10^{99}$$

故 100 位十进制数的素数密度为 $\dfrac{1}{100\ln 10}$。

一般可证 n 位十进制数中素数的密度当 n 充分大时约为 $\dfrac{1}{n\ln 10}$。素数的稀疏性从此可知。一般说来，判定一个数是素数比较难，而否定它是素数要容易得多。比如偶素数只

有 2 这一个，故偶数应排除。同样，最后一位为 5 的数也不是素数。我们希望有一个简便而有效的算法能将大量合数“淘汰”出去，对余下的数再进行素数判定，好在素数的研究在数论中是比较成熟的。

确定素数的一种十分直观的“筛法”，以下面序列为例：

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

在上面数列中除去 2 的倍数的数，即其中有“\”记号的数；剩下的数列中第 1 个数 3 便是素数。再在余下的数列中除去以 3 为因数的数，即其中打“/”记号的数；余下的第一个数“5”便是素数。再在余下的数列中，除去“5”的倍数的数，即其中有“—”记号的数；余下的第 1 个数 7 是素数。再从剩下的数列中除去 7 的倍数，即有“|”记号的数；11 便是素数，依此类推。

可见，若 n 为合数 $n_1 n_2$，则 n_1, n_2 必有一个因数小于$\sqrt{n}$。要是 n 不是合数，则必须用小于$\sqrt{n}$的所有素数来除，而且除不尽。以 n 为 200 位十进制数为例，$\sqrt{n}$将为十进制数 100 位，小于十进制数 100 位的素数数目约为

$$\frac{10^{100}}{100\ln 10} = \frac{10^{98}}{\ln 10}$$

也就是说要作大致可达此数目的除法，以每秒能完成 10^7 大数除法运算的计算机来计算需时

$$T = \frac{10^{98}}{\ln 10} \Big/ 3.1536 \times 10^{14}$$
$$= 10^{84}/3.1536 \cdot \ln 10 \text{（年）}$$

7.1.2 判定素数的确定型多项式算法

上面的讨论我们已看到，若用常规的“筛法”判定 n 是素数，必须除以小于$\sqrt{n}$的所有的素数，包括等于$\sqrt{n}$，若$\sqrt{n}$是素数。当 n 的位数为 100 位或更大时，实际上是不可行的。

在相当一段时间，密码学在寻找大素数时，都用的是概率算法。具体地说，任给一随机数用 Fermat 定理，若 n 是素数，对任意数 a 必然有

$$a^{n-1} \equiv 1 \bmod n$$

但 $a^{n-1} \equiv 1 \bmod n$ 不一定是素数，只是机会少一些，若对 $a_1, a_2, \cdots, a_k$，都满足

$$a_i^{n-1} \equiv 1 \bmod n \quad i = 1, 2, \cdots, k$$

则 n 不是素数的可能性将随 k 的增大而减小，这就是概率算法的思想，后面将讨论它。

2002 年印度的数学家 Agrawal 等人在一篇“Primes is in P”的论文中提出了一种确定型的多项式算法。确定型是对概率算法型而言的。Agrawal 等人的算法是直接判定 n 是素数的方法，它用到许多数论中的概念，所以证明从略，下面只叙述它的判定过程。

S1. $r \leftarrow 2$；

S2. 当 $r < n$，转 S3；

S3. 若 $\gcd(n, r) \neq 1$，则判定 n 是合数；

S4. 若 r 是素数，q 是 $r-1$ 的最大素数因子，若 $(q\geqslant 4\sqrt{r}\log n)\wedge(n^{\frac{r-1}{q}}\not\equiv 1 \bmod r)$，则转 S6，否则转 S5；

S5. $r\leftarrow r+1$，转 S2；

S6. 对 $1\leqslant a\leqslant 2\sqrt{r}\log n$，

若 $(x-a)^n\not\equiv(n^n-a) \bmod (x^r-1,n)$，

则 n 是合数；

S7. n 是素数。

这里 log 是以 e 为底的自然对数。

算法的复杂性是 $O((\log n)^{15})$。

证明从略。

n 不是素数的机会随 $k\to\infty$ 而渐近于 0。这将在 7.2 节讨论。即判断素数而采用了素数的必要条件，而非充分条件。算法的基本思想是根据等式：

若 a 与 p 互素，则 p 是素数的充要条件是：

$$(x-a)^p\equiv(x^p-a) \bmod p$$

这个等式的证明很简单。

对于 $0\leqslant i<p$，$((x-a)^p-(x^p-a))$ 中 x^i 的系数是 $(-1)^i\binom{p}{i}a^{p-i}$。若 p 是素数，则

$$\binom{p}{i}\equiv 0 \bmod p$$

根据 Fermat 定理

$$a^p\equiv a \bmod p$$

故必要性成立。

若 p 是合数，设 q 是 p 的一个因数，$q^k\mid p$，则 q^k 不能除尽 $\binom{p}{q}$，而且和 a^{p-q} 互素，故 x^q 项的系数不为零(mod p)，故

$$((x-a)^p-(x^p-a))\not\equiv 0 \bmod p$$

等式证毕。

对素数判定有一个充要条件，叫做威尔逊(Wilson)定理。

定理 7.2（威尔逊定理） n 是素数的充要条件是

$$(n-1)!\equiv -1(\bmod n)$$

证明 必要性。n 是素数，所以在 $\{1,2,\cdots,n-1\}$ 中每一个元素 a 都有一元素 a^{-1}，使得 $aa^{-1}\equiv 1(\bmod n)$，除 1 和 $n-1$ 这两个数它的逆元素为自身外，即

$$1\cdot 1\equiv 1(\bmod n),\quad (n-1)\cdot(n-1)\equiv 1(\bmod n)$$

其他 $n-3$ 个数，$a^{-1}\not\equiv a$，故成对出现 $aa^{-1}\equiv 1(\bmod n)$。

$$(n-1)!\equiv(n-1)\equiv -1(\bmod n)$$

充分性。用反证法。即 $(n-1)!\equiv -1 \bmod n$ 成立，但假定 n 不是素数。

设 $n=a\cdot b$，其中 $a,b>1$，则 $a\mid(n-1)!$ 同样理由 $a\mid[(n-1)!]+1]$，故 $a\mid[(n-1)!+$

$1-(n-1)!]$，但$[(n-1)!+1]-(n-1)!=1$。这与$a>1$的假设矛盾。从而证明n是素数。

威尔逊定理给出判定素数的充要条件，有很高的理论价值。但用来寻找素数并不适当。

关于素数还有一个 Fermat 定理。此定理给出素数p的必要条件，若不满足，则可断定它不是素数。

定理内容为，若p是素数，则对于任意的整数a，应有

$$a^{p-1}\equiv 1 \pmod p$$

遗憾的是存在一种数，满足 Fermat 定理，但不是素数。即 Fermat 定理的逆不真。$a^{p-1}\equiv 1 \bmod p$只是p为素数的必要条件。$a^{p-1}\not\equiv 1 \bmod p$，$p$必非素数。

定义 7.1 n是合数，若对于$Z_n^*=\{x \mid 1\leqslant x\leqslant n,\gcd\{x,n\}=1\}$中任一元素$a$恒有

$$a^{n-1}\equiv 1 \bmod n$$

则称n为卡密沙尔(Carmichael)数。

已经知道最小的卡密沙尔数是 561。$561=3\times 11\times 17$。

$$F_n \triangleq \{a \mid 0\leqslant a\leqslant n,\gcd\{a,n\}=1,a^{n-1}\equiv 1 \bmod n\}$$

显然，当n是素数时，F_n和Z_n^*是一样的。

7.2 素数的 Miller-Rabin 概率测试法

令$n-1=2^l m$，其中l是非负整数，m是正奇数。若$b^m\equiv 1 \pmod n$或$b^{2^j m}\equiv -1 \pmod n$，$0\leqslant j\leqslant i-1$，则称$n$通过以$b$为基的 Miller-Rabin 测试。

定理 7.3 若n是素数，b是整数，且$n\nmid b$，则n必然通过以b为基的米勒测试。

证明 令

$$S_k \equiv b^{\frac{n-1}{2^k}} \pmod n \equiv b^{2^{l-k}m} \pmod n \quad k=l,l-1,\cdots,2,1,0$$

其中$S_l\equiv b^m \pmod n$，$S_0=b^{n-1} \pmod n$。

若n是素数，则由 Fermat 定理可知

$$S_0 \equiv b^{n-1} \equiv 1 \pmod n$$

必然成立。它的必然结果是

$$S_1 \equiv b^{(n-1)/2} \equiv 1 \pmod n \text{ 或 } S_1 \equiv -1 \pmod n$$

必然成立。而且$S_1\equiv 1 \pmod n$或$S_1\equiv -1 \pmod n$成立时，Fermat 定理必然成立。因$S_0\equiv S_1^2\equiv 1 \pmod n$。同理，若$S_2\equiv 1 \pmod n$或$S_2\equiv -1 \pmod n$成立时，则$S_1\equiv 1 \pmod n$，因而满足 Fermat 定理。

依此类推，若存在一整数$k>0$，使

$$S_{k-1}\equiv 1 \pmod n \quad \text{或} \quad S_{k-1}\equiv -1 \pmod n$$

则

$$S_k \equiv S_{k-1} \equiv \cdots \equiv S_0 \equiv 1 \pmod n$$

即 Fermat 定理成立。

定理说明 Miller-Rabin 测试一旦通过，Fermat 定理便可满足。注意，在计算 S_k 的过程中证明从 $k=l,l-1,l-2,\cdots,2,1,0$ 依此进行，最终为 S_0，即为 $b^{n-1}\equiv 1(\bmod n)$。

7.3 关于因数分解的讨论

非对称密码系统中以 RSA 最为看好，它的安全性依赖于因子分解的困难性，即大数的因子分解的计算困难。

$n=pq$ 若被分解，则系统便被攻破，所以对 n 的选择十分重要，必须选择好 p 和 q 使之分解变得计算上十分困难或不可能。

近来数学家在大数分解上取得了令人注目的进步，特别是 1994 年 4 月，Lenstra 领导的跨国科研群体，约 600 余人，集中了 1600 台计算机，在网络上作分布计算，花了 8 个月的时间，分解了称为 RSA129 这个长 129 位十进制合数。这说明 n 长为 129 位的 RSA 已被攻破，虽然代价很高。究竟 n 应取多长才算保险？Lenstra 的分解因数方法是否具有普遍意义？对此数学家作了复杂性的估计，每增加 10 位十进制数，分解的复杂性增加 10 倍，也就是说分解 n 的位数为 139 位的难度，大致为分解长为 129 位的 10 倍。按此估计，应该说 RSA 还没有受到严重威胁，前锋离并预设 n 取 200 位十进制数还有很长一段距离。对于商业上的应用，在一段时间内取 n 为 512 比特还是比较安全。

n 的长度只是问题的一方面，在 n 的长度确定之后，p 和 q 的选择极为重要，加密密钥 e 的选择也有关系，分别讨论于后。

7.3.1 $p-1$ 因数分解法

在引进强素数之前，先介绍 Pollard $p-1$ 分解法。知道了 Pollard 的 $p-1$ 分解法，才明白应该避免 n 被分解的要求是什么。假定要对合数 n 进行因数分解，p 是 n 的某一个因数，若 $p-1$ 不存在大的素数因子，则下面方法可用来找出因数 p。

S1. 选一整数 k，要求 k 为比界 B 小的所有数的倍数，比如取 $k=B!$，即 k 为比 B 小的所有数的公倍数；

S2. 在 2 和 $n-2$ 之间取一数 a，a 可以是 2，也可以是 3，或是随机的；

S3. 利用快速求模幂运算法计算

$$a^k \bmod n$$

S4. 利用欧几里得算法和 $a^k \bmod n$ 计算 $\gcd(a^k,n)$；

S5. 若 d 不是 n 的非平凡的除数，则由重新选择 a 和 k，转到 S1 重新开始。

算法的根据说明如下：由于 k 被小于 B 和小于等于 B 的正整数除尽，p 是 n 的因子，倘若 $p-1$ 是小于 B 的素数的幂之积，则 k 是 $p-1$ 的倍数，根据 Fermat 定理 $a^k\equiv 1 \bmod p$，故

$$p \mid \gcd(a^k-1,n)$$

失败的可能仅仅由于

$$a^k \equiv 1 \bmod n$$

例如，

$$n=540143,\quad B=8,\quad k=840,\quad a=2$$

$$2^{840} \bmod n \equiv 53047$$

$$540143 = 10 \times 53046 + 9683$$

$$9683 = 540143 - 10 \times 53046$$

$$53046 = 5 \times 9683 + 4631$$

$$4631 = 53046 - 5 \times 9683$$

$$9683 = 2 \times 4631 + 421$$

$$421 = 9683 - 2 \times 4631$$

$$4631 = 11 \times 421$$

所以

$$\gcd(53046, 540143) = 421$$

$$540143 = 1283 \times 421$$

540143 之所以被因子分解，由于

$$421 - 1 = 420 = 2^2 \times 3 \times 5 \times 7$$

即 540143 有一因数 421，而 421－1 的所有因数都是小素数和它的幂。

和 $p-1$ 分解法类似的有 $p+1$ 法，也就是说若 $p+1$ 可分解为小素数和它的幂的乘积时，p 是数 n 的因数时，n 可被因数分解。即若 $n=r$，

$$p - 1 = \prod_{i=1}^{\cdot} p_i^{\alpha_i} \quad 或 \quad p \pm 1 = \prod_{j=1}^{\cdot} q_j^{\beta_j}$$

所有的 $p_i \leqslant B, q_j \leqslant B$，$B$ 是已知小的数，则 n 可被因子分解。

所以 RSA 中 $n=pq$，p 和 q 还必须附加上它的 $p-1$ 和 $p+1$ 的所有因数不是小的数，一般要求 p 的长度大于 512 比特，$p-1$ 至少有大素数因子 q，q 的长度应接近 p 的长度。

7.3.2 关于 p 和 q 的讨论

1. 若 p 和 q 很小，显然 $n=pq \approx p^2$。

先估计

$$p \approx \sqrt{n} \approx q$$

再根据

$$\left(\frac{p+q}{2}\right)^2 - \left(\frac{p-q}{2}\right)^2 = pq = n$$

所以

$$\left(\frac{p+q}{2}\right)^2 - n = \left(\frac{p-q}{2}\right)^2$$

$\frac{p+q}{2}$ 是 p 和 q 的平均数，若 $p \approx q$，$\frac{p+q}{2} \approx \sqrt{n}$，可给一估计值，若为 $\bar{p}$，使

$$\bar{p}^2 - n = \left(\frac{p-q}{2}\right)^2$$

右端正好为一数的平方，则可得

$$\frac{p+q}{2} \quad 和 \quad \frac{p-q}{2}$$

例如，$n=94245$，

$$\left(\frac{p+q}{2}\right)^2 - n = \left(\frac{p-q}{2}\right)^2$$

给$\left(\frac{p+q}{2}\right)$估值307.则

$$(307)^2 = 94249$$

$$94249 - 94245 = 4 = 2^2$$

$$\frac{1}{2}(p+q) = 307$$

$$\frac{1}{2}(p-q) = 2$$

所以

$$p = 309$$

$$q = 305$$

故 p 和 q 的差必须比较大，比如差几个字节。

2. $p-1$ 和 $q-1$ 的最大公因数应很小。

若 $p-1$ 和 $q-1$ 的最大公因数很小，则若攻击者截获密文

$$C \equiv m^e \bmod n$$

后作

$$C_1 = C$$

$$C_2 \equiv C_1^e \bmod n$$

$$\equiv (m^e)^e \bmod n \equiv m^{e^2} \bmod n$$

$$C_3 \equiv C_2^e \bmod n \equiv m^{e^3} \bmod n$$

$$\vdots$$

$$C_k \equiv C_{k-1}^e \bmod n \equiv (m^{e^{k-1}})^e \bmod n \equiv m^{e^k} \bmod n$$

若 $\gcd(p-1, q-1)$ 很小，设为 g。

根据欧拉定理有

$$e^{\phi(n)} \equiv 1 \bmod n$$

若存在 l 使 $e^l \equiv 1 \bmod \phi(n)$，则有 $e^l \equiv h\phi(n)+1$。

由欧拉定理

$$m^{\phi(n)} \equiv 1 \bmod n$$

$$C_l \equiv m^{e^l} \bmod n \equiv m \bmod n$$

假定 l 很小，则可通过以上办法恢复明文，而无需因数分解 n，达到攻击 RSA 的目的。

如果 g 很大，则$\frac{p-1}{g}$和$\frac{q-1}{g}$将很小。根据 $\phi(n) = \phi(p)\phi(q) = (p-1)(q-1)$ 有

$$\phi(n) = \left(\frac{p-1}{g}\right)\left(\frac{q-1}{g}\right)g^2 = p_1 q_1 g^2$$

其中

$$p_1 = \frac{p-1}{g}, \quad q_1 = \frac{q-1}{g}$$

若 $g=2$，

$$\phi(n) = \frac{(p-1)(q-1)}{2} \times 2 = 2l$$

$$l = \frac{1}{2}(p-1)(q-1)$$

$$(e^l)^2 \equiv 1 \bmod \phi(n)$$

$$e^l \equiv 1 \bmod \phi(n)$$

一般有

$$(e^{\frac{1}{g}(p-1)(q-1)})^g \equiv 1 \bmod n$$

若

$$(e^{\frac{1}{g}(p-1)(q-1)}) \equiv 1 \bmod n$$

则

$$l = \frac{1}{g}(p-1)(q-1)$$

所以最理想 $\gcd\{p-1, q-1\}=2$，使 l 达到最大。

7.4 关于 e 和 d 的讨论

(1) e 只要求 $(e, \phi(n))=1$. 为了加快加密速度，自然想到取 e 尽可能小一些，但太小了也欠妥，比如 $e=3$。

$$C \equiv m^3 \bmod n$$

若 $m^3 < n$ 时，加密过程不存在模 n 的操作，这时，解密导致对 C 的开立方。

特别是若有 3 个人，他的加密密钥都是 3。若某甲欲传递相同的 m 给 3 个人，设

$$C_1 = m^3 \bmod n_1$$

$$C_2 = m^3 \bmod n_2$$

$$C_3 = m^3 \bmod n_3$$

假定 n_1, n_2, n_3 两两互素，如若不然，可通过求公因数使 n_1, n_2, n_3 被因子分解，系统被攻。

若 n_1, n_2, n_3 两两互素，可利用中国剩余定理求得

$$m^3 \equiv C \bmod (n_1 n_2 n_3)$$

由于 m 分别小于 n_1, n_2, n_3，故 $m^3 < n_1 n_2 n_3$。

解 $m^3 \equiv C \bmod (n_1 n_2 n_3)$ 导致求立方根。

总之，e 不能太小，比如 16 比特，比较合适。

同时应选择 e 使得 $e^l \equiv 1 \bmod \phi(n)$ 的 l 可取最大值 $\frac{1}{2}(p-1)(q-1)$，这一点在前面已讨论过了。

(2) d 的选择应大于 $\sqrt[4]{n}$ 许多场合要求 d 短一些，以减少解密或签名所需要的时间，比如 IC 卡的应用，IC 卡的运算能力比主机低得多。所以 e 可稍大些，使 d 缩短，将矛盾转让给主机，问题发生于其结果是否会降低 RSA 的安全性？d 太短，对 RSA 进行攻击的搜索空间将缩小，这是无疑的。有人指出，n 为 512 比特时，d 的长度小于 n 的长度的 1/4 时，利用连分数法在多项式时间内破译成功。

总的考虑是宁可降低 e 的长度，也不要降低 d 的长度。

7.5 随机数产生器

密钥的产生必须是随机地，而且对于各种可能密钥的概率应该是等概率的。但这很难实现。非随机的密钥往往给敌人以推测的可能。好在完全均匀分布对密钥并非是本质的，最根本的还是不可预测。

产生随机数的最原始方法是掷银币、扔骰子。主密钥产生后很少更换，可以用这样的人工方法来实现，由完全可信赖的人来完成。其他任何方法都多少有点可被预测的危险，银币和骰子的一些偏差对产生的随机数没有明显的变化，不必介意。

主密钥是如此重要，故它被两重、甚至三重加密保存。主密钥及加密主钥的密钥可授权不同的人来产生，以分散主密钥的安全责任。不至于由一个人的失误而全局皆输在他手里。

但人工产生密钥的方法对大量的密钥是不合适的。对于频繁地需要新密钥，可采用物理的办法或伪随机数产生器来完成。

电噪声对放大器设计是一个讨人厌的问题，但可以派上好用场，电阻即使在低温下也产生噪声，宽频带的放大器将这些噪声转换为信号，使开关开或关。但并不是任何一种噪声都是可利用来产生随机数的。要求随机数序列是不混进非随机的分量，是宽频带而且是正态分布的平稳随机过程。

不同原理的噪声产生电路有真空管的、稳压二极管的，还有专用的噪声芯片，但它们都多少带有非随机量的影响。

关于电噪声随机数产生器的线路构造这里不拟深入讨论。下面仅介绍若干伪随机数产生器的例子。

有许多由数学方法产生的序列具有随机性，遗憾的是，通过统计测试发现这些伪随机数都有一定的模式。但是好的加密算法可帮助生成伪随机序列。假如该算法是好的加密算法，它的密文将产生一好的伪随机序列，反之，如若它产生的伪随机序列有一定的模式，则该算法必然抗攻击能力弱。下面将介绍如何利用加密算法构造实用的伪随机数产生器。

例如选定了密钥 k，对自然数 $1,2,\cdots$，加密的产生随机数序列。即序列 R_n 为

$$R_n = E_k(n)$$

若用的密钥相同，则两个伪随机序列将相同。所以伪随机数产生器应从随机数开始，这个数称为“种子”，从种子产生的序列不可能是无穷长，但可以是非常长，可以考虑用人工方法产生一随机序列

$$m_1, m_2, \cdots, m_n$$

再用人工方法产生一随机数 k，最后得随机序列

$$E_k(m_1), E_k(m_2), \cdots, E_k(m_k)$$

下面介绍一种利用加密算法构造随机数产生器方案。

1. 方案之一：其流程图如图 7.1 所示，其中

$$X_i = DES_{k_1}(V_i)$$

$$Y_i = DES_{x_i}^{-1}(X_i)$$

$$V_{i+1} = DES_{k_2}(Y_i)$$

$$W_i = DES_{x_i}^{-1}(V_i)$$

$$R_i = DES_{k_2}(W_i)$$

$$Z_{i+1} = DES_{k_1}^{-1}(Z_i)$$

V_1 = Date(64 比特)

Z_1 = Time(64 比特)

R_i 是所求的随机数。

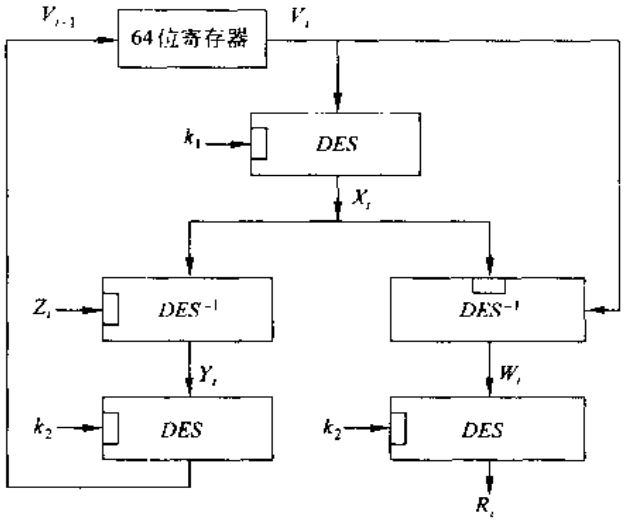


图 7.1

其中 k_1，k_2 是 64 比特的密钥，Z_i 的产生器，如图 7.2 所示。

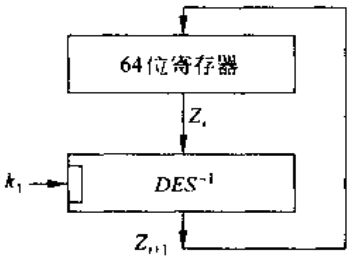


图 7.2

2. 方案之二，其流程图如图 7.3 所示。

其中 k_1，k_2 都是 128 比特的密钥，Z_0 = Date，X_1 = Time，则

$$Z_i = IDEA_{k_1}^{-1}(Z_{i-1})$$

$$Y_i = IDEA_{k_2}(Z_i)$$

$$W_i = Y_i \oplus X_i$$

$$R_i = IDEA_{k_1}^{-1}(W_i)$$

$$X_{i+1} = IDEA_{k_2}(Y_i \oplus R_i)$$

上面两例都是利用 64 比特的 Date，Time 作为种子。R_i 是产生的随机数。

3. 方案之三，其流程如图 7.4 所示。

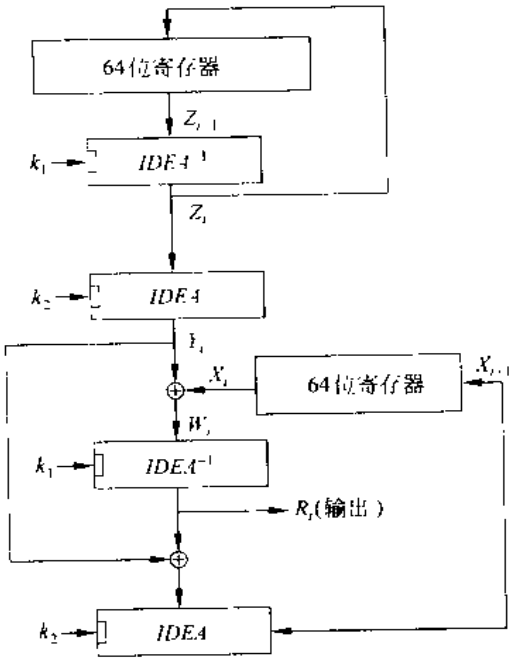


图 7.3

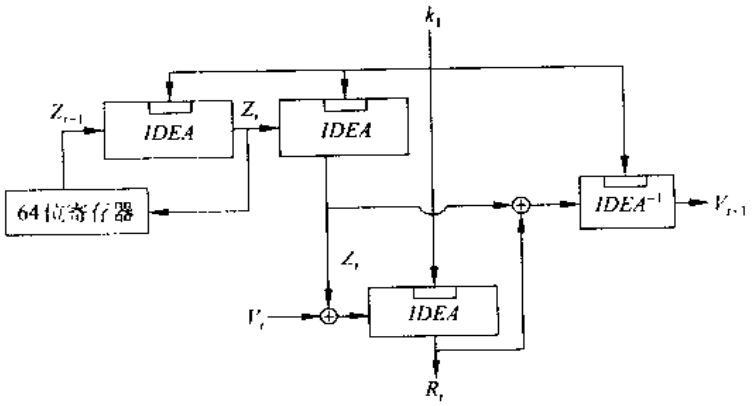


图 7.4

k_1 是 128 比特的密钥，

Z_0 是 64 比特的 Date(日期)，

V_1 是 64 比特的 Time(时间)，

$Z_i = IDEA_{k_1}(Z_{i-1})$，

$Z_i = IDEA_{k_2}(Z_i)$，

$R_i = IDEA_{k_1}(V_i \oplus Z_i)$，

$V_{i+1} = IDEA_{k_1}^{-1}(Z_i \oplus R_i)$。

4. BBS 随机数产生器。

BBS 是 3 位开发者 Blum，Blum，Shub 的缩写，先选择两个大素数 p 和 q，满足条件：

$$p \equiv q \equiv 3 \bmod 4$$

令 $n=pq$，令 S 是满足与 n 互素的随机数，即 S 没有因数 p 或 q。

BBS 产生器产生 B_i 比特的序列的步骤是：

S1. $X_0 = S^2 \bmod n$，

$i \leftarrow 1$；

S2. $X_i \leftarrow (X_{i-1})^2 \bmod n$

$R_i \leftarrow X_i \bmod 2$

$i \leftarrow i+1$，转到 S2。

每轮迭代可选取其最小有效位。

下面是一例子，$n=383\times503=192649$，$S=101355$

i	X_i	R_i
0	20749	
1	143135	1
2	177671	1
3	97048	0
4	89992	0
5	174051	1
6	80649	1
7	45663	1
8	69442	0
9	186894	0
10	177046	0
11	137922	0
12	123175	1
13	8630	0
14	114386	0
15	14863	1
16	133015	1
17	106065	1
18	45870	0
19	137171	1
20	48060	0

7.6 序列的随机性统计检验

7.6.1 χ^2 检验

从信息安全的角度来看伪随机序列，影响它的性能标准最关键的因素：首先是周期长度，当然周期长度越长越好；其次是可预测性如何？无疑序列的规律性越差，其可预测性也就越小。

对其性能的预测用到统计学中的 χ^2 检验法，它是用来测试随机序列的概率分布是否接近估计的某一模式。

先举一例，设有两个骰子，同时掷出可能出现以下几种可能，即点数为

$$2,3,4,5,6,7,8,9,10,11,12$$

若骰子是正常的，则出现点数为 i 的概率 p_i 应为

$$i=2,3,4,5,6,7,8,9,10,11,12$$

$$p_i=\frac{1}{36},\frac{2}{36},\frac{3}{36},\frac{4}{36},\frac{5}{36},\frac{6}{36},\frac{7}{36},\frac{8}{36},\frac{9}{36},\frac{10}{36},\frac{11}{36},\frac{12}{36}$$

现作 n 次试验，若出现点数之和为 i 的次数为 X_i，欲判断所掷的两个骰子是否正常，引进公式

$$\chi^2=\sum_{i=1}^{k}(X_i-np_i)^2/np_i$$

其中

$$\sum_{i=1}^{k}X_i=n,\quad \sum_{i=1}^{k}p_i=1$$

k 是可能出现的事件的个数，在本例中 $k=11$，n 是试验次数，np_i 是第 i 种事件的期望值，即作 n 次试验中可能出现的次数的期望值，若

$$X_i=np_i,\quad i=1,2,\cdots,k$$

即 X_i 等于它的期望值，则 $\chi^2=0$。

偏差 X_i-np_i 越大，说明序列偏离假定的概率分布越远。所以 χ^2 的值是这偏离程度的度量。由 k 和 χ^2 值查 χ^2 表，概率 p。

判断的规则：

(1) $0\leqslant p\leqslant 0.01$ 或 $0.99\leqslant p\leqslant 1$，则否定的判定予以拒绝。

(2) $0.2\leqslant p\leqslant 0.3$ 或 $0.7\leqslant p\leqslant 0.8$，则予以肯定。

(3) $0.3\leqslant p\leqslant 0.7$ 则认为十分优秀。

其余情况为不能确定。

为了准确起见，要求试验次数 n 满足

$$\min_i\{np_i\}\geqslant 5$$

一般说来 χ^2 检验也必须进行多次。比如 3 次以上。每次试验应取不同的样本。

假定一八面体，若该八面体正常则各面出现的几率均等，各为 1/8。现作 800 次试验，测得出现的次数 X_i 如下：

i：1 2 3 4 5 6 7 8

X_i：91 110 89 108 112 93 107 90

$$\chi^2 = \{(91-100)^2 + (110-100)^2 + (89-100)^2 + (108-100)^2 + (112-100)^2 + (93-100)^2 + (107-100)^2 + (90-100)^2\}/100$$

$$= \{81 + 100 + 121 + 64 + 144 + 49 + 47 + 100\}/100$$

$$= 708/100 = 7.08$$

查 χ^2 表，$v=k-1=7$ 行中可见

$$\chi^2 = 7.08 < 9.03715$$

可得

$$p > 0.25$$

故该八面体正常的假定成立。

若有另一种试验结果

i：1 2 3 4 5 6 7 8

X_i：85 84 85 90 90 120 124 122

$$\chi^2 = \{(85-100)^2 + (84-100)^2 + (85-100)^2 + (90-100)^2 + (90-100)^2 + (120-100)^2 + (124-100)^2 + (122-100)^2\}/100$$

$$= \{225 + 196 + 225 + 100 + 100 + 400 + 576 + 484\}/100$$

$$= 2481/100 = 24.81$$

查 χ^2 表，$v=7$ 行可见

$$0.005 < p < 0.001$$

故应作不正常的判断。

χ^2 表

v\Q	0.995	0.99	0.975	0.95	0.9	0.75	0.5
1	(−5)3.92704	(−4)1.57088	(−4)9.82069	(−3)3.93214	0.0157908	0.101531	0.454937
2	(−2)1.00251	(−2)2.01007	(−2)5.06356	0.102587	0.210720	0.575364	1.38629
3	(−2)7.17212	0.114832	0.215795	0.351846	0.584375	1.212534	2.36597
4	0.206990	0.297110	0.384419	0.710721	1.063623	1.92255	3.35670
5	0.411740	0.554300	0.831211	1.145476	1.61031	2.67460	4.35146
6	0.675727	0.872085	1.237347	1.63539	2.20413	3.45460	5.34812
7	0.989265	1.239043	1.68987	2.16735	2.83311	4.25485	6.34581
8	1.344419	1.646482	2.17973	2.73264	3.48954	5.07004	7.34412
9	1.734926	2.087912	2.70039	3.32511	4.16816	5.89883	8.34283
10	2.15585	2.55821	3.24697	3.94030	4.86518	6.73721	9.34182
11	2.60321	3.05347	3.81575	4.57481	5.57779	7.58412	10.3410
12	3.07382	3.57056	4.40379	5.22603	6.30380	8.43842	11.3403
13	3.56503	4.10691	5.00874	5.89186	7.04150	9.29906	12.3398
14	4.07468	4.66043	5.62872	6.57063	7.78953	10.1653	13.3393

续表

$v\backslash Q$	0.995	0.99	0.975	0.95	0.9	0.75	0.5
15	4.60094	5.22935	6.26214	7.26094	8.54675	11.0365	14.3389
16	5.14224	5.81221	6.90766	7.96164	9.31223	11.9122	15.3385
17	5.69724	6.90766	7.56418	8.67176	10.0852	12.7919	16.3381
18	6.26481	7.01491	8.23075	9.39046	10.8649	13.6753	17.3379
19	6.84398	7.63273	8.90655	10.1170	11.6509	14.5620	18.3376
20	7.43386	8.26040	9.59083	10.8508	12.4426	15.4518	19.3374
21	8.03366	8.89720	10.28293	11.5913	13.2396	16.3444	20.3372
22	8.64272	9.54249	10.9823	12.3380	14.0415	17.2369	21.3370
23	9.26042	10.19567	11.6885	13.0905	14.8479	18.1373	22.3367
24	9.88623	10.8564	12.4011	13.8484	15.6587	19.0372	23.3367
25	10.5197	11.5240	13.1197	14.6114	16.4734	19.9393	24.3366
26	11.1603	12.1981	13.8439	15.3791	17.2919	20.8434	25.3364
27	11.8076	12.8786	14.5733	16.1513	18.1138	21.7494	26.3363
28	12.4613	13.5648	15.3079	16.9279	18.9392	22.6572	27.3363
29	13.1211	14.2565	16.0471	17.7083	19.7677	23.5666	28.3362
30	13.7867	14.9535	16.7908	18.4926	20.5992	24.4776	29.3360
40	20.7065	22.1643	24.4331	26.5093	29.0505	33.6603	39.3354
50	27.9907	29.7067	32.3574	34.7642	37.6886	42.9421	49.3349
60	35.5346	37.4848	40.4817	43.1879	46.4589	52.2938	59.3347
70	43.2752	45.4418	48.7576	51.7393	55.3290	61.6983	69.3344
80	51.1720	53.5400	57.1532	60.3915	64.2778	71.1445	79.3343
90	59.1963	61.7541	65.6466	69.1260	73.2912	80.6247	89.3342
100	67.3276	70.0648	74.2219	77.9295	82.3581	90.1332	99.3341
x	-2.5758	-2.3263	-1.9600	-1.6449	-1.2816	-0.6745	0.0000

$v\backslash Q$	0.25	0.1	0.05	0.025	0.01	0.005	0.001	0.0005	0.0001
1	1.32330	2.70544	3.84146	5.02389	6.63490	7.87944	10.828	12.116	15.137
2	2.77259	4.60517	5.99147	7.37776	9.21034	10.5966	13.816	15.202	18.421
3	4.10835	6.25139	7.81473	9.34840	11.3449	12.8381	16.266	16.730	21.108
4	5.38527	7.77944	9.48773	11.1433	13.2767	14.8602	178.467	19.997	23.513
5	6.62568	9.23635	11.0705	12.8325	15.0863	16.7496	20.515	22.105	25.745
6	7.84080	10.6446	12.5916	14.4494	16.8119	18.5476	22.458	24.103	27.856
7	9.03715	12.0170	14.0671	16.0128	18.4753	20.2777	24.322	26.018	29.877
8	10.2188	13.3616	15.5073	17.5348	20.0902	21.9550	26.125	27.868	31.828
9	11.3887	14.6837	16.9190	19.0228	21.6660	23.5893	27.8717	29.666	33.720
10	12.5489	15.9871	18.3070	20.4831	23.2093	25.1882	29.588	31.420	35.564
11	13.7007	17.2750	19.6751	21.9200	24.7250	26.7569	31.264	33.137	37.367
12	14.8454	18.5494	21.0261	23.3367	26.2170	28.2995	32.909	34.821	39.134
13	15.9839	19.8119	22.3621	24.7356	27.6883	29.8194	34.528	36.478	40.871
14	17.1170	21.0642	23.6848	26.1190	29.1413	31.3193	36.123	38.109	42.579

续表

$v\backslash Q$	0.25	0.1	0.05	0.025	0.01	0.005	0.001	0.0005	0.0001
15	18.2451	22.3072	24.9958	27.4884	30.5779	32.8013	37.697	39.719	44.263
16	19.3688	23.5418	26.2962	28.8454	31.9999	34.2672	39.252	41.308	45.925
17	20.4887	24.7690	27.5871	30.1910	33.4087	35.7185	40.790	42.879	47.566
18	21.6049	23.9894	28.8693	31.5264	34.8053	37.1564	42.312	44.434	49.189
19	22.7178	27.2036	30.1435	32.8523	36.1908	38.5822	43.820	45.973	50.796
20	23.8277	28.4120	31.4104	34.1696	37.5662	39.9968	45.315	47.498	52.386
21	24.9348	29.6151	32.6705	35.4789	38.9321	41.4010	46.797	49.011	53.962
22	26.0393	30.8133	33.9244	36.7807	40.2894	42.7956	48.268	50.511	55.525
23	27.1413	32.0069	35.1725	38.1757	41.6382	44.1813	49.728	52.000	57.075
24	28.2412	33.1963	36.4151	39.3641	42.9798	45.5585	51.179	53.479	58.613
25	29.3389	34.3816	37.6525	40.6465	44.3141	46.9278	52.620	54.947	60.140
26	30.4345	35.5631	38.8852	41.9232	45.6417	48.2899	54.052	56.407	64.657
27	31.5284	36.7412	40.1133	43.1944	46.9630	49.6449	55.476	57.858	63.164
28	32.6205	37.9159	41.3372	44.4607	48.2872	50.9933	56.892	59.300	64.662
29	33.7109	39.0875	42.5569	45.7222	49.5879	52.3356	58.302	60.735	66.152
30	34.7998	40.2560	43.7729	46.9792	50.8922	53.6720	59.703	62.162	67.633
40	45.6160	51.8050	55.7585	59.3417	63.6907	66.7659	73.402	76.095	82.062
50	56.3336	63.1671	67.5048	71.4202	76.1539	79.4900	86.661	89.560	95.969
60	66.9814	74.3970	79.0819	83.2976	88.3794	91.9517	99.607	102.695	109.503
70	77.5766	85.5271	90.5312	95.0231	100.425	104.215	112.317	115.578	122.755
80	88.1303	96.5782	101.879	106.629	112.329	116.321	124.839	128.261	135.783
90	98.6499	107.565	113.145	118.136	124.116	128.299	137.208	140.782	148.627
100	109.141	118.498	124.342	129.561	135.807	140.169	149.449	153.167	161.319
x	0.6745	1.2816	1.6449	1.9600	2.3263	2.5758	3.0902	3.2905	3.7190

7.6.2 随机性的检验方法

1. 频数检验

设二进制序列的长度为 n，出现 0 和 1 的概率各为 1/2，设测试结果有 n_0 个 0，n_1 个 1，$n_0+n_1=n$。

$$\chi^2=\frac{2}{n_0+n_1}\left\{\left(n_0-\frac{n_0+n_1}{2}\right)^2+\left(n_1-\frac{n_0+n_1}{2}\right)^2\right\}$$

$$=\frac{2}{n_0+n_1}\left\{\left(\frac{(n_0-n_1)^2}{4}\right)+\frac{(n_1-n_0)^2}{4}\right\}$$

$$=\frac{1}{n}(n_0-n_1)^2=\frac{1}{n}(n-2n_1)^2$$

查 χ^2 表 $v=1$ 的行。

2. 连缀检验

目的在于考察转移概率是否合理。

设 n_{ij} 为出现 ij 状态的数目，$i,j=0,1$。期望值应为

$$n_{00}=n_{01}=n_{10}=n_{11}=\frac{n-1}{4}$$

$$\chi^2=\left\{\left(n_{00}-\frac{n-1}{4}\right)^2+\left(n_{01}-\frac{n-1}{4}\right)^2+\left(n_{10}-\frac{n-1}{4}\right)^2+\left(n_{11}-\frac{n-1}{4}\right)^2\right\}\Big/\left(\frac{n-1}{4}\right)$$

$$=\frac{4}{n-1}\left\{\sum_{i=0}^{1}\sum_{j=0}^{1}n_{ij}^2-\frac{1}{2}(n-1)\sum_{i=0}^{1}\sum_{j=0}^{1}n_{ij}+\frac{1}{16}(n-1)^2\right\}$$

$$=\frac{4}{n-1}\sum_{i=0}^{1}\sum_{j=0}^{1}n_{ij}^2-2\sum_{i=0}^{1}\sum_{j=0}^{1}n_{ij}+\frac{1}{4}(n-1)$$

查 χ^2 表，$v=2$，$p=0.05$ 时 χ^2 值为 5.99147，只要

$$\chi^2\leqslant 5.99147$$

就算通过连缀的跟随试验。

3. 扑克试验

每次取出 m 个比特（m 为比 n 小得多的正整数），将长为 n 的序列分成 $\frac{n}{m}$ 组。

长为 m 比特的码共有 2^m 种状态。

用 n_i 表示第 i 种状态出现的次数，每种状态出现的期望值为 $\frac{n}{(m2^m)}=\left(\frac{n}{m}\right)/2^m$。

$$\chi^2=\frac{m2^m}{n}\left\{\sum_{i=0}^{2^m-1}\left(n_i-\frac{n}{m2^m}\right)^2\right\}$$

$$=\frac{m2^m}{n}\left\{\sum_{i=0}^{2^m-1}n_i^2-\frac{m}{m2^{m-1}}\sum_{i=0}^{2^m-1}n_i+\frac{n^2}{m^2 2^{2m}}2^m\right\}$$

$$=\frac{m2^m}{n}\sum_{i=0}^{2^m-1}n_i^2-2\sum_{i=0}^{2^m-1}n_i+\frac{n}{m2^m}2^m$$

$$=\frac{m2^m}{n}\sum_{i=0}^{2^m-1}n_i^2-F$$

因为

$$\sum_{i=0}^{2^m-1}n_i=F=\frac{n}{m}$$

查 χ^2 表，$p=0.05$，自由度 $v=2^m-1$，得到的值与 χ^2 作比较，以 $m=4$，$2^4-1=15$，$p=0.05$ 的 χ^2 值为 24.9958，故 $\chi^2\leqslant 24.9958$ 就算通过扑克检验。

4. 自相关检验

自相关检验是要考察序列的可预测性。首先要计算序列 $\{a_i\}$ 的自相关函数

$$A(d)=\sum_{i=1}^{n}(a_i\oplus a_{i+d})$$

$$a_i\in\{0,1\},\quad i=1,2,\cdots,n,\quad 1\leqslant d<n$$

将序列头尾相接成环，即将序列于转移位 d 比特后作模 2 加，好的序列出现 1 的次数的期望值应为 $\frac{n}{2}$，对每一个 d 值，对 $A(d)$ 作 0—1 的频数试验，办法见 1。

5. 游程检验

用 n_{0m} 和 n_{1m} 分别表示序列 $\{a_i\}_{i=1}^{n}$ 中长度为 m 的 0 游程和 1 游程的数目，它们的期望值为

$$\frac{n}{2^{m+1}}, \quad m = 1,2,\cdots$$

$$n_0 = n_1 = \frac{n}{2}$$

n_0 和 n_1 分别表示 0 游程和 1 的游程的函数。

$$\chi^2 = \sum_{i=0}^{1}\left(n_{im} - \frac{n}{2^{m+1}}\right)^2 \Big/ \frac{n}{2^{m+1}}$$

对于自由度 $v=1$，$p=0.05$ 对应的 χ^2 值为 3.84146，所以计算出的 $\chi^2 \leqslant 3.84146$，就算通过游程检验。

7.7 Fermat 因数分解法

因数分解 $n=pq$ 是有效地对 RSA 公钥密码进行攻击的办法，因此对大整数因子分解这样一个古老问题又掀起了研究的浪潮，特别是借助于计算机网进行分布式计算取得了新的进展，记录一次次地被刷新。1988 年用 400 台计算机联网运行 326 天，对 100 位十进制数分解成功；1990 年又将 155 位的第 9 个 Fermat 数 $F_9=2^{2^9}+1$ 分解成功；最新的记录已提高到 129 位十进制数。下面再介绍若干分解算法。

先引进若干概念。n 是正奇整数，若 $n=ab$，则 n 可写成 u^2-v^2，假定 $a>b$，则

$$u = \frac{1}{2}(a+b), \quad v = \frac{1}{2}(a-b)$$

所以 u 和 v 为非负整数。

反过来 n 是正奇数，n 可表示为 $n=u^2-v^2=(u+v)(u-v)$，故

$$a=u+v, b=u-v, n=ab$$

这里可提供一种因数分解办法。

因若 $n=ab$，且 a,b 比较接近，则 $\frac{1}{2}(a-b)$ 将是一个很小的数，而 $\frac{1}{2}(a+b)$ 比 $\sqrt{n}$ 稍大，这样可从 $\lfloor\sqrt{n}\rfloor$ 开始，$\lfloor\sqrt{n}\rfloor+1$，$\lfloor\sqrt{n}\rfloor+2$，…，直到找到 $(\lfloor\sqrt{n}\rfloor+t)^2-n=s^2$ 为止。这样 n 便可以表示成 u^2-v^2 形式，从而可分解为 $(u+v)(u-v)$。

例 7.1 求 200819 的因数分解。

$$\lfloor\sqrt{200819}\rfloor = 448$$

$$(449)^2 - 200819 = 201601 - 200819 = 782, \ \sqrt{782} \text{ 非整数}$$

$$(450)^2 - 200819 = 202500 - 200819 = 1681 = (41)^2$$

故

$$200819 = (450)^2 - (41)^2 = (450+41)(450-41) = 491 \times 409$$

还可以将上面步骤拓展为求 t,s 使 $(\lfloor\sqrt{kn}\rfloor+\tau)^2-kn=s^2$。令 $\lfloor\sqrt{kn}\rfloor+\tau=t$，则 $t^2-s^2=kn$，即 $(t+s)(t-s)=kn$。问题引出求 $\gcd\{t\pm s,n\}$。

例 7.2 求 141467 的因数分解。

$\lfloor\sqrt{141467}\rfloor=376$，若求$(376+t)^2-141467$，$t=1,2,\cdots$，将发现很不容易找到差$(376+t)^2-141467$正好是某一整数的平方。$3\times141467=424401$，若从$\lfloor\sqrt{3\times141467}\rfloor+1=651+1$开始。

$$(652)^2-424401=425104-424401=703,\quad \sqrt{703}\approx26.5$$

$$(653)^2-424401=426409-424401=2008,\quad \sqrt{2008}\approx44.8$$

$$(654)^2-424401=427716-424401=3315,\quad \sqrt{3315}\approx57.5$$

$$(655)^2-424401=429025-424401=4624=(68)^2$$

$$\gcd\{68+655,141467\}=\gcd\{723,141467\}=241$$

故

$$141467=241\times587$$

进一步将上述办法拓广为：若存在$u>v$，且

$$u^2\equiv v^2 \bmod n$$

则因数分解n，可从求

$$\gcd\{u+v,n\}\text{ 或 }\gcd\{u-v,n\}$$

得n的因数。

例 7.3 $118^2\equiv25 \bmod 4633$，即

$$118^2\equiv5^2 \bmod 4633$$

$$\gcd\{118+5,4633\}=\gcd\{123,4633\}=41$$

$$\gcd\{118-5,4633\}=\gcd\{113,4633\}=113$$

$$4633=41\times113$$

定义 7.2 $B=\{p_1,p_2,\cdots,p_k\}$称为因数基，其中$\{p_1,p_2,\cdots,p_k\}$是一组不同的素数，p_1可以是-1，若$b^2 \bmod n$可以表达为B的数之积，则称b关于数n为B数。

例 7.4 $n=4633$，$B=\{-1,2,3\}$ $67^2=4489$

故

$$67^2\equiv-144 \bmod 4633$$

$$-144=-(4\times3)^2=-2^4\times3^2$$

故 67 关于 4633 是B数。又

$$68^2=4624,\quad 68^2\equiv-9 \bmod 4633$$

故 68 关于 4633 也是B数。

$$69^2=4761,\quad 69^2\equiv128 \bmod 4633\quad 128=2^7$$

故 69 关于 4633 也是B数。

以例 7.4 为例。下面引进向量$\vec{\beta}=(\alpha_1,\alpha_2,\alpha_3)$，对于每一个$B$数$b$

$$b^2 \bmod n$$

因子分解后若含有-1的因数，则$\alpha_1=1$，否则为 0。若含有 2 的因数，方次为偶数，则$\alpha_2=0$，否则$\alpha_2=1$。同理，若$b^2 \bmod n$因数中含有 3 的因数的次方为偶数时，则$\alpha_3=0$，否则$\alpha_3=1$。比如

$67^2\equiv-144 \bmod 4633$，$-144=-2^4\times3^2$，故对应一向量$\vec{\beta}_1=(1,0,0)$。

$68^2\equiv-3^2 \bmod 4633$，故对应一向量$\vec{\beta}_2=(1,0,0)$。

$69^2\equiv2^7 \bmod 4633$，故对应一向量$\vec{\beta}_3=(0,1,0)$。

此概念可推广到一般。它用以表达因数基各因数的次方的奇偶性。

给定了 n 及因数基 $B=\{p_1,p_2,\cdots,p_k\}$，b 关于 n 是 B 数，对应一向量

$$\vec{\beta}=(\alpha_1,\alpha_2,\cdots,\alpha_k),\quad \alpha_i=0\text{ 或 }1,\quad i=1,2,\cdots,k$$

$\alpha_i=0$ 说明 $b^2 \bmod n$ 因数中含有 p_i 的次方为偶数(包括 0)；若 $\alpha_i=1$，则含 p_i 的因数次方为奇数。若 $\vec{\beta}$ 是 0 向量，即 $\alpha_1-\alpha_2-\cdots-\alpha_k=0$，则若存在 $b_1,b_2,\cdots,b_h$，使 $i=1,2,\cdots,h$

$$b_i^2 \bmod n \equiv a_i = p_1^{\alpha_{i1}} p_2^{\alpha_{i2}}\cdots p_k^{\alpha_{ik}}$$

所有的 $\alpha_{i1},\alpha_{i2},\cdots,\alpha_{ik}$ 都是整数，

$$a_1a_2\cdots a_h = p_1^{\sum_{i=1}^{h}\alpha_{i1}} p_2^{\sum_{i=1}^{h}\alpha_{i2}}\cdots p_k^{\sum_{i=1}^{h}\alpha_{ik}}$$

若所有的指数 $\sum_{i=1}^{h}\alpha_{ij}$，$j=1,2,\cdots,k$ 都是偶数，则右端是完全平方。

令
$$r_j=\frac{1}{2}\sum_{i=1}^{h}\alpha_{ij},\quad j=1,2,\cdots,k$$

则
$$\prod_{i=1}^{h}a_i=\left(\prod_{j=1}^{k}p_j^{r_j}\right)^2$$

令
$$b=\prod_{i=1}^{h}b_i \bmod n$$

$$c=\prod_{j=1}^{k}p_j^{r_j-1} \bmod n$$

有
$$b^2\equiv c^2 \bmod n$$

还是以 $n=4633$，$B=(-1,2,3)$ 为例。

$67^2\equiv -144 \bmod n$ 对应于 $\vec{\beta}=(1,0,0)$，$b_1=67$。

$68^2\equiv -9 \bmod n$ 对应于 $\vec{\beta}=(1,0,0)$，$b_2=68$。

$$(-144)(-9)=(-1)^2(2)^4(3)^4$$

$$r_1=1,r_2=2,r_3=2,b_1b_2=67\times 68=4556$$

所以
$$b\equiv 4556 \bmod 4633$$

$$b\equiv -77 \bmod 4633$$

$$c=(2)^2(3)^4 \bmod 4633\equiv 36$$

满足
$$b^2\equiv c^2 \bmod 4633$$

$$\gcd\{-77+36,4633\}=41,\quad 4633=41\times 113$$

例 7.5 $n=1829$，取 b_i 为 $\lfloor\sqrt{1829k}\rfloor$，$\lfloor\sqrt{1829k}\rfloor+1$，$k=1,2,3,4$。$\lfloor\sqrt{1829}\rfloor=42$，$\lfloor\sqrt{1829\times 2}\rfloor=60$，$\lfloor\sqrt{1829\times 3}\rfloor=74$，$\lfloor\sqrt{1829\times 4}\rfloor=85$，即取 $b_1=42,b_2=43,b_3=60,b_4=61,b_5=74,b_6=75,b_7=85,b_8=86$

$(42)^2\equiv 1764 \bmod n\equiv -65$，　　$-65=-5\times 13$

$(43)^2\equiv 1849 \bmod n\equiv 20$，　　$20=4\times 5=2^2\times 5$

$(60)^2\equiv 3600 \bmod n\equiv -58$，　　$-58=-2\times 29$

$(61)^2\equiv 63 \bmod n$，　　$63=3^2\times 7$

$(74)^2\equiv -11 \bmod n$，

$(75)^2 \equiv 138 \bmod n$，　　$138 = 2 \times 3 \times 23$

$(85)^2 \equiv -91 \bmod n$，　　$-91 = -7 \times 13$

$(86)^2 \equiv 80 \bmod n$，　　$80 = 2^4 \times 5$

令　　$B = \{-1, 2, 3, 5, 7, 11, 13, 23, 29\}$

选择表 7.1 中的若干行，使得对应的列的和都是偶数，取之如下：

$$b_1 = 42, b_2 = 43, b_3 = 61, b_4 = 85$$

$$(42 \times 43 \times 61 \times 85)^2 = 2^2 \cdot 3^2 \cdot 5^2 \cdot 7^2 \cdot 13^2 \bmod n$$

$$42 \times 43 \times 61 \times 85 = 9564110 \equiv 1459,$$

$$2 \times 3 \times 5 \times 7 \times 13 = 2730 \equiv 901 \bmod 1829$$

即　　$(956411)^2 \equiv (2730)^2 \bmod n$

或　　$(1459)^2 \equiv (901)^2 \bmod n$

$$\gcd\{1459 + 901, 1829\} = \gcd\{2360, 1829\} = 59$$

$$1829 = 59 \times 31$$

表 7.1

b_i \ p_j	−1	2	3	5	7	11	13	23	9
42	1			1			1		
43		2		1					
60	1								1
61			2		1				
74	1					1			
75		1	1					1	
85					1		1		
86		4		1					

7.8　连分式因数分解法

7.8.1　实数的连分式表示

定义 7.3

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \ddots + \cfrac{1}{a_m}}}$$

用$[a_0, a_1, a_2, \cdots, a_m]$表示它，称为连分式。其中 $a_0 \in Z, a_i \in N, 1 \leqslant i \leqslant m$。

若 $m \to \infty$，表以$[a_0, a_1, a_2, \cdots]$。

显然任一有限连分数是一有理数。

$$a_{m-1}+\frac{1}{a_m}=\frac{a_{m-1}a_m+1}{a_m}$$

$$\frac{1}{a_{m-1}+\frac{1}{a_m}}=\frac{a_m}{a_{m-1}a_m+1}$$

引理 7.1 任一有理数都可以用有限的连分数来表示。

证明 证明的过程也就是求连分式的步骤，即证明是构造性的，

设$\frac{a}{b}$是有理数，令

$$s_0=a,s_1=b,s_i=q_is_{i-1}+s_{i-2},\quad 0\leqslant s_{i-2}<s_{i-1},\quad i\geqslant 0$$

直到 $s_{m+2}=0$ 为止，这时 m 为奇整数，

$$s_m=q_ms_{m+1}$$

所以

$$\frac{a}{b}=\frac{s_0}{s_1}=\frac{q_0s_1+s_2}{s_1}=q_0+\frac{s_2}{s_1}$$

$$=q_0+\frac{s_2}{q_1s_2+s_3}=q_0+\frac{1}{\frac{q_1s_2+s_3}{s_2}}$$

$$=q_0+\frac{1}{q_1+\frac{s_3}{s_2}}=q_0+\frac{1}{q_1+\frac{1}{\frac{s_2}{s_3}}}$$

$$=q_0+\frac{1}{q_1+\frac{1}{\frac{q_2s_3+s_4}{s_3}}}=q_0+\frac{1}{q_1+\frac{1}{q_2+\frac{s_4}{s_3}}}$$

$$=\cdots=q_0+\cfrac{1}{q_1+\cfrac{1}{q_2+\ddots\ q_{m-1}+\frac{s_{m+1}}{s_m}}}$$

$$=q_0+\cfrac{1}{q_1+\cfrac{1}{\ddots\ q_{m-1}+\frac{1}{q_m}}}$$

故$\frac{a}{b}$可表以连分数$[q_0,q_1,\cdots,q_m]$，其中 $q_i>0,i\geqslant 1$。

算法：已知 α 为有理数，

S1. $r_0\leftarrow\alpha$，　$i\leftarrow 0$；

S2. $s_i=\lfloor r_i\rfloor$，

$$r_{i-1}=\frac{1}{r_{i-1}-s_i},\quad i\leftarrow i+1$$

S3. 若 $r_{i-1}=s_i$，则转 S2，否则转 S4；

S4. 输出$[s_0,s_1,\cdots]$。

例如，
$$\alpha=\frac{11}{9},\ r_0=\frac{11}{9},\ s_0=\left\lfloor\frac{11}{9}\right\rfloor=1$$

$$r_1=\frac{1}{\frac{11}{9}-1}=\frac{9}{2},\ s_1=\left\lfloor\frac{9}{2}\right\rfloor=4$$

$$r_2=\frac{1}{\frac{9}{2}-4}=2,\ s_2=0$$

所以
$$\frac{11}{9}=1+\frac{1}{4+\frac{1}{2}}$$

例如，
$$\alpha=\frac{294}{159}$$

$$r_0=\frac{294}{159},\qquad s_0=\left\lfloor\frac{294}{159}\right\rfloor=1$$

$$r_1=\frac{1}{\frac{294}{159}-1}=\frac{1}{\frac{135}{159}}=\frac{159}{135}$$

$$s_1=\left\lfloor\frac{159}{135}\right\rfloor=1,\quad r_1=\frac{1}{\frac{159}{135}-1}=\frac{1}{\frac{24}{135}}=\frac{135}{24}$$

$$s_2=\left\lfloor\frac{135}{24}\right\rfloor=5,\quad r_2=\frac{1}{\frac{135}{24}-5}=\frac{24}{15}$$

$$s_3=\left\lfloor\frac{24}{15}\right\rfloor=1,\quad r_3=\frac{1}{\frac{25}{14}-1}=\frac{15}{9}$$

$$s_4=\left\lfloor\frac{15}{9}\right\rfloor=1,\quad r_4=\frac{1}{\frac{15}{9}-1}=\frac{9}{6}$$

$$s_5=\left\lfloor\frac{9}{6}\right\rfloor=1,\quad r_5=\frac{1}{\frac{9}{6}-1}=\frac{6}{3}=2$$

$$s_6=2$$

所以
$$\frac{294}{159}=1+\cfrac{1}{1+\cfrac{1}{5+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\frac{1}{2}}}}}}$$

例如，　$\pi=3.14159$，

$r_0=3.14159$，　　$s_0=\lfloor 3.14159\rfloor=3$.

$r_1=\dfrac{1}{0.14159}$，　　$s_1=\left\lfloor\dfrac{1}{0.14159}\right\rfloor=7$.

$r_2=\dfrac{1}{\dfrac{100000}{14159}-7}=\dfrac{14159}{887}$，　　$s_2=\left\lfloor\dfrac{14159}{887}\right\rfloor=15$.

$r_3=\dfrac{1}{\dfrac{14159}{887}-15}=\dfrac{887}{854}$，　　$s_3=\left\lfloor\dfrac{887}{854}\right\rfloor=1$,

$r_4=\dfrac{1}{\dfrac{887}{854}-1}=\dfrac{854}{33}$，　　$s_4=\left\lfloor\dfrac{854}{33}\right\rfloor=25$,

$r_5=\dfrac{1}{\dfrac{854}{33}-25}=\dfrac{33}{29}$，　　$s_5=\left\lfloor\dfrac{33}{29}\right\rfloor=1$,

$r_6=\dfrac{1}{\dfrac{33}{29}-1}=\dfrac{29}{4}$，　　$s_6=\left\lfloor\dfrac{29}{4}\right\rfloor=7$.

$r_7=\dfrac{1}{\dfrac{29}{4}-7}=4$

所以

$$\pi=3+\cfrac{1}{7+\cfrac{1}{15+\cfrac{1}{1+\cfrac{1}{25+\cfrac{1}{1+\cfrac{1}{7+\cfrac{1}{4}}}}}}}$$

已知连分式$[a_0,a_1,\cdots,a_m]$，$0\leqslant k\leqslant m$，则$[a_0,a_1,\cdots,a_k]$和c_k表示它。

例如，　$\pi=[3,7,15,1,25,1,7,4]$

$c_1=[3,7]=3+\dfrac{1}{7}=\dfrac{22}{7}$

$c_2=[3,7,15]=3+\cfrac{1}{7+\cfrac{1}{15}}=3+\dfrac{15}{106}=\dfrac{333}{106}$

例如，　$\sqrt{3}=a_0+\cfrac{1}{a_1+\cfrac{1}{a_2+\ddots+\cfrac{1}{a_k}}}$

$a_0=\lfloor\sqrt{3}\rfloor=1$，　$x_1=\dfrac{1}{\sqrt{3}-1}=\dfrac{\sqrt{3}+1}{2}=1+\dfrac{1}{2}(\sqrt{3}-1)$

$a_1=\lfloor x_1\rfloor=1$，　$x_2=\dfrac{2}{\sqrt{3}-1}=\sqrt{3}+1$

$$a_2=\lfloor x_2\rfloor=2,\quad x_3=\frac{1}{\sqrt{3}-1}=\frac{1}{2}(\sqrt{3}+1),\cdots$$

$$\frac{b_0}{c_0}=1,\quad \frac{b_1}{c_1}=1+1=2,\quad \frac{b_2}{c_2}=1+\frac{1}{1+\frac{1}{2}}=\frac{5}{3}$$

$$\frac{b_3}{c_3}=1+\frac{1}{1+\frac{1}{2+1}}=\frac{7}{4},\quad \frac{b_4}{c_4}=1+\frac{1}{1+\frac{1}{2+\frac{1}{1+\frac{1}{2}}}}=\frac{9}{11},\cdots$$

定理 7.4 设$\{a_i\}i\geqslant 0$是有限或无限序列，除a_0外其余$a_i>0$，令

$$r_k=[a_0,a_1,\cdots,a_k]=\frac{b_k}{c_k}$$

则下面递推关系成立

$$b_0=a_0,\qquad b_1=a_0a_1+1$$
$$c_0=1,\qquad c_1=a_1$$
$$b_k=a_kb_{k-1}+b_{k-2},\qquad k\geqslant 2$$
$$c_k=a_kc_{k-1}+c_{k-2},\qquad k\geqslant 2$$

证明 证明是采用数学归纳法。

$$k=0,\qquad r_0=\frac{b_0}{c_0}=a_0$$

所以 $$b_0=a_0,\quad c_0=1$$

$$k=1,\quad r_1=[a_0,a_1]=a_0+\frac{1}{a_1}=\frac{a_0a_1+1}{a}=\frac{b_1}{c_1}$$

所以 $$b_1=a_0a_1+1,\quad c_1=a_1$$

设 $r_k=[a_0,a_1,\cdots,a_k]=\frac{b_k}{c_k}=\frac{a_kb_{k-1}+b_{k-2}}{a_kc_{k-1}+c_{k-2}}$为真，则

$$r_{k+1}=[a_0,a_1,\cdots,a_k,a_{k+1}]=\left[a_0,a_1,\cdots,a_k+\frac{1}{a_{k+1}}\right]$$

$$=\frac{\left(a_k+\frac{1}{a_{k+1}}\right)b_{k-1}+b_{k-2}}{\left(a_k+\frac{1}{a_{k+1}}\right)c_{k-1}+c_{k-2}}=\frac{a_{k+1}(a_kb_{k-1}+b_{k-2})+b_{k-1}}{a_{k+1}(a_kc_{k-1}+c_{k-2})+c_{k-1}}$$

$$=\frac{a_{k+1}b_k+b_{k-1}}{a_{k+1}c_k+c_{k-1}}=\frac{b_{k+1}}{c_{k-1}}$$

引理 7.2 设 $r_k=[a_0,a_1,\cdots,a_k]=\frac{b_k}{c_k}$，则

$$b_kc_{k-1}-b_{k-1}c_k=(-1)^{k-1}$$

证明 $k=1$，$b_1c_0-b_0c_1=(a_0a_1+1)1-a_0a_1=1$ 为真。

设 $b_kc_{k-1}-b_{k-1}c_k=(-1)^k$ 正确。则

$$b_{k+1}c_k-b_kc_{k+1}=(a_{k+1}b_k+b_{k-1})b_k-b_k(a_{k+1}c_k+c_{k-1})$$
$$=b_{k-1}c_k-b_kc_{k-1}=(-1)(-1)^{k-1}=(-1)^k$$

推论 7.1 $r_k=\dfrac{b_k}{c_k}$，则 $\gcd\{b_k,c_k\}=1$。

证明 由 $b_{k-1}c_k-b_kc_{k-1}=(-1)^{k-1}$，$\gcd\{b_k,c_k\}$除尽$(-1)^{k-1}$。

定理 7.5 $r_k=\dfrac{b_k}{c_k}$，则

(1) $r_k-r_{k-1}=\dfrac{(-1)^{k-1}}{c_{k-1}c_k}$，　$k\geqslant 1$；

(2) $r_k-r_{k-2}=\dfrac{(-1)^k a_k}{c_{k-1}c_k}$，　$k\geqslant 2$；

(3) $r_0<r_2<r_4<\cdots<r_5<r_3<r_1$。

证明
$$r_k-r_{k-1}=\frac{b_k}{c_k}-\frac{b_{k-1}}{c_{k-1}}=\frac{b_kc_{k-1}-b_{k-1}c_k}{c_{k-1}c_k}=\frac{(-1)^{k-1}}{c_{k-1}c_k}$$

$$c_k-c_{k-2}=\frac{b_k}{c_k}-\frac{b_{k-2}}{c_{k-2}}=\frac{b_kc_{k-2}-b_{k-2}c_k}{c_kc_{k-2}}$$

$$=\frac{(a_kb_{k-1}+b_{k-2})c_{k-2}-b_{k-2}(a_kc_{k-1}+c_{k-2})}{c_{k-2}c_k}$$

$$=\frac{a_kb_{k-1}c_{k-2}-a_kb_{k-2}c_{k-1}}{c_{k-2}c_k}=\frac{a_k(-1)^k}{c_{k-2}c_k}$$

这就证明

$$c_{2k}-c_{2k-2}>0,\quad c_{2k+1}-c_{2k-1}<0$$

即

$$c_{2k}<c_{2k-2}<c_{2k+3}<c_{2k-1}$$

定理 7.6 $x>1$ 是一实数，b_i/c_i 是 x 的第 i 级连分数 l，则

$$|b_i^2-x^2c_i^2|<2x$$

证明 根据推论，即 b_{i-1}/c_{i-1} 和 b_i/c_i 位于 x 的两侧，距离不大于$\dfrac{1}{c_ic_{i-1}}$，故有

$$|b_i^2-x^2c_i^2|=c_i^2\left|x-\frac{b_i}{c_i}\right|\left|x+\frac{b_i}{c_i}\right|<c_i\frac{1}{c_ic_{i+1}}\left(x+\left(x+\frac{1}{c_ic_{i+1}}\right)\right)$$

所以
$$|b_i^2-x^2c^2|-2x<2x\left(-1+\frac{c_i}{c_{i+1}}+\frac{1}{2xc_i^2+1}\right)$$

$$<2x\left(-1+\frac{c_i}{c_{i-1}}+\frac{1}{c_{i+1}}\right)$$

这是考虑到 $x>1$，$c_{i-1}\geqslant 1$。又因

$$\frac{c_i}{c_{i+1}}+\frac{1}{c_{i-1}}=\frac{1+c_i}{c_{i+1}}$$

$i>0$，$c_i>1$，而且 c_i 和 c_{i-1} 不相等，否则若 $c_i=c_{i-1}$，则

$$\frac{b_{i-1}}{c_{i-1}}-\frac{b_i}{c_{i+1}}=\frac{b_{i+1}-b_i}{c_{i-1}}=\frac{(-1)^i}{c_i^2+1}$$

则
$$b_{i+1}-b_i=\frac{(-1)^i}{c_{i+1}}$$

与 b_i 和 b_{i+1} 是正整数的假定矛盾。所以 $c_i\neq c_{i-1}$，而且 $c_{i-1}>c_i+1$，故

$$\frac{c_i}{c_{i-1}}+\frac{1}{c_{i-1}}<\frac{c_{i+1}}{c_{i-1}}=1$$

$$| b_i^2 - x^2 c_i^2 | - 2x < 2x\left(-1 + \frac{c_{i+1}}{c_{i+1}} \right) = 0$$

所以 $$| b_i^2 - x^2 c_i^2 | < 2x$$

定理 7.7 n 是一个非整数平方的正整数，且 b_i/c_i，$i=0,1,2,\cdots$，收敛于$\sqrt{n}$，则

$$b_i^2 \bmod n < 2\sqrt{n}$$

证明 以 $x=\sqrt{n}$代入$|b_i^2-x^2c_i^2|<2x$，又

$$b_i^2 - nc_i^2 \equiv b_i^2 \bmod n$$

故 $$b_i^2 \bmod n < 2\sqrt{n}$$

例 将$\sqrt{3}$化为连分数。

$$a_0 = \lfloor \sqrt{3} \rfloor = 1, \quad x_0 = \sqrt{3} - 1, \quad a_1 = \lfloor 1/a_0 \rfloor = 1$$

$$x_1 = \frac{1}{\sqrt{3}-1} - 1 = \frac{1-\sqrt{3}+1}{\sqrt{3}-1} = \frac{2-\sqrt{3}}{\sqrt{3}-1} = \frac{\sqrt{3}-1}{2}$$

7.8.2 连分式因数分解法

假定 n 是待因数分解整数。下面运算都在 mod n 下进行。

$$b_{-1} \leftarrow 1, b_0 \leftarrow a_0 = \lfloor \sqrt{n} \rfloor, \quad x_0 \leftarrow \sqrt{n} - a_0, \quad b_0^2 \bmod n$$

令 $i=1,2,3,\cdots$，进行以下运算，并观察最后 $b_i^2 \bmod n$ 各数。

$$a_i \leftarrow \lfloor 1/x_{i-1} \rfloor, \quad x_i \leftarrow \frac{1}{x_{i-1}} - a_i$$

$$b_i \leftarrow a_i b_{i-1} + b_{i-2} \bmod n$$

$$b_i^2 \bmod n$$

得一表将它们因数分解成小素数之积，构成因数基 B。对于每个 $b_i^2 \bmod n$，它含有属于 B 的因数表以对应向量 $\boldsymbol{\beta}$，举例说明如下。

例 7.6 求 9073 的因数分解。

$i=0$，

$$b_{-1} = 1, \quad b_0 = a_0 = \lfloor \sqrt{9073} \rfloor = 95, \quad x_0 = \lfloor \sqrt{9073} \rfloor - 95$$

$$b_0^2 \equiv 9025 \bmod n \equiv -48 \bmod n$$

$i=1$，

$$a_1 = \left\lfloor \frac{1}{\sqrt{9073}-95} \right\rfloor = \left\lfloor \frac{\sqrt{9073}+95}{48} \right\rfloor = 3$$

$$x_1 = \frac{1}{\sqrt{9073}-95} - 3 = \frac{\sqrt{9073}+95}{48} - 3 = \frac{\sqrt{9073}-49}{48}$$

$$b_1 = a_1 b_0 + b_{-1} = 3 \cdot 95 + 1 = 286$$

$$b_1^2 \equiv 81796 \equiv 139 \bmod n$$

$i=2$，

$$a_2 = \left\lfloor \frac{48}{\sqrt{9073}-49} \right\rfloor = \left\lfloor \frac{48(\sqrt{9073}+49)}{6672} \right\rfloor = 1$$

$$x_2 = \frac{48}{\sqrt{9073} - 49} - 1 = \frac{97 - \sqrt{9073}}{\sqrt{9073} - 49}$$

$$b_2 = a_2 b_1 + b_0 = 286 + 95 = 381$$

$$b_2^2 \equiv 145161 \equiv -7 \bmod n$$

$i = 3$,

$$a_3 = \left\lfloor \frac{\sqrt{9073} - 49}{97 - \sqrt{9073}} \right\rfloor = 26$$

$$x_3 = \frac{\sqrt{9073} - 49}{97 - \sqrt{9073}} - 26$$

$$b_3 = a_3 \cdot b_2 + b_1 = 26 \cdot 381 + 286 = 10192 = 1119 \bmod n$$

$$b_3^2 \bmod n \equiv 1252161 \bmod n \equiv 87$$

$i = 4$,

$$a_4 = 2, \quad b_4 = 2619, \quad b_4^2 \bmod n = -27$$

如表 7.2 所示，$i=0,4$，两行分别为

$$(1,4,1,0,0,0)$$

$$(1,0,3,0,0,0)$$

这两行分量之和均为偶数。

$$(b_0^2)(b_4^2) \equiv (-48)(-27) = 4^2 \cdot 9^2 \bmod 9073$$

$$b_0 b_4 = 95 \cdot 2619 = 248805 \equiv 3834 \bmod 9073$$

所以

$$(3834)^2 \equiv (36)^2 \bmod 9073$$

$$\gcd\{3834 + 36, 9073\} = \gcd\{3870, 9073\} = 43$$

$$9073 = 211 \cdot 43$$

表 7.2

p_i / $b_i^2 \bmod n$	-1	2	3	7	87	139
$-48 = -2^4(3)$	1	4	1			
$139 - 139$						1
$-7 = (-1)(7)$	1			1		
$87 - 87$					1	
$-27 - -3^3$	1		3			

例 7.7 分解 637753。

$$b_{-1} = 1, \ b_0 = \lfloor \sqrt{637753} \rfloor = 798, \quad a_0 = 798$$

$i = 0$,

$$b_0^2 \equiv 636804 \equiv 2^2 \times 3^2 \times 1789$$

$$x_0 = \sqrt{637753} - 798$$

$i = 1$,

$$a_1 = \left\lfloor \frac{1}{\sqrt{637753} - 798} \right\rfloor = 1, \quad x_1 = \frac{1}{\sqrt{637753} - 798} - 1$$

$$x_1 = \frac{799 - \sqrt{637753}}{\sqrt{637753} - 798}$$

$$b_1 = b_0 + b_{-1} = 798 + 1 = 799$$

$$b_1^2 = 638401 \equiv 648 \bmod n$$

$$648 = 2^3 \times 3^4$$

$i = 2$,

$$a_2 = \left\lfloor \frac{\sqrt{637753} - 798}{799 - \sqrt{637753}} \right\rfloor = 1$$

$$x_2 = \frac{\sqrt{637753} - 798}{799 - \sqrt{637753}} - 1 = \frac{2\sqrt{637753} - 1597}{799 - \sqrt{637753}}$$

$$b_2 = b_1 + b_0 = 799 + 798 = 1597$$

$$b_2^2 = 2550409 \equiv -603 \bmod n$$

$$-603 = (-1)(3)(201) = (-1)(3)^2(67)$$

$i = 3$,

$$a_3 = \left\lfloor \frac{799 - \sqrt{637753}}{2\sqrt{637753} - 1597} \right\rfloor = 2$$

$$x_3 = \frac{799 - \sqrt{637753}}{2\sqrt{637753} - 1597} - 2 = \frac{3993 - 3\sqrt{637753}}{2\sqrt{637753} - 1597}$$

$$b_3 = 2 \times 1597 + 799 = 3993$$

$$b_3^2 = 15944049 \equiv 224 \bmod n$$

$$244 = 2^5 \times 7$$

后面的计算不再详细给出，只叙述结果。因数基 $B=\{-1,2,3,7,13,59,67,73\}$相应的 B 表如表 7.3 所示。

表 7.3

$b_i^2 \bmod n$ \ p_i	−1	2	3	7	13	59	67	73
$-13 \cdot 73$	1				1			1
$2^4 \cdot 3^4$		1						
$-3^2 \cdot 67$	1						1	
$2^5 \cdot 7$		1		1				
$-3 \cdot 349$								
$3 \cdot 149$								
-907								
$2^3 \cdot 3^2 \cdot 7$		1		1				

从表 7.3 可知，$2^5 \cdot 7$ 行和 $2^3 \cdot 3^2 \cdot 7$ 行对应元素之和为偶数。$b_5 \equiv 3993 \bmod n$，$b_7 \equiv 114199 \bmod n$。

$$3993 \times 114199 \equiv 3212 \bmod n$$

$$(3212)^2 \equiv 2^8 \cdot 3^2 \cdot 7^2 \bmod n, \quad \text{即} (3212)^2 \equiv (2^4 \cdot 3 \cdot 7)^7 \bmod n$$

$$2^4 \cdot 3 \cdot 7 = 336$$

$$\gcd\{3212 + 336, 637753\} = \gcd\{3548, 637753\} = 887$$

$$\gcd\{3212 - 336, 637753\} = \gcd\{2876, 637753\} = 719$$

$$887 \times 719 = 637753$$

7.9 离散对数计算法

7.9.1 离散对数

若 p 是一大素数，a 是 $\{0,1,2,\cdots,p-1\}$ 中与 p 互素的数，或称 a 是域 $GF(p)$ 上的本原元素，计算幂函数 $f(x)=a^x \bmod p$ 并不困难。令

$$x = b_0 + b_1 2 + b_2 2^2 + \cdots + b_{n-1} 2^{n-1}$$

则

$$a^x = a^{b_0 + b_1 2 + b_2 2^2 + \cdots + b_{n-1} 2^{n-1}}$$

$$= (a)^{b_0} \times (a^2)^{b_1} \times (a^{2^2})^{b_2} \times \cdots \times (a^{2^{n-1}})^{b_{n-1}}$$

且

$$(a^{2^i})^{b_i} = \begin{cases} 1, & b_i = 0 \\ a^{2^i}, & b_i = 1 \end{cases}$$

所以先计算

$$a^2 = a \times a$$

$$a^{2^2} = a^2 \times a^2$$

$$\vdots$$

$$a^{2^{n-1}} = a^{2^{n-2}} \times a^{2^{n-2}}$$

再计算 a^x。

例 7.8 $p=1823$，$a=5$，求 5^{375}。

由于 375 用二进制数表示为 $(101110111)_2$，即

$$375 = 1 + 2 + 2^2 + 2^4 + 2^5 + 2^6 + 2^8$$

$$= 1 + 2 + 4 + 16 + 32 + 64 + 256$$

故

$$(5)^{375} = 5\ (5^2)\ (5^{2^2})\ (5^{2^4})\ (5^{2^5})\ (5^{2^6})\ (5^{2^8})$$

$$5^2 \bmod 1823 \equiv 25 \bmod 1823$$

$$5^4 \bmod 1823 \equiv 625 \bmod 1823$$

$$5^8 \bmod 1823 \equiv (625)^2 \bmod 1823 \equiv 390625 \bmod 1823 \equiv 503$$

$$5^{16} \bmod 1823 \equiv (503)^2 \bmod 1823 \equiv 1435$$

$$5^{32} \bmod 1823 \equiv 2059225 \bmod 1823 \equiv 1058$$

$$5^{64} \bmod 1823 \equiv 42$$

$$5^{128} \bmod 1823 \equiv 1764$$

$$5^{256} \bmod 1823 \equiv 1658$$

所以 $5^{377} \bmod 1823 \equiv 5 \times 25 \times 625 \times 1435 \times 1058 \times 42 \times 1658 \bmod 1823 \equiv 591$

在 $GF(p)$ 上已知 x 计算 $a^x = y$，反之，已知 y 求 x，则 x 用 $\log_a y$ 表示，也称之为求离散对数。已知 y 求 x，不如已知 x 求 y 那么容易了。Diffie-Hellman 便是利用了求离散对数的困难，建立了 Pohlig-Hellman 密码体制。该体制可用于网络上保密通信，确定一大数 p 及 $GF(p)$ 的本元元素 a。设一用户 A 选择一随机整数 x_A，计算

$$y_A \equiv a^{x_A} \bmod p$$

将 y_A 公开，x_A 保密，只有 A 自己才掌握。用户 B 欲与 A 进行保密通信，查得 y_A，计算他们间的通信密钥

$$k_{AB} \equiv (y_A)^{x_B} \bmod p \equiv a^{x_A x_B} \bmod p$$

A 收到密文后，可以先计算出通信密钥

$$k_{AB} \equiv a^{x_B x_A} \bmod p \equiv (y_B)^{x_A} \bmod p$$

即 A 可查得公开的密钥 y_B，计算出 k_{AB}，然后利用它来解密。公开 y_A 或 y_B 要求 x_A 或 x_B 等价于求离散对数

$$\log_a(y_A) \quad 或 \quad \log_a(y_B)$$

7.9.2 Pohlig-Hellman 离散对数求法

已知 x 和 y 都是正整数，且 x 是 $GF(p)$ 域上的本原元素。求正整数 r，满足

$$y \equiv x^r (\bmod n)$$

假定整数 n 为素数 p，比 $p-1$ 只有小素数因子。

设

$$p - 1 = p_1^{n_1} p_2^{n_2} \cdots p_k^{n_k}, \quad n_i > 0, \ 1 \leqslant i \leqslant k$$

根据中国剩余定理，只要能确定

$$r_i \equiv r (\bmod p_i^{n_i}), \quad 1 \leqslant i \leqslant k$$

则 r 便能求得。下面讨论求 r_i 的方法。以求 r_1 为例，求 $r_2, r_3, \cdots, r_k$ 类似。

若

$$M_i = (p-1)/p_i^{n_i}, \quad i = 1, 2, \cdots, k$$

$$M_i y_i \equiv 1 (\bmod p_i^{n_i}), \quad i = 1, 2, \cdots, k$$

则

$$r = r_1 M_1 y_1 + r_2 M_2 y_2 + \cdots + r_k M_k y_k$$

由于

$$y \equiv x^r \bmod p$$

故

$$y^{(p-1)/p_1} \equiv (x^r)^{(p-1)/p_1} (\bmod p)$$

$$\equiv (x^{r_1 M_1 y_1 + r_2 M_2 y_2 + \cdots + r_k M_k y_k})^{(p-1)/p_1} (\bmod p)$$

$$\equiv (x^{r_1 M_1 y_1})^{(p-1)/p_1} (x^{r_2 M_2 y_2})^{(p-1)/p_1} \cdots (x^{r_k M_k y_k})^{(p-1)/p_1} (\bmod p)$$

由于 $M_1 y_1 \equiv 1 (\bmod p_i^{n_i})$，令 $M_1 y_1 = h p_1^{n_1} + 1$

故 $$(x^{r_1 M_1 y_1})^{(p-1)/p_1} \equiv (x^{r_1 (h p_1^{n_1}+1)})^{(p-1)/p_1} (\bmod p)$$

$$\equiv (x^{r_1 h p_1^{n_1-1}})^{p-1} (x^{r_1})^{(p-1)/p_1} (\bmod p)$$

由于 $$x^{p-1} \equiv 1 (\bmod p)$$

故 $$(x^{r_1 M_1 y_1})^{(p-1)/p_1} \equiv (x^{r_1})^{(p-1)/p_1} (\bmod p)$$

另一方面，$p_1 \mid M_j$，$j = 2, 3, \cdots, k$，故 $j \neq 1$ 时

$$(x^{r_j M_j y_j})^{(p-1)/p_1} \equiv (x^{p-1})^{r_j M_j y_j / p} \equiv 1 (\bmod p)$$

所以 $$y^{(p-1)/p_1} \equiv (x^{r_1})^{(p-1)/p_1} \equiv (x^{r_1})^{(p-1)/p_1} (\bmod p)$$

由于 $$y^{(p-1)/p_1} \equiv (x^{r_1})^{(p-1)/p_1} \equiv (\bmod p)$$

$$r_1 = r_{10} + r_{11} p_1 + r_{12} p_1^2 + \cdots + r_{1(n_1-1)} p_1^{n_1-1}$$

$$0 \leqslant r_{1j} \leqslant p_1 - 1, j = 1, 2, \cdots, n_1 - 1$$

故 $$y^{(p-1)/p_1} \equiv x^{r_{10}(p-1)/p_1} (\bmod p)$$

比较等号两端可以确定 r_{10}，但

$$x^{r_1 - r_{10}} = x^{r_{11} p + r_{12} p^2 + \cdots + r_{1(n_1-1)} p_1^{n_1-1}}$$

若令 $$y_1 = y x^{-r_{10}}$$

则有 $$(y x^{-r_{10}})^{(p-1)/p_1} = (x^{r_{11}(p-1)})(x^{r_{12}(p-1)p_1}) \cdots (x^{r_{1(n_1-1)}(p-1)p_1^{n_1-2}})$$

$$\equiv (x^{(p-1)/p_1})^{r_{11}} (\bmod p)$$

由上面的等式两端可以确定 r_{11}。类似理由，令

$$y_2 = y_1 x^{-r_{11} p_1}$$

则 $$(y_2)^{(p-1)/p_1^2} \equiv x^{r_{12}(p-1)/p_1}$$

由上式可确定 r_{12}。依此类推，可求得 $r_{13}, \cdots, r_{1(n_1-1)}$。为了计算方便，令 $\beta_1 = x^{(p-1)/p_1}$，预先计算 $\beta_1, \beta_1^2, \cdots, \beta_1^{p_1-1}$。

例 7.9 $p = 8101$，$x = 6$，$y = 7833$。求 r 使之满足 $y \equiv x^r (\bmod p)$。

$$8101 = 6 \times 1350 + 1$$

$$6(-1350) \equiv 1 (\bmod 8101)$$

则 $$6^{-1} \equiv 6751 (\bmod 8101)$$

对于 $p_1 = 2$，有

$$\beta_1 \equiv 6^{8100/2} \equiv 6^{4050} \equiv 8100 (\bmod 8101)$$

β_1	β_1^2
8100	1

对于 $p_2 = 3$，有

$$\beta_2 = 6^{8100/3} = 6^{2700} \equiv 5883 (\bmod 8101)$$

β_2	β_2^2	β_2^3
5883	2217	1

对于 $p_3=5$，有

$$\beta_3=6^{8100/5}=6^{1620}\equiv 3547(\bmod 8101)$$

β_3	β_3^2	β_3^3	β_3^4	β_3^5
3547	356	7077	5221	1

$M_1y_1\equiv 1(\bmod 4)$，　$M_1=3^4\cdot 5^2=1215$

$M_1y_1\equiv 0(\bmod 3^4)$，　$M_1y_1\equiv 2025(\bmod 8101)$

$M_1y_1\equiv 0(\bmod 5^2)$，

$M_2y_2\equiv 0(\bmod 2^2)$，　$M_2=2^2\cdot 5^2=100$

$M_2y_2\equiv 1(\bmod 3^4)$，　$M_2y_2\equiv 6400(\bmod 8101)$

$M_2y_2\equiv 0(\bmod 5^2)$，

$M_3y_3\equiv 0(\bmod 2^2)$，　$M_3=2^2\cdot 3^4=324$

$M_3y_3\equiv 0(\bmod 3^4)$，　$M_3y_3\equiv 7776(\bmod 8101)$

$M_3y_3\equiv 1(\bmod 5^2)$，

(a)　$p_1=2$，　$n_1=2$

$y=7833$，　$y^{8100/2}\equiv 8100$

故　$r_{10}=1$，　$y_1=yx^{-1}\equiv 5356$

$(y_1)^{8100/2^2}\equiv 1(\bmod 8101)$

即　$r_{11}=0$

所以　$r_1=1$

(b)　$p_2=3$，　$n_2=4$

$y=7833$，　$y^{8100/3}\equiv 8100$，　$r_{20}=2$

$y_1=yx^{-2}$，　$6^{-2}\equiv 7876(\bmod 8101)$

故　$y_1\equiv 7833\times 7876\equiv 3593(\bmod 8101)$

$(y_1)^{8100/3^2}\equiv (3592)^{900}\equiv 2217(\bmod 8101)$

故　$r_{21}=2$

$y_2=y_1x^{-6}\equiv 3593\times 7482(\bmod 8101)\equiv 3708$

$(y_2)^{8100/3^3}\equiv (3708)^{300}\equiv 5883(\bmod 8101)$

故　$r_{22}=1$

$y_3=y_2x^{-2^2}\equiv 3708\times 2960\equiv 6926(\bmod 8101)$

故　$(y_3)^{8100/3^4}\equiv (6926)^{100}\equiv 5883(\bmod 8101)$

故　$r_{23}=1$

$r_2=2+2\times 3+3^2+3^3=44$

(c)　$p_3=5$，　$n_3=2$

$y=7833$，$y^{8100/5}\equiv (7833)^{1620}\equiv 356(\bmod 8101)$

故　$r_{30}=2$

$y_1=yx^{-2}\equiv 7833\times 7876\equiv 3593$；$y_1^{8100/5^2}\equiv 356(\bmod 8101)$

故
$$r_{31}=2$$
$$r_3=2+2\times 5=12$$
由中国剩余定理得 $r=r_1M_1y_1+r_2M_2y_2+r_3M_3y_3$
$$=2025+44\times 6400+12\times 7776=376937$$
$$\equiv 4337(\bmod 8101)$$

当然，Pohlig-Hellman 方法建立在 $p-1$ 可以因子分解成 $p_1^{n_1}p_2^{n_2}\cdots p_k^{n_k}$ 的基础上，也就是在 $p_1,p_2,\cdots,p_k$ 都是小素数时才可能。

7.9.3 求离散对数的 Shank 法

下面介绍一种比较有效的 Shank 算法，不仅速度快，而且需要的存储也较少。

$$y=a^x \bmod p,\quad 0\leqslant x<n$$

n 是 a 的乘法周期，即 $a^n\equiv 1 \bmod p$，已知 y 求 x。

Shank 算法的基本思想如下，运算在 $GF(p)$ 上进行。

(1) 选一 $d\approx\sqrt{n}$

$$x=qd+r,\quad 0\leqslant r<d,\quad q\leqslant\frac{x}{d}\approx\sqrt{n}$$

$$r<d\approx\sqrt{n}$$

(2) 建立一表格 $(\lambda,\log_a\lambda)$，$\log_a\lambda=0,1,\cdots,d-1$，$\lambda$ 按顺序排序，以便于检索。

(3) 由于假定 $y=a^x=a^{qd+r}$，所以

$$y(a^{-d})^q=a^{qd+r}a^{-qd}=a^r$$
$$y(a^{-d})^q=y(a^{n-d})^q$$

计算 $y(a^{n-d})^q$，$q=0,1,2,\cdots$，直到结果等于表中某个 r，$r=\log_a x$，$x=qd+r$。Shank 算法如下，其中
$$r=\log_a\gamma$$

S1. 选择 $d\approx\sqrt{n}$，$r\leftarrow 0$，$\gamma\leftarrow 1$；

S2. $(\lambda,\log_a\gamma)$ 进入表格；

S3. 若 $r=d-1$，则作

始 对表格中的 γ 进行排序，转 S4；

终。否则，作

始 $\gamma\leftarrow a\gamma$，$r\leftarrow r+1$，转 S2；

终。

S4. 计算 $A\leftarrow a^{-d}$，$B\leftarrow y$，$q\leftarrow 0$；

S5. 若存在 (γ,r)，其中 $\gamma=B$，则作

始 $x\leftarrow qd+r$，输出 x，结束；

终。否则，作

始 $B\leftarrow AB$，$q\leftarrow q+1$，转 S5；

终。

本算法的流程图见图 7.5。

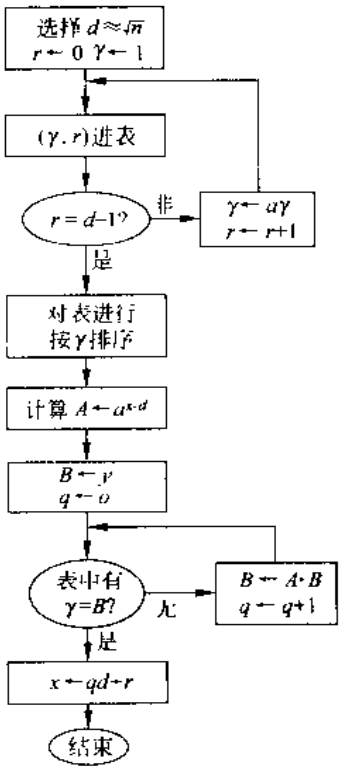


图 7.5

例 7.10 在 $GF(23)$ 上求 $\log_5 3$。5 的乘法周期 n 为 22，故 5 是 $GF(23)$ 的本原元素。

$$g = 5 \approx \sqrt{22}$$

在 $GF(23)$ 上计算 $g=a^x$，$x=1,2,\cdots$

$$5^0 \equiv 1,\quad 5^1 \equiv 5,\quad 5^2 \equiv 2,\quad 5^3 \equiv 10,\quad 5^4 \equiv 4,$$
$$5^5 \equiv 20,\quad 5^6 \equiv 8\quad 5^7 \equiv 17,\quad 5^8 \equiv 16,\quad 5^9 \equiv 11,$$
$$5^{10} \equiv 9,\quad 5^{11} \equiv 22,\quad 5^{12} \equiv 18,\quad 5^{13} \equiv 21,\quad 5^{14} \equiv 13,$$
$$5^{15} \equiv 19,\quad 5^{16} \equiv 3$$

若用穷举法，可得 $x=16$，即

$$g = 5^{16} \equiv 3 \bmod 23 \quad 或 \quad \log_5 3 = 3$$

其实，$r=0,1,2,3,4$，γ 分别对应的值依 γ 的顺序列表于后

γ	1	2	4	5	10
r	0	2	4	1	3

$5^{d} \equiv 5^{22-5} \equiv 5^{17} \equiv 15$，$A \leftarrow 15$。

$q \leftarrow 0$，$y \leftarrow 3$，表中无 $\gamma=3=5^{16}$。

$3\times 15 \equiv 5^{16-17} \equiv 5^{33} \equiv 5^{11} \equiv 22$，$B \leftarrow 22$，$q \leftarrow 1$，表上无 $\gamma=22$ 的行。

$22\times 15 \equiv 5^{11+17} \equiv 5^{6} \equiv 8$，$B \leftarrow 8$，$q \leftarrow 2$，表上无 $\gamma=8$ 的行。

$8\times 15 = 5^{6+17} \equiv 5^{23} \equiv 5$，表中存在 (γ,r)，其中 $\gamma=5$，$r=1$，$q=3$，$x=qd+r=3\cdot 5+1=16$。

算法的复杂性留给读者思考。

下面介绍当 n 可以因子分解时求离散对数的方法，这时可以简化计算步骤。

若 a 的乘法周期 $N=n_1n_2$，即 $a^N \equiv 1$。则 $a_1=a^{n_2}$ 的乘法周期为 n_1　$a_2=a^{n_1}$ 的乘法周期为 n_2，若

$$b = a^x, 0 \leqslant x \leqslant N$$

则
$$x = x_2n_1 + x_1, 0 \leqslant x_1 \leqslant n_1, 0 \leqslant x_2 \leqslant n_2$$

令
$$b_1 = b^{n_2} = a^{x_1n_2} = a^{x_2n_1n_2+x_1n_2}$$

由于
$$a^{n_1n_2} = a^N \equiv 1$$

故
$$b_1 = a^{n_2x_1} = a_1^{x_1}$$

即
$$x_1 = \log_{a_1} b_1$$

令
$$b_2 = ba^{-x_1} = ba^{N-x_1}$$
$$= a^{x_2n_1+x_1}a^{-x_1} = a^{x_2n_1} = a_2^{x_2}$$

所以
$$x_2 = \log_{a_2} b_2$$

故得：若 $N=n_1n_2$，求 $x=\log_a b$ 的算法：

S1. $a_1 \leftarrow a^{n_2}$，$b_1 \leftarrow b^{n_2}$

S2. 求 $x_1 \leftarrow \log_{a_1} b_1$

S3. $a_2 \leftarrow a^{n_1}$，$b_2 \leftarrow a^{N-x_1}$

S4. 求 $x_2 \leftarrow \log_{a_2} b_2$

S5. $x = x_2n_1 + x_1$

a 是 $GF(p)$ 上的本原元素，$y=a^x$，当 $N=p-1$，有小素数因子时，离散对数 $x=\log_a y$ 不困难。利用离散对数作为密码时最好 $p-1=2p_1$，p_1 是大素数。

特别是 $GF(2^m)$ 上，当 $m=127$ 时，$N=2^{127}-1$ 是素数，$\sqrt{N}\approx 10^{19}$。

当 $m=521$ 时，$N=2^{521}-1$ 也是素数，$\sqrt{N}\approx 10^{78}$。

也可以按下列方法求 m，使满足：

$$29^m \equiv 30 \bmod 97$$

先造一表：

$$p_i \equiv 29^i \bmod 97, \quad i=0,1,2,\cdots,9$$

i	0	1	2	3	4	5	6	7	8	9
p_i	1	29	65	42	54	14	18	37	6	77

设 $m=10j+i$，$0\leqslant i\leqslant 9$，则

$$29^m \equiv 30 \cdot 29^{10j} \bmod 97$$

但
$$29^{-1}\equiv 87 \bmod 97, \quad 87^{10}\equiv 49 \bmod 97$$

故
$$29^m \equiv 30 \cdot 87^{10j} \bmod 97 \equiv 30 \cdot 49^j \bmod 97$$

令 $j=0,1,2,\cdots,9$ 计算

$$q_j \equiv 30 \cdot 49^j \bmod 97$$

与 $p_i \equiv 29^i \bmod 97$ 进行比较。

由于 $m<97$，

所以
$$j\leqslant\left\lfloor\frac{97}{10}\right\rfloor=9$$

不难发现 $i=5$，$j=4$ 有

$$30 \cdot 49^4 \equiv 29^5 \bmod 97$$

$m=4\times 10+5=45$，有

$$26^{45} \equiv 30 \bmod 97$$

7.9.4 求离散对数的 Pollard ρ 法

求 m 使满足

$$C \equiv a^m \bmod q$$

将 $GF(q)$ 的 p 阶乘法子群分解为 3：G_0, G_1, G_2。

$$x \in G_i \Leftrightarrow x \equiv i \bmod 3$$

$$x_{i+1}=f(x_i)=\begin{cases} x_i^2 \bmod q, & x_i \in G_0 \\ cx_i \bmod q, & x_i \in G_1 \\ ax_i \bmod q, & x_i \in G_2 \end{cases}$$

设 $x_0=1$，可得序列 $\{x_i\}$，$i\geqslant 0$。

设 $x_i=a^{a_0}c^{b_0}$，对应可得序列 $\{a_i\}$，$i\geqslant 0$，$\{b_i\}$，$i\geqslant 0$，$a_0=b_0=0$。

$$a_{i+1}=\begin{cases}2a_i \bmod p, & x_i\in G_0\\ a_i & x_i\in G_1\\ a_{i+1} \bmod p, & x_i\in G_2\end{cases}$$

$$b_{i+1}=\begin{cases}2b_i \bmod p, & x_i\in G_0\\ b_{i+1} \bmod p, & x_i\in G_1\\ b_i & x_i\in G_2\end{cases}$$

$$x_{i+1}=x_i^2\equiv(\alpha^{a_i}c^{b_i})^2 \bmod q=\alpha^{a_{i+1}}c^{b_{i+1}},\ x_i\in G_0$$

$$x_{i+1}\equiv cx_i\equiv c\alpha^{a_i}c^{b_i} \bmod q=\alpha^{a_{i+1}}c^{b_{i+1}},\ x_i\in G_1$$

$$x_{i+1}=\alpha x_i\equiv\alpha\cdot\alpha^{a_i}c^{b_i} \bmod q=\alpha^{a_{i+1}}c^{b_{i+1}},\ x_i\in G_2$$

若存在 i 和 j 使 $x_0=x_j$，则

$$\alpha^{a_i}c^{b_i}=\alpha^{a_j}c^{b_j}$$

即

$$\alpha^{a_i-a_j}=c^{b_j-b_i}$$

则

$$m=\frac{a_j-a_i}{b_i-b_j} \bmod p$$

当然前提是 $b_i\neq b_j$，b_i-b_j 存在逆。

如若 $b_i=b_j$，可令 $c'=c\alpha$，问题改为求 m'，$c'\equiv\alpha^{m'} \bmod q$，$m'=m+1$。

为了找 i 和 j，使 $x_i=x_j$，可采用以下步骤以节省存储单元，即从(x_1,x_2)开始依次计算(x_2,x_4)，(x_3,x_6)，…，(x_i,x_{2i})，(x_{i+1},x_{2i+2})，…

从(x_i,x_{2i})计算(x_{i+1},x_{2i+2})，如下进行

$$x_{i+1}=f(x_i),\quad x_{2i+2}=f(f(x_{2i}))$$

举例说明如下：求 m 满足

$$121^m\equiv 3435 \bmod 4679$$

$$q=4679,q-1=2\times 2339$$

$121=11^2$，11 是 $GF(4679)$的本原元素。

121 是阶为 2339 的 $GF(4679)$的乘法群的一个子群的生成元素。

其中

$$3435^{2339}\equiv 1 \bmod 4679$$

$$q=4679,\ \alpha=121,\ c=3435,\ p=2339$$

$$x_0=1\in G_1,\ a_0=b_0=1$$

$$x_1\equiv cx_0 \bmod q\equiv 3475\in G_1$$

$$x_2\equiv cx_1 \bmod q\equiv(3475)^2 \bmod 4679\equiv 691\in G_1$$

$$x_3\equiv cx_2 \bmod q\equiv 3475\times 691 \bmod 4679\equiv 898\in G_1$$

$$x_4\equiv 3475\times 898 \bmod 4679\equiv 2595\in G_0$$

$$\vdots$$

若采用 Floyd 的方法有

$$(x_1,x_2)=(3475,691)$$

$$(x_2, x_4) = (691, 2595)$$
$$\vdots$$

$i=76$ 时有 $x_{76}=492, x_{152}=492$

与之相应有

$$a_1 = a_1 = 0, b_1 = 1$$

由于 $x_1 \in G_1$，有 $a_2=0, b_2=b_1+1=2, \cdots$

$i=76$ 时，有 $a_{76}=84, b_{76}=2191, a_{152}=286, b_{152}=915, m \equiv \frac{286-84}{2191-915} \bmod 2339$。

$2191-915=1276,$

$2339=1276+1063,$ $\quad 1063=2339-1276,$

$1276=1063+213,$ $\quad 213=1276-1063,$

$1063=4\times213+211,$ $\quad 211=1063-4\times213,$

$213=211+2,$ $\quad 2=213-211$

$$211 = 105 \times 2 + 1$$

$$\begin{aligned} 1 &= 211 - 105 \times 2 = 211 - 105(213-211) \\ &= 106 \times 211 - 105 \times 213 \\ &= 106(1063 - 4 \times 213) - 105 \times 213 \\ &= 106 \times 1063 - 529 \times 213 \\ &= 106 \times 1063 - 529(1276 - 1063) \\ &= 635 \times 1063 - 529 \times 1276 \\ &= 635(2339 - 1276) - 529 \times 1276 \\ &= 635 \times 2339 - 1164 \times 1276 \end{aligned}$$

所以

$$1276(-1164) \equiv 1 \bmod 2339$$
$$1276^{-1} \equiv -1164 \bmod 2339 \equiv 1175$$

故

$$\begin{aligned} m &\equiv 202 \times 1175 \bmod 2339 \equiv 237350 \bmod 2339 \\ &\equiv 1111 \end{aligned}$$

7.9.5 求离散对数的另一种方法

循环群 $G=\{e, g, g^2, \cdots, g^{r-1}\}$，$g^n=e$，$g$ 是 G 的生成元素。

问题：已知 h，求 m 满足 $g^m=h$。

步骤：

S1. 选择一 G 的子集 S，使得 G 的大部分元素可表为 S 的元素之积。设 S 的规模为 l；

S2. 选一足够多的指数 i 的集合，使得 g^i 可表为

$$g^i = s_1^{k_{i1}} s_2^{k_{i2}} \cdots s_l^{k_{il}}, \quad i \in I$$

$$i \equiv k_{i1} \log_g S_1 + k_{i2} \log_g S_2 + \cdots + k_{il} \log_g S_l, \quad \bmod n, i \in I$$

S3. 选一随机数 r，使得 $g^r h (=g^{r+m})$ 可表为 S 集合元素之积，则

$$r + m \equiv v_1 \log_g S_1 + v_2 \log_g S_2 + \cdots + v_l \log_g S_l \bmod n$$

假定 $\log_g S_i$ 在 S_2 中已解出来。

现在讨论 $GF(p)$ 域上的乘法群。

设 $G=\{1,2,\cdots,p-1\}$，g 是生成元素。选 $\{p_1,p_2,\cdots,p_k\}$ 为前面 k 个素数。当 k 充分大时，G 的元素可表为这 k 个素数之积。

例如，$p=541$，$g=2$，$p-1=540=2^2\cdot 3^3\cdot 5$

$$2^{\left(\frac{541-1}{2}\right)}\equiv 540,\ \mathrm{mod}\ 541$$

$$2^{\left(\frac{541-1}{3}\right)}\equiv 129,\ \mathrm{mod}\ 541$$

$$2^{\left(\frac{541-1}{5}\right)}\equiv 48,\ \mathrm{mod}\ 541$$

若选取 2,3,5,7 前面四个素数。

经过若干次试验得：$2^{14},2^{81},2^{207},2^{214},2^{300}$ 为

$$2^{14}\equiv 16384\equiv 154\equiv 2\cdot 7\cdot 11\ \mathrm{mod}\ 541$$

$$2^{81}\equiv 294\equiv 2\cdot 3\cdot 7^2\ \mathrm{mod}\ 541$$

$$2^{207}\equiv 275\equiv 5^2\cdot 11\ \mathrm{mod}\ 541$$

$$2^{214}\equiv 35\equiv 5\cdot 7\ \mathrm{mod}\ 541$$

$$2^{300}\equiv 352\equiv 2^5\cdot 11\ \mathrm{mod}\ 541$$

令

$$m_1=\log_2 2,\quad m_2=\log_2 3,\quad m_3=\log_2 5,\quad m_4=\log_2 7,\quad m_5=\log_2 11$$

$$2^{14}\equiv 2\cdot 7\cdot 11\equiv 2^{\log_2 2}\cdot 2^{\log_2 7}\cdot 2^{\log_2 11}\equiv 2^{m_1}\cdot 2^{m_4}\cdot 2^{m_5}\ \mathrm{mod}\ 541$$

所以 $14\equiv m_1+m_4+m_5\ \mathrm{mod}\ 540$

类似有

$$2^{81}\equiv 2^{\log_2 2}\cdot 2^{\log_2 3}\cdot 2^{\log_2 7}\equiv 2^{m_1}\cdot 2^{m_2}\cdot 2^{2m_4}\ \mathrm{mod}\ 541$$

故 $81\equiv m_1+m_2+2m_4\ \mathrm{mod}\ 540$

$$2^{207}\equiv 2^{2\log_2 5}\cdot 2^{\log_2 11}\equiv 2^{2m_3}\cdot 2^{m_5}\ \mathrm{mod}\ 541$$

所以 $207\equiv 2m_3+m_4\ \mathrm{mod}\ 540$

同理 $214\equiv m_3+m_4\ \mathrm{mod}\ 540$

$300\equiv 5m_1+m_5\ \mathrm{mod}\ 540$

$$\begin{aligned}
&m_1+ && && m_4+m_5 &&\equiv 14\ \mathrm{mod}\ 540\\
&m_1+m_2 && +2m_4 && &&\equiv 81\ \mathrm{mod}\ 540\\
& && 2m_3 && +m_5 &&\equiv 204\ \mathrm{mod}\ 540\\
& && m_3 && +m_4 &&\equiv 214\ \mathrm{mod}\ 540\\
&5m_1 && && +m_5 &&\equiv 300\ \mathrm{mod}\ 540
\end{aligned}$$

解联立同余方程组得

$$m_1=1,\quad m_2=104,\quad m_3=496,\quad m_4=258,\quad m_5=295$$

或

$$2^1\equiv 2\ \mathrm{mod}\ 541,\ 2^{104}\equiv 3\ \mathrm{mod}\ 541,\ 2^{496}\equiv 5\ \mathrm{mod}\ 541$$

$$2^{258}\equiv 7\ \mathrm{mod}\ 541,\ 2^{295}\equiv 11\ \mathrm{mod}\ 541$$

求解 $2^m\equiv 345\ \mathrm{mod}\ 541$

$345=3\cdot5\cdot23$

$2^{13}\cdot345\equiv2^3\cdot7 \bmod 541 \quad 2^{13}\cdot345\equiv2^{13}\cdot2^m\equiv2^{13+m} \bmod 540$

$2^3\cdot7=2^3\cdot2^{\log_2^7}=2^{3+\log_2^7}=2^{3\log_2^2+\log_2^7}$

所以 $13+m=3m_1+m_4\equiv3+258\equiv261 \bmod 540$

$m\equiv248 \bmod 540$

即 $2^{248}\equiv345 \bmod 541$

{1,2,…,540}中可表为S的元素之积的元素有：

{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,18,20,21,22,24,25,27,28,30,32,33,35,36,40,42,44,45,48,49,50,54,55,56,60,63,64,66,70,72,75,77,80,81,84,88,90,96,98,99,100,105,108,110,112,120,121,125,126,128,132,135,140,144,147,150,154,160,162,165,168,175,176,180,189,192,196,198,200,210,216,220,224,225,231,240,242,243,245,250,252,256,264,270,275,280,288,294,297,300,308,315,320,324,330,336,343,350,352,360,363,375,378,384,385,392,396,400,405,420,432,440,441,448,450,462,480,484,486,490,495,500,504,512,525,528,539,540}

第 8 章　椭圆曲线与椭圆曲线上的公钥密码

近若干年对椭圆曲线的研究，特别是它在近代密码学的应用，可谓是异军突起，前景很好。一方面它在因数分解上有很突出的表现，威胁了 RSA 的安全。尤其在椭圆曲线上建立公钥密码系统效率极高。有报告作过比较，在椭圆曲线上 150 比特的密钥，对它进行攻击的工作量以每秒百万次的计算机要进行 3.8×10^{10} 年。而一般 RSA 密码系统，512 比特的密钥，对它攻击只要 3×10^{4} 年。由此可见一斑。

8.1　椭圆曲线导论

椭圆曲线的研究源于椭圆积分

$$\int \frac{\mathrm{d}x}{\sqrt{E(x)}}$$

其中 $E(x)$ 是 x 的三次方或四次方多项式。它不能用初等函数来表示。

椭圆曲线指的是由 Weierstrass 方程：

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{8-1}$$

所确定的曲线。

为方便起见，先从

$$y^2 = x^3 + a_2x^2 + a_4x + a_6$$

入手，探讨椭圆曲线的图像，令

$$f(x) = x^3 + ax^2 + bx + c$$

则 $f(x)=0$ 的根有 3 个，若 $f(x)=0$ 有 3 个实数根，则 Weierstrass 方程的图像如图 8.1 所示。

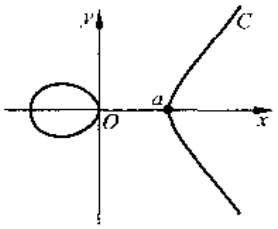


图　8.1

令　　$F(x,y) \equiv y^2 - f(x)$

$$\frac{\partial F}{\partial x} = -f'(x), \quad \frac{\partial F}{\partial y} = 2y$$

若 Weierstrass 方程的曲线 C 不存在这样的点，使得

$$\frac{\partial F}{\partial x} = 0 \quad \frac{\partial F}{\partial y} = 0$$

由于　　$F(x,y)=0$

所以

$$\frac{\partial F}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}x} + \frac{\partial F}{\partial x} = 0$$

$$\frac{\mathrm{d}y}{\mathrm{d}x} = -\frac{\partial F}{\partial x}\Big/\frac{\partial F}{\partial y}$$

所以不存在这样的点使得 $F_x=F_y=0$。说明曲线上所有的点都存在切线。

若在(x_0,y_0)点$\frac{\partial F}{\partial x}$和$\frac{\partial F}{\partial y}$同时为零，即 $y_0=0$，$f'(x_0)=0$，说明$(x_0,0)$是曲线 C 或 $f(x)=0$的二重根。

例如，$y^2=x^3+x^2$，$(-1,0)$点是曲线上的奇点，如图 8.2 所示。

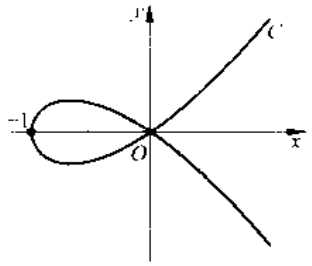


图 8.2

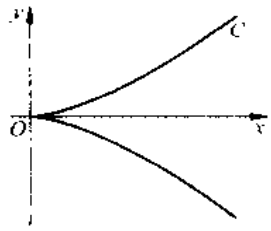


图 8.3

例如，$y^2=x^3$，(0,0)点也是奇点，如图 8.3 所示。

令 $x=X/Z$，$y=Y/Z$，代入 Weierstrass 方程，可得关于射影平面(X,Y,Z)的方程：

$$Y^2Z=X^3+aX^2Z+bXZ^2+cZ^3$$

$Z=0$ 代入可得 $X=0$ 有二重根，这意味着在无穷远交于 3 点，用 $\mathcal{O}$ 表示无穷远点。

Weierstrass 曲线称之为椭圆曲线，表示为 C。若 P,Q 是 C 上两点，如图 8.4 所示，过 P,Q 两点引一直线，交 C 于第 3 点 $P*Q$。过 $P*Q$ 引 x 轴的垂直线，交 C 于点 $P+Q$。设 $Q=(x,y)$，则 $-Q=(x,-y)$。Q 和 $-Q$ 的连线便是垂直于 x 轴的垂直线。它和 C 交于第 3 点即 $\mathcal{O}$ 点(见图 8.5)。

$\mathcal{O}$ 和 $\mathcal{O}$ 的连线还是无穷远线，交 C 于 $\mathcal{O}$。故

$$-\mathcal{O}=\mathcal{O}$$

令

$$P_1=(x_1,y_1),\quad P_2=(x_2,y_2)$$

P_1 和 P_2 的连线交 C 于(x_3,y_3)点，用 $P_1*P_2=(x_3,y_3)$来表示，则

$$P_1+P_2=(x_3,-y_3)$$

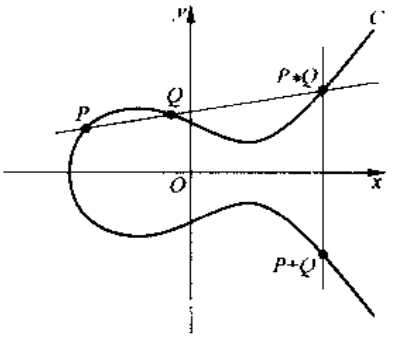


图 8.4

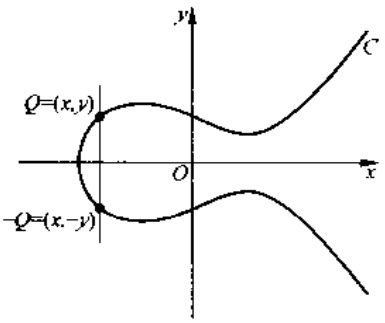


图 8.5

如图 8.6 所示，过 $P_1(x_1,y_1)$，$P_2(x_2,y_2)$的直线方程

$$y = mx + \nu$$

$$m = \frac{y_2 - y_1}{x_2 - x_1}, \quad y_1 - mx_1 = y_2 - mx_2$$

$$y^2 = (mx + \nu)^2 = x^3 + ax^2 + bx + c$$

设

$$x^3 + (a - m^2)x^2 + (b - 2m\nu)x + (c - \nu^2)$$
$$= (x - x_1)(x - x_2)(x - x_3)$$

设 $P_1 * P_2$ 点为(x_3, y_3)，$P_1 + P_2$ 为$(x_3, -y_3)$，根据根与系数关系有

$$m^2 - a = x_1 + x_2 + x_3$$

$$x_3 = m^2 - a - x_1 - x_2$$

$$y_3 = mx_3 + \nu$$

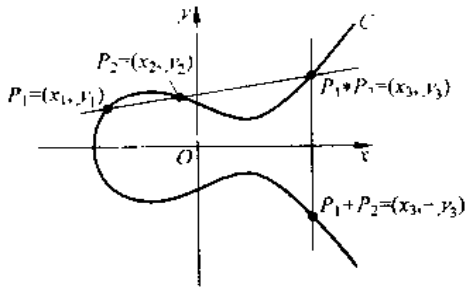


图 8.6

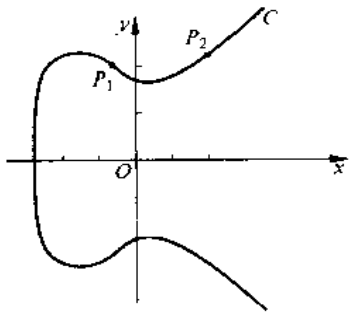


图 8.7

例 8.1 椭圆曲线

$$y^2 = x^3 + 17$$

上 $P_1=(-1,4)$，$P_2=(2,5)$，如图 8.7 所示。

过 P_1，P_2 的直线

$$y = \frac{1}{3}x + \frac{13}{3}$$

过$(-1,4)$，$(2,5)$两点的直线方程为

$$y = \frac{1}{3}x + \frac{13}{3}$$

以之代入 $y^2=x^3+17$ 得

$$\frac{1}{9}x^2 + \frac{26}{9}x + \frac{169}{9} = x^3 + 17$$

所以

$$x^3 - \frac{1}{9}x^2 - \frac{26}{9}x - \frac{16}{9} = 0$$

有 3 个根 $x_1=-1$，$x=2$，第 3 个根设 x_3

所以

$$x_3 = \frac{1}{9} - 2 + 1 = -\frac{8}{9}$$

$$y_3 = \frac{1}{3}x_3 + \frac{13}{3} = -\frac{8}{27} + \frac{13}{3} = \frac{109}{27}$$

所以
$$P_2+P_1=(x_3,-y_3)=\left(-\frac{8}{9},-\frac{109}{27}\right)$$
进一步求 $2P_1$，$P_1=(-1,4)$，P_1 和 P_1 的连线实即 C 在 P_1 点的切线。
$$y^2=x^3+17$$
$$2y\frac{\mathrm{d}y}{\mathrm{d}x}=3x^2$$
$$\frac{\mathrm{d}y}{\mathrm{d}x}=3x^2/2y$$
故 P_1 点的切线斜率为 $m=-3/8$，切线方程为
$$y=+\frac{3}{8}x+u$$
过$(-1,4)$点，$4=-\frac{3}{8}+u$，$u=4+\frac{3}{8}=\frac{35}{8}$

所以
$$y=+\frac{3}{8}x+\frac{35}{8}$$
以 $y=\frac{3}{8}x+\frac{35}{8}$代入 $y^2=x^3+17$ 得
$$\frac{9}{64}x^2+\frac{210}{64}x+\frac{1225}{64}=x^3+17$$
所以
$$x^3-\frac{9}{64}x^2-\frac{210}{64}x-\frac{137}{64}=0$$
但
$$x^3-\frac{9}{64}x^2-\frac{210}{64}x-\frac{137}{64}=(x+1)^2\left(x+\frac{137}{64}\right)$$
故 $P=(-1,4)$点切线交 $y^2=x^3+17$ 于(x^*,y^*)
$$x^*=\frac{137}{64}$$
$$y^*=\frac{8}{3}\cdot\frac{137}{64}+\frac{35}{8}=\frac{2651}{512}$$
故 $2P=\left(\frac{137}{64},-\frac{2651}{512}\right)$。

8.2 有限域上的椭圆曲线

假定讨论的 Weierstrass 方程是
$$y^2+a_1xy+a_3y=x^3+a_2x^2+a_4x+a_6$$
$$a_1,a_2,a_3,a_4,a_6\in\bar{F} \tag{8-2}$$
其中
$$\bar{F}=GF(q)$$
q 是一素数，称为是 $GF(q)$的特征。

射影平面上两点
$$(X_1,Y_1,Z_1),\quad (X_2,Y_2,Z_2)$$

若存在 $u\in\bar{K}$ 使

$$X_1 = uX_2,\quad Y_1 = uY_2,\quad Z_1 = uZ_2$$

则表以

$$(X_1,Y_1,Z_1)\sim(X_2,Y_2,Z_2)$$

即(X_1,Y_1,Z_1)和(X_2,Y_2,Z_2)对等的意思。

令 Weierstrass 方程为

$$F(X,Y,Z) = Y^2Z + a_1XYZ + a_3YZ^2 - X^3 - a_2X^2Z - a_4XZ^2 - a_6Z^3 = 0$$

所有满足 $F(X,Y,Z)=0$ 的点，若

$$\frac{\partial F}{\partial X},\quad \frac{\partial F}{\partial Y},\quad \frac{\partial F}{\partial Z}$$

不全为零，即至少其中有一项不为零，则称该点为非奇异的点，否则是奇异的点。

所有满足 $F(X,Y,Z)=0$ 的非奇异的点，包含无穷远点 O 在内，构成一椭圆曲线，用 E 表示它。

无穷远点 $O=(0,1,0)$。

若 $a_1,a_2,a_3,a_4,a_6\in F$，则称式(8-2)在域 F 上定义一椭圆曲线 E，表以 E/F，E 上属于域 F 的有理点记以 $E(F)$，也就是坐标属于 F 的点，包括无穷远点在内，有时也简记为 E。

两个椭圆曲线：

$$\begin{aligned} E_1:\ & y_1^2 + a_1x_1y_1 + a_3y_1 = x_1^3 + a_2x_1^2 + a_4x_1 + a_6 \\ E_2:\ & y_2^2 + a_1x_2y_2 + \bar{a}_3y_2 = x_2^3 + \bar{a}_2x_2^2 + \bar{a}_4x_2 + a_6 \end{aligned} \tag{8-3}$$

若存在 $u,r,s,t\in F,u\neq0$，使得变换

$$\begin{aligned} \psi:\ x_2 &= u^2x_1 + r \\ y_2 &= u^3y_1 + u^2sx_1 + t \end{aligned}$$

使 E_1 变为 E_2，则称 E_1 和 E_2 关于域 F 是同构的。

同样的理由存在变换

$$\begin{aligned} \phi:\ x_1 &= u^{-2}(x_2 - r) \\ y_1 &= u^{-3}(y_2 - sx_2 - t + rs) \end{aligned}$$

可将 E_2 转换为 E_1，则称 E_1 和 E_2 是同构的。E_1 和 E_2 同构用

$$E_1\cong E_2$$

表示它。

定理 8.1 若式(8-3)定义的两椭圆曲线同构的充分必要条件是：存在 $u,r,s,t\in F$，$u\neq0$，使

$$\begin{aligned} u\bar{a}_1 &= a_1 + 2s \\ u^2\bar{a}_2 &= a_2 - sa_1 + 3r - s^2 \\ u^3\bar{a}_3 &= a_3 + ra_1 + 2t \\ u^4\bar{a}_4 &= a_4 - sa_3 + 2ra_2 - (t + rs)a_1 + 3r^2 - 2st \\ u^6a_6 &= a_6 + ra_4 + r^2a_2 + r^3 - ta_3 - t^2 - rta_1 \end{aligned}$$

只要直接代入验证即可。

8.3 椭圆曲线上的群

在椭圆曲线 E 上的点关于"+"法构成 Abel 群。$\forall P,Q\in E$,

a. $P+\mathcal{O}=P,\mathcal{O}+P=P$。

b. $-\mathcal{O}=\mathcal{O}$。

c. 若 $P=(x_1,y_1)\neq\mathcal{O}$,则 $-P=(x_1,-y_1-a_1x_1-a_3)$。

注意 P 和 $-P$ 是椭圆曲线上惟一一个与 P 有共同 x 坐标的点。

d. 若 $Q=-P$,则 $P+Q=\mathcal{O}$。

e. 若 $P\neq\mathcal{O},Q\neq\mathcal{O},\mathcal{O}\neq-P,P(\neq Q)$ 和 Q 的连线交椭圆曲线于第三点,若 $P=Q,P$ 点的切线交 E 于 R,则 $P+Q=-R$。

直观上容易看清上面的结论,$\mathcal{O}=(0,1,0)$,表示(x,∞)点,即 x 坐标为不定式,而 y 坐标为∞点,如图 8.8 所示。

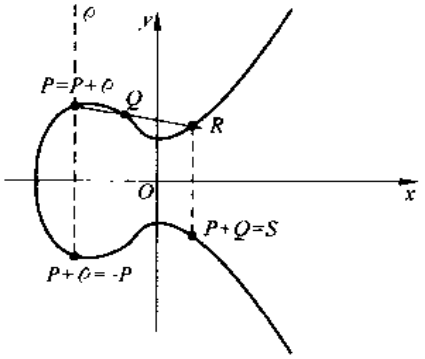


图 8.8

椭圆曲线上的点,关于"+"法构成 Abel 群,以 $\mathcal{O}$ 为单位元素。$P+Q=Q+P$,交换律成立。要证明成群,困难在于证结合律成立,即

$$(P+Q)+R=P+(Q+R)=P+Q+R$$

令 $P=(x_1,y_1),Q=(x_2,y_2),P+Q=S=(x_3,y_3)$,关键在于求 x_3,y_3 的计算,现讨论于下:

$P=Q$ 时,$R=2P$。

PQ 联线的斜率

$$m=\begin{cases}\dfrac{y_2-y_1}{x_2-x_1}, & \text{若 } P\neq Q\\[2ex] \dfrac{3x_1^2+2a_2x_1+a_4-a_1y_1}{2y_1+a_1x_1+a_3}, & \text{若 } P=Q\end{cases}$$

$P=-Q$ 时,$m=\dfrac{y_2-y_1}{x_2-x_1}$,显然成立。

$P=Q$,问题导致求 P 点的切线斜率。对式(8-2)两端对 x 求导得

$$(2y+a_1x+a_3)\frac{\mathrm{d}y}{\mathrm{d}x}+a_1y=3x^2+2a_2x+a_4$$

所以 $P=(x_1,y_1)$点的斜率

$$\left.\frac{\mathrm{d}y}{\mathrm{d}x}\right|_{P=(x_1,y_1)}=\frac{3x_1^2+2a_2x_1+a_4-a_1y_1}{2y_1+a_1x_1+a_3}$$

问题导致求直线

$$y=mx+b$$

和椭圆曲线的交点,以 $y=mx+b$ 代入式(8-2)得

$$x^3+a_2x^2+a_4x+a_6-(mx+b)^2-a_1x(mx+b)-a_3(mx+b)=0 \qquad (8\text{-}4)$$

若上式有 x_1,x_2,x_3 3 个根,即上式等于

$$(x-x_1)(x-x_2)(x-x_3)=0$$

根据根与系数的关系，有

$$-(x_1+x_2+x_3)=(8\text{-}3)\text{式展开后 }x^2\text{ 项的系数}$$

即

$$(x_1+x_2-x_3)=-a_2+m^2+a_1m$$

$$x_3=m^2+a_1m-a_2-x_1-x_2$$

$$y_3=-mx_3-b$$

由于

$$y(y+a_1x+a_3)=x^3+a_2x^2+a_4x+a_6$$

故得

$$x_3=\left(\frac{y_2-y_1}{x_2-x_1}\right)^2+a_1\left(\frac{y_2-y_1}{x_2-x_1}\right)-a_2-x_1-x_2$$

$$y_3=\left(\frac{y_2-y_1}{x_2-x_1}\right)(x_1-x_3)-y_1-a_1x_3-a_3$$

总之 P 和 Q 已知求 $P+Q=R=(x_3,y_3)$。可在确定而且有限时间内完成。常数 b 不难求得。结合律的证明留给读者。

8.4 判别式 Δ 与不变式 j

对于式(8-2)的 Weierstrass 方程，定义

$$d_2=a_1^2+4a_2$$

$$d_4=2a_4+a_1a_3$$

$$d_6=a_3^2+4a_6$$

$$d_8=a_1^2a_6+4a_2a_6-a_1a_3a_4+a_2a_3^2-a_4^2$$

$$c_4=d_2^2-24d_4$$

$$\Delta=-d_2^2d_8-8d_4^3-27d_6^2+9d_2d_4d_6$$

$$j(E)=c_4^3/\Delta$$

定理 8.2 若 E 是一椭圆曲线，即 Weierstrass 方程(8-3)是非奇异的，当且仅当 $\Delta\neq0$。

定理 8.3 两个椭圆曲线 E_1/K 和 E_2/K 同构，则 $j(E_1)=j(E_2)$。

8.4.1 关于 F 的特征 $Char(F)\neq2,3$ 的椭圆曲线

先假定 F 的 $Char(F)\neq2$，$Char$ 是 F 域的特征，则变换

$$x_1=x$$

$$y_1=y-\frac{1}{2}a_1x-\frac{1}{2}a_3$$

将 E：$y^2+a_1xy+a_3y=x^3+a_2x^2+a_4x+a_6$ 变换成

$$E'/F:\ y_1^2=x_1^3+b_2x_1^2+b_4x_1+b_6$$

关于 F，$E\cong E'$。

若 F 的 $Char(F)\neq2,3$，则变换

$$x_1=\frac{x-3b_2}{36}$$

$$y_1 - \frac{y}{216}$$

将 E'变换成

$$E''/F:\ y^2 = x^3 + ax + b$$

注意：$E' \cong E''$，故 $E \cong E''$。

所以 F 的 $Char(F) \neq 2,3$，可将 E/F 假定是

$$y^2 = x^3 + ax + b, \quad a,b \in F$$

即 $Char(F) \neq 2,3$ 时，可假定选择 Weierstrass 方程，使

$$a_1 = a_2 = a_3 = 0$$

这时

$$\Delta = -16(4a^3 + 27b^2)$$

$$j(E) = -1728(4a)^3/\Delta$$

因假定 E 是非奇导，故 $\Delta \neq 0$。

定理 8.4 椭圆曲线 $E_1/F: y_1^2 = x_1^3 + ax_1 + b$，和 $E_2/F: y_2^2 = x_2^3 + \bar{a}x_2 + \bar{b}$ 关于 F 同构的充分必要条件是：存在 $u \in F$，使得 $u^4\bar{a} = a$，$u^6\bar{b} = b$。

若 $E_1 \cong E_2$，则同构由

$$\phi: E_1 \to E_2$$

$$x_2 = u^{-2}x_1, \quad y_2 = u^{-3}y_1$$

或

$$\psi: E_2 \to E_1$$

$$x_1 = u^2x_2, \quad y_1 = u^3y_2$$

而且当 $P_1 = (x_1, y_1) \in E$，$-P = (x_1, -y_1)$，$Q = (x_2, y_2) \in E$，$Q \neq -P$，则 $P + Q = (x_3, y_3)$

$$x_3 = m^2 - x_1 - x_2$$

$$y_3 = m(x_1 - x_3) - y_1$$

$$m = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1}, & \text{若 } P \neq Q \\ \dfrac{3x_1^2 + a}{2y_1}, & \text{若 } P = Q \end{cases}$$

$$2y\frac{dy}{dx} = 3x^2 + a$$

所以

$$\left.\frac{dy}{dx}\right|_P = \frac{3x_1 + a}{2y_1}$$

当 $P = Q$ 时，由于

$$2y\frac{dy}{dx} = 3x^2 + a$$

所以

$$\left.\frac{dy}{dx}\right|_P = \frac{3x_1 + a}{2y_1}$$

例 8.2 $E: y^2 = x^3 + x + 6$，$F = F_{11}$，$a = 1$，$b = 6$，

$$\Delta = -16(4 + 27 \times 36) \equiv 6(4 + 5 \times 3)$$

$$\equiv 6(19) \equiv 6 \times 8 = 48 \equiv 4 \bmod 11$$

$$E(F_{11}) = \{\mathcal{O},(2,4),(3,5),(3,6),(5,2),$$
$$(5,9),(7,2),(7,9),(8,3),(8,8),$$
$$(10,2),(10,9)\}$$

8.4.2 关于特征等于 2 的域的讨论

$Char(F)=2$，椭圆曲线有两个标准形式，即通过适当变换，使方程(8-1)导致

$$y^2+xy=x^3+ax^2+c$$
$$y^2+xy=x^3+bx+c$$

设

$$E:\ y^2+a_1xy+\bar{a}_3y=x^3+\bar{a}_2x^2+\bar{a}_4x+\bar{a}_6$$
$$d_2=\bar{a}_1^2+4\bar{a}_2=\bar{a}_1^2$$
$$d_4=2\bar{a}_4+\bar{a}_1\bar{a}_3=\bar{a}_1\bar{a}_3$$
$$d_6=\bar{a}_3^2+4\bar{a}_6=\bar{a}_3^2$$
$$d_8=\bar{a}_2^2-24\bar{a}_4=\bar{a}_1^4$$
$$\Delta=d_2^2d_8-8d_4^3-27d_6^2+9d_2d_4d_6$$
$$=\bar{a}_1^6-\bar{a}_3^4+\bar{a}_1^3\bar{a}_3^3$$
$$j(E)=c_4^3/\Delta=(d_2^2-24d_4)/\Delta$$
$$=\bar{a}_1^{12}/\Delta$$

8.4.3 $j(E)\neq 0$ 的讨论

即 $\bar{a}_1\neq 0$，令变换

$$x=\bar{a}_1^2x_1+\frac{\bar{a}_3}{\bar{a}_1},\quad y=\bar{a}_1^3y_1+\frac{\bar{a}_1^2\bar{a}_4+\bar{a}_3^2}{\bar{a}_1^3}$$

代入式(8-3)得

$$\left(\bar{a}_1^3y_1+\frac{\bar{a}_1^2\bar{a}_4+\bar{a}_3^2}{\bar{a}_1^3}\right)^2+\bar{a}_1\left(\bar{a}_1^2x_1+\frac{\bar{a}_3}{\bar{a}_1}\right)\left(\bar{a}_1^3y_1+\frac{\bar{a}_1^2\bar{a}_4+\bar{a}_3}{\bar{a}_1^3}\right)+\bar{a}_3\left(\bar{a}_1^3y+\frac{\bar{a}_1^2\bar{a}_4+\bar{a}_3^2}{\bar{a}_1^3}\right)$$
$$=\left(\bar{a}_1^2x_1+\frac{\bar{a}_3}{\bar{a}_1}\right)^3+\bar{a}_2\left(\bar{a}_1^2x_1+\frac{\bar{a}_3}{\bar{a}_1}\right)^2+\bar{a}_4\left(\bar{a}_1^2x_1+\frac{\bar{a}_3}{\bar{a}_1}\right)+\bar{a}_6$$

展开整理除以 $\bar{a}_1^6$ 后得

$$E_1/F:\ y_1^2+x_1y_1=x_1^3+a_2x_1^2+a_6$$

E_1 的 $\Delta=a_6$，$j(E_1)=1/a_6$。

若 $j(E)=0$，即 $\bar{a}_1=0$，则作变换：

$$x=x_1+\bar{a}_2$$
$$y=y_1$$

将 E 变为

$$E_2/F:\ y_1^2+a_3y=x^3+a_4x+a_6$$

对于 E_2：$\Delta=a_3^4$，$j(E_2)=0$。

证明与前面类似，故从略。

若 $P=(x_1,y_1)\in E_1$，因 $y_1(y_1+x_1)=x^3+a_2x_1^2+a_6$，故 $-P=(x_1,y_1+x_1)$。

若 $Q=(x_2,y_2)\in E_1$，且 $Q\neq -P$，则

$$P+Q=(x_3,y_3)$$

$$x_3=\begin{cases}\left(\dfrac{y_1-y_2}{x_1-x_2}\right)^2+\dfrac{y_1-y_2}{x_1-x_2}-x_1-x_2-a_2, & P\neq Q\\ x_1^2+\dfrac{a_6}{x_1^2}, & P=Q\end{cases}\tag{8-5}$$

$$y_3=\begin{cases}\left(\dfrac{y_1+y_2}{x_1+x_2}\right)(x_1+x_3)+y_1, & P\neq Q\\ x_1^2+\left(x_1+\dfrac{y_1}{x_1}\right)x_3, & P=Q\end{cases}\tag{8-6}$$

证明如下：过 P,Q 直线的斜率为

$$m=\frac{y_2-y_1}{x_2-x_1}=\frac{y_2+y_1}{x_2+x_1}$$

直线方程：

$$y=mx+b$$

由于过(x_1,y_1)点，所以

$$y_1=mx_1+b$$

$$b=y_1+mx_1=y_1+\frac{y_2+y_1}{x_2+x_1}x_1$$

所以令

$$y=\frac{y_2-y_1}{x_2-x_1}x+\left(y_1-\frac{y_2-y_1}{x_2-x_1}x_1\right)$$

代入

$$y^2+xy=x^3+a_2x^2+a_6$$

而且(x_1,y_1)，(x_2,y_2)都满足这方程。

$$\left[\frac{y_2+y_1}{x_2+x_1}x+\left(y_1+\frac{y_2+y_1}{x_2+x_1}x_1\right)\right]^2+x\left[\frac{y_2+y_1}{x_2+x_1}x+\left(\frac{y_2+y_1}{x_2+x_1}\right)x_1\right]$$
$$=x^3+a_2x^2+a_6\tag{8-7}$$

x^3：项系数为 1，

x^2：项系数为$-\left(\dfrac{y_2-y_1}{x_2-x_1}\right)^2-\dfrac{y_2-y_1}{x_2-x_1}+a_2$。

根据根与系数关系，如若上面(8-7)方程分解为

$$(x-x_1)(x-x_2)(x-x_3)=0$$

显然有

$$x_1+x_2+x_3=\left(\frac{y_2+y_1}{x_2+x_1}\right)^2+\frac{y_2+y_1}{x_2+x_1}+a_2$$

所以 $$x_3 = \left(\frac{y_2+y_1}{x_2+x_1}\right)^2 + \frac{y_2+y_1}{x_2+x_1} + x_2 + x_1 + a_2$$

$$y = mx + b$$

所以 $$y_3 = \left(\frac{y_2+y_1}{x_2+x_1}\right)x_3 + \left[y_1 + \frac{y_2+y_1}{x_2+x_1}x_1\right] + x_3$$

$$= \left(\frac{y_2+y_1}{x_2+x_1}\right)(x_3+x_1) + y_1 + x_3$$

$$= \frac{y_2+y_1}{x_2+x_1}(x_3+x_1) + y_1 + x_3, \quad P \neq Q \tag{8-8}$$

若 $$P=Q:$$

$$y^2 + xy = x^3 + a_2x^2 + a_6 \tag{8-9}$$

等号两端对 x 求导得

$$(2y+x)\frac{dy}{dx} + y = 3x^2 + 2a_2x$$

所以 $$x_1\left.\frac{dy}{dx}\right|_P + y_1 = x_1^2$$

$$m = \left.\frac{dy}{dx}\right|_P = \frac{x^2+y}{x}$$

令 $y=mx+b$，代入方程(8-9)

$$b = y_1 + mx_1$$

$$(mx+b)^2 + x(mx+b) = x^3 + a_2x^2 + a_6$$

x^2 项系数：a_2+m^2+m

所以 $$2x_1 + x_3 = a_2 + m^2 + m$$

$$x_3 = m^2 + m + a_2$$

$$y = mx + (y_1 + mx_1)$$

$$y_3 = mx_3 + y_1 + \left(x_1 + \frac{y_1}{x_1}\right)x_1 + x_3$$

$$= mx_3 + x^2 = \frac{x^2+y}{x}x_3 + x^2 + x_3$$

8.4.4 $j(E)=0$ 的讨论

令 $P=(x_1,y_1)\in E_2$，$-P=(x_1,y_1+a_3)$，若 $Q=(x_2,y_2)\in E_2$，$Q\neq -P$，

$$P+Q=(x_3,y_3)$$

则有

$$x_3 = \begin{cases} \left(\frac{y_1+y_2}{x_1+x_2}\right)^2 + x_1 + x_2, & P \neq Q \\ \frac{x_1^4+a_4^2}{a_3^2}, & P = Q \end{cases}$$

$$y_3 = \begin{cases} \left(\dfrac{y_1+y_2}{x_1+x_2}\right)(x_1+x_3)+y_1+a_3, & P \neq Q \\ \left(\dfrac{x_1^2+a_4}{a_3}\right)(x_1+x_3)+y_1+a_3, & P = Q \end{cases}$$

从式(8-4)可知 $\bar{a}_1=0$

$$y^2+a_3y = x^3+a_4x+a_6 \tag{8-10}$$

设　过 P,Q 的直线斜率为

$$m = \frac{y_2+y_1}{x_2+x_1}$$

直线方程：　　$y=mx+(y_1+mx_1)$

代入式(8-10)得

$$[mx+(y_1+m_1x_1)]^2+a_3[mx+(y_1+mx_1)] = x^3+a_4x+a_6$$

x^2 项系数：m^2，设上式可化为

$$(x-x_1)(x-x_2)(x-x_3) = 0$$

显然有：

$$x_1+x_2+x_3 = a_2+m^2$$

$$x_3 = \left(\frac{y_2+y_1}{x_2+x_1}\right)^2+(x_1+x_2)$$

$$y_3 = \left(\frac{y_1+y_2}{x_1+x_2}\right)x_3+\left(\frac{y_1+y_2}{x_1+x_2}\right)x_1+y_1+a_3$$

$$= \left(\frac{y_1+y_2}{x_1+x_2}\right)(x_1+x_3)+y_1+a_3$$

类似可讨论 $P=Q=(x,y)$ 的情况。

有关结果如下：

(1) 椭圆曲线的标准形式及对应的判别式 Δ 和不变数 j

① $Char(F)=2,3$。

$$y^2 = x^3+a_4x+a_6$$

$$\Delta = -16(4a_4^3+27a_6^2), \quad j = 1728\frac{4a_4^3}{4a_4^3+27a_6^2}$$

② $Char(F)=3, j\neq 0$。

ⓐ $y^2=x^3+a_2x^2+a_6$

$$\Delta = -a_2^3a_6, \quad j=\frac{a_2^3}{a_6}$$

ⓑ $y^2=x^3+a_4x+a_6$，　$j=0$

③ $Char(F)=2$。

ⓐ $y^2+xy=x^3+a_2x^2+a_6$

$$\Delta = a_6, \quad j=\frac{1}{a_6}$$

ⓑ $y^2+a_3y=x^3+a_4x+a_6$

$$\Delta = a_3^4, \quad j = 0$$

(2) 椭圆曲线及对应的求和公式，$P=(x_1,y_1)$，$Q=(x_2,y_2)$

$$P+Q=(x_3,y_3)$$

ⓐ $y^2+a_1xy+a_3y=x^3+a_2x^2+a_4x+a_6$

$$x_3=\begin{cases}\left(\dfrac{y_2-y_1}{x_2-x_1}\right)^2+a_1\left(\dfrac{y_2-y_1}{x_2-x_1}\right)-a_2-x_1-x_2, & P\neq Q\\ \left(\dfrac{3x^2+2a_2x+a_4-a_1y}{2y+a_1x+a_3}\right)^2+a_1\left(\dfrac{3x^2+2a_2x+a_4-a_1y}{2y+a_1x+a_3}\right)-a_2-2x, & P=Q=(x,y)\end{cases}$$

$$y_3=\begin{cases}\dfrac{y_2-y_1}{x_2-x_1}(x_1-x_3)-y_1-(a_1x_3+a_3), & P\neq Q\\ \left(\dfrac{3x^2+2a_2x+a_4-a_1y}{2y+a_1x+a_3}\right)(x-x_3)-y-(a_1x_3+a_3), & P=Q\end{cases}$$

ⓑ $Char(F)=2$，$j\neq 0$，$y^2+xy=x^3+a_2x^2+a_6$，

$$x_3=\begin{cases}\left(\dfrac{y_1+y_2}{x_1+x_2}\right)^2+\dfrac{y_1+y_2}{x_1+x_2}+a_2+x_1+x_2, & P\neq Q\\ x^2+\dfrac{a_6}{x^2}, & P=Q=(x,y)\end{cases}$$

$$y_3=\begin{cases}\dfrac{y_1+y_2}{x_1+x_2}(x_1+x_3)+y_1+x_3, & P\neq Q\\ \dfrac{x^2+y}{x}x_3+x^2+x_3, & P=Q\end{cases}$$

ⓒ $Char(F)=2$，$j=0$，$y^2+a_3y=x^3+a_4x+a_6$

$$x_3=\begin{cases}\left(\dfrac{y_1+y_2}{x_1+x_2}\right)^2+x_1+x_2, & P\neq Q\\ \left(\dfrac{x^2+a_4}{a_3}\right)^2, & P=Q=(x,y)\end{cases}$$

$$y_3=\begin{cases}\dfrac{y_1+y_2}{x_1+x_2}(x_1+x_3)+y_1+a_3, & P\neq Q\\ \dfrac{x^2+a_4}{a_3}(x+x_3)+y+a_3, & P=Q\end{cases}$$

8.5 Hasse 定 理

先看一个例子，在 F_5 上的椭圆曲线：

$$y^2=x^3+x+1 \tag{8-11}$$

令 $x=0,1,2,3,4$ 代入式(8-11)，得 9 个点：

$$E(F_5):\{\theta,(0,\pm 1),(2,\pm 1),(3,\pm 1),(4,\pm 2)\}$$

故 $E(F_5)$构成 9 个元素的 Abel 群。它是一循环群，或是两个阶为 3 的群的乘积。令 $P=(0,1)$，可得 $2P=(4,2)$，$3P=(2,1)$，$4P=(3,-1)$，…所以 $E(F_5)$是阶等于 9 的循环群。两个阶为 3 的点是$(2,\pm 1)$。其他所有非零的点都是阶等于 9。现论证如下：

$P=(0,1)$，对式(8·11)等号两端对 x 求导得

$$2y\frac{dy}{dx}=3x^2+1$$

$$m=\left.\frac{dy}{dx}\right|_P=\left.\frac{3x^2+1}{2y}\right|_{(0,1)}=\frac{1}{2}$$

P 点的切线方程：

$$y=\frac{1}{2}x+b$$

在 F_5 域上 $2^{-1}=3$，故 $y=3x+1$

$$(3x+1)^2=x^3+x+1$$

x^2 项系数：

$$-9\equiv 1 \bmod 5$$

设 $y=3x+1$ 交 E 于(ξ,η)，则

$$\xi=-1\equiv 4 \bmod 5$$

$$\eta=3 \bmod 5$$

故

$$2P=(4,-3)=(4,2)$$

$2P=(4,2)$和$(0,1)$的联线，斜率 m 为

$$m=\frac{2-1}{4-0}=\frac{1}{4}$$

$$4^{-1}=4$$

连线方程：$y=4x+b$，过$(0,1)$故 $b=1$。

连线方程：$y=4x+1$，交 $y^2=x^3+x+1$ 于(ξ,η)点

$$(4x+1)^2=x^3+x+1$$

x^2 项系数：

$$-16\equiv 4 \bmod 5$$

若

$$x^3-(4x+1)^2+x+1=x(x-4)(x-x_3)$$

则

$$4+\xi=-4,\xi=-8\equiv 2 \bmod 5$$

$$\eta=4x_3+1=9\equiv 4 \bmod 5$$

故 $3P=(2,1)$。

$2P=(4,2)$点的切线斜率：

$$m=\left.\frac{3x^2+1}{2y}\right|_{(4,2)}=\frac{49}{4}\equiv\frac{4}{4}=1$$

$(4,2)$点切线方程：

$$y=x+3$$

$$(x+3)^2=x^3+x+1$$

x^2 项系数：-1，故$(4,2)$点切线交 $y^2=x^3+x+1$ 于(ξ,η)有

$$2\cdot 4+x_3=1$$

所以

$$\xi=-7=3$$

$$\eta=1$$

所以

$$4P=(3,-1)$$

同理：$4P+P=5P=(3,-4)=(3,1)$

再以 $3P=(2,1)$ 为例。

过(2,1)的切线斜率：在 $GF(5)$ 上有

$$m=\left.\frac{3x^2+1}{2y}\right|_{(2,1)}=\frac{13}{2}=\frac{3}{2}=2^{-1}3=9=4$$

切线方程：

$$y=4x+b$$

过(2,1)点，故

$$b=1-8=-7=3$$

即

$$y=4x+3$$

$(4x+3)^2=x^3+x+1$，交点设为(ξ,η)。

$$x^3-(4x+3)^2+x+1=(x-2)^2(x-\xi)=0$$

所以

$$16=2\cdot 2+\xi,\ \xi=12\equiv 2 \bmod 5$$

$$\eta=11\equiv 1$$

故

$$6P=(2,-1)$$

$$3P+6P=9P=O$$

如本例所示，在 F_p 域上 $E(P)$ 的点是有限的。

定理 8.5(Hasse,Weil) 若 E 是在 $GF(p)$ 上定义的非奇异不可化约的椭圆曲线，则 E 上坐标在 $GF(p)$ 的点数 $E(F_p)$ 有

$$E(F_p)=P+1+\varepsilon$$

对于椭圆曲线 E，有

$$-2\sqrt{P}\leqslant E(F_p)-p-1\leqslant 2\sqrt{P}$$

证明从略。

由前面的讨论可见：椭圆曲线上的点对所定义的加法运算构成阿贝尔(Abel)群。

8.6 椭圆曲线上的公钥密码

若我们能找到一椭圆曲线 E，将明文通过编码嵌入到 E 的点，然后在 E 上进行加密。加密变换也是一种编码。但从明文到椭圆曲线 E 上的点的编码结果不是密码。假定嵌入变换已解决，下面讨论在椭圆曲线上的加密，它实际上是将熟知的加密运算移植到椭圆曲线上。

1. Diffie-Hellman 公钥系统

已知 q 及 $GF(q)$ 是大家共同确定的。用户 A 任选一整数 a，$1\leqslant a\leqslant q-1$，将 a 保密，将 $g^a\in GF(q)$ 公开。g 是 $GF(q)$ 上一本原元素。用户 B 任选一整数 b，$1\leqslant b\leqslant q-1$ 予以保密，将 $g^b\in GF(q)$ 公开。如 A 和 B 秘密通信，他们间的通信密钥为 g^{ab}：

$$g^{ab}=(g^a)^b=(g^b)^a$$

而第三者只能知道 g^a 和 g^b 无法得知 g^{ab}，从 g^a 或 g^b 要计算 a 或 b，这是离散对数问题，是为人熟知的困难问题。

下面介绍如何在椭圆曲线上实现 Diffie-Hellman 的体制。假定椭圆曲线 E 定义在 $GF(q)$域上。

设 $P\in E$,要求由 P 产生的群的元素足够多,P 的作用相当于上述的 g。A 和 B 分别选 a 和 b 予以保密,但将 $aP,bP\in E$ 公开。A,B 间通信用的密钥为 abP,这是第三者无法得知的。

2. Massey-Omura 公钥体制

已知有限域 $GF(q)$,用户 A 选一加密密钥 e_A,$0\leqslant e_A\leqslant N$,满足$(e_A,q-1)=1$,通过欧几里得算法求得 d_A 满足:

$$e_A d_A \equiv 1(\mathrm{mod}(q-1))$$

根据类似的理由,若用户 B 有 $e_B d_B\equiv 1(\mathrm{mod}(q-1))$而且$(e_B,q-1)=1$,若 A 欲向 B 送去信息 m,则 A 送去 m^{e_A},B 无法获得 m,因他既不知道 e_A,也不知道 d_A,B 退还 A 以 $m^{e_A e_B}$。A 收到 $m^{e_A e_B}=c$ 后,作 $c^{d_A}=m^{e_B}$ 并送给 B,B 对此结果作$(m^{e_B})^{d_B}=m^{e_B d_B}\equiv m(\mathrm{mod}\ q)$,从而获得明文 m。

现在来讨论 Massey-Omura 密码体系在椭圆曲线上的实现。

假定 m 嵌入到 E 上的 P_m 点。设 E 上的点数 N 为已知的大素数,每一个用户随机选择一数 e,$1<e<N$,$(e,N)=1$,d 是 e 的逆,即 $de\equiv 1(\mathrm{mod}\ N)$。假如 A 要送 B 信息 m,A 首先送去 $e_A P_m$,这里 e_A 表示属于用户 A 的 e。B 退还给 A 以 $e_B e_A P_m$,A 再送去 $d_A e_B e_A P_m=e_B P_m$。B 乘以 d_B 得 $d_B e_B P_m=P_m$,从而获得了 P_m。

3. ElGamal 密码系统

假定在一个有限域 $GF(q)$上讨论。设 $g\in GF(q)$,g 为非 0 元素。每一用户随机地选择一数 α,$0<\alpha<q$,α 保密,而将 g^α 公开,若要向 A 送去信息 m,可随机产生一整数 k,送给 A 以下列一对数:

$$(g^k,\ mg^{\alpha_A k})$$

由于 $g^{\alpha_A k}=(g^{\alpha_A})^k$,所以虽然不知道 α_A,也可以求得 $g^{\alpha_A k}$,但 A 掌握 α_A,他可从 g^k 这个元素得知 $g^{\alpha_A k}$,并用 $g^{\alpha_A k}$ 去"除"第二个数 $mg^{\alpha_A k}$ 以恢复 m。这里"除"的含意应理解为用 $g^{\alpha_A k}$ 的逆元素来乘 $mg^{\alpha_A k}$。

同样可在椭圆曲线上实现这个系统,假定明文 m 嵌入到 E 上 P_m 点。选一点 $B\in E$,它相当于上述的元素 g,每用户都选择一数 α,$0<\alpha<N$,N 是已知数,α 保密,但将 αB 公开。欲向 A 送 m,可送去下面一对数偶:

$$(kB,\ P_m+k(\alpha_A B))$$

k 是随机产生的整数。A 可从 kB 求得 $k\alpha_A B$。通过:

$$P_m+k(\alpha_A B)-\alpha_A kB=P_m$$

恢复 P_m。

8.7 因数分解的 Lenstra 算法

因数分解问题由于密码学的研究而重新被重视。若大数的因数分解获成功,则 RSA 等密码系统便宣告失败。下面介绍椭圆曲线在这方面的应用,它属于 H. W. Lenstra 的

工作。

1. Pollard 的 $p-1$ 方法

波拉德(Pollard)方法假定 n 是合数，p 是 n 的素数因子，但 $p-1$ 没有大的因数，方法如下：

S1. 求一整数 k，它可被小于某一整数 b 的所有素数除尽，可以取 $k=b!$，也可取 $k=\mathrm{lcm}\{1,2,\cdots,b\}$；

S2. 选取整数 a，$2\leqslant a\leqslant n-2$；

S3. 计算 $a^k \bmod n$；

S4. 利用 S3 的结果计算 $\alpha=\gcd\{a^k-1,n\}$；

S5. 若 α 为 2 或 n 本身，则转 S2，选新的 a 重新开始。

根据假定 $p-1$ 没有大的因数，可被小于 b 的某正整数除尽，因而 k 是 $p-1$ 的倍数。由 Fermat 定理：$a^{p-1}\equiv 1(\bmod p)$。所以 $a^k\equiv 1(\bmod p)$，因此 p 可以除尽 $\gcd(a^k-1,n)$，当 $a^k\equiv 1(\bmod n)$时方法失败。

例 8.3 设 $n=4731$，取 $b=5$，$k=\mathrm{lcm}\{1,2,3,4,5\}=60$，$a=2$

$$2^{60}(\bmod 4731)\equiv 2^{4\times 3\times 5}(\bmod 4731)$$
$$\equiv (4096)^5(\bmod 4731)$$
$$\equiv 4096\cdot(4096^2)^2(\bmod 4731)$$

因为
$$4096^2\equiv 1090(\bmod 4731)$$

所以
$$2^{60}(\bmod 4731)\equiv 4096\times(1090)^2(\bmod 4731)$$
$$\equiv 4096\times 619(\bmod 4731)\equiv 4339(\bmod 4731)$$

又
$$4731=4338+393$$
$$4338=11\times 393+15$$
$$393=26\times 15+3$$
$$15=5\times 3$$

所以
$$4731=3\times 1577$$

2. mod n 导出的椭圆曲线

定义 8.1 假定 p 是 n 的素因子，且 $p>3$，m 是任一整数，若 x_1，x_2 是任意两个有理数，其分母与 m 互素，但 x_1-x_2 化约到最后，分子为被 m 除尽的分数时，写作 $x_1\equiv x_2(\bmod m)$。

以上对于两个有理数 x_1 和 x_2，$x_1=x_2(\bmod m)$是同余式的概念的拓广。可证若 x_1 是一分母与 m 互素的分数，则存在惟一的整数 x_2，$0\leqslant x_2\leqslant m-1$，使得 $x_1\equiv x_2(\bmod m)$，并记 $x_2\equiv(x_1 \bmod m)$。

若
$$x_1=\frac{a}{b},\quad (b,m)=1$$

则
$$x_1-x_2=\frac{a-bx_2}{b}$$
$$bx_2\equiv a(\bmod m) \tag{8-12}$$

在$(b,m)=1$时有惟一解x，$0\leqslant x_2\leqslant m-1$。

现在将上述的同余概念用于椭圆曲线E。

定义 8.2 设$P(x,y)\in E$，定义

$$P(\bmod m)\triangleq(x(\bmod m),\ y(\bmod m)) \tag{8-13}$$

则$P(\bmod m)$为$E(\bmod m)$椭圆曲线上的点。

前面关于椭圆曲线的讨论均可推广到$E(\bmod m)$上来，只不过它的每一项都理解为mod m意义下的同余。比如$y^2=x^3+ax+b$，其中a和b都理解为$a(\bmod m)$和$b(\bmod m)$。特别是$P_1(\bmod m)+P_2(\bmod m)$也就是根据公式(8-13)，不过其中每一项都是mod m的同余；分母(x_1-x_2)理解为(x_1-x_2)的逆，即$(x_1-x_2)^{-1}$使$(x_1-x_2)\cdot(x_1-x_2)^{-1}\equiv 1(\bmod m)$。尤其是$E(\bmod p)$上的无穷远点$0(\bmod m)$表示椭圆曲线$E$上点坐标的分母有$m$的因子的点。

定理 8.6 令$E:y^2=x^3+ax+b$，$a,b\in Z$，$gcd\{4a^3+27b^2,n\}=1$。P_1和P_2是E上两点，且$P_2\neq -P_1$，它们坐标的分母与n互素，则$P_1+P_2\in E$的坐标其分母与n互素的充要条件是：不存在素数$p\mid n$，使得曲线$E(\bmod p)$上$P_1(\bmod p)$和$P_2(\bmod p)$之和为$E(\bmod p)$上的无穷远点$0(\bmod p)$，其中$E(\bmod p)$为$GF(p)$域上的椭圆曲线。$0(\bmod p)$为$GF(p)$域上的椭圆曲线的无穷远点(即指的是$(x,y)\in E$上坐标的分母有p因子的点)。

证明 必要性。即已知$P_1=(x_1,y_1)$，$P_2=(x_2,y_2)$，$P_1+P_2=(x_3,y_3)\in E$，它们的坐标的分母都和n互素，p是n的素因子，则

$$P_1(\bmod p)+P_2(\bmod p)\neq 0(\bmod p)$$

分$x_1\not\equiv x_2(\bmod p)$和$x_1\equiv x_2(\bmod p)$两种情况进行讨论。先讨论第一种情况。

(a) $x_1\not\equiv x_2(\bmod p)$，即$P_1\neq P_2$。根据

$$x_3=\left(\frac{y_2-y_1}{x_2-x_1}\right)^2-x_1-x_2,\quad y_3=-y_1+\left(\frac{y_2-y_1}{x_2-x_1}\right)(x_1-x_3) \tag{8-14}$$

显然$P_1(\bmod p)+P_2(\bmod p)\neq 0(\bmod p)$成立。

(b) $x_1\equiv x_2(\bmod p)$，又分$P_1=P_2$和$P_1\neq P_2$两种可能，若$P_1=P_2$，则$P_1+P_2=2P_1=(x_3,y_3)$，这时

$$x_3=\left(\frac{3x_1^2+a}{2y_1}\right)^2-2x_1,\quad y_3=-y_1+\left(\frac{3x_1^2+a}{2y_1}\right)(x_3-x_1) \tag{8-15}$$

其分母$2y_1$不被p除尽。如若不然，必有$3x_1^2+a$被p除尽，这说明函数x^3+ax+b和它的导数有相同的零点$x=x_1$，因$4a^3+27b^2\not\equiv 0(\bmod p)$，这和mod p无重根的假定相矛盾。这也就证明了$P_1(\bmod p)+P_2(\bmod p)\neq 0(\bmod p)$。最后假定$x_2\equiv x_1(\bmod p)$，但$P_1\neq P_2$，且$x_2\neq x_1$，令$x_2=x_1+p^r x$，$x$的分母和分子均无$p$的因子，$r\geqslant 1$，根据$P_1+P_2$的坐标的分母不含$p$的因子的假设。

$$x_3=\left(\frac{y_2-y_1}{x_2-x_1}\right)^2-x_1-x_2,\quad y_3=-y_1+\left(\frac{y_2-y_1}{x_2-x_1}\right)(x_1-x_3)$$

故$y_2-y_1=p^r y$必然成立，这就证明了

$$y_2\equiv y_1(\bmod p)\ \text{或}\ P_1(\bmod p)=P_2(\bmod p)$$

另一方面

$$
\begin{aligned}
y_2^2 &= (x_1 + p^r x)^3 + a(x_1 + p^r x) + b \\
&= x_1^3 + 3x_1^2 p^r x + 3x_1 p^{2r} x^2 + p^{3r} x^3 + ax_1 + ap^r x + b \\
&\equiv x_1^3 + 3p^r x_1^2 x + ax_1 + ap^r x + b (\bmod p^{r+1}) \\
&= x_1^3 + ax_1 - b + p^r x(3x_1^2 + a) (\bmod p^{r+1}) \\
&= y_1^2 + p^r x(3x_1^2 + a) (\bmod p^{r+1})
\end{aligned} \tag{8-16}
$$

由于 $P_1 \neq P_2$，但 $x_1 \equiv x_2 (\bmod p)$，$y_2 \equiv y_1 (\bmod p)$，所以

$$P_1 (\bmod p) + P_2 (\bmod p) = 2P_1 (\bmod p)$$

$2P_1 (\bmod p)$为 $0 (\bmod p)$的充要条件是 $y_1 \equiv y_2 \equiv 0 (\bmod p)$，这便有：$(y_2^2 - y_1^2) = (y_2 - y_1)(y_2 + y_1)$的分子可被 p^{r+1}除尽，导致

$$3x_1^2 + a \equiv 0 (\bmod p)$$

这是不可能的。因多项式 $x^3 + ax + b (\bmod p)$没有重根。这就证明了必要条件。

充分性　假定 n 的所有素因子 p 有 $P_1 (\bmod p) - P_2 (\bmod p) \neq 0 (\bmod p)$。证明 $P_1 + P_2$ 点坐标的分母与 n 互素，也就是它们坐标的分母不被 n 的任意因子 p 除尽。

(1) 若 $x_2 \neq x_1 (\bmod p)$，根据公式(8-13)可知 x_3，y_3 的分母不被 p 除尽，这时定理自然成立。

(2) 若 $x_2 \equiv x_1 (\bmod p)$，因 $y^2 = x^3 + ax + b$。故 $y_2 \equiv \pm y_1 (\bmod p)$，但 $P_1 (\bmod p) + P_2 (\bmod p) \neq 0 (\bmod p)$，故

$$y_2 \equiv y_1 \not\equiv 0 (\bmod p)$$

在 $x_1 \equiv x_2 (\bmod p)$，$y_1 \equiv y_2 (\bmod p)$前提下，有两种可能，$P_1 = P_2$ 或 $P_1 \neq P_2$。

若 $P_1 = P_2$，$P_1 + P_2 = 2P_2$，从公式(8-15)，这时分母不存在 p 的因子，若 $P_1 \neq P_2$，令 $x_2 = x_1 + p^r x$，$r \geqslant 1$，x 的分母不含 p 的因子。

所以
$$x_2 - x_1 = p^r x$$

$$\frac{y_2^2 - y_1^2}{x_2 - x_1} \equiv 3x_1^2 + a (\bmod p)$$

所以
$$y_1 + y_2 \equiv 2y_1 (\bmod p)$$

$\dfrac{y_2^2 - y_1^2}{(y_2 + y_1)(x_2 - x_1)} = \dfrac{y_2 - y_1}{x_2 - x_1}$的分母不存在 p 的因子。公式(8-14)的分母不存在 p 的因子，也就是说 $P_1 + P_2$ 的坐标的分母不含 p 的因子。证毕。

定理 8.7　是任斯特拉(Lenstra)因数分解的基础。给定一合数 n，要找一素数 p，使得 $p \mid n$，$1 < p < n$。

(a) 首先要随机地生成一椭圆曲线 $E: y^2 = x^3 + ax + b$ 及 E 上的点 $P(x, y)$，解法是先随机产生 $P(x, y)$点，及整数 a，$b = y^2 - x^3 - ax$。这样，曲线 E 及曲线 E 上的 P 点便产生了。再检验 $4a^3 + 27b^2 = 0$ 否？以保证方程 $x^3 + ax + b = 0$ 没有重根。若 $4a^3 + 27b^2 = 0$，则改变 a，重复以上过程。

(b) 给出界 B 和 C，计算

$$k = \prod_{p_i \leqslant B} p_i^{a_i}, \quad p_i^{a_i} \leqslant C, \quad \forall p_i \leqslant B$$

(c) 计算 $kP (\bmod n)$，根据公式(8-14)，(8-15)，x_3 和 y_3 的分母分别为 $x_2 - x_1$ 及

$2y_1$。求(x_0-x_1)和$2y_1 \pmod n$的逆时，若发生困难，正好说明分母含有n的某一因数，这正是我们需要的。转而求它和n的最大公因数。根据定理 8.6，P_1-P_2的坐标含有n的因数p的分母的充要条件是

$$P_1 \pmod p + P_2 \pmod p = 0 \pmod p, \quad p \mid n$$

算法是在$E \pmod n$上进行的。实际上也是在$E \pmod p$上进行的，p是n的任意因数。

或将 Lenstra 椭圆曲线算法归结如下：

设$n \geqslant 2$是一合数，

S1. 在$1 \sim n$间随机选取整数x_1, y_1, b；

S2. 令曲线E：

$$y^2 = x^3 + bx + c$$

$$P = (x_1, y_1) \in E, \text{其中 } c = y_1^2 - x_1^3 - bx_1$$

S3. 计算$d = \gcd(4b^3 + 27c^2, n)$，若$d = n$则转 S1 重新选择$b$；若$d$是$1 \sim n$间的数，则它是$n$的因数；

S4. 选择k是小素小幂次的乘积，例如

$$k = \operatorname{lcm}[1, 2, \cdots, h]$$

h是一整数；

S5. 计算

$$kP = \left(\frac{a_k}{d_k^2}, \frac{b_k}{d_k^3}\right)$$

S6. 计算$g = \gcd(d_k, n)$，若$1 < g < n$，则g是n的素数因数。若$g = 1$，即d_k和n互素，则转 S4，提高k，也可以转 S_3 选择新的椭圆曲线。若$g = n$，则转 S_4 降低k。

在计算kP的过程中存在一个k_1，使得$k_1 P \pmod p = 0 \pmod p$。也就是$k_1$为$P \pmod p$点阶的倍数，在计算过程中求 mod n 某分母的逆时，代以求n和分母的最大公因数，这个最大公因数，除非就是n本身，一般说来是n的因数。所以在计算$kP \pmod n$时，对某一$p \mid n$，k是$P \pmod p$的阶时，我们将获得一个n的真因子。

任斯特拉方法当所选的E和P不成功时，换上新的一组重新开始，若失败的概率为λ，则连续l次均失败的概率为λ^l，故连续l次失败的概率很小。

算法中涉及到选两个整数界B和C，B和C越大当然使得找到$kP \pmod p = 0 \pmod p$的概率增加，然而计算的时间也越长。

例 8.4 已知$n = 5429$，试通过椭圆曲线因数分解n。

设$B = 3$，$C = 92$，取$E: y^2 = x^3 + 2x - 2$。令$2P = (\xi_1, \eta_1)$，根据公式(8·4)，可得

$$\xi_1 = \left(\frac{3+2}{2}\right)^2 - 2 = \left(\frac{5}{2}\right)^2 - 2 = \frac{17}{4}$$

$$\eta_1 = -1 + \left(\frac{5}{2}\right)\left(1 - \frac{17}{4}\right) = -\frac{73}{8}$$

即

$$2P = \left(\frac{17}{4}, -\frac{73}{8}\right)$$

令 $$\frac{17}{4} \equiv u(\bmod 5429),\ 4u \equiv 17(\bmod 5429)$$

$$u \equiv 4076(\bmod 5429)$$

令 $$-\frac{73}{8} \equiv v(\bmod 5429),\ 8v \equiv -73(\bmod 5429)$$

$$v \equiv 3384(\bmod 5429)$$

故 $$2P(\bmod 5429) = (4076, 3384)$$

当然也可以用下面的步骤计算 $2P(\bmod 5429)$

$$u \equiv \left(\frac{3+2}{2}\right)^2 - 2 \equiv \left(\frac{5}{2}\right)^2 - 2(\bmod 5429)$$

但 $\frac{1}{2}$ 理解为 $2^{-1}(\bmod 5429)$，即上面的计算在 $E(\bmod 5429)$ 上进行。$2^{-1}(\bmod 5429) = 2715$，所以

$$\begin{aligned} u &\equiv (2715\times 5)^2 - 2 \equiv (2717)^2 - 2 \\ &\equiv 4078(\bmod 5429) \end{aligned}$$

同理

$$\begin{aligned} v &\equiv -1 + \left(\frac{5}{2}\right)\left(\frac{-13}{4}\right) \\ &\equiv -1 + (2715\times 5)(-13\times 4072) \\ &\equiv -1 + (2717)(1354)(\bmod 5429) \\ &\equiv 3384(\bmod 5429) \end{aligned}$$

即 $2P(\bmod 5429) = (4076, 3384)$，结果是一样的。这里 $4^{-1}(\bmod 5429) = 4072$。

依类似的算法计算 $4P(\bmod 5429)$，令 $2^2P(\bmod 5429) = (u_2, v_2)$。下面计算假定在 $GF(p)$ 上进行，其中 $P|5429$。

$$u_2 \equiv \left(\frac{3\times 4076+2}{6768}\right)^2 - 8152(\bmod 5429)$$

$$\equiv \left(\frac{1549+2}{1339}\right)^2 - 2723(\bmod 5429)$$

$$1339^{-1} \equiv 2826(\bmod 5429)$$

所以

$$\begin{aligned} u_2 &\equiv (1551\times 2826)^2 - 2723(\bmod 5429) \\ &\equiv (1923)^2 - 2723(\bmod 5429) \\ &\equiv 780 - 2723(\bmod 5429) \\ &\equiv -1943(\bmod 5429) \\ &\equiv 3486(\bmod 5429) \end{aligned}$$

$$\begin{aligned} v_2 &\equiv -3384 + (1551\times 2826)(4076-3486)(\bmod 5429) \\ &\equiv -3384 + 1923\times 590(\bmod 5429) \\ &\equiv -3384 + 5338(\bmod 5429) \\ &\equiv 1954(\bmod 5429) \end{aligned}$$

所以 $$4P(\text{mod } 5429)=(3486,1954)$$

类似步骤计算 $2^{\alpha}3^{\beta}P(\text{mod } 5429)$。

当 $\alpha=6,\beta=2$ 时，发现分母与 5429 有公因子 61，$5429=61\times89$。

附带说明取 $C=92$ 的依据：$n=5429$，$\sqrt{5429}=73.68$，即 n 的因数 $p<\sqrt{n}=73.68$，定义在 $GF(p)$ 上的椭圆曲线 E 上的属于 $GF(p)$ 的点的数目 N 应满足

$$N\leqslant p+1+2\sqrt{p}=74+2\sqrt{73}<74+2\sqrt{81}=92$$

取 $C=92$ 是出于 $C>N$ 的考虑。

例 8.5 $n=17\ 15\ 76\ 15\ 13$

$$2^{17\ 15\ 76\ 15\ 14}\equiv 93082891 \bmod 17\ 15\ 76\ 15\ 13$$

根据 Fermat 定理 $2^{n-1}\not\equiv 1 \bmod n$，可知 n 不是素数。

由 $\sqrt{17\ 15\ 76\ 15\ 13}\approx 42422$ 可知 n 有小于 42422 的素数因子 p。

选 E：

$$y^2=x^3+x-9$$

$$P=(2,1)\in E$$

令 k 为

$$\text{lcm}[1,2,3,\cdots,17]=12252240$$

$$k=12252240=(101110101111110001010000)_2$$

$$=2^4+2^6+2^{10}+2^{12}+2^{13}+2^{14}+2^{15}+2^{17}+2^{19}+2^{20}+2^{21}+2^{23}$$

计算 $kP(\text{mod } n)$：

$$P=(2,1)$$

$$2y\frac{dy}{dx}=3x^2+1$$

所以 $$\left.\frac{dy}{dx}\right|_P=\left.\frac{3x^2+1}{2y}\right|_{(2,1)}=\frac{13}{2}$$

过(2,1)点的切线：

$$y-1=\frac{13}{2}(x-2),\ y=\frac{13}{2}(x-2)+1$$

$$y^2=\left[\frac{13}{2}x-\frac{11}{2}\right]^2=x^3+x-9$$

或

$$x^3-\frac{169}{4}x^2+\left(\frac{143}{2}+1\right)x-\left(9+\frac{121}{4}\right)=0$$

但 $$1=1715761513\quad 4(428940378)$$

或 $$4(-428940378)\equiv 1 \bmod n$$

$$4^{-1}\equiv -428940378\equiv 1286821135 \bmod 1715761513$$

同理

$$2^{-1}\equiv -857880756\equiv 85788075 \bmod 1715761513$$

所以过(2,1)点切线交 E 于(ξ,η)。

则 $2\cdot 2+\xi=1286821135\times 169$

$$\xi=217472771815-4$$

$$=217472771811\equiv 1286821173 \bmod 1715761513$$

$$\eta=1672350709$$

即 $2P-(1286821173,1072350709)$

同理可得

$$2^2P=(1334478523,112522703)$$

$$2^3P=(912789305,77695868)$$

$$2^4P=(385062894,618628731)$$

$$2^5P=(866358838,450284374)$$

$$2^6P=(904716938,169383608)$$

$$2^7P=(808696477,1201030016)$$

$$2^8P=(572301268,107111567)$$

$$2^9P=(1512647092,1695275444)$$

$$2^{10}P=(1858186,1224662922)$$

$$2^{11}P=(1550404618,825515387)$$

$$2^{12}P=(1519325194,1657497846)$$

$$2^{13}P=(522917322,524407354)$$

$$2^{14}P=(25207285,1375034461)$$

$$2^{15}P=(781360494,1457273929)$$

$$2^{16}P=(1108412304,25813532)$$

$$2^{17}P=(435914774,323718902)$$

$$2^{18}P=(1399483199,1203611423)$$

$$2^{19}P=(778823593,192206539)$$

$$2^{20}P=(853199887,1012680972)$$

$$2^{21}P=(501929966,910060788)$$

$$2^{22}P=(1315182921,305331854)$$

$$2^{23}P=(257200250,318342966)$$

$$2^4P=2^{14}P=(385062894,618628731)$$

$$2^4P+2^6P=16P+64P=80P$$

过(385062894,618628731)和(904716938,169383608)点引直线

$$y=mx+b$$

其中

$$m=\frac{618628731-169383608}{385062894-904716938}=\frac{-449245123}{519654044}$$

过 2^4P 点。故得

$$b = y - mx$$

$$= 618628731 - \frac{449245123}{519654044}385062894$$

$$= 618628731 - \frac{172987627177765962}{519654044}$$

$$172987627177765962 \equiv 884611387 \bmod 1715761513$$

所以 $$b=618628731-\frac{884611387}{519654044}$$

可证 $\gcd(884611387,519654044)=1$

令

$$\frac{884611387}{519654044}\equiv n$$

$$519654044n \equiv 884611387$$

求 $519654044^{-1} \bmod n$

$$n=519654044^{-1} \cdot 884611387$$

$$b\equiv 618628731-n$$

若 $y=mx+b$ 与曲线 E 交于(α,β)，则

$$\alpha + 385062894 + 9047169 + 38 \equiv 3m^2 b$$

$$S_2 = (2^4 + 2^6)P = 80P = (\alpha, -\beta)$$

$$S_2 = (2^4 + 2^6)P = 80P = (831572269,1524749605)$$

计算过程从略。

$$S_3 = S_2 + 2^{10}P = 1104P = (1372980126,1361189875)$$

$$S_4 = S_3 + 2^{12}P = 5200P = (303639172,374618943)$$

$$S_5 = S_4 + 2^{13}P = 13392P = (243887465,4582074)$$

$$S_6 = S_5 + 2^{14}P = 29776P = (317266292,18428261)$$

$$S_7 = S_6 + 2^{15}P = 62544P = (425351528,95871325)$$

$$S_8 = S_7 + 2^{17}P = 193616P = (845805016,877478209)$$

$$S_9 = S_8 + 2^{19}P = 717904P = (1070680397,744910034)$$

$$S_{10} = S_9 + 2^{20}P = 1766480P = (543241897,1394639550)$$

$$S_{11} = S_{10} + 2^{21}P = 3863632P = (331651823,1280931959)$$

$$S_{12} = S_{11} + 2^{23}P = 12252240P = (421401044,664333727)$$

即

$$kP = 12252240P = (421401044,664333727) \bmod 1715761513$$

kP 没能对 n 的素因子提供任何信息，搜索 $b=3,4,\cdots,253$，都未有任何结果。$b=254$ 时，如前面例子所示的方法，可得线 E：$y^2=x^3+254x-515$，

$$(2^4 + 2^6 + 2^{10} + \cdots + 2^{20} + 2^{21})P = 3863632P$$

$$= (390104967,128395638) \bmod n$$

$$2^{23}P = (520835552, 974225979) \bmod n$$

为了求得 kP，必须求 (390104967,128395638)和(520835552,974225979)的连线和曲线 E 的交点。

$$390104967 - 520835552 = -130730585 \equiv 1585030928 \bmod n$$

$$\gcd\{1585030928, 1715761513\} = 26927$$

$$n = 1715761513 = 26927 \times 63719$$

$$1585030928 = 26927 \times 58864$$

3. Pollard 因子分解 ρ 算法

定义 8.3 设 $a \in Z_n \setminus \{0\}$，$n$ 是奇合数二次同类方程：$x^2 \equiv c \bmod n$ 的解 $x = \pm a$ 称为平凡解。若 $b \in Z_n$，$b \neq a \bmod n$，也满足 $x^2 \equiv a^2 \bmod n$，则称 b 为非平凡解。因为

$$b^2 \equiv a^2 \bmod n$$

故

$$n \mid (b+a)(b-a)$$

从而对整数 n 因数分解问题引出求

$$\gcd\{n, b+a\}, \quad \gcd\{n, b-a\}$$

例 8.6 $x^2 \equiv 4 \bmod 77$

$x = \pm 2$ 是上面 mod 77 的平方根的平凡解，± 9 是非平凡解。但

$$\gcd\{77, 9+2\} = \gcd\{77, 11\} = 11$$

$$\gcd\{77, 9-2\} = \gcd\{77, 7\} = 7$$

实际上

$$77 = 11 \times 7$$

令 p 是 n 的未知因子，设序列 $a_0, a_1, \cdots$，满足

$$a_0 = 1, \quad a_{i+1} \equiv a_i^2 + 1 \bmod p, \quad i \geqslant 0$$

若存在 h 和 k 使 $a_h \equiv a_k \bmod p$，$k > h$，显然有 $\gcd(a_k - a_h, n)$ 被 p 除尽，而且非常可能这个 gcd 就是 p，由于 p 尚不知道，故代以 n，$a_0 = 1$，$a_{i+1} \equiv a_i^2 + 1 \bmod n$，$i \geqslant 0$。

现将叙述实用的 Pollard 启发式因数分解 n 的 ρ 法：

S1. $i \leftarrow 1$，$x_1 \leftarrow 1$，

$y \leftarrow x_1$，$k \leftarrow 2$；

S2. $i \leftarrow i+1$，$x_i \leftarrow (x_{i-1}^2 - 1) \bmod n$，

$d \leftarrow \gcd\{y - x_i, n\}$；

S3. 若 $d \neq 1$ 且 $y \neq n$ 则作

始 打印 d，停止，

终；

S4. 若 $i = k$ 则作

始 $y \leftarrow x_i$，

$k \leftarrow 2k$

终；

S5. 转 S2。

例 8.7 $n = 1387$

1. $i = 1$，$x_1 = 2$，$y = x_1 = 2$，$k = 2$；

2. $i = 2$，$x_2 = 3$，$d = 1$.

$y=x_2=3,k=4$；

3. $i=3,x_3=8,d=\gcd\{5,1387\}=1$；
4. $i=4,x_4=63,d=\gcd\{60,1387\}=1$，
   $i=k,y=63,k=8$；
5. $i=5,x_5\equiv 1194,d=\gcd\{1131,1387\}=1$；
6. $i=6,x_6\equiv 1425635 \bmod 1387\equiv 1186$，
   $d=\gcd\{1123,1387\}=1$；
7. $i=7,x_7\equiv 1406595 \bmod 1387\equiv 177$，
   $d=\gcd\{177-63,1387\}=\gcd\{114,1387\}=19$。

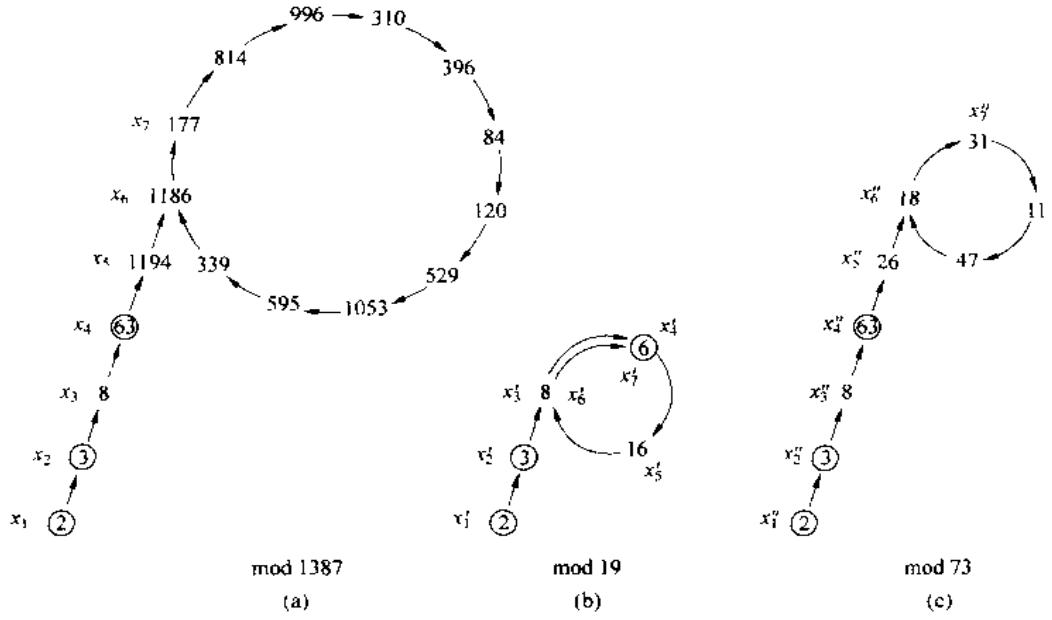


图 8.9

图 8.9(a)是分解 $n=1387$ 的例子。图 8.9(b)和图 8.9(c)分别是 mod 19 和 mod 73 的例子，Pollard ρ 法因此而得名，至于 $n=73$ 的 ρ 法分解因子的过程如下：

1. $i=1,x_1=2,y=x_1,k=2$；
2. $i=2,x_2=3,d=1,y=3,k=4$；
3. $i=3,x_3=8,d=\gcd\{5,73\}=1$；
4. $i=4,x_4=63,d=\gcd\{40,73\}=1$；
   $i=k,y=63,k=8$；
5. $i=5,x_5\equiv(63^2-1) \bmod 73\equiv 3968 \bmod 73\equiv 26$，
   $a=\gcd\{37,73\}=1$；
6. $i=6,x_6\equiv(26^2-1) \bmod 73\equiv 675 \bmod 73\equiv 18$，
   $d=\gcd\{45,73\}=1$；
7. $i=7,x_7\equiv(18^2-1) \bmod 73\equiv 31$，
   $d=\gcd\{32,31\}=1$。

第9章 密码协议

9.1 密码协议举例

随着网络应用普及到各个领域，如何在网络环境下完成各种任务必须要有协议。所谓协议指的是双方或多方通过一系列规定的步骤来完成某项任务。一系列规定指的是自始至终的步骤序列需依序进行。利用RSA公钥密码进行数字签名就有通信双方约定的协议：

(1) A先用自己的解密密钥对信息 m 签名得

$$s = D_A(m)$$

(2) A再用B的加密密钥对 s 进行加密得密文

$$c = E_B(s) = E_B(D_A(m))$$

并将密文 c 送给B

(3) B先用他自己掌握的解密密钥对 c 解密得

$$D_B(c) = D_B(E_B(s)) = s = D_A(m)$$

(4) B再用A的加密密钥对 s 进行加密，即

$$E_A(s) = E_A(D_A(m)) = m$$

这样B可以确认收到的信息 m 的确是A送来的，因 s 是由只有A才掌握的解密密钥得来的。

为了使A了解B是否已确实完整收到他发出的信息 m，很自然想到如下协议：

(1) A送给B下面密文

$$c_1 = E_B(D_A(m))$$

(2) B收到密文 c_1 后，作

$$E_A(D_B(c_1)) = E_A(D_A(m)) = m$$

(3) B送给A下面密文

$$c_2 = E_A(D_B(m))$$

(4) A收到 c_2 后，作

$$E_B(D_A(c_2)) = \overline{m}$$

A将 m 与自己发送给B的 m 作比较。若 $m=\overline{m}$，说明B确已完整收到A送去的信息 m。

其实这协议是不安全的。比如F截获A送给B的密文

$$E_B(D_A(m))$$

F将 c 送给B，并宣称是他自己发送给B的，按协议，B机械执行

$$E_F(D_B(c)) = E_F(D_B(E_B(D_A(m))))$$
$$= E_F(D_A(m))$$

并将它送给 F，F 可以此获得 m。

这个协议只要将 B 收到密文 $E_B(D_A(m))$后不是简单地机械地执行“回执”的步骤，而是观察解密的结果是否有意义，若收到的密文解密后获得有意义的信息，将 m 收下，然后给 A 送去“回执”$E_A(D_B(m))$，否则予以拒绝。这样不安全因素可以避免。

9.2 Shamir 协议

下面介绍一种 Shamir 协议，任一对秘密通信的双方都不必建立公钥或私钥，通过该协议便可以获得保密通信。

下面协议中 E_A 和 D_A 均由 A 自己掌握，秘密保存，E_B，D_B 也一样。

(1) A 用自己的加密密钥给 m 以签名得 $E_A(m)$。并将它送给 B。

(2) B 收到 $E_A(m)$后，用自己的加密密钥再加密得 $E_B(E_A(m))$，并将它送还 A。

(3) A 收到 $E_B(E_A(m))=E_A(E_A(m))$后，用他自己掌握的解密密钥对之解密得

$$D_A(E_A(E_B(m))) = E_B(m)$$

并将 $E_B(m)$再送给 B。

(4) B 收到 $E_B(m)$后用 B 自己掌握的解密密钥，解密得

$$D_B(E_B(m)) = m$$

Shamir 协议可形象地看作是 A 先将保密信息 m 装入保险箱子，锁上只有他自己方能开启的一把锁，然后寄给 B。B 收到后，他不能开启这箱子，只是再加上只有 B 自己才能打开的另一把锁，再退还给 A。这时箱子上共有两把锁，A 开启他自己锁上的只有他自己才能打开的锁，然后重新寄给 B，B 最后打开他自己加上的锁，取出保密信息 m。

必须特别指出，Shamir 协议没有确认，只有保密。在保密通信前缺少对发信方的身份验证。

Shamir 协议要求加密算法是可交换的，即

$$E_i(E_j(m)) = E_j(E_i(m))$$

并非所有的密码系统都具有这一性质，比如 DES 就不是可交换的。

一次一密密码系统是可交换的，但用在 Shamir 协议上是不安全的。假如 A 的密钥为 $k_A=k_1k_2\cdots k_n\cdots$，B 的密钥 $k_B=k_1'k_2'\cdots k_n'$，其中 k_i，k_i' 都是 0 或 1，$i=1,2,\cdots,n$，设 $m=m_1m_2\cdots m_n$ 为明文。

$$c_1 = c_1^{(1)}c_2^{(1)}\cdots c_n^{(1)}, \quad c_i^{(1)} = m_i \oplus k_i, \quad i = 1,2,\cdots,n$$

A 将 c_1 送给 B，B 作 $c_i^{(2)}=c_i^{(1)}\oplus k_i'=m_i\oplus k_i\oplus k_i'$，$i=1,2,\cdots,n$。

B 将 $c_2=c_1^{(2)}c_2^{(2)}\cdots c_n^{(2)}$ 送给 A，A 将 c_2 与 k 作按位⊕运算得

$$c_3 = c_1^{(3)}c_2^{(3)}\cdots c_n^{(3)}$$

$$c_i^{(3)} = c_i^{(2)} \oplus k_i = m_i \oplus k_i \oplus k_2' \oplus k_i$$

$$c_i^{(3)} = m_i \oplus k_i', \quad i = 1,2,\cdots,n$$

攻击者可将 c_1，c_2，c_3 作按位⊕运算，即可得到 m，因

$$c_i^{(1)} \oplus c_i^{(2)} \oplus c_i^{(3)} = m_i \oplus k_i \oplus (m_i \oplus k_i \oplus k_i') \oplus (m_i \oplus k_i \oplus k_i' \oplus k_i)$$

$$= m_i, \quad i = 1,2,\cdots,n$$

所以这时 Shamir 协议是不安全的。

可以取 p 一素数，$\varphi(p)=p-1$，各用户随机选择一数 e，$1<e<p$，$\gcd(e,p-1)=1$，并计算一 d，使 $ed\equiv 1 \bmod(p-1)$。

这样的(e,d)由各用户自己秘密保存，可用来通过 Shamir 协议进行通信。加解密运算：$c\equiv m^e \bmod p$，$m\equiv c^d \bmod p$。

还可以考虑选择一 $GF(2^m)$域，使得 2^m-1 是一素数，

$$c \equiv m^e \bmod (2^m-1)$$

$$m \equiv c^d \bmod (2^m-1)$$

其中 e 和 d 为各用户自己随机选择，并自己秘密保存。

$$1<e<2^m-1, \quad ed \equiv 1 \bmod (2^m-1)$$

当 $p\approx 2^m$ 时，在 $GF(2^m)$上进行幂运算比 $GF(p)$上运算要快得多，但较不安全。

9.3 典型的协议举例：会话密钥分配

计算机网络上不仅仅可以保密通信，还可以通过协议完成另一些项目。

密钥改换是十分重要的，但绝非轻而易举。它的重要性是显而易见的，正如我们所了解的那样，攻击者对密文掌握得越多，危险性也越大。不同密钥的密文可帮助攻击者分析加密算法的结构。相同密钥的密文有助于揭露密钥本身。密钥经常更换，不让攻击者有机可乘。

前面已介绍过通过 KDC 发放会话密钥的办法，也就是发放会话密钥的协议。用的是对称密码算法，下面介绍通过非对称密码，比如说是 RSA 公钥密码，发送密钥的协议。假定 A 的公开密钥为 $k_e^{(A)}$，A 的秘密密钥为 $k_d^{(A)}$，KDC 的公钥设为 $k_e^{(D)}$，相应的密钥用 $k_d^{(D)}$ 表示它。$k_e^{(D)}$ 为用户普遍掌握。

S1. A 向 KDC 提出申请，要和 B 保密通信，向 KDC 送去：

$$(A,B)$$

S2. KDC 用它的密钥 $k_d^{(D)}$，对$(k_e^{(B)},B)$加密，得

$$S = \mathrm{RSA}_{k_d^{(D)}}(k_e^{(B)},B)$$

S3. A 用 $k_e^{(D)}$ 对 S 解密，得 $k_e^{(B)}$。

这里 $\mathrm{RSA}_{k_d^{(D)}}(k_e^{(B)},B)$没有保密措施，谁都有可能解开它。但只有 KDC 才能产生它。

A 有了 B 的公钥可以马上和 B 通信，只要用 $k_e^{(B)}$ 通过 RSA 加密信息 m，

$$C = \mathrm{RSA}_{k_e^{(B)}}(m)$$

只有 B 才能读到它。但 B 没法相信它是来自 A，有可能是第三者假冒 A 发出的。下面的协议有：

S1. A 向 KDC 提出申请：我是 A，我要求 B 的公钥 $k_e^{(B)}$；

S2. KDC 向 A 送去用 $k_d^{(D)}$ 签的名的

$$S = \mathrm{RSA}_{k_d^{(D)}}(k_e^{(B)},B)$$

S3. A 解密得 $k_e^{(B)}$，并向 B 送去

$$C = \mathrm{RSA}_{k_e^{(B)}}(A,r_A)$$

这里 r_A 是 A 的标识符；

S4. B 向 KDC 提出申请：给我 A 的公钥 $k_e^{(A)}$；

S5. 同样办法 KDC 向 B 送去

$$RSA_{k_d^{(0)}}(k_e^{(A)}, A)；$$

S6. B 收到后向 A 送去

$$RSA_{k_e^{(A)}}(r_A, r_B)；$$

S7. A 接到后从 r_A 可知信息来自 B；

S8. A 向 B 送去

$$RSA_{k_e^{(B)}}(k_s, r_A)。$$

k_s 是 A,B 间的会话密钥。会话还是通过对称密码进行。

9.4 对称密码的数字签名协议

对称密码的数字签名需要一公证机构 C。每一用户在 C 存有共享的密钥，用户 A 在 C 加密保存的密钥设为 k_A。

S1. A 向 C 送去经过 k_A 加密的信息：$E_{k_A}(A, E_{k_1}(m), B)$

即 A 欲向 B 寄去 m，及签名 $S=E_{k_1}(m)$；

S2. C 在确认 S 来自 A 后，向 B 送去

$$E_{k_B}(m, A, E_{k_A}(m))$$

S3. B 用 k_B 解密得 A 送给的 m，对 $E_{k_A}(m)$，B 无法解密，更无法修改它。B 将 $E_{k_1}(m)$保存。

若 A 抵赖，B 可出示$(m, E_{k_A}(m))$。

9.5 扑克游戏

通过一种利用指数加密方法，做电子游戏。甲和乙选一大素数 p，并分别各自选一密钥 k_1, k_2 作为加密变换。令

$$E_1(m) = m^{k_1} \bmod p$$
$$E_2(m) = m^{k_2} \bmod p$$

其中 m 是明文。若 h_1 和 h_2 分别满足

$$h_1 k_1 \equiv 1 \bmod (p-1), \quad h_2 k_2 \equiv 1 \bmod (p-1)$$

则 E_1 和 E_2 的脱密变换分别为

$$D_1(c) = c^{h_1} \bmod p, \quad D_2(c) = c^{h_2} \bmod p$$

其中 c 是密文。所以

$$\begin{aligned} E_2(E_1(m)) &\equiv E_2(m^{k_1}) \bmod p \\ &\equiv (m^{k_1})^{k_2} \bmod p \\ &\equiv (m^{k_2})^{k_1} \bmod p \\ &\equiv E_1(E_2(m)) \end{aligned}$$

假如52张扑克牌分别用相应的52个信息 $m_1, m_2, \cdots, m_{52}$ 来表示。若甲和乙要通过通信网络玩扑克牌。步骤如下(假定甲发牌)：

(1) 甲通过自己的加密变换对52个信息加密得 $E_1(m_1), E_1(m_2), \cdots, E_1(m_{52})$。并将这52个密文随机排列，将它送给乙。

(2) 乙随机地选取其中的5个，并将这5个送还给甲，甲利用自己的解密算法求得这些明文，但是乙对这些明文不得而知，因为乙无法破译甲的加密变换。

(3) 乙从其他密文中随机地选取5个。设为 c_1, c_2, c_3, c_4, c_5。其中

$$c_i = E_1(m_{j_i}), \quad i = 1,2,\cdots,5$$

并将密文 $c_1, c_2, \cdots, c_5$ 用他自己的加密变换进行加密，得

$$c_i' = E_2(c_i) = E_2(E_1(m_{j_i})), \quad i = 1,2,\cdots,5$$

乙将这5个经过双重加密后的密文送给甲。

(4) 甲对乙送来的 $c_1', c_2', \cdots, c_5'$ 进行解密，由于

$$E_2(E_1(m)) = E_1(E_2(m))$$

所以

$$\begin{aligned} D_1(c_i') &= D_1(E_2(E_1(m_{j_i}))) \\ &= D_1(E_1(E_2(m_{j_i}))) \\ &= E_2(m_{j_i}) \end{aligned}$$

并将它送还乙，也就是甲知道的只是 m_{j_i} 经 E_2 加密后的结果。

(5) 乙将甲送来的密文进行解密，得明文 $m_{j_1}, m_{j_2}, \cdots, m_{j_5}$。

游戏继续进行过程中需要增加发牌可重复以上的步骤。直到结束时双方都公开他们的密钥，用以说明他们并没有欺骗行为。

但是以上的协议不能保证完全不存在欺骗。这个问题在此不加以讨论。

9.6 掷银币游戏

假定利害冲突的A，B双方远在两个城市，若要依靠掷银币定胜负，可以通过网络依如下的协议进行：

(1) A选取 p 和 q 两个大素数，告诉B它们的积 $n=pq$。

(2) B随机选择 u，满足 $0<u\leqslant n/2$，将 $z\equiv u^2 \bmod n$ 告诉A。

(3) A计算 $z\equiv x^2 \bmod n$ 的4个根：$\pm x, \pm y$。

由于A掌握 p 和 q，他能做到：

令 $$\xi=\min\{x, n-x\}, \ \eta=\min\{y, n-y\}$$

A猜测 $u=\xi$ 或 $u=\eta$。

(4) B告诉A他是正确或不正确，并将 u 的值告诉A。

(5) A公开 n 的因子。

这个协议依据是因数分解 n 是困难的。若A猜对了，B作假，这就要求B能因数分解 n，也就是他能掌握 ξ 和 η，这是B做不到的。A不掌握 u，故只能猜，猜中的概率为

1/2。这个协议实际上用猜 ξ 和 η 以模拟掷银币。B 若掌握 ξ 和 η，则求 $\gcd(\xi-\eta,n)$ 或 $\gcd(\xi+\eta,n)$ 便可求得 p 和 q。

9.7 一种身份认证协议

比如 A 将利用信用卡在读卡机上取款。A 必须证明他是信用卡的合法拥有者，银行在向 A 发放信用卡时，在上面储存有：①A 的身份号码 Id_A，②A 的密钥 k_A，k_A 连 A 自己也不知道，而且不可以对它进行存取。

$$k_A = E_k(Id_A)$$

E 是某分组密码，k 是银行用密钥，由银行掌管，在每一台读卡机上都储存 k。

在所有读卡机上储存所有用户的密钥 k_A 是不现实的。

当读卡机接收到用户 A 的卡时，它被要求提供 Id_A。读卡机接着计算 k_A，因 $k_A = E_k(Id_A)$。

接着读卡机产生一 n 位的随机数 r，作为对卡的响应。

信用卡向读卡机送去

$$R^* = E_{k_A}(r)$$

读卡机只要计算 $E_{k_A}(r)$，若 $E_{k_A}(r)=R^*$ 说明信用卡上储存有 k_A，即卡是合法的。否则予以拒绝。

卡	读卡机
(已知 k_A, ID_A)	(已知 k)
S1. 送去 Id_A	
S2.	(1) 计算 $k_A=E_k(ID_A)$， (2) 产生随机数 r， (3) 送回 r。
S3.	(1) 计算 $E_{k_A}(r)=R^*$， (2) 送出 R^*，
S4.	(1) 计算 $E_{k_A}(r)=R$， (2) 比较 $R=R^*$ 是否成立。

若相等则接收，否则拒绝。

$R=R^*$ 说明卡确实有 k_A。

信用卡也可以利用以上协议来测试读卡机是否是真的，以防止利用伪读卡来窃取用户的信息，达到犯罪的目的。也就是卡向读卡机发去 Id_A，读卡机计算

$$k_A^* = E_k(Id_A)$$

并向卡作出回应。若

$$k_A = k_A^*$$

则证明读卡机掌握正确的 k。否则读卡机是假的。

9.8 计算机选举

计算机选举是对一争议的问题 v 给一个"是"或"非"的答案。首先只有被授权的人才能参加，每人只能投一次票，没有其他任何人能知道他投的是什么。

假定一个公钥密码系统 R，两个投票人可能投的票是相同的。协议必须保证每个投票的人能够识别他或她投的那个票，但无需知道别人投的是什么。

假定投票的有 A，B，C 3 个人，每一个参加投票的人给 v 一个"是"或"非"。而假定他们投的票分别为 v_A，v_B，v_C。

每一个人分别进行如下步骤，以 A 为例，其余类推。

(1) 计算 $R_A(R_B(R_C(v_A,r_A)))$，设令它为 w_A，r_A 是随机数。

(2) 附加上随机的 0，1 符号串 $r_1^{(A)}$，$r_2^{(A)}$，$r_3^{(A)}$ 继续作

$$R_A(r_1^{(A)},R_B(r_2^{(A)},R_C(r_3^{(A)},w_A)))$$

并记下 $r_1^{(A)}$，$r_2^{(A)}$，$r_3^{(A)}$ 的值。

同理 B 和 C 分别有

$$R_A(r_1^{(B)},R_B(r_2^{(B)},R_C(r_3^{(B)},w_B)))$$

$$R_A(r_3^{(C)},R_B(r_2^{(C)},R_C(r_1^{(C)},w_C)))$$

各自记录下中间结果，可以用来确定自己的选票是否在内。

(3) 他们把结果都汇集到 A，A 作 R_A^{-1} 的运算，由于

$$R_A^{-1}(R_A(r_1^{(i)},R_B(r_2^{(i)},R_C(r_3^{(i)},w_i)))) = (r_3^{(i)},R_B(r_2^{(i)},R_C(r_1^{(i)},w_i)))$$

$$i = A,B,C$$

A 接着去掉 $r_2^{(i)}$，结果是

$$R_B(r_2^{(i)},R_C(r_1^{(i)},w_i)),\quad i = A,B,C$$

A 将这些结果顺序搅乱后转给 B。

(4) B 用自己的私钥解密，并查看自己的选票是否在其中，再去掉附加的随机数 $r_2^{(i)}$，得

$$R_C(r_1^{(i)},w_i),\quad i = A,B,C$$

B 将结果的顺序搅乱，然后送给 C。

(5) C 类似地解密，并查看自己的选票是否在其中，去掉随机数 $r_1^{(i)}$，$i=A,B,C$。最后是

$$w_A = R_A(R_B(R_C(v_A)))$$

$$w_B = R_A(R_B(R_C(v_B)))$$

$$w_C = R_A(R_B(R_C(v_C)))$$

并将它们一起送给 A。

(6) A 对它们解密，检查自己的选票是否在其中，然后对结果签上自己的名送给 B 和 C，结果是

$$S_A(E_B(E_C(v_i,r_i))),\quad i = A,B,C$$

(7) B 用 A 的公钥除去 A 的签名，然后再用 B 的密钥解密，观察自己的选票还在其

中，再签上名送 A 和 C。结果将是

$$S_B(E_C(v_i, r_i)), \quad i = A, B, C$$

(8) C 用 B 的公钥去掉 S_B，再用自己的密钥解密得

$$(v_i, r_i), \quad i = A, B, C$$

再签名送 A、B，形式为

$$S_C(v_i, r_i), \quad i = A, B, C$$

(9) 每位都可以除去 C 的签名，检查自己的选票是否在其中。并统计选举结果。

9.9 签合同协议

S1. A 随机地选 $2n$ 个某对称密码系统的密钥：

$$k_1, k_2, \cdots, k_{2n}$$

并将它们分成 n 组：

$$(k_1, k_{n+1}), (k_2, k_{n+2}), \cdots, (k_n, k_{2n})$$

S2. A 计算

$$C_i = E_{k_i}(S_l), \ C_{n+i} = E_{k_{n+i}}(S_r), \quad i = 1, 2, \cdots, n$$

S 是合同的签名文件，S_l、S_r 是 S 的左、右两部分。A 将 $\{C_i\}$ 寄给 B，B 已有合同的对于各个密钥加密的密文的文本；

S3. A 答应 B 只要能出示 (k_i, k_{n+i})，i 为 $1, 2, \cdots, n$ 中某一个值，就算 A 签了合同；

S4. B 类似于 A 执行 S1～S3 的步骤，令

$$D_j = E_{h_j}(S), \quad j = 1, 2, \cdots, 2n$$

S5. A 按“若无其事”的传输协议方式向 B 送去

$$(k_i, k_{n+i}), \quad i = 1, 2, \cdots, n$$

即 A 向 B 送去 k_i 或 k_{n+i}，B 也类似向 A 送去 h_j 或 h_{n+j}，$j = 1, 2, \cdots, n$

S6. j 从 1 到 l 作

始 A 将 k_i 的第 j 比特寄给 B，

B 将 h_i 的第 j 比特寄给 A，

$$i = 1, 2, \cdots, n$$

终

其中 l 是 h_i、k_i 的密钥比特位数；

S7. A 和 B 解余下的一半秘密对，合同被签署。

9.10 挂号信协议

A 送一信给 B，要 B 在收到后给收据证明。A 给 B 的信 m 假如是加密的，需等到 B 反馈足够的信息后才将密钥寄去。B 必须在收到密钥后，取得 m 才给 A 发出收据。

S1. A 产生某一对称密码系统的 $n+1$ 个随机密钥：

$$k_0, k_1, \cdots, k_n$$

S2. A 计算

$$C = E_{k_0}(M)$$

并将 C 送给 B，M 是新寄的信息；

S3. A 计算并给 B 寄去

$$C_i = E_{k_i}(S), \quad i = 1,2,\cdots,2n$$

其中 $k_{n+i} = k_0 \oplus k_i$, $i = 1,2,\cdots,n$

S 是某些信息。这时 B 已有 C 及 S 的 $2n$ 个加密的文本；

S4. B 选 $2n$ 个密钥：$h_1, h_2, \cdots, h_{2n}$，并计算

$$D_i = E_{h_i}(S), \quad i = 1,2,\cdots,2n$$

将 $D_1, D_2, \cdots, D_{2n}$ 寄给 A；

S5. B 送一说明给 A，如果 A 能出示一对 (h_i, h_{n+i}) 及 $k_1, k_2, \cdots, k_{2n}$。

B 将给收据；

S6. 如签合同的协议一样，A 和 B 互换通过"若无其事"的协议方式交换

$$(k_1, k_{n+1}), (k_2, k_{n+2}), \cdots, (k_n, k_{2n})$$

和

$$(h_1, h_{n+1}), (h_2, h_{n+2}), \cdots, (h_n, h_{2n})$$

也就是 B 收到 k_i 或 k_{n+i}，A 和 B 都不知道它是什么，A 收到每个 (h_i, h_{n+i}) 也不知道它是什么；

S7. j 从 1 到 l 作

始

A 将 k_i 的第 j 比特位寄给 B，

B 将 h_i 的第 j 比特位寄给 A，

$$i = 1,2,\cdots,2n$$

终

l 是 k_i 和 h_i 的长度。

注意 $k_{n+i} = k_0 \oplus k_i$

所以 $k_{n+i} \oplus k_i = k_0$

所以 B 只要知道任意一对 (k_i, k_{n+i})，他就可以对 $C = E_{k_0}(M)$ 解密。

9.11 其他有意义的协议

前面已经讨论了许多协议，下面再介绍几个有意义的协议。

1. 不可否认的签名协议

一般的数字签名都是由送信方（设为 A）将信息 m 加密签名后送给收信方，任何一个只要知道 A 的公开密钥者都可以对此签名进行验证。不可否认的签名则要求对它的验证需要 A 参加，B 不能向第三者证明该签名的正确性。

协议之一：

p 为大素数，g 是其本原元素，p，g 公开，A 的秘密密钥为 x，公开密钥为

$$y \equiv g^x \bmod p$$

信息设为 m，A 计算

$$z \equiv m^t \bmod p$$

送 c 给收信方(设为 B)。

B 的验证步骤：

(1) B 选择两个小于 p 的随机数 a 和 b，

$$c \equiv (z)^a (y)^b \bmod p$$

并给 A。

(2) A 计算 $t \equiv x^{-1} \bmod (p-1)$，

$$d \equiv c^t \bmod p$$

并将 d 送给 B。

(3) B 验证

$$d \equiv m^a g^b \bmod p$$

是否成立，如果成立，则签名有效，否则无效。

协议之二：

本协议与协议之一的不同之处在于验证步骤，其他和前面相同。验证步骤如下：

(1) B 选择小于 p 的两个随机数 a 和 b，并计算

$$c = m^a g^b \bmod p$$

B 将 c 送给 A。

(2) A 选择小于 p 的随机数 q，计算

$$s_1 \equiv cg^q \bmod p$$

$$s_2 \equiv (cg^q)^x \bmod p$$

并将 s_1、s_2 送给 B。

(3) B 将 a、b 送给 A，以证明步骤(1)不曾欺骗 A。

(4) A 将 q 送给 B。

(5) B 验证

$$s_1 \equiv cg^q \bmod p$$

$$s_2 \equiv y^{b+q} z^a \bmod p$$

是否成立，若成立签名有效，否则无效。

2. 盲签名

A 要 B 对信息 m 签名，但不让 B 知道 m 的内容。设 B 有 RSA 密码系统中的公开的加密密钥 e，及秘密的解密密钥 d。

(1) A 随机选择 $k<n$，计算

$$t = mk^e \bmod n$$

(2) A 送 t 给 B，B 作

$$t^d \equiv (mk^e)^d \bmod n$$

的计算后还给 A。

(3) A 收到 $t^d \bmod n$ 后，只要作

$$\frac{t^d}{k} \bmod n$$

计算。不难证明

$$\frac{t^d}{k} \equiv m^d \bmod n$$

3. 多方安全计算协议

例 9.1 已知 p 是一大素数，g 是 $GF(p)$ 域的本原元素，x 为小于 p 的整数。A 要求得离散对数 $y=\log_g x$，即求 y 使

$$g^y \equiv x \bmod p$$

求助于 B。B 有足够大的计算能力，但 A 不愿 B 得知 x 这个数，即在 A 不愿暴露 x 这个数的前提下，让 B 协助解决离散对数问题。协议如下：

(1) A 选择随机数 $r<p$，计算

$$\xi \equiv xg^r \bmod p$$

(2) A 要求 B 计算 η，使得

$$g^\eta \equiv \xi \bmod p$$

即计算离散对数

$$\eta \equiv \log_g \xi \bmod p$$

(3) B 将 η 送给 A 后，A 计算

$$y \equiv (\eta - r) \bmod (p-1)$$

不难验证

$$g^y \equiv g^\xi / gr \equiv x \bmod p$$

例 9.2 百万富翁问题：设 A 手里掌握数 α，B 手里掌握数 β。在不让对方知道自己手里掌握的具体数字的前提下，作出判断究竟是

$$\alpha > \beta? \quad \text{还是} \quad \alpha \leqslant \beta?$$

可按如下协议进行。

不妨假定 i 和 j 是 1 到 100 间的某个数，而且 B 有一公开的密钥和秘密的密钥。

(1) A 选一大随机数 x，并用 B 的公开密钥加密得

$$c = E_B(x)$$

并告诉 B，x 的范围。

A 计算 $c-\alpha$，并将结果告诉 B。

(2) B 计算下面 100 个数

$$y_u = D_B(c - i + u), \quad 1 \leqslant u \leqslant 100$$

其中 $D_B(\cdot)$ 表示用 B 的秘密密钥加密。

B 随机选择一大素数 $p<x$，并计算下面 100 个数

$$z_u \equiv y_u \bmod p, \quad 1 \leqslant u \leqslant 100$$

B 验证对所有 $u \neq v$，恒有

$$|z_u - z_v| \geqslant 2$$

对所有的 u，恒有

$$0 < z_u < p-1$$

如若不然，B 选择其他素数，再作试验。

(3) B将下面一串数送给A。

$$z_1, z_2, \cdots, z_\beta, z_{\beta-1}+1, z_{\beta-2}+1, \cdots, z_{100}+1, p$$

(4) A测试该串数中的第α个数是否与$x \bmod p$同余，若是，则结论为$\alpha \leqslant \beta$，否则$\alpha > \beta$。

A将结论告诉B。

(2)的过程是为了保证(3)产生的序列不至于出现一个数出现两次。否则若$z_a = z_b$，A知道$a \leqslant \beta < b$。

本协议存在一种可能：A欺骗B。

例9.3 RSA公钥系统$n=55$，B的公钥为7，秘密密钥为23。A的$a=4$，B的$\beta=2$。α和β可能为1,2,3,4。

(1) A取$x=39$，$c=E_B(39)\equiv 39^7 \bmod 55$

$c\equiv 19$。

$c-a=15$，A将它送给B。

(2) B计算

$$y_1 = D_B(15+1) \equiv 16^{23} \bmod 55 - 26$$
$$y_2 = D_B(15+2) \equiv 17^{23} \bmod 55 = 18$$
$$y_3 = D_B(15+3) \equiv 18^{23} \bmod 55 = 2$$
$$y_4 = D_B(15+4) \equiv 19^{23} \bmod 55 = 39$$

B取$p=31$，计算

$$z_1 = 26 \bmod 31 = 26$$
$$z_2 = 18 \bmod 31 = 18$$
$$z_3 = 2 \bmod 31 = 2$$
$$z_4 = 39 \bmod 31 = 8$$

z_1, z_2, z_3, z_4满足条件使若$u \neq v$，有$|z_u - z_v| \geqslant 2$。

B将下列序列送给A：

$$26, 18, 2+1, 8+1, 31$$

即

$$26, 18, 3, 9, 31$$

(3) A将其中第4个数9，检验它是否符合与$x \bmod p$同余，由于

$$9 \not\equiv 39 \bmod 31$$

故

$$\alpha > \beta$$

协议的原理留给读者思考。

9.12 零知识证明

9.12.1 零知识证明概念

在以后各节，集中讨论近代密码中十分引人入胜的问题。设证明者P掌握某些秘密信息，可以是某些长期没有解决的猜想问题的证明。如Fermat最后定理，图的三色问

题，也可以是缺乏有效算法的难题解法，如大整数因数分解等，信息的本质是可以验证的，即可通过具体的步骤来检测它的正确性。V 是验证者，P 设法让 V 相信他确实掌握这信息。以假定 P 掌握了大数 n 的因数 p 和 q 为例，当然 P 可以直截了当地把 p 和 q 告诉 V，这样 V 也可以向他人宣布他也掌握了大数 n 的因数。这样就不是零知识证明。

问题要求：假如 P 想说服 V，使 V 相信他确实知道 n 的因数 p 和 q，但要求一比特的秘密也不泄露，从而 V 无从获得这秘密信息本身。

最简单的步骤如下：

S1. V 随机地选择一大整数 x，计算 $x^4 \bmod n$ 的值，并且将它告诉 P。

S2. P 求 $x^2 \bmod n$ 并将它告诉 V。

V 知道求 $x^2 \bmod n$ 等价于 n 的因数分解，若不掌握 n 的因数 p 和 q，求平方根是一困难问题。此外，他毫无所得，因为求 $x^2 \bmod n$ 他本来就能做到。若 P 不知道 n 的因数，多次重复以上步骤一直到成功的可能性是极小的。

下面我们假定秘密信息是定理证明，并假定证明者 P 不可能欺骗 V，即 P 不掌握定理的证明能说服 V 使他相信的可能性几乎没有。同时，验证者 V 也不能欺骗证明者，V 除了"P 知道证明"外，他不知道哪怕是一点点的证明细节，因此他不可能对其他人证明那个定理。

举一交互过程非零知识的例子：身份证明协议。

密码协议是一系列的约定，通信系统中的成员都必须遵守以建立系统内部的信息交换和联系。例如下面证明身份的协议。

每一用户，设为 A，秘密选择一正整数 a 并计算 $y_A = \alpha^a$。并用 (A, y_A) 形式公布在通信录上，其中 A 为用户的身份，假定每一用户都有一本通信录。

若 A 向 B 证明自己的身份，步骤如下：

(1) A 向 B 送去一正整数 α。

(2) B 收到 α 后随机选择一正整数 R，计算 α^R，并将 $y_2 = \alpha^R$ 送给 A。

(3) A 计算 $y_3 = (y_2)^a = \alpha^{aR}$，并送 y_3 给 B。

(4) B 计算 $(y_A)^R = \alpha^{aR}$，并检查 $(y_A)^R = y_3$？若等号成立，A 的身份便得到了证明。

此协议就不是零知识的。因为若 B 送去的 $y_2 = R$，则他将获得 $y_3 = R^a$。这个结果就不是他所能得到的，虽然这距离他想知道的 a 还很远。

又例

若已知正整数 x 和 y，$0 < y < x$，$\gcd(x, y) = 1$。P 宣称他知道不存在 $z \in Z$，使得

$$z^2 \equiv y \pmod{x}$$

验证者 V 便产生一组整数序列 $z_1, z_2, \cdots, z_n$ 及一组 0,1 序列 $b_1, b_2, \cdots, b_n$。满足 $0 < z_i < x$，$\gcd(z_i, x) = 1$，$i = 1, 2, \cdots, n$。计算

$$w_i = \begin{cases} z_i^2 \pmod{x} & \text{若 } b_i = 0 \\ z_i^2 y \pmod{x} & \text{若 } b_i = 1 \end{cases} \quad i = 1, 2, \cdots, n$$

V 将 $w_1, w_2, \cdots, w_n$ 送给 P。

P 对此进行判断，令

$$c_i=\begin{cases}0, & \text{若存在 } z_i \text{ 满足 } z_i^2\equiv w_i(\bmod x)\\ 1, & \text{其他}\end{cases}$$

P 将 $c_1,c_2,\cdots,c_n$ 送给 V。

若对所有的 i 有 $c_i=b_i$ 成立，则 P 断言：不存在 z 使得 $z^2\equiv y(\bmod x)$ 得到了证明。

表面上 V 得到的回答是他自己已知道的事实，但实际上 P 已向 V 泄露了 V 本不掌握的信息：即不存在 z 满足 $z^2\equiv y \bmod x$。

9.12.2 3 SAT 问题的零知识证明

例 9.4 $f=(x_1\vee x_2\vee\overline{x}_4)\wedge(x_2\vee\overline{x}_3\vee x_4)\wedge(\overline{x}_1\vee x_2\vee x_3)$
$\wedge(\overline{x}_1\vee\overline{x}_2\vee\overline{x}_3)\wedge(x_1\vee x_3\vee x_4)\wedge(\overline{x}_2\vee x_3\vee x_4)$

上面的 x_i 是逻辑变量，$i=1,2,3,4$，只取 T(真)，F(伪)两种值，$x_i=$T，则 $\overline{x}_i=$F，反之 $x_i=$F，则 $\overline{x}_i=$T，$\vee$ 为逻辑和，$\wedge$ 为逻辑乘，它们满足

$\vee$	T	F
T	T	T
F	T	F

$\wedge$	T	F
T	T	F
F	F	F

所谓可满足性问题 f，即对变量 x_i 的值给予指派，使最后结果 $f=$T。例如 $x_1=x_3=x_4=$T，$x_2=$F 时 $f=$T。

上面例中 f 是由 6 项通过 $\wedge$ 关系结合在一起，每项都由 x_1,x_2,x_3,x_4 或它们的非 $\overline{x}_1,\overline{x}_2,\overline{x}_3,\overline{x}_4$ 中 3 个用 $\vee$ 结合在一起，其中每一项称为一句子。例如 $x_1\vee x_2\vee\overline{x}_4$，$x_2\vee\overline{x}_3\vee x_4$ 等都称为句子。

这个例子中逻辑变量的个数为 4，句子的数目为 8。可将它推广到逻辑变量个数为 r，句子数目设为 t，即

$$f=C_1\wedge C_2\wedge\cdots\wedge C_t$$

其中 $C_1,C_2,\cdots,C_t$ 都由 $x_1,x_2,\cdots,x_n,\overline{x}_1,\overline{x}_2,\cdots,\overline{x}_n$ 中取 3 个通过运算 $\vee$ 构成的，当然 x_i 和 $\overline{x}_i$ 不能同时出现，如何给变元 $x_1,x_2,\cdots,x_n$ 以指派，使满足 $f=$T 的问题称为 3 SAT。SAT 是可满足性问题，它属于典型的 NPC 类，是没有有效算法的一类问题。

假定证明者 P 企图在不泄露任何秘密的前提下，说服验证者 V 相信他掌握关于 f 的指派。

P 准备了如下箱子：

关于变量的 B_i，$1\leqslant i\leqslant 2n$；

关于真值的 B_i^t，$1\leqslant i\leqslant 2n$。

对于 $2n$ 对 (x,y)，其中 x 是逻辑变量，y 是逻辑值 T 和 F，存在一 i 使得 x 锁在 B_i 里，y 锁在 B_i^t 里，而且 (x,y) 随机地对应于 (B_i,B_i^t)。以上述 f 为例：

B_i：$B_1(x_4)$，$B_2(x_2)$，$B_3(x_1)$，$B_4(x_4)$，$B_5(x_3)$，$B_6(x_4)$，$B_7(x_1)$，$B_8(x_2)$

B_i^t：B_1^t(T)，B_2^t(F)，B_3^t(F)，B_4^t(F)，B_5^t(T)，B_6^t(F)，B_7^t(T)，B_8^t(T)

P 同时还准备：

关于指派的 $B_{i,j,k}$：$1\leqslant i,j,k\leqslant 2n$，$\overline{1}\leqslant i,j,k\leqslant\overline{2n}$，$B_{i,j,k}$ 里含数 0 或 1，$B_{i,j,k}$ 含 1 当

且仅当 $i'=i$ 或 $\bar{i}$，$j'=j$ 或 $\bar{j}$，$k'=k$ 或 $\bar{k}$ 时，f 含有一句子，其中 3 个变元出现在箱子里 B_i，B_j，B_k，其对应逻辑值出现在 B_i^l，B_j^l，B_k^l。

以上述 f 为例。出现 1 的箱子有

$$B_{7,2,\bar{1}},\quad B_{2,\bar{5},1},\quad B_{\bar{7},2,5}$$
$$B_{\bar{7},\bar{2},\bar{5}},\quad B_{7,\bar{5},1},\quad B_{\bar{2},5,1}$$

以 $B_{7,2,\bar{1}}$ 为例。$B_7(x_1)$，$B_2(x_2)$，$B_1(x_4)$ 对应于第一个句子 $(x_1 \vee x_2 \vee \bar{x}_4)$，且 $B_7^l(\mathrm{T})$，$B_2^l(\mathrm{F})$，$B_1^l(\mathrm{T})$。由于 $B_7^l(\mathrm{T})$ 故 $B_{7,2,\bar{1}}$ 含 1。$B_{2,\bar{5},1}$ 由于 $B_2(x_2)$，$B_5(x_4)$，$B_1(x_4)$，故对应于第 2 句子 $(x_2 \vee \bar{x}_5 \vee x_4)$，且 $B_2^l(\mathrm{F})$，$B_5^l(\mathrm{T})$，$B_1^l(\mathrm{T})$，故 $F_{2,\bar{5},1}$ 含 1。

再以 $B_{\bar{7},\bar{2},\bar{5}}$ 为例。$B_i(x_1)$，$B_2(x_2)$，$B_5(x_3)$，故 $B_{\bar{7},\bar{2},\bar{5}}$ 对应于句子 $(\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$。又 $B_7^l(\mathrm{T})$，$B_2^l(\mathrm{F})$，$B_5^l(\mathrm{T})$，故 $B_{\bar{7},\bar{2},\bar{5}}$ 箱子含数 1。

协议可模仿图的三着色进行。若 V 要求 P 打开除逻辑值的箱子 B_i^l 以外的所有箱子，V 从 $B_{i,j,k}$ 含 1 的箱子可以看到只是 f 的句子，此外他什么也得不到。

V 也可以要求 P 打开所有关于逻辑值的箱子 B_i^l 和指派箱子 $B_{i,j,k}$，V 如何验证 P 的指派是合理的等。这些都留给读者作为练习。

9.12.3 身份的零知识证明

智能卡、信用卡、计算机的口令等身份鉴别技术是 P 显示他的身份 $I(\mathrm{P})$，这种 $I(\mathrm{P})$ 储存或打印在卡片上，敌方和不诚实的验证方合作可能获得一卡的拷贝或知道 $I(\mathrm{P})$，敌方可以利用 $I(\mathrm{P})$ 假冒 P 从事非法活动。

这个问题的解决办法是利用零知识证明技术。P 可以不透露 $I(\mathrm{P})$ 的一个比特使 V 确信 P 确实掌握 $I(\mathrm{P})$。

(1) 下面的协议假定存在一可信赖的机构，它的职责在于公布模数 n，n 等于两个大素数 p 和 q 之积，p 和 q 都是 mod 4 与 3 同余。

$I(\mathrm{P})$ 必须包含 P 的许多信息。假定 $I(\mathrm{P})$ 包含 P 的保密身份的 k 个数 $c_1, c_2, \cdots, c_k$，$1 \leqslant c_i < p$，$i = 1, 2, \cdots, k$；还有他的公开身份的另外 k 个数 $d_1, d_2, \cdots, d_k$，$1 \leqslant d_i < p$，$i = 1, 2, \cdots, k$，而且满足

$$d_j c_j^2 \equiv \pm 1 \bmod n \quad j = 1, 2, \cdots, k$$

验证者 V 知道 n 及 P 的公开身份。P 为了使 V 相信他掌握 $I(\mathrm{P})$，下面 4 步骤算是一轮。通过轮数越多，P 作假的概率越小。

S1. P 选一随机数 r，计算 $\pm r^2 \bmod n$，P 取其中一个告诉 V，称之为 x。

S2. V 从 $\{1, 2, \cdots, k\}$ 中选一子集 S 告诉 P。

S3. P 告诉 V

$$y = rT_c \bmod n$$

其中

$$T_c = \prod_{j \in S} c_j$$

S4. V 验证

$$x = \pm y^2 T_d \bmod n?$$

其中

$$T_d = \prod_{j \in S} d_j$$

若等号成立，开始新的一轮验证或停止，否则予以拒绝。

由于 $d_j c_j^2 \equiv \pm 1 \bmod n$，$j=1,2,\cdots,k$。所以

$$y^2 T_d \equiv r^2 T_c^2 T_d = r^2 \prod_{j \in S} c_j^2 d_j \equiv \pm r^2 \equiv \pm x \bmod n$$

随机数 r 是必要的，否则 V 选 $S=\{j\}$，从而找到 c_j，对 c_j 要求 $(c_j, n) \neq 1$，$j=1,2,\cdots,k$，否则 n 可能被因数分解。

例 9.5 可信赖的机构公开宣布 $n=2773$。

P 的秘密 $I(P)$ 包含有

$$c_1=1901,\ c_2=2114,\ c_3=1509,\ c_4=1400,\ c_5=2001,\ c_6=119$$

$$c_1^2 \bmod n=582,\ c_2^2 \bmod n=1693,\ c_3^2 \bmod n=448$$

$$c_4^2 \bmod n=2262,\ c_5^2 \bmod n=2562,\ c_6^2 \bmod n=296$$

P 选择他的公开身份有

$$d_1=81,\ d_2=2678,\ d_3=1207,\ d_4=1183,\ d_5=2681,\ d_6=2595$$

它们满足

$$d_j c_j^2 \equiv \pm 1 \bmod n$$

假定 P 选 $r=1221$，P 计算

$$-r^2 \equiv -1490841 \equiv 1033 \bmod n$$

令 $x=1033$，P 将 x 告诉 V。

V 选子集 $S=\{1,4,5,6\}$，告诉 P。

P 计算：

$$T_c=1901\times 1400\times 2001\times 119 \equiv 96 \bmod n$$

$$y \equiv rT_c \bmod n \equiv 1221\times 96 \bmod n \equiv 117216 \bmod n \equiv 750$$

V 收到 y 后计算

$$T_d = 81\times 1183\times 2681\times 2595 \bmod n \equiv 1116 \bmod n$$

$$y^2 T_d \equiv (750)^2 \times 1116 = 627750000 \bmod n \equiv 1033 \bmod n$$

验证通过。

(2) Fiat-Shamir 协议适合于网上身份验证。假定 P 身份有 k 个秘密的数 x_{p1}、x_{p2}、$\cdots$、x_{pk}。令 $n=pq$，作

$$y_{pi} \equiv x_{pi}^2 \bmod n$$

公开文件上在 P 的姓名后面记录上

$$ID:\ y_{p1},\ y_{p2},\cdots,y_{pk}$$

S1. P 随机选择一数 $r \in Z_n$，计算

$$r^2 \bmod n$$

P 给 V 送去 (P, r^2)

S2. V 给 P 送去 $b=(b_1,b_2,\cdots,b_k)$

b_i 是随机产生的 0 或 1，即 $b_i \in \{0,1\}$，$i=1,2,\cdots,k$。

S3. P 计算

$y=rc_1c_2\cdots c_k$，并将 y 送给 V。

其中

$$c_i = \begin{cases} 1, & \text{若 } b_i = 0 \\ 0, & \text{若 } b_i = 1 \end{cases}$$

S4. V 检验，若

$$y^2 = r^2 \prod_{i=1}^{k} y_{p_i}^{b_i} \bmod m$$

则接受，否则拒绝。

实际上

$$y^2 = r^2 \prod_{i=1}^{k} (x_{p_i}^2)^{b_i}$$

不掌握 x_{p_i}，$i=1,2,\cdots,k$ 而欺骗成功的概率为 $\frac{1}{2^k}$。

9.12.4 Schnorr 身份验证

Schnorr 的身份验证也是基于求离散对数的困难性。系统的参数为 p，q 两个素数，q 是 $p-1$ 的素数因子，$g \neq 1$，且 $g^q \equiv 1 \bmod q$。P 取 x_P，计算 $y_P \equiv g^{-x_P} \bmod p$。

P：已知 x_P，y_P，p，q，g。　V：已知 p，q，g。

Schnorr 身份验证的步骤：

S1. (1) P 产生一随机数 $r_1 \in GF(p)$，$r_1 \neq 0$

(2) 计算 $S \equiv g^{r_1} \bmod p$

(3) P 将 (y_P, S) 送给 V

S2. V 产生一随机数 r_2，并将 r_2 寄给 P。

S3. P 计算 $v = r_1 + r_2 x_P \bmod p$，并将 v 寄给 V。

S4. V 校验

$$g^v = S(y_P)^{r_2} ?$$

如果相遇则接受 P，否则予以拒绝。

因　$g^v \equiv g^{(r_1 + r_2 x_P)} \bmod p \equiv g^{r_1} \cdot (g^{x_P})^{r_2} \bmod p$

$\equiv g^r \cdot (y_P)^{r_2} \bmod p \equiv S \cdot (y_P)^{r_2}$

9.13 量子密码学

量子密码学是一门很有前途的新领域，许多国家的人员都在研究它，而且在一定的范围内进行了实验。离实际应用只有一段不很长的距离。

在介绍量子密码学之前，先引进量子力学若干基本知识，其中之一是“测不准原理”。测不准原理是量子力学的基本原理。微观世界的粒子有许多共轭量，比如位置和速度，时间和能量就是一对共轭量，人们能对一对共轭量之一进行测量，但不能同时测得另一个与之共轭的量，比如对位置进行测量的同时，破坏了对速度进行测量的可能性。

量子密码学便是利用量子的不确定性，构造一安全的通信信道，使任何在信道上的窃听行为不可能不对通信本身产生影响，使达到窃听失败的目的，以保证信道的安全。

根据量子力学，微观世界的粒子不可能确定它存在任何位置，它以不同的概率存在于若干不同的地方。

同时还得介绍一物理概念，光子在传输过程会在上、下、左、右等方向产生震荡，或按一角度震荡。

当一大群光子被极化，它可在同一方向震荡，偏震器只允许被某一方向极化了的光子通过，其余则被挡住。比如一水平方向的偏震器只让在水平方向极化的光子通过。将偏震器转 90°，只有垂直方向极化了的光子能通过。

比如有一水平方向极化的光子脉冲。我们想让它们通过水平极化的偏震器。再慢慢地转动偏震器到 90°，光子通过的数目逐渐减少，直到没有光子通过为止。这是因为光子被极化了。但是量子力学中，每个微粒子都有突然切换它的极化与偏震器相配的几率，若角度很小，其概率比较大，当角度达到 90°，其概率为零。当角度为 45°时，通过偏震器的概率只有 1/2。

极化可以在任意两个正交的方向进行测量，比如水平和垂直两个方向，或称正交轴的方向；也可以是左分角线和右分角线两个方向，或称对角线方向。若光子脉冲在某一坐标轴方向极化，可在该坐标轴方向进行测量。若在其他错误的轴向进行测量，将得到随机结果。利用以上结果可产生一会话密钥，步骤如下：

S1. A 送给 B 一串光子脉冲，每一脉冲随机地沿水平、垂直、左分角线、右分角线 4 个方向极化。例如 A 送 B

| · / — — \ | — /

S2. B 有一极化偏震接收器，B 将极化偏震接收器用以接收水平、垂直的极化方向（或两对角方向），但不可能两者兼顾，因为量子力学原理不允许，测其一必然破坏另一方面测量的可能性。B 可以随机地设置接收器的方向。例如为

× + + × × × +

若 B 设置的接收器方向和 A 设置的方向一致，则 B 收到的极化方向将和 A 发出的一致。比如 B 的接收器方向正好是正交轴的方向，A 的脉冲极化方向也是正交轴方向，则 B 便可收到 A 发出的光子极化方向的光子。如若 B 的接收器装置测量对角方向，而 A 的脉冲是极化正交轴方向，则收到的将是随机的，本例若 B 得到

/ | — \ / \ — /

S3. B 通过非保密信道告诉 A 他的接收器的设置。

S4. A 告诉 B，哪些设置是正确，也就是说与 A 的脉冲极化方向一致，例如本例为 2，6，7，9。

S5. A 和 B 只能肯定这些极化方向是正确的，本例为只有以下的位是肯定的，* 是不可信的位的标志。

* | * * * \ — * — *

A 和 B 利用预先安排的编码，比如水平和左对角线为 1，垂直和右对角为 0，于是他们得 4 位码

0 0 1 1

A 和 B 依此办法可获得足够多的位。

B 有 50%的概率猜测正确，A 大致可通过 $2n$ 次光子脉冲以产生 n 个比特位。可用来作为对称密码的密钥，还可以产生足够多的比特位作为一次一密的密钥流。

应该说窃听者也可以和 B 一样进行测量，他也和 B 一样有 1/2 的出错概率，问题是窃听者的猜测错误将光子极化，导致 A 的判断和 B 实际收到的串有不同。步骤 S3，B 告诉 A 他的接收器的设置，A 可以从此推断 B 接收到的正确的位，A 和 B 只要牺牲若干位进行比较，若出现矛盾，则可断定窃听者在行动，若比较结果没有矛盾，他们只要牺牲作比较的几位，用余下的部分。

第 10 章　密 钥 管 理

密钥管理是密码的重要核心，公钥密码就是为了解决密钥管理而提出的，但是它的效率低。密钥若被泄露，则保密便不复存在，特别是利用对称密码通信的系统，这问题尤为突出。所有用户在 KDC 都储存着各自的密钥。窃取密钥成为攻击者的目标，特别是预防潜伏于内部的敌人作案。这里介绍的密钥管理是 IBM 的方案。

10.1　密 钥 等 级

密钥分为三级，最高一级有两个主密钥 $k_0^{(m)}$，$k_1^{(m)}$；中间一层是终端密钥 k_t，最下一层才是会话密钥 k_s。

会话密钥是用来加密信息的。

$k_0^{(m)}$ 和 $k_1^{(m)}$ 是分别用来加密会话密钥 k_s 和终端密钥 k_t 的。也就是说会话密钥 k_s 一旦离开密码装置就利用 $k_1^{(m)}$ 加密，而终端密钥在存储区内都是以

$$E_{k_1^{(m)}}(k_t)$$

形式出现。

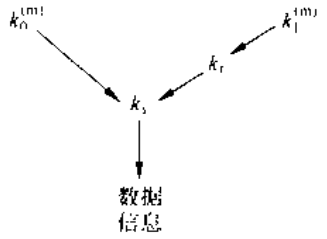


图　10.1

三级密钥的关系可用图 10.1 表示。

终端密钥 k_t 和会话密钥存放在密码装置之外，一是为了减轻密码装置的负担，也为了方便操作系统的存取控，也就是对这些密钥的存取控制。主密钥必须放在密码装置内部，密码装置是有防攻击的设施的。

10.2　主机对数据信息的加密解密操作

主机的密码装置可以被利用来对终端来的数据信息进行加密或解密。其操作过程如图 10.2 所示。

它不是简单的加、解密问题，因为会话密钥 k_s 是用 $k_1^{(m)}$ 主密钥加密形式出现的。

会话密钥的分配。

图 10.3 是说明如何用密码操作来分配会话密钥的。因为它送到终端是以

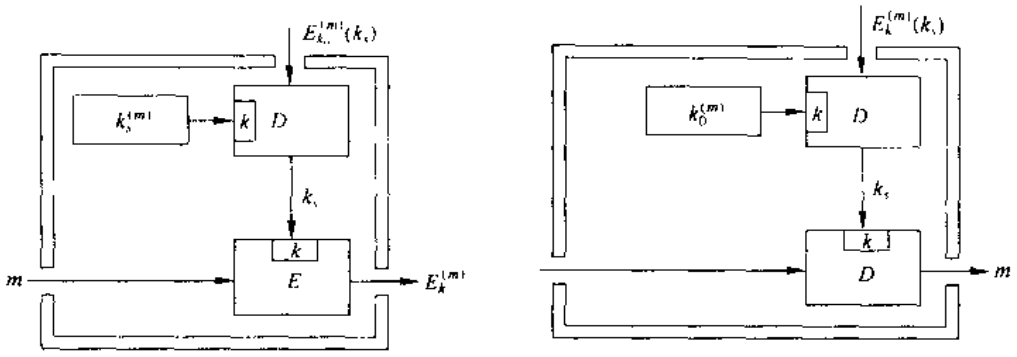
$$E_{k_t}(k_s)$$

出现的。而 k_t 和 k_s 都需要在密码装置外面用加密形式来保护的。k_t 是用主密钥 $k_1^{(m)}$ 加密的形式储存。操作的第一步是解密得 k_t。然而 k_s 是用

$$E_{k_0^{(m)}}(k_s)$$

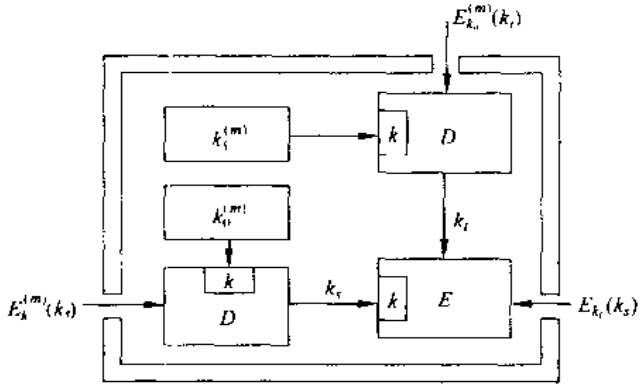
形式储存。首先在密码装置内部解密，然后再用 k_t 加密得

$$E_{k_t}(k_s)$$



(a) 主机加密　　(b) 主机解密

图　10.2



RFMK

图　10.3

这个过程以从主密钥加密改为用终端密钥加密。用 RFMK 表示它。RFMK 是 Reencipher From Master Key 的缩写。会话密钥是随机产生的，但为了防止潜伏于系统内部的敌人搭线窃听，故不允许在密码装置以外以明文形式出现，所以改为随机产生 $E_{k_0^{(m)}}(k_s)$，$E_{k_0^{(m)}}(k_t)$ 反正也是随机的。

一点注意事项，即主密钥为什么有 $k_0^{(m)}$，$k_1^{(m)}$ 之别，如若不然，比如说只有一个主密钥，k_s 有可能在密码装置以外的地方，泄露如图 10.4 所示。

若 k_t 也用 $k_0^{(m)}$ 加密，则解密装置就可以使 k_s 泄露。产生和安装一级密钥是十分麻烦的事。IBM 方案是从 $k_0^{(m)}$ 产生 $k_1^{(m)}$，只要对其中一个的位倒置即可，反正它们都是保密的，而且它们加密的性质也不一样。

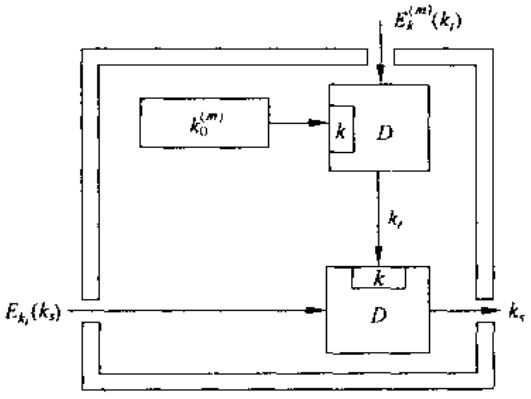


图　10.4

10.3 终端密钥分配的产生

会话密钥可频繁地改变，但终端密钥则不然。当它需要改变时，必须将新的终端密钥取到终端，然后注入终端主机，要求整个过程不以明文形式出现。最后还要以 $E_{k_1^{(m)}}(k_t)$ 形式储存。办法是这样的：用到一个特殊的数 $E_{k_1^{(m)}}(k_1^{(m)})$，这个数在系统一开始就加以生成。还要用到 RFMK 装置，如图 10.5 所示。

至于 $E_{k_1^{(m)}}(k_1^{(m)})$ 的产生如图 10.6 所示。

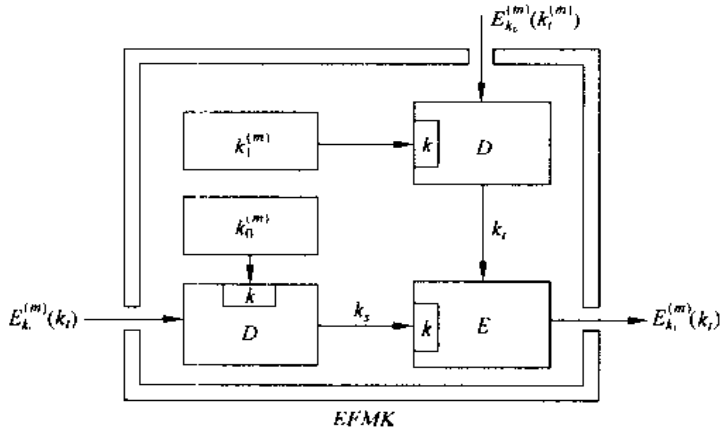


图 10.5

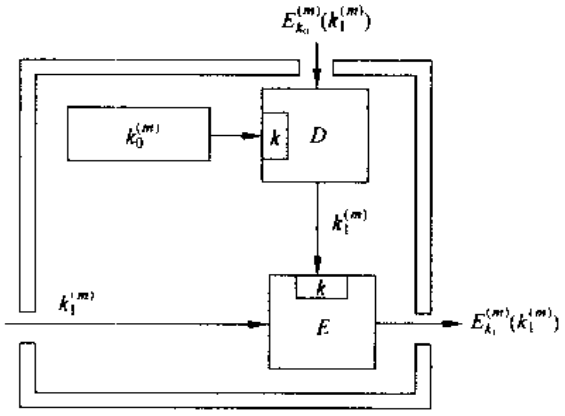


图 10.6

$E'_{k_0^{(m)}}(k_1^{(m)})$ 则利用 EFMK 装置产生，一旦这过程结束，除了储存在密码装置内的数以外，主密钥储存的值全部销毁。

前面已讨论过 $E_{k_0^{(m)}}(k_t)$ 不允许存在，它将导致暴露 k_t 本身。所以这个过程必须十万分小心，要注意软件的可靠性，还有中间数据有没有被复制储存的可能。

10.4 文件保密的密钥管理

在多域网有多个主机和终端，各终端的文件保密的密钥结构和通信加密的结构十分相似，也分三级。第一级是主密钥 $k_0^{(m)}$，$k_2^{(m)}$。$k_0^{(m)}$ 是用来对文件加密密钥 k_f 加密的。$k_2^{(m)}$ 是对二级密钥 k_g 加密用的，加密的结果放在主机。第三级 k_f 是对文件加密用的，加密的结果 $E_{k_g}(k_f)$ 放在文件头上，无需保密，因不掌握 k_g，无法得知 k_f。而 k_g 是由文件的主人掌握。一个 k_g 管一批文件，将 k_g 送到用户手上，就可以将一批文件交给他。

文件密钥的生成和检索。

图 10.7 表达一种密码操作，可用于检索一文件密钥，为对一文件加密和解密。称之为 RTMK(即 Reencipher To Master Key 的缩写)。

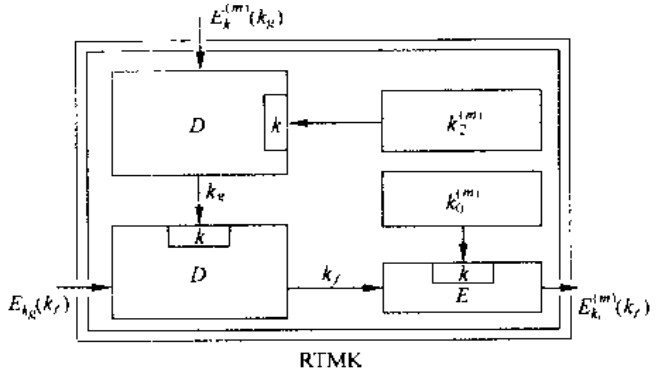


图 10.7

图 10.7 的上方输入的是 $E_{k_2^{(m)}}(k_g)$，它是储存在主机、从左方输入的另一个数是新生成的随机数，在密码机内部产生了 k_f、k_f 也是随机数，输出的是由 $k_0^{(m)}$ 加密的文件密钥。文件密钥不可在密码装置外部暴露。这个操作过程对加密和解密都是需要的。

$E_{k_g}(k_f)$ 可以和文件在一起。如果需要解密，这个数通过 RTMK 操作产生 $E_{k_0^{(m)}}(k_f)$，作为解密的密钥。这样 RFMK 和 RTMK 几乎是互为逆操作，但是是用不同的主密钥：$k_1^{(m)}$ 和 $k_2^{(m)}$。

当 $k_1^{(m)}$ 和 $k_2^{(m)}$ 相同时，RTMK 可以作为 RFMK 的逆。$E_{k_f}(k_s)$ 被窃取，可以用来产生 $E_{k_0^{(m)}}(k_s)$，可用于解密。

两个主机间的文件传输。

一个主机的密钥管理已讨论于前，将之推广到多个主机间的保密通信，设 $k_{0,i}^{(m)}$ $k_{0,j}^{(m)}$ 分别表示主密钥。令第 i 点建成的会话密钥为 $E_{k_{i,0}^{(m)}}(k_s)$，RFMK 操作可用来对 k_s 用终端密钥加密。

用 k_f^{ij} 表示从 i 点到 j 点的终端密钥，相当于终端密钥 k_f，也是二级密钥。图 10.8 表示会话密钥 k_s 从 i 点通过中间形式：

$$E_{k_f^{ij}}(k_s)$$

传输到 j 点储存。这个会话密钥可用于加密和解密。

图 10.8 的操作是可能的。因为在发出点 i 用主密钥 $k_{1,i}^{(m)}$，对接收点则用 $k_{1,j}^{(m)}$。我们已讨论如何利用 $E_{k_1^{(m)}}(k_1^{(m)})$ 取得 $E_{k_1^{(m)}}(k_t)$，同样可以建立 $E_{k_2^{(m)}}(k_t)$，$E_{k_1^{(m)}}(k_2^{(m)})$ 这个数是需要，它也在系统建立时生成，然后储存。显然，这些步骤用于 k_t''，加密储存在点 i。

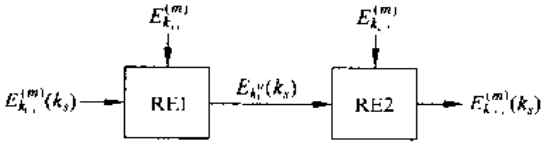


图 10.8

加密的文件的传输。

将一文件从 i 点传输到 j 点，其操作可如下图 10.9 所示。

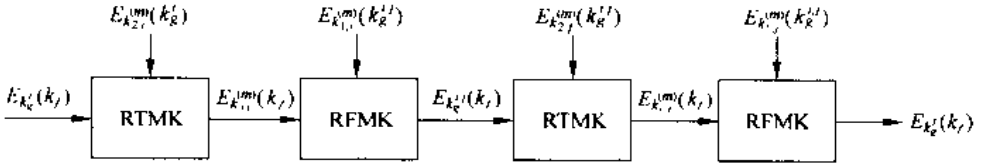
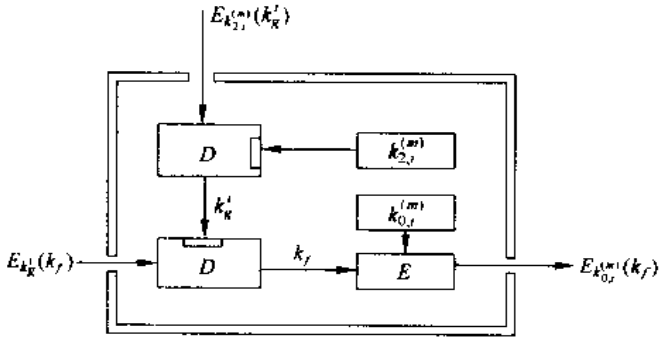


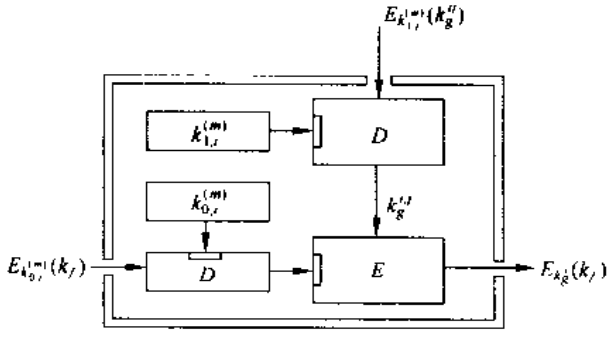
图 10.9

其结果是从 $E_{k_g'}(k_f)$ 转换为 $E_{k_g''}(k_f)$，从这过程来看安全完全依赖于在主机内的存取控制和管理功能。

现将图 10.9 的前两级分别通过 RTMK，RFMK 的输入，输出关系表示如图 10.10。



RTMK
(a)



RFMK
(b)

图 10.10

第 11 章　信息的认证技术

网上的数据安全除了保密之外，还有防止“假冒”和“篡改”，信息的保密则通过“密码”加密方法。这一章集中讨论防止假冒或篡改。当然通过“加密”也能作到“防伪”。因为对方不掌握通信密钥，无法造假。有时信息本身无需保密，但必须确认的确是真的，也就是确认它的确是发信方发出的，一律进行加密，代价太高了，并不可取。传统密码通信双方都掌握相同的密钥，一方可以造假，另一方可以抵赖，如果发生纠纷无法判断，数字签名，则可解决这个问题。

11.1　Hash 函 数

在讨论 Hash 函数前，先介绍单向函数。

定义 11.1　函数 $f: A \to B$ 若满足如下两条件，则称为单向函数。

(1) 对数 $x \in A$，计算 $y = f(x)$ 是容易的，其 $y \in B$。

(2) 对于 $y \in B$，求 $x \in A$，使满足

$$f(x) = y$$

是计算上不可能的。

设 A 是 $\{0,1\}$，A^*：0,1 符号串构成的集合，设 h：将 A^* 映射为 A^n，即将任意长度的 0,1 符号串映射为固定长度 n 的 0,1 符号串，h 满足以下三个性质之一，称为安全保密的 Hash 函数。

H1. H 是单向函数。即对于几乎所有的 $y \in A^n$，求 $x \in A^*$ 使 $H(x) = y$ 在计算上是不可能的。

H2. 已知 x，找 $x^* \in A^*$，使 $H(x) = H(x^*)$ 在计算上是不可能的。

H3. 找一对 x 和 x^*，$x \neq x^*$，使 $H(x) = H(x^*)$ 也是计算上不可能的。

信息认证码记为 MAC(Message Authentication Code)，它是利用密码产生的长度固定的一段数据，作为认证用，附在信息后面，它的应用模式有以下几种：

记 $H_k(m)$ 表示含有参数 k 的 Hash 函数，k 可以是密钥。

(1) 最常用也是最基本的是 A 向 B 送去信息 m 的同时，附上他们之间通信用的密钥 k 产生的 $H_k(m)$，即

$$\mathrm{A} \xrightarrow{(m, H_k(m))} \mathrm{B}$$

B 收到后，利用他们共享的密钥 k，检验 $H_k(m)$ 是否正确，设 B 计算得 $H_k'(m)$ 若和 $H_k(m)$ 相等，则 m 得到确认、予以接受，否则拒绝。因他们间的通信密钥是相同的，故 $H_k(m)$ 应该相同。

(2) A 向 B 送去 $(m, E_k(H(m)))$，即 $H(m)$ 用密钥 k 加密得 $E_k(H(m))$ 附在 m 后面

寄给 B，B 首先解密得 $H(m)$，再按(1)的步骤予以检验 $H(m)$以确定是否接受。这里 $H(m)$可以是不含密钥 k。$H(m)$的长度固定，$E_k(H(m))$开销有限。

(3) A 向 B 寄去$(m, H(m))$的密文 $E_k(m, H(m))$，B 收到后首先解密得$(m, H(m))$，然后计算 $H(m)$得 $H'(m)$，检验 $H(m)$是否与 $H'(m)$相等。若 $H'(m) \neq H(m)$，说明 m 被篡改了。

(4) A 向 B 送去$(m, D_A(H(m)))$，其中 $D_A(H(m))$是 A 对 $H(m)$利用公钥密码系统中 A 的密钥进行的数字签名。B 收到后利用 A 的公钥 E_A 作

$$E_1(D_A(H(m))) = H(m)$$

然后如(3)一样，对 $H(m)$进行检验。

(5) A 向 B 寄去的是

$$E_k(m, D_A(H(m)))$$

与(4)不同的是对$(m, D_A(H(m)))$加密，B 收到后需先解密，按(4)的办法处理。

(6) A 向 B 寄去的是$(m, H(m \| S))$，S 是双方约定的鉴别用的信息，$\|$ 是字符联接的符号。

(7) A 向 B 寄去的是

$$E_k(m, H(m \| S))$$

与(6)不同的是对$(m, H(m \| S))$加密。

11.2 DSA 算法

这里的 DSA 是 Decimal Shift and Add 的缩写，意即只用到移位和求和两种运算。这算法是由 Sievi 提出的，请 ISO 推荐作双方通信的 MAC。它将信息看作是十进制数，即将 m 分划成一组 10 位十进制数的序列。若最后一组不够 10 位，可补以 0 使达到 10 位。在介绍 DSA 算法之前，先引入两个符号。

(1) $R(d)m$ 表示让 m 向右旋转移动 d 位后的结果。例如 $m=1234567890$，则旋转右移 3 位后即

$$R(3)m = 8901234567$$

(2) $S(d)m$ 表示 $R(d)m+m \bmod 10^{10}$。

例 11.1 $m=1234567890$

$$\begin{aligned} R(3) \equiv\ & 8901234567 \\ & 8901234567 \\ & \underline{1234567890} \end{aligned}$$

所以 $$S(3)m = 0135802457$$

下面通过实例介绍 DSA 算法，不作形式化叙述。若收发信双方必须掌握的两个密钥 k_1 和 k_2，都是两个 10 位十进制数。设

$$k_1 = 5379224863$$

$$k_2 = 2562357284$$

$$m = 238643781322356376$$

将 m 分为

$$m_1 = 2386437813$$
$$m_2 = 2235637600$$

原来 m_2 不是 10 位，补上两个 0。

第 1 步

$$m_1 = 2386437813$$
$$\underline{k_1 = 5379224863}$$
$$p_1^{(1)} \equiv m_1 + k_1 \equiv 7765662676 \bmod 10^{10}$$
$$m_1 = 2386437813$$
$$\underline{k_2 = 2562357284}$$
$$p_2^{(1)} \equiv m_1 + k_2 \equiv 4948795097 \bmod 10^{10}$$

k_1 的第 1 位为 5，k_2 的第 1 位为 2。作

$$R(2)p_1^{(1)} = 7677656626$$
$$\underline{p_1^{(1)} = 7765662676}$$
$$k_1^{(1)} = S(2)p_1 \equiv 5443319302$$
$$R(5)p_2^{(1)} = 9509749487$$
$$\underline{p_2^{(1)} = 4948795097}$$
$$k_2^{(1)} = S(5)p_2 = 4458544584$$

第 2 步

$$m_2 = 2235637600$$
$$\underline{k_1^{(1)} = 5443319302}$$
$$p_1^{(2)} = 7678956902 \bmod 10^{10}$$
$$m_2 = 2235637600$$
$$\underline{k_2^{(1)} = 4458544584}$$
$$p_2^{(1)} = 6694182184 \bmod 10^{10}$$

k_1 的第 2 位为 3，k_2 的第 2 位为 5，作

$$R(5)p_1^{(2)} = 5690276789$$
$$\underline{p_1^{(2)} = 7678956902}$$
$$z_1 = S(5)p_1^{(3)} = 3369233691$$
$$R(3)p_2^{(1)} = 1846694182$$
$$\underline{p_2^{(1)} = 6694182184}$$
$$z_2 = S(3)p_2^{(1)} = 8540876366$$
$$z_1 + z_2 \equiv 3369233691 + 8540846366 \bmod 10^{10}$$
$$\equiv 1910110057$$

或

$$z_1 = 3369233691$$
$$\underline{z_2 = 8540876366}$$
$$z_1 + z_2 \equiv 1910110057$$

用 z_1+z_2 作为 m 的鉴别码，随着 m 寄给收信方。

例 11.2 若 m 为 23864378132235637600852143589 1

$$k_1 = 8642136534$$

$$k_2 = 4267652345$$

则

$$m_1 = 2386437813$$

$$m_2 = 2235637600$$

$$m_3 = 8521435891$$

第 1 步

$$\begin{aligned} m_1 &= 2386437813 \\ \underline{+\,k_1} &\underline{= 8642136534} \\ p_1^{(1)} &= 1028574347 \bmod 10^{10} \end{aligned}$$

$$\begin{aligned} m_1 &= 2386437813 \\ \underline{+\,k_2} &\underline{= 4267652345} \\ p_2^{(1)} &= 6654090158 \bmod 10^{10} \end{aligned}$$

k_1 的第 1 位为 8，k_2 的第 1 位为 4，

计算 $S(4)p_1^{(1)}$：

$$\begin{aligned} & 4349102857 \\ & \underline{1028574349} \\ k_1^{(1)} &= 5377677206 \end{aligned}$$

计算 $S(8)p_2^{(1)}$：

$$\begin{aligned} & 5409015866 \\ & \underline{+6654090158} \\ k_2^{(1)} &= 2063106024 \end{aligned}$$

第 2 步

$$\begin{aligned} m_2 &= 2235637600 \\ \underline{k_1^{(1)}} &\underline{= 5377677206} \\ p_1^{(2)} &= 7613314806 \end{aligned}$$

$$\begin{aligned} m_2 &= 2235637600 \\ \underline{k_2^{(1)}} &\underline{= 2063106024} \\ p_2^{(2)} &= 4298743624 \end{aligned}$$

k_1 的第 2 位为 6，k_2 的第 2 位为 2，

计算 $S(2)p_1^{(2)}$：

$$\begin{aligned} R(2)p_1^{(2)} &= 0676133148 \\ \underline{p_1^{(2)}} &\underline{= 7613314806} \\ k_1^{(2)} &= S(2)p_1^{(2)} = 8289447954 \end{aligned}$$

计算 $S(6)p_2^{(2)}$：

$$R(6)p_2^{(2)} = 7436244298$$
$$\underline{p_2^{(2)} = 4298743624}$$
$$k_2^{(2)} = S(6)p_2^{(2)} = 1734987922$$

第 3 步

$$m_3 = 8521435891$$
$$\underline{k_1^{(2)} = 8289447954}$$
$$p_1^{(3)} = 6810883845$$

$$m_3 = 8521435891$$
$$\underline{k_2^{(2)} = 1734987922}$$
$$p_2^{(3)} = 025642813$$

k_1 的第 3 位为 4，k_2 的第 3 位为 6，

计算 $S(6)p_1^{(3)}$：

$$R(6)p_1^{(3)} = 8838456810$$
$$\underline{p_1^{(3)} = 6810883845}$$
$$z_1 = 5649340655$$

计算 $S(4)p_2^{(3)}$：

$$R(4)p_2^{(3)} = 3813025642$$
$$\underline{p_2^{(3)} = 0256423813}$$
$$z_2 = 4069449455$$
$$5699340655$$
$$\underline{4069449455}$$
$$z_1 + z_2 \equiv 9768790110$$

故给 m 以 $z_1 + z_2 \bmod 10^{10}$ 作为认证码，即以

9768790110

作为认证码，附在 m 的后面寄给收信方。DSA 容易实现，不需要什么基础。

11.3 利用 DES 构造 Hash 函数

利用密码构造信息认证码 MAC，是密码用途最广的一种应用。下面介绍利用 DES 构造 Hash 函数的若干模式，实际上也是利用和 DES 类似、明、密文长度和密钥长度一样的密码的构造 Hash 函数的模式。

设 $m = m_1 m_2 \cdots m_l$，$m_1, m_2, \cdots, m_l$ 都是 64 比特的明文块，k 是密钥，也是 64 比特。

(1) 第 1 种构造模式。

$H_k(m)$的过程可以形象地表示如图 11.1 所示。或形式地写成：

S1. $i \leftarrow 1, K \leftarrow k$；

S2. $K \leftarrow DES_k(m_i)$， $i \leftarrow i+1$；

S3. 若 $i \leqslant l$，则转 S2；否则作 $H_k(m) \leftarrow K$。

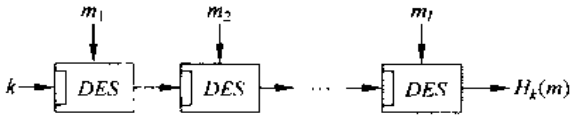


图 11.1

(2) 第 2 种构造模式，如图 11.2 所示。

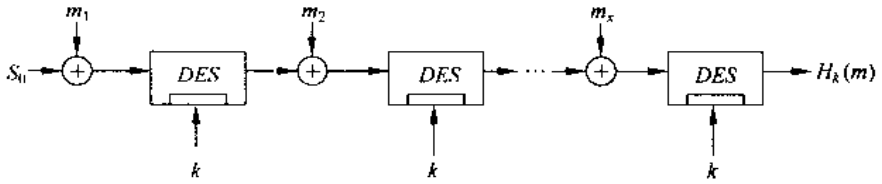


图 11.2

它的算法是：

S1. $i \leftarrow 1, S \leftarrow S_0$；

S2. $m_i \leftarrow m_i \oplus S, S \leftarrow DES_k(m_i)$；

S3. 若 $i \leqslant l$，转 S2；否则作 $H_k(m) \leftarrow S$。

其中 S_0 是预先给定的参数，它的作用也相当于密钥 k，由通信双方秘密约定。

(3) 第 3 种构造模式，如图 11.3 所示。

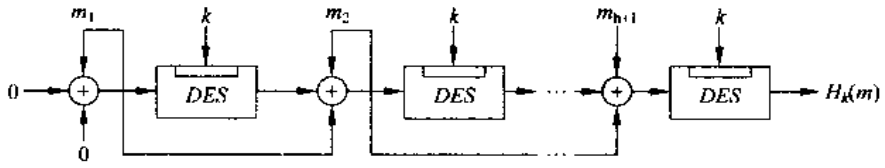


图 11.3

S1. $i \leftarrow 1, m_0 \leftarrow 0, H_0 \leftarrow 0, m_{h+1} \leftarrow V$；

S2. $H_i \leftarrow DES_k(m_i \oplus m_{i-1} \oplus H_{i-1}); i \leftarrow i+1$；

S3. 若 $i \leqslant l+1$，转 S2；否则 $H_k(m) \leftarrow H_{l+1}$。

其中 V 为和 k 类似的参数，其作用相当于密钥。H_0 和 M_0 都取为零。

(4) 第 4 种构造模式，如图 11.4 所示。

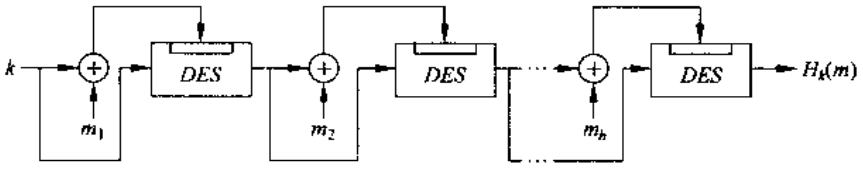


图 11.4

S1. $i \leftarrow 1, H_0 \leftarrow k$；

S2. $H_i \leftarrow DES_{m_i} \oplus H_{i-1}(H_{i-1})$；

S3. 若 $i \leqslant l$，则转 S2；否则 $H_k(m) \leftarrow H_k$。

(5) 第 5 种构造模式，如图 11.5 所示。

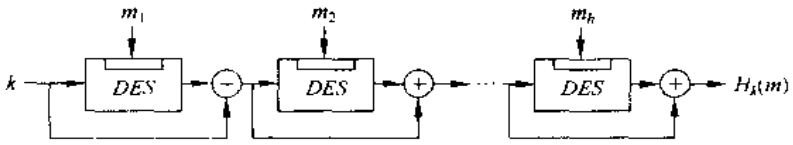


图 11.5

S1. $i \leftarrow 1, H_0 \leftarrow k$；

S2. $Hi \leftarrow DES_m(H_{i-1}) \oplus H_{i-1}$；

S3. 若 $i \leqslant l$，则转 S2；否则 $H_k(m) \leftarrow H$。

(6) 第 6 种构造模式，如图 11.6 所示。

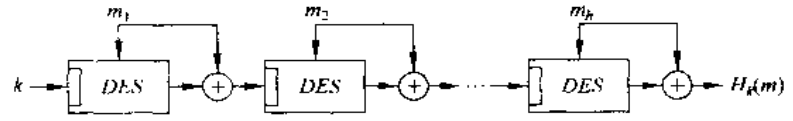


图 11.6

S1. $i \leftarrow 1, H_0 \leftarrow k$；

S2. $Hi \leftarrow DES_{H_{i-1}}(m_i) \oplus m_i; i \leftarrow i+1$；

S3. 若 $i \leqslant l$，则转 S2；否则 $H_k(m) \leftarrow H_k$。

11.4 利用 IDEA 构造 Hash 函数

实际是一种和 IDEA 类似的加密算法都适用的模式。密钥长度是明密文长度的一倍。

(1) 假定，$m = m_1 m_2 \cdots m_k$，m_i 长 64 比特，$i = 1, 2, \cdots, k$

$k = k_1 k_2 \cdots k_{64} k_{65} \cdots k_{128}$

令 $k_1 = k_1 k_2 \cdots k_{64}$

$k_r = k_{65} k_{66} \cdots k_{128}$

而 $k = k_1 \| k_r$

S1. $i \leftarrow 1, k_1 \leftarrow k_1 k_2 \cdots k_{64}$，

$k_r \leftarrow k_{65} k_{66} \cdots k_{128}$；

S2. $k \leftarrow k_1 \| k_r$，

$H_i \leftarrow IDEA(k, m_i)$，

$i \leftarrow i+1$；

S3. 若 $i \leqslant l$，则作

始

$k_r \leftarrow k_1, k_1 \leftarrow H_i$，

转 S2，

终；否则转 S4。

S4. $H_k(m)\leftarrow H_h$。

上面 $k_1 \| k_r$ 是将 k_1 和 k_r 连接的意思，下同此。

(2) 假定 k 为 64 比特的符号串(如图 11.7 所示)。

S1. $i\leftarrow 1, H_0\leftarrow k$；

S2. $K\leftarrow H_{i-1} \| m_i$，

$H_i\leftarrow IDEA(K, H_{i-1})$，

$i\leftarrow i+1$；

S3. 若 $i<l$，则转 S2，否则作

$H(k,m)\leftarrow H_h$。

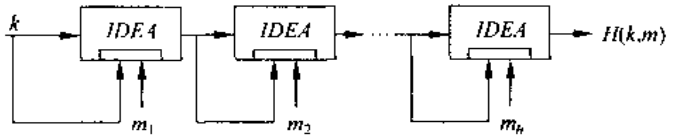


图 11.7

(3) 假定 $k=k_1k_2\cdots k_{64}k_{65}\cdots k_{128}$，令

$$k_g=k_1k_2\cdots k_{64}, k_h=k_{65}k_{66}\cdots k_{128}$$

S1. $i\leftarrow 1, G_0\leftarrow k_g, H_0\leftarrow k_h$；

S2. $w_i\leftarrow IDEA(G_{i-1} \| m_i, H_{i-1})$，

$G_i\leftarrow G_{i-1}\oplus IDEA(m_i \| w_i, G_{i-1})$，

$Hi\leftarrow w_i\oplus H_{i-1}$，

$i\leftarrow i+1$；

S3. 若 $i\leqslant l$，则转 S2，否则作 $H_k(m)\leftarrow(G_l, H_l)$，如图 11.8 所示。

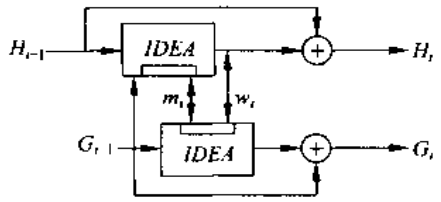


图 11.8

这个算法最后的 $H(k,m)$是 128 比特。

(4) S1. $G_0\leftarrow k_g, H_0\leftarrow k_h, i\leftarrow 1$；

S2. $G_i\leftarrow G_{i-1}\oplus IDEA(m_i \| H_{i-1}\urcorner G_{i-1})$，

$H_i\leftarrow H_{i-1}\oplus IDEA(G_{i-1} \| M_i, H_{i-1})$，

$i\leftarrow i+1$；

S3. 若 $i\leqslant l$，则转 S2，否则作

$H_{k(m)}\leftarrow G_h \| H_h$

其中$\urcorner G_{i-1}$是对 G_{i-1}按位取反运算，即 0 和 1 互换。图 11.9 中的 0 是取反的符号。

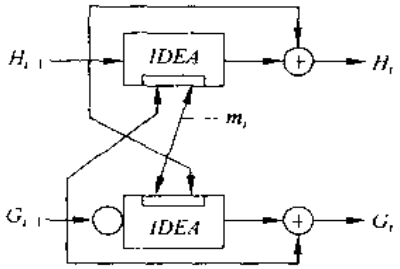


图 11.9

11.5 利用 DES 构造 Hash 函数的其他形式

(1) 64 比特的 Hash 函数对于重要的信息已不够，下面是借助 DES 构造 128 比特 Hash 函数的方法

S1. $i \leftarrow 1, H1_0 \leftarrow k_1, H2_0 \leftarrow k_2$，$k_1$ 和 k_2 都是 64 比特；

S2. $T1_i \leftarrow DES(M1_i, H1_{i-1} \oplus m2_i) \oplus m2_i$，

$T2_i \leftarrow DES(M2_i, H2_{i-1} \oplus m1_i \oplus T1_i) \oplus m1_i$，

$H1_i \leftarrow H1_{i-1} \oplus H2_{i-1} T2_i$，

$H2_i \leftarrow H2_{i-1} \oplus H2_{i-1} T1_i$，

$i \leftarrow i+1$；

S3. 若 $i \leqslant l$，则转 S2，否则作

$H(k,m) \leftarrow H1_k \| H2_k$

这里 $H1_k \cdot H2_k$ 表示将 $H1_k$ 和 $H2_k$ 连接起来，如图 11.10 所示。

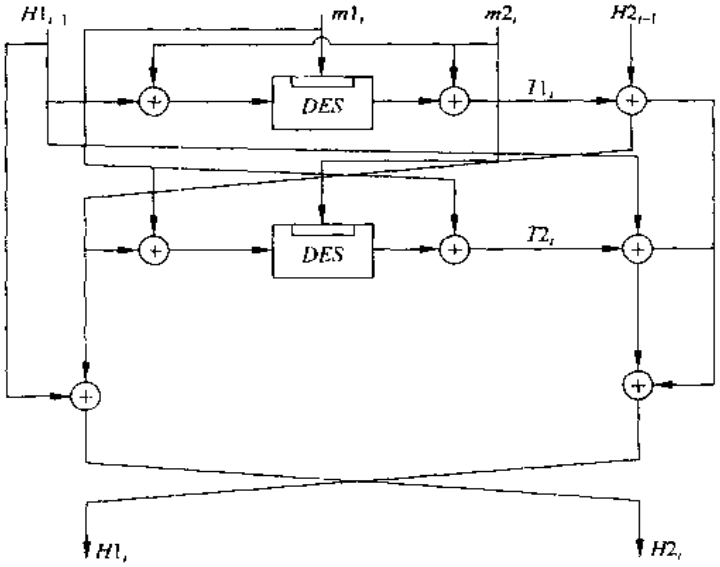


图 11.10

(2) MDS-4。MDC-2 和 MDC-4 为 IBM 公司首先开发的，利用 DES 可产生 128 比特的 Hash 值。MDC-2 用了两次 DES，MDC-4 用了 4 次 DES 加密运算，如图 11.11 所示。

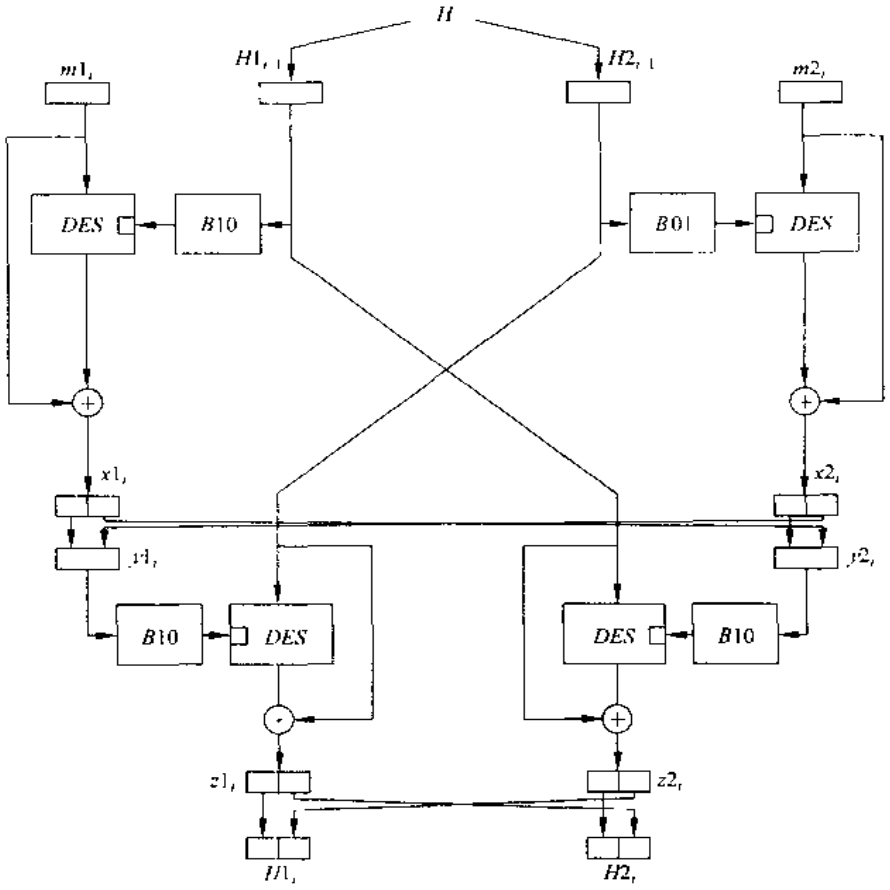


图 11.11

(3) 下面介绍 MDC-4，设已知 $m=m_1m_2\cdots m_t$，m_i 是 128 比特的 0，1 符号串，$k=k_1k_2\cdots k_{128}$，即密钥也是长 128 比特。

S1. $H1_0\leftarrow k_1k_2\cdots k_{64}$，$H2_0\leftarrow k_{65}k_{66}\cdots k_{128}$，

$M1_1\leftarrow m_1m_2\cdots m_{64}$，$m2_1\leftarrow m_{65}m_{66}\cdots m_{128}$，$i\leftarrow 1$；

S2. 若 $i>1$，则转 S4，否则作

始

$X1_i=DES(H1'_{i-1},m1_i)\oplus m1_i$，

$X2_i=DES(H2'_{i-1},m2_i)\oplus m2_i$，

其中 $H1'_{i-1}$ 为 $H1_{i-1}$ 的第 1 比特置 1，第 2 比特置 0。

其中 $H2'_{i-1}$ 为 $H2_{i-1}$ 的第 1 比特置 0，第 2 比特置 1，分别是图 11.11 中的 $B10$ 和 $B01$ 的结果。这样做的目的在于使对称性质受到破坏，避免出现 $DES(\bar{k},\bar{m})=\overline{DES(k,m)}$。

$Y1_i=$（$X1_i$ 的左 32 比特）·（$X2_i$ 的右 32 比特），

$Y2_i$=($X2_i$ 的左 32 比特)·($X1_i$ 的右 32 比特)。

$Z1_i = DES(Y1_i', H2_{i-1}) \oplus H2_{i-1}$,

$Z2_i = DES(Y2_i', H1_{i-1}) \oplus H1_{i-1}$。

其中 $Y1_i'$ 为将 $Y1_i$ 的第 1 比特置 1,第 2 比特置 0,

$Y'2_i$ 为将 $Y2_i$ 的第 1 比特置 0,第 2 比特置 1。

$H1_i$=($Z1_i$ 的左 32 比特)·($Z2_i$ 的右 32 比特),

$H2_i$=($Z2_i$ 的左 32 比特)·($Z1_i$ 的右 32 比特)。

终;

S3. $i \leftarrow i+1$,转 S2;

S4. $H(k,m) \leftarrow H1_h \| H2_h$,结束。

(4) 与 MDC-4 类似,下面也是一种利用 64 比特的分组密码构造 128 比特的 H 函数的方法。

S1. $G_0 \leftarrow V_1$, $H_0 \leftarrow V_2$, $i \leftarrow 1$;

S2. $S_i \leftarrow DES(m1_i, G_{i-1} \oplus m2_i) \oplus m2_i$,

$T_i \leftarrow DES(m2_i, H_{i-1} \oplus m1_i \oplus S_i) \oplus m1_i$,

$G_i \leftarrow G_{i-1} \oplus H_{i-1} \oplus T_i$,

$H_i \leftarrow G_{i-1} \oplus H_{i-1} \oplus S_i$,

$i \leftarrow i+1$;

S3. 若 $i \leqslant l$ 转 S2,否则转 S4;

S4. $H(m) \leftarrow G_h \cdot H_h$,结束。

其中 V_1,V_2 是初始的 0,1 符号串,各 64 比特。$M1_i$ 和 $M2_i$ 是相邻的两组信息分组,或明文块,如图 11.12 所示。

设 $m=(m1_1, m2_1)(m1_2 m2_2)\cdots(m1_l m2_l)$。

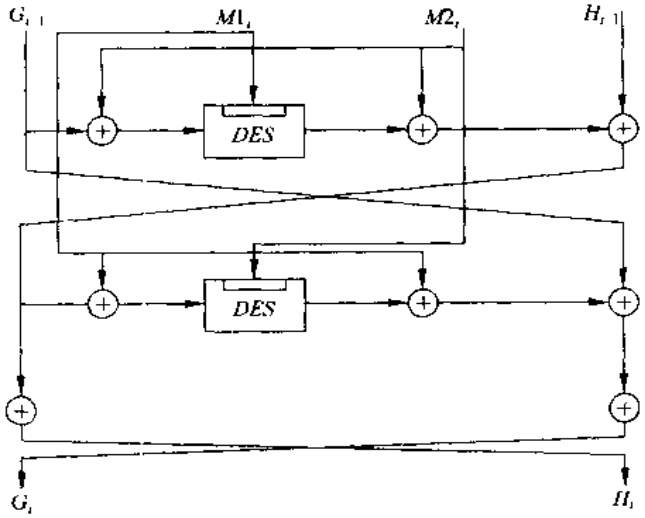


图 11.12

11.6 MD5

下面将介绍 MD5 的算法。

MD5 是由 R. Rivest 在 MIT 开发研究的 Hash 函数，它和前面介绍的利用 DES 或 IDEA 构造的 Hash 函数不一样，它仅仅是一种压缩函数，不含任何参数，即不论多长的信息 m，通过 MD5，最后输出都是 128 比特的 0,1 符号串。该符号串的每一比特都是 m 的每一比特非常复杂、非常敏感的函数。

这样出来的 Hash 函数值是靠密码来加密传输的。比如利用 RSA 系统的签名功能来完成。说得具体些，A 欲向 B 寄去信息 m，A 计算

1、$H=MD5(m)$。

2、$S\equiv H^{d_A} \bmod n_A$。

A 向 B 寄去(m,S)，B 收到后对 S

1. 计算 $H'=MD5(m)$。

2. 计算 $H\equiv S^{e_A} \bmod n_A$。

3. 若 $H=H'$ 则接受，否则拒绝。

还可将(m,S)储存于数据库，当然这仅是用法的一种，利用 MD5 的步骤；

S1. 信息是 m 被补充使其长度比特数 b 满足

$$b \equiv 448 \bmod 512$$

也就是说补充上去的位数，使之作为 512 的倍数少 64 比特，例如 m 的位数本身就是 512 比特。则补充上 448 比特达到 960 比特，补充的内容除第一位 1 以外，跟随以足够的 0，如图 11.13 所示；

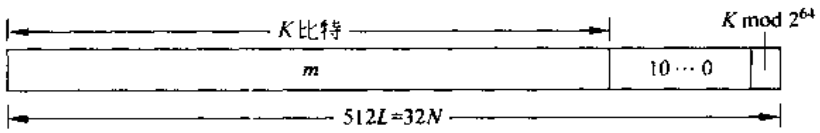


图 11.13

S2. 附上 64 比特以表示原来信息 m 长度的比特数。

若 m 的原来长度超过 2^{64} 比特，只保留低阶的 64 比特。

S1 和 S2 的输出是 512 比特一组，设为

$$Y_0, Y_1, \cdots, Y_{L-1}$$

故最后的信息长度为 $512L$，即 L 组每组 16 个字，每字 32 比特。令 $M_0[0,1,\cdots,N-1]$为最后的字序列串，$N=16L$；

S3. 128 比特的缓冲区用来存放中间结果及最后结果，128 比特分为(A,B,C,D)4 个寄存器，每一寄存器 32 比特，它们的初值(用十六进制数表示)：

$$A = 0x01234567$$
$$B = 0x89abcdef$$
$$C = 0xfedcba98$$
$$D = 0x76543210$$

S4. 将信息处理成 16 个字块：算法的核心是将信息压缩：共 4 轮，如图中的 MD5，它的逻辑如图 11.14 所示。

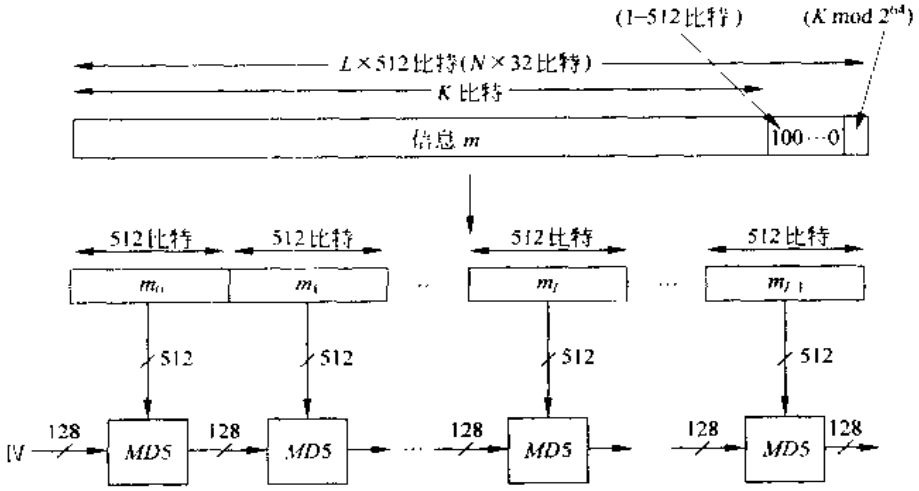


图 11.14

其中 4 轮的结构相似，但每个轮各有一逻辑函数，分别用 FF,GG,HH,II 表示它，如图 11.15 所示。其中 FF,GG,HH,II 分别是 $FF(a,b,c,d,M[k],S,T[i])$，$GG(a,b,c,d,M[k],S,T[i])$，$HH(a,b,c,d,M[k],S,T[i])$，$II(a,b,c,d,M[k],S,T[i])$的缩写，它们的运算在后面介绍。

每一轮的输入为 512 比特(Y_q)，和 128 比特的缓冲区($ABCD$)，并更改缓冲区内容。每一轮用由 sine 的四分之一函数构造的表 $T[1\cdots64]$e，表示即

$$T[i] = \langle 2^{32} \mid \sin(i) \mid \rangle, \quad i = 1,2,\cdots,64$$

这里 i 是弧度，$T[i]$是 32 比特，用作随机数。

S5. 输出：当 $512L$ 比特处理完毕，输出是 128 比特。

其中 $M[k]$是将 512 比特的明文块再分成 16 份，每块 32 比特，$k=0,1,2,\cdots,15$，$M[k]$是其中的第 k 块。

$$FF(a,b,c,d,M[k],S,T[i])$$
$$a \leftarrow b + ((a + F(b,c,d) + M[k] + T[i]) <<< S)$$

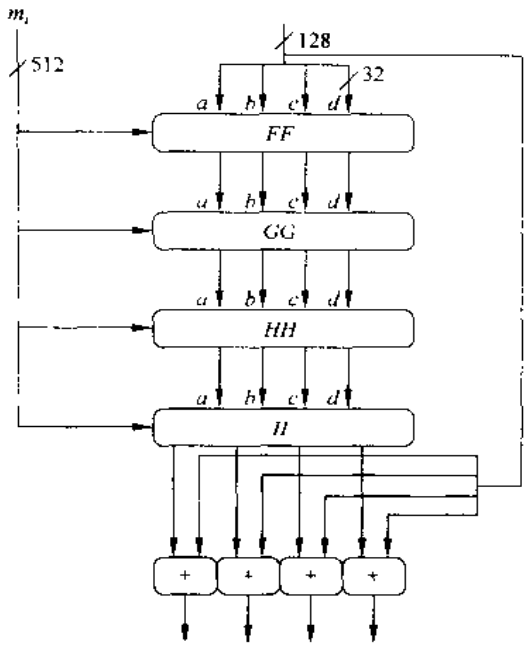


图 11.15

$T[i]$表

$T[1]$=D76AA478	$T[17]$=F61E2562	$T[33]$=FFFA3942	$T[49]$=F4292244
$T[2]$=E8C7B756	$T[18]$=C040B340	$T[34]$=8771F681	$T[50]$=432AFF97
$T[3]$=242070DB	$T[19]$=265E5A51	$T[35]$=699D6122	$T[51]$=AB9423A7
$T[4]$=C1BDCEEE	$T[20]$=E9B6C7AA	$T[36]$=FDE5380C	$T[52]$=FC93A039
$T[5]$=F57C0FAF	$T[21]$=D62F105D	$T[37]$=A4BEEA44	$T[53]$=655B59C3
$T[6]$=4787C62A	$T[22]$=02441453	$T[38]$=4BDECFA9	$T[54]$=8F0CCC92
$T[7]$=A8304613	$T[23]$=D8A1E681	$T[39]$=F6BB4B60	$T[55]$=FFEFF47D
$T[8]$=FD469501	$T[24]$=E7D3FBC8	$T[40]$=BEBFBC70	$T[56]$=85845DD1
$T[9]$=698098D8	$T[25]$=21E1CDE6	$T[41]$=289B7EC6	$T[57]$=6FA87E4F
$T[10]$=8B44F7AF	$T[26]$=C33707D6	$T[42]$=EAA127FA	$T[58]$=FE2CE6E0
$T[11]$=FFFF5BB1	$T[27]$=F4D50D87	$T[43]$=D4EF3085	$T[59]$=A3014314
$T[12]$=895CD7BE	$T[28]$=455A14ED	$T[44]$=04881D05	$T[60]$=4E0811A1
$T[13]$=6B901122	$T[29]$=A9E3E905	$T[45]$=D9D4D039	$T[61]$=F7657E82
$T[14]$=FD987193	$T[30]$=FCEEA3F8	$T[46]$=E6DB99E5	$T[62]$=BD3AF235
$T[15]$=A679438E	$T[31]$=676F02D9	$T[47]$=1FA27CF8	$T[63]$=2AD7D2BB
$T[16]$=49B40821	$T[32]$=8D2A4C8A	$T[48]$=C4AC5665	$T[64]$=EB86D391

$GG(a,b,c,d,M[k],S,T[i])$；

$a \leftarrow b+((a+G(b,c,d)+M[k]+T[i]) <<< S)$，

$HH(a,b,c,d,M[k],S,T[i])$；

$a \leftarrow b+((a+H(b,c,d)+M[k]+T[i]) <<< S)$，

$II(a,b,c,d,M[k],S,T[i])$；

$a \leftarrow b+((a+I(b,c,d)+M[k]+T[i]) <<< S)$。

其中 $F(b,c,d)=(b \wedge c) \vee (\bar{b} \wedge d)$，

$G(b,c,d)=(b \wedge d) \vee (c \wedge \bar{d})$，

$H(b,c,d)=b \oplus c \oplus d$，

$I(b,c,d)=c \oplus (b \vee \bar{d})$。

+：为 mod 2^{32} 的加法运算。

最后一轮最后一步的输出 a,b,c,d 分别与开始时 a,b,c,d 的初值作+运算，结果记录在 a,b,c,d 中输出，如图 11.16 所示。

MD5 的操作过程可示于图 11.16。其中 R 在 1～4 轮分别代以 F,G,H,I。

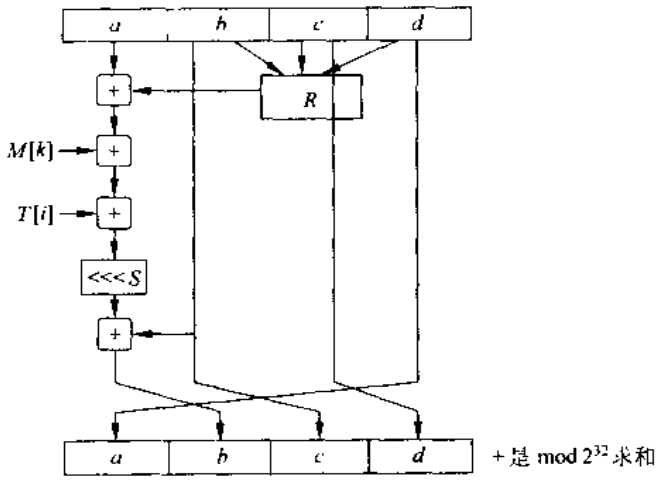


图 11.16

逻辑运算(AND,OR,NOT,XOR)分别用($\wedge$,$\vee$,$-$,$\oplus$)表示。

函数 F 是条件函数：if b then c else d；G 函数：if d then b else c；F,G,H,I 这 4 个函数的功能见表 11.1。

$M[k]$的意义已介绍于前面。在第 1 轮中 16 个字 $M[k]$正好被用上一次。从第 2 轮到第 4 轮则依次通过下面的置换来实现。

$$\rho_2(i) \equiv (1+5i) \bmod 16$$

$$\rho_3(i) \equiv (5+3i) \bmod 16$$

$$\rho_4(i) \equiv 7i \bmod 16$$

表 11.1

b	c	d	F	G	H	I
0	0	0	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	1	0	1	0	1
1	1	0	1	1	0	0
1	1	1	1	1	1	0

例如第二轮 $\rho_2(i)\equiv 5+3i \bmod 16$，依次得 5,8,11,14,1,4,7,10,13,0,3,6,9,12,15,2。64 个 32 比特的字的 T 每轮每一步被应用一次，同样注意每一步，只有 abcd 的 4 字节之一被改变。最后 4 个不同的向左循环移位被应用于每轮，而且每轮不一样，下面的算法是针对 512 比特分组进行处理的。

第 1 轮

$FF(a,b,c,d,M[0],7,0xd76aa478)$
$FF(d,a,b,c,M[1],12,0xe8c7b756)$
$FF(c,d,a,b,M[2],17,0x242070db)$
$FF(b,c,d,a,M[3],22,0xc1bdceee)$
$FF(a,b,c,d,M[4],7,0xf57c0faf)$
$FF(d,a,b,c,M[5],12,0x4787c62a)$
$FF(c,d,a,b,M[6],17,0xa8304613)$
$FF(b,c,d,a,M[7],22,0xfd469501)$
$FF(a,b,c,d,M[8],7,0x698098d8)$
$FF(d,a,b,c,M[9],12,0x8b44f7af)$
$FF(c,d,a,b,M[10],17,0xffff5bb1)$
$FF(b,c,d,a,M[11],22,0x895cd7be)$
$FF(a,b,c,d,M[12],7,0x6b901122)$
$FF(d,a,b,c,M[13],12,0xfd987193)$
$FF(c,d,a,b,M[14],17,0xa679438e)$
$FF(b,c,d,a,M[15],22,0x49b40821)$

第 2 轮

$GG(a,b,c,d,M[1],5,0xf61e2562)$
$GG(d,a,b,c,M[6],9,0xc040b340)$
$GG(c,d,a,b,M[11],14,0x265e5a51)$
$GG(b,c,d,a,M[0],20,0xe9b6c7aa)$
$GG(a,b,c,d,M[5],5,0xd62f105d)$
$GG(d,a,b,c,M[10],9,0x02441453)$
$GG(c,d,a,b,M[15],14,0xd8a1e681)$
$GG(b,c,d,a,M[4],20,0xe7d3fbc8)$
$GG(a,b,c,d,M[9],5,0x21e1cde6)$
$GG(d,a,b,c,M[14],9,0xc33707d6)$
$GG(c,d,a,b,M[3],14,0xf4d50d87)$
$GG(b,c,d,a,M[8],20,0x455a14ed)$
$GG(a,b,c,d,M[13],5,0xa9e3e905)$
$GG(d,a,b,c,M[2],9,0xfcefa3f8)$
$GG(c,d,a,b,M[7],14,0x676f02d9)$
$GG(b,c,d,a,M[12],20,0x8d2a4c8a)$

第 3 轮

$HH(a,b,c,d,M[5],4,0xfffa3942)$
$HH(d,a,b,c,M[8],11,0x8771f681)$
$HH(c,d,a,b,M[11],16,0x6d9d6122)$
$HH(b,c,d,a,M[14],23,0xfde5380c)$
$HH(a,b,c,d,M[1],4,0xa4beea44)$
$HH(d,a,b,c,M[4],11,0x4bdecfa9)$

第 4 轮

$II(a,b,c,d,M[0],6,0xf4292244)$
$II(d,a,b,c,M[7],10,0x411aff97)$
$II(c,d,a,b,M[14],15,0xab9423a7)$
$II(b,c,d,a,M[5],21,0xfc93a039)$
$II(a,b,c,d,M[12],6,0x655b59c3)$
$II(d,a,b,c,M[3],10,0x8f0ccc92)$

$HH(c,d,a,b,M[7],16,0xf6bb4b60)$　　$II(c,d,a,b,M[10],15,0xffeff47d)$

$HH(b,c,d,a,M[10],23,0xbebfbc70)$　　$II(b,c,d,a,M[1],21,0x85845dd1)$

$HH(a,b,c,d,M[13],4,0x289b7ec6)$　　$II(a,b,c,d,M[8],6,0x6fa87e4f)$

$HH(d,a,b,c,M[0],11,0xeaa127fa)$　　$II(d,a,b,c,M[15],10,0xfe2ce6e0)$

$HH(c,d,a,b,M[3],16,0xd4ef3085)$　　$II(c,d,a,b,M[6],15,0xa3014314)$

$HH(b,c,d,a,M[6],23,0x04881d05)$　　$II(b,c,d,a,M[13],21,0x4e0811a1)$

$HH(a,b,c,d,M[9],4,0xd9d4d039)$　　$II(a,b,c,d,M[4],6,0xf7537e82)$

$HH(d,a,b,c,M[12],11,0xe6db99e5)$　　$II(d,a,b,c,M[11],10,0xbd3af235)$

$HH(c,d,a,b,M[15],16,0x1fa27cf8)$　　$II(c,d,a,b,M[2],15,0x2ad7d2bb)$

$HH(b,c,d,a,M[2],23,0xc4ac5665)$　　$II(b,c,d,a,M[9],21,0xeb86d391)$

11.7 SHA

11.7.1 SHA 的描述

SHA是美国标准与技术国家研究所(National Institute of Standard and Technology)开发并公布作为信息压缩的标准,和 DSS 合并使用,DSS 是数字签名标准,将在 11.13 节讨论。

SHA 的输入长度只要求少于 2^{64} 比特,输出 160 比特。输入被分成 512 比特块。

S1. 附加信息位使达到它的长度 mod 512 与 448 同余。请注意:即使信息的长度满足要求,即 mod 512 和 448 同余。即便信息的长度即满足这个要求,附加位也是需要,所以附加的位数从 1 到 512 位。

附加的位数是包含 1 个 1 跟以足够的零,这一点和 MD5 颇为类似。

S2. 一组 64 比特附在后面,看作是 64 比特的整数,在前面为最高位,包含原来未加附加位的信息的长度。

S3. MD 缓冲器的初始化,160 比特的缓冲区用来储存中间结果及最后的 Hash 值,缓冲区分为 5 个寄存器(A,B,C,D,E),每个 32 比特,它们的初值(十六进制)

A =67 45 23 01

B =EF CD AB 89

C =98 BA DC FE

D =10 32 54 76

E =C3 D2 E1 F0

注意前 4 个值和 MD5 相同,但按 Big-Endian 模式,即最高位放在最低位地址,初值的十六进制为

字 A: 67 45 23 01

字 B: EF CD AB 89

字 C: 98 BA DC FE

字 D: 10 32 54 76

字 E：C3 D2 E1 F0

S4. 对 512 比特组的信息的处理：算法的核心是 4 轮操作，每轮 20 步，如图 11.17 所示。

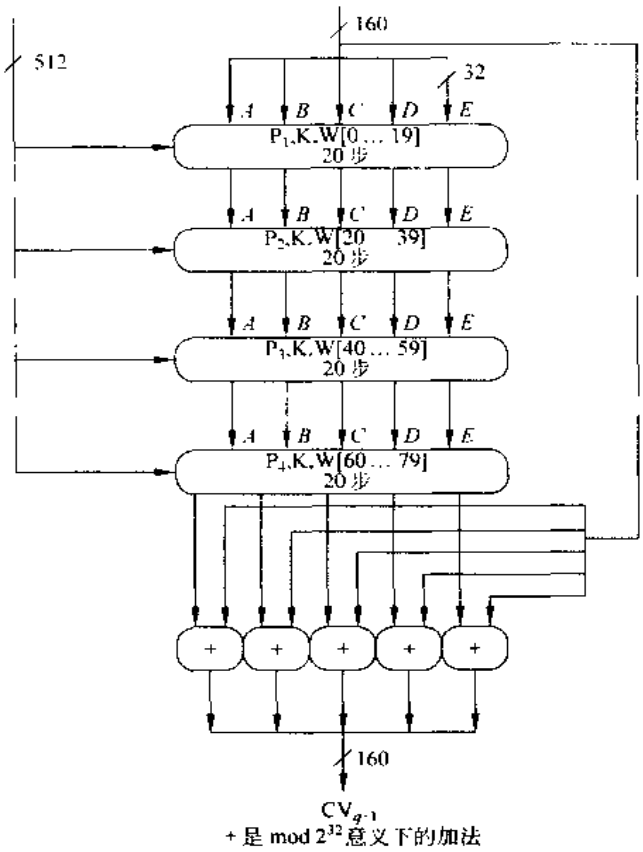


图 11.17

每轮的结构类似，但逻辑函数不同，设分别为 f_1, f_2, f_3, f_4 每轮的输入 512 比特，160 比特的输出。

$ABCDE$，每轮用的常数 K_t，$0 \leqslant t \leqslant 79$，其实只有 4 个不同常数

$$K_t = \begin{cases} 5A\ 82\ 79\ 99 & 0 \leqslant t \leqslant 19 \\ 6E\ D9\ EB\ A1 & 20 \leqslant t \leqslant 39 \\ 8F\ 1B\ BC\ DC & 40 \leqslant t \leqslant 59 \\ CA\ 62\ C1\ D6 & 60 \leqslant t \leqslant 79 \end{cases}$$

它们分别是$\lfloor 2^{30}\sqrt{2} \rfloor$，$\lfloor 2^{30}\sqrt{3} \rfloor$，$\lfloor 2^{30}\sqrt{5} \rfloor$，$\lfloor 2^{30}\sqrt{10} \rfloor$。

第四轮的输出和第一轮的输入相加产生加法是在 mod 2^{32} 意义下独立地进行。

S5. 当 $512L$ 组运算完结后，由第 L 步的输出 160 比特作为 $H(m)$。

每轮的计算在下节介绍。

11.7.2 SHA 的压缩函数

SHA 是每一步作为下面流程图所示的计算，

$$(A,B,C,D,E) \leftarrow ((E + f(t,B,C,D) + S^5(A) + W_t), A, S^{30}(B), C, D)$$

如图 11.18 所示。

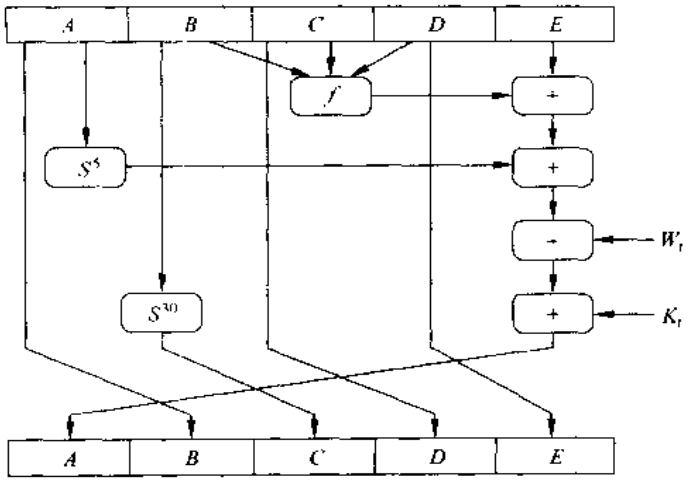


图 11.18

其中 A,B,C,D,E 为缓冲区的 5 个字。

t 为步数，$0 \leqslant t \leqslant 79$。

$f(t,B,C,D)$ 为第 t 步逻辑函数。

$<<<k$ 为向左旋转移位 k 位。

W_t 为由 512 比特的输入组导出 32 比特字的第 t 块。

K_t 为附加的常数，有 4 个不同的值，已定义于前面。

$+$ 为 mod 2^{32} 的加法。

每一个函数的输入为 3 个 32 比特字，输出是一个 32 比特字，由一系列的逻辑运算而得到的，见图 11.19。输出的第 n 位为 B,C,D 3 个输入第 n 位的函数。

$$f(t,B,C,D) = \begin{cases} (B \wedge C) \vee (\overline{B} \wedge D) & 0 \leqslant t \leqslant 19 \\ B \oplus C \oplus D & 20 \leqslant t \leqslant 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & 40 \leqslant t \leqslant 59 \\ B \oplus C \oplus D & 60 \leqslant t \leqslant 79 \end{cases}$$

$\wedge$，$\vee$，$-$，$\oplus$ 分别是逻辑运算 AND，OR，NOT，XOR 的符号，至于 W_t 它自 512 比特的信息导出 32 比特的字，如图 11.19 所示。

W_t 的前 16 个数是直接由信息的 16 个分组取出，其余的为：

$$W_t = S^1(W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-1})$$

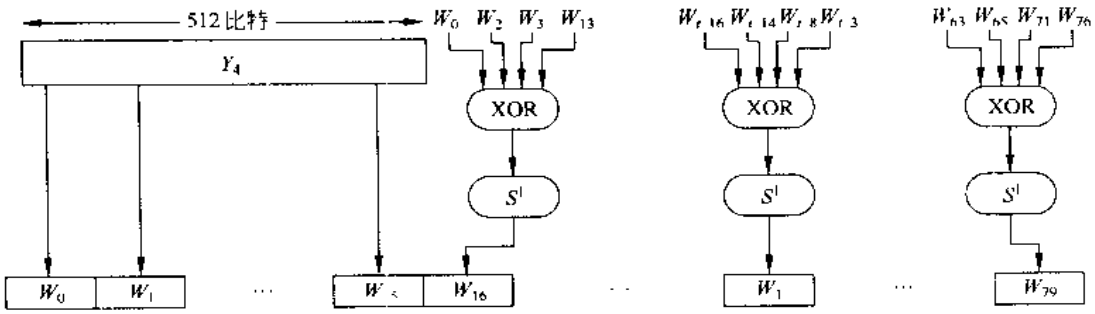


图 11.19

B	C	D	$f_{0\cdots19}$	$f_{20\cdots39}$	$f_{40\cdots59}$	$f_{60\cdots79}$
0	0	0	0	0	0	0
0	0	1	1	1	0	1
0	1	0	0	1	0	1
0	1	1	1	0	1	0
1	0	0	0	1	0	1
1	0	1	0	0	1	0
1	1	0	1	0	1	0
1	1	1	1	1	1	1

11.7.3 SHA 和 MD5 的比较

SHA 和 MD5 颇为相似。

(1) 最主要的不同点，SHA 比 MD5 长 32 比特，若采用强行攻击，SHA 显然比 MD5 抗攻击能力比较强。

(2) MD5 已有密码分析的攻击法，故较脆弱。

(3) MD5 效率比 SHA 高。

(4) 两者都比较容易实现。

(5) MD5 采用小端(Little-endian)格式，SHA 则采用大端(Big-endian)格式。

11.8 生日问题与生日攻击

问题：有多少人在一起，至少有$\frac{1}{2}$的概率存在两个人有相同的生日？这就是著名的“生日问题”，为后面讨论方便，称为Ⅰ型生日问题。

令 P_m 为 m 个人在一起，不存在相同生日的概率。根据假定 $m-1$ 个人中无相同生日的概率为 P_{m-1}，$m-1$ 人共有生日 $m-1$ 日。第 m 个人与后面 $m-1$ 人无相同生日的概率应为

$$[365-(m-1)]/365=\frac{366-m}{365}$$

故得递推关系：

$$P_m = \frac{366-m}{365} P_{m-1}$$

$$P_1 = 1$$

所以

$$P_2 = \frac{366-2}{365} P_1 = \frac{364}{365}$$

$$P_3 = \frac{363}{365} P_2 = \frac{364}{365} \cdot \frac{363}{365} = \frac{1}{365^2} \cdot \frac{364!}{362!}$$

$$P_4 = \frac{362}{365} P_3 = \frac{1}{365^3} \cdot \frac{364!}{361!}$$

$$P_m = \frac{1}{365^{m-1}} \frac{364!}{(365-m)!}$$

不难验证当 $m \geqslant 23$ 时 $P_m < 1/2$。即 23 人在一起时，无相同生日的概率小于 $\frac{1}{2}$。这结果很有点出人意料。

但是，若已知某 A 的生日日期，问有多少人在一起时，至少有 $\frac{1}{2}$ 的概率使有一人和 A 的生日相同？我们称之为Ⅱ型生日问题，假定所有的人的生日均匀分布于一年的 365 天。

设一人和 A 有相同生日的概率为 1/365，有不同生日的概率则为

$$1 - \frac{1}{365} = \frac{364}{365}$$

k 个人与 A 生日都不同的概率应为

$$\left(\frac{364}{365}\right)^k$$

k 个人至少有一人与 A 的生日相同的概率则为

$$1 - \left(\frac{364}{365}\right)^k \geqslant \frac{1}{2}$$

所以

$$\left(\frac{364}{365}\right)^k \leqslant \frac{1}{2}$$

$$k \geqslant \frac{-\ln 2}{\ln\left(\frac{364}{365}\right)} \geqslant \frac{0.6931471}{0.0027370} \geqslant 253$$

即 k 至少为 253 人。

还请注意：$2^8 < 365 < 2^9$，而 $2^4 < 23 < 2^5$ 这两组上、下界间数量关系，而 $2^8 = 256$。

与生日问题类似的，若一文件 m 的 $H(m)$ 为 32 比特，以此为例。试问至少有多少的文件在一起，其中有两个文件它的 $H(m)$ 值以至少 1/2 的概率相同。不过 365 改为 2^{32}，可得

$$P_m = \frac{2^{32} - m + 1}{2^{32}} P_{m-1}, \quad P_1 = 1$$

P_m 为 m 个长度为 32 比特的 0，1 符号串不存在两个相等的概率。类似可得

$$P_m = \frac{1}{(2^{32})^{m-1}} \frac{(2^{32}-1)!}{(2^{32}-m)!}$$

简单的验算可得 $m>2^{17}(2^{17}-131072)$时 $P_m<\frac{1}{2}$，则有相同的 0,1 符号串的概率超过$\frac{1}{2}$。

生日攻击便是根据这个原理，如若一潜伏的攻击者 E 为被攻击者起草一文件，在不改变原意的前提下，行文可以有许多无关紧要的差别。例如林先生，改为林同志，“假如”改为“如若”，“十分感激”改为“非常感谢”，若有 16 处这样的差异。于是有 2^{16}种的不同文件，令这些文件的 $H(m)$的集合为 A。

如若 E 为被攻击者起草另一文件，关键的信息相反，E 如法炮制，也留下 16 个无关紧要的差异，也有 2^{16}个文件，设这文件的 $H(m)$集合为 B。

A 与 B 的全体为 32 比特的 0,1 符号串，共 $2\times2^{16}=2^{17}$个。根据“生日问题”的结果，以$\frac{1}{2}$的概率存在两个相同的符号串。若一个在 A，另一个在 B，攻击者 E 便可以用 B 中对应的文件取代 A 中的文件，或用 A 中的文件代以 B 中的文件达到攻击的目的。比直接找一攻击文件，和确定的文件有相同的 $H(m)$要比较简单。如果说“生日攻击”是属于Ⅰ型生日问题，则后者属于Ⅱ型生日攻击。

11.9 中间相遇攻击

最典型的问题当推对两重的 DES 的一种攻击。所谓两重 DES 指的是取独立的两个密钥 k_1 和 k_2。

令

$$C = DES_{k_2}(DES_{k_1}(m))$$

相应的解密步骤为

$$m = DES_{k_1}^{-1}(DES_{k_2}^{-1}(C))$$

如图 11.20 所示。

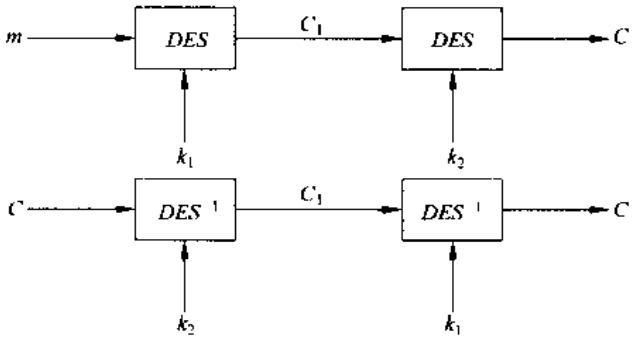


图 11.20

令 $$DES_{k_1}(m) = C_1$$

则 $$DES_{k_2}(C_1) = C$$

所以

$$C_1 = DES_{k_2}^{-1}(C)$$

$$= DES_{k_1}(m) = DES_{k_2}(C)C_1$$

可对 m 采取一切可能的 k_1 加密并储存，再对 C 取一切可能的 k_2 解密，并与储存的 $DES_{k_1}(m)$ 比较，若有相等的，则 k_1 和 k_2 便可获得。

二重 DES 并不像人们想像那样可提高密钥长度到 $2\times56=112$ 比特，根据生日攻击原理，而是相当复杂度为 2^{57} 的问题，即密钥长相当 57 比特，而不是 112 比特。

这在上面生日攻击中也有可以借鉴的。先将 2^{32} 份原文件的求其 $H(m)$ 值并储存，然后计算假文件的 $H(m)$ 值，并到储存的真 $H(m)$ 中寻找相等，而不是全部计算真为两套各 2^{33} 个文件的 $H(m)$ 值，再去寻找其中是否存在相等的一对。

对于 Hash 函数的中间相遇攻击也类似于对二重 DES 的攻击，现在不过 h 重。可将有害信息分成两部分，一部分有 r_1 个可变的，另一部分有 r_2 个可变的。一个从头向前计算到中间，另一部分从后面后退，计算到中间，在中间进行比较。如图 11.21 所示。

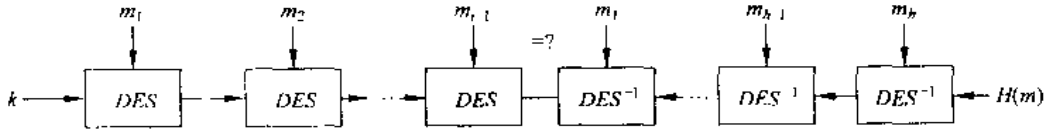


图 11.21

11.10 Lamport-Diffie 数字签名

前面讨论的认证技术主要集中于 Hash 函数方法。A 向 B 送去信息 m，附上 $H(k,m)$。k 是 A 和 B 双方通信的密钥，B 收到后，计算 $H(k,m)$ 与附上的 $H(k,m)$ 进行比较，若不等则拒绝接收。下面介绍数字签名方法。数字签名也是认证技术的一种，它还具有防抵赖的功能。如 RSA 的数字签名就是其典型，但失之大数模幂运算，速度太慢。

Lamport Diffie 数字签名是利用传统分组密码进行数字签名的一例子。

设用户 A 随机生成一密钥序列：

$$\lambda_1,\mu_1,\lambda_2,\mu_2,\cdots,\lambda_n,\mu_n,\cdots,\lambda_q,\mu_q$$

A 将这密钥秘密保存。

A 再生成随机序列

$$u_1,v_1,u_2,v_2,\cdots,u_n,v_n,\cdots,u_q,v_q$$

利用 λ_i,μ_i 分别对 u_i,v_i 加密得

$$U_i = E_{\lambda_i}(U_i),\quad V_i = E\mu_i(V_i),\quad i=1,2,\cdots,q$$

A 将下面序列公开在公钥文件上：

$$u_1,v_1,u_2,v_2,\cdots,u_n,v_n,\cdots,u_q,v_q$$

$$U_1,V_1,U_2,V_2,\cdots,U_n,V_n,\cdots,U_q,V_q$$

若 A 欲向 B 送去信息 m：

$$m = m_1m_2\cdots m_n\quad m_i = 0\text{ 或 }1\quad i=1,2,\cdots,n$$

令 $S=s_1s_2\cdots s_n$，其中

$$s_i = \begin{cases} \lambda_i, m_i = 0 \\ \mu_i, m_i = 1 \end{cases}$$

$$i = 1,2,\cdots,n$$

A 将明文 m 随同 S 送给 B。S 便是 A 的数字签名。

B 收到后认证如下。

$$E_{v_i}(u_i) = U_i, \quad 若\ m_i = 0$$

$$E_{v_i}(v_i) = V_i, \quad 若\ m_i = 0$$

$$i = 1,2,\cdots,n$$

若上面等式全部成立，则认证通过，接受 m，否则予以拒绝。

因为只有 A 才能掌握所用的 n 个密钥。这种签名的致命弱点是数据的膨胀率极大，而且密钥用过立即作废。以 DES 为例，数字签名的结果：每一比特，数字签名是 64 比特。

11.11 Fiat-Shamir 与 Schnorr 数字签名

11.11.1 Fiat-Shamir 数字签名原理

Fiat-Shamir 数字签名，有时简记为 FS 数字签名。它比 RSA 效率高得多，仅及 RSA 的百分之几。FS 签名的理论依据是求离散对数的困难性。

有一授权的管理中心 CA，它是可信赖的。已知一大数 n，它是两大素数 p 和 q 之积。n 是公开的，p 和 q 保密。只有 CA 掌握这秘密。对于用户 A，CA 产生一序列：v_1，$v_2,\cdots,v_k$，$0<v_i<n$，$i=1,2,\cdots,k$ 作为 A 的公钥。

在讨论 Fiat-Shamir 数字签名之前先引进“平方剩余”概念。

给定整数，对于任意正整数 k，$0<k<n$，并不一定存在整数 x，$0<x<n$，使

$$x^2 \equiv k \bmod n$$

若存在 x 满足上面同余式，则称 k 为 mod n，x 的平方剩余。也就是说不是任何数 k，mod n 都是平方剩余。例如 $n=35$，只有

$$1,4,9,11,14,15,16,21,25,29$$

才是平方剩余。其余如 k 为

$$2,3,5,6,7,8,10,12,13,17,18,19,20,22,23,24,26,27,28,30,31,32,33,34$$

都不存在 x，使

$$x^2 \equiv k \bmod 35$$

(1) $k=1$，$x^2 \equiv 1 \bmod 35$ 有解：

$$1,6,29,34$$

1 的逆是自身，即

$$1 \cdot 1 \equiv 1 \bmod 35$$

(2) $k=4$，$x^2 \equiv 4$ 有解：

$$2,12,23,33$$

而 4 的逆可求之如下：

$$35=8\times4+3,\quad 3=35-8\times4$$

$$4=3+1,\quad 1=4-3$$

所以

$$\begin{aligned}1&=4-3=4-(35-8\times4)\\&=4-35+8\times4\\&=9\times4-35\end{aligned}$$

所以

$$9\times4\equiv1 \bmod 35$$

$$4^{-1}\equiv9 \bmod 35$$

(3) $k=9, x^2\equiv9 \bmod$ 有解：

$$3,17,18,32$$

$$9^{-1}\equiv4 \bmod 35$$

(4) $k=15, x^2\equiv15 \bmod 35$ 有解：

$$15,20$$

但 15 没有逆，即不存在 h，使

$$15h\equiv1 \bmod 35$$

11.11.2 Fiat-Shamir 算法的签名过程

CA 给用户 A 产生公开密钥

$$V_1,V_2,\cdots,V_k$$

其中 V_i^{-1} 属于 mod n 的平方剩余，即存在 S_i，使

$$S_i^2\equiv V_i^{-1} \bmod n$$

给 A 分配密钥

$$S_i\equiv(V_i^{-1})^{1/2} \bmod n$$

$$i=1,2,\cdots,k$$

而

$$S_i^2\equiv V_i^{-1} \bmod n$$

也就是说 V_i 选择必须使它 mod n 存在逆，而且它的逆还必须是 mod n 的平方剩余。

签名过程：已知 A 有

$$V_1,V_2,\cdots,V_t\quad(\text{公开})$$

$$S_1,S_2,\cdots,S_t\quad(\text{保密})$$

S1. A 取 t 个小于 n 的随机整数。

$$r_1,r_2,\cdots,r_t$$

A 计算

$$x_j\equiv r_j^2 \bmod n$$

$$j=1,2,\cdots,t$$

S2. A 计算由 m 联接以 x_j 的 Hash 函数值：($m_j\|x_j$ 表 m_j 联接以 x_j)

$$H(m\|x_j),\quad j=1,2,\cdots,t$$

取其前面 k 位：

$$e_{ij},i=1,2,\cdots,k$$

$$j=1,2,\cdots,t$$

S3. A 计算

$$y_j \equiv r_j \prod_{e_{ij}=1} s_i$$

$$j=1,2,\cdots,t$$

A 送 B 的关于 m 的签名包含

$$\{y_j\} \text{ 和 } \{e_{ij}\}$$

$$j=1,2,\cdots,t$$

$$i=1,2,\cdots,k$$

S4. B 利用 A 的公钥计算

$$Z_j = y_j^2 \prod_{e_{ij}=1} v_i$$

$$j=1,2,\cdots,t$$

S5. B 计算

$$H(m \| Z_j), j=1,2,\cdots,t$$

取前面 k 比特：

$$e_{ij}^* \quad i=1,2,\cdots,k$$

$$j=1,2,\cdots,t$$

B 比较

$$e_{ij}^* = e_{ij}? \quad i=1,2,\cdots,k$$

$$j=1,2,\cdots,t$$

若等式全部成立，则通过认证，否则拒绝接受。

由于

$$S_i \equiv v_i^{-1} \bmod n$$

$$x_j \equiv r_j^2 \bmod n$$

$$y_j \equiv r_j \prod_{e_{ij}=1} s_i$$

$$Z_j \equiv y_j^2 \prod_{e_{ij}=1} s_i$$

实际上是判断

因为

$$z_j = x_j? \quad j=1,2,\cdots,t$$

$$y_j^2 \equiv r_j^2 \prod_{e_{ij}=1} s_i^2 \bmod n$$

$$\equiv r_j^2 \prod_{e_{ij}=1} v_i^{-1} \bmod n$$

$$z_j \equiv y_j^2 \prod_{e_{ij}=1} v_i^{-2} \equiv r_j^2 \prod_{e_{ij}=1} v_i \prod_{e_{ij}=1} v_i^{-1}$$

$$\equiv r_j^2 \bmod n \equiv x_j$$

例如 $n=35, v_1=4, v_2=11, v_3=16, v_4=29, s_1=3, s_2=4, s_3=9, s_4=8, s_i^2 \equiv v_i^{-1} \bmod 35$。A 选{4,11,16,29}作为公钥；{3,4,9,8}作为私钥。

$$r=16, x\equiv r^2=16^2\equiv 11 \bmod 35$$
$$11=1011_2, y_j x_j=16\times 3\times 9\times 8\equiv 26 \bmod 35$$
$$z=26^2\times 4\times 16\times 29=11 \bmod 35$$

11.11.3 Fiat-Shamir 数字签名的其他形式及补充

1. Fiat 和 Shamir 建议 $k\geqslant 9, t\geqslant 8$。这样$\{e_n\}$至少有 72 个元素，超过 2^{72}种状态。

2. FS 也可以改为如下步骤。

S1. A 选择 $r_j, j=1,2,\cdots,t$，计算

$$x_j \equiv r_j^2 \bmod n, j=1,2,\cdots,t$$

A 收$\{x_j\}$送给 B；

S2. B 计算

$$H(m\| x_j), j=1,2,\cdots,t$$

取前面 k 位

$$e_{ij}, i=1,2,\cdots,k$$
$$j=1,2,\cdots,t$$

B 将$\{e_{ij}\}$送给 A；

S3. A 计算

$$y_j \equiv r_2 \prod_{e_{ij}=1} s_i \bmod n$$
$$j=1,2,\cdots,t$$

A 将$\{y_j\}$送 B；

S4. B 证明

$$x_j \equiv y_j^2 \prod_{e_{ij}=1} v_i \bmod n$$
$$j=1,2,\cdots,t$$

11.11.4 Schnorr 数字签名

系统已知 p 和 q 两素数，q 是 $p-1$ 的一个素数因子，g 是本原元素 $g^q\equiv 1 \bmod p$。A 取私钥 x_A，令 $y_A\equiv g^{-x_A} \bmod p$ 作为 A 的公钥。

A 已知 x_A, y_A, p, q, g。B 已知 p, q, g。

签名过程：

S1. A 作

(1) 选一随机数 $r_1(<q)$，计算

$$R\equiv g^{r_1} \bmod p$$

(2) 将信息 m 和 R 一起作 Hash 函数

$$h=H(m\| R)$$

(3) 计算

$$y\equiv (r_1+hx_A) \bmod q$$

(4) 将(h,y)作为m的数字签名寄给B。

S2. B作

(1) 计算

$$R^* \equiv g^{y}y_A^{h} \bmod p$$

(2) 计算

$$H(m \| R^*) = h^*$$

若$h^* = h$则接受m，否则予以拒绝。

因$g^{y}y_A^{h} \bmod p \equiv g^{r_1 - hx_A}(g^{x_A})^{h} \bmod p \equiv g^{r_1} \bmod p = R$。

11.12 ElGamal数字签名

11.12.1 ElGamal系统

ElGamal的数字是基于在有限域$GF(p)$计算离散对数的困难，其中p是素数，它用于传送明文，附以数字签名。设m是明文，$m \in GF(p)$，$0 \leqslant m \leqslant p-1$系统除$p$外，选择$GF(p)$上的本原元素$g$。系统的$p$和$g$是公开的。用户A任选一元素$r_A \in GF(p)$，计算

$$k_A \equiv g^{r_A} \bmod p$$

k_A作为公钥存放公钥文件里，也是公开的。r_A作为A的私钥，由A秘密保存。设为了对信息m进行签名，发信方选一$s \in GF(p)$，要求

$$\gcd(s, p-1) = 0$$

计算$W \equiv g^{s} \bmod p$，由

$$m \equiv rW + sV \bmod (p-1)$$

解出V，A将(W,V)作为m的数字签名，即将(m,W,V)寄给B，由于

$$g^{m} \equiv g^{rW+sV} \equiv (g^{r_A})^{W}(g^{s})^{V} \bmod p$$

所以

$$g^{m} \equiv k_A^{W}W^{V} \bmod p \quad (*)$$

故B可以验证上式$(*)$是否正式，若成立则接收，并将(m,W,V)存储于数据库，否则拒绝，若A抵赖，则B可出示(m,W,V)。

例11.3 已知$p=11$，$GF(11)$的本原元素有2,6,7,8。若取$g=7$，则有$g^{0}=1$，$g^{1}=7$，$g^{2}=5$，$g^{3}=2$，$g^{4}=3$，$g^{5}=10$，$g^{6}=4$，$g^{7}=6$，$g^{8}=9$，$g^{9}=8$，$g^{10}=1$。

$r_A=3$，则公钥$k_A=g^{3}=7^{3} \bmod 11 \equiv 2$，故公钥

$$(k_A, g, p) = (2,7,11)$$

若$m=6$，选取$s=7$，满足$(s, p-1)=(7,10)=1$

$$W \equiv g^{s} \bmod p \equiv 7^{7} \bmod 11 \equiv 6$$

$$m \equiv r_A W + sV \bmod (p-1)$$

所以

$$6 \equiv 3 \times 6 + 7V \bmod 10$$

$$7V \equiv 12 \equiv 8$$

$$V \equiv 7^{-1} \cdot 8 \bmod 8$$

但

$$7^{-1} \equiv 3$$

所以 $$V \equiv 4$$

故密文$(m,W,V)=(6,6,4)$。

收信方B收到密文后，计算验证下列等式是否成立。

$$g^m \equiv k^W W^V \bmod p$$

右端：$k^W W^V \equiv 2^5 6^4 \equiv 82944 \bmod 11 \equiv 4$

左端：$g^m \bmod p \equiv 7^6 \bmod 11 \equiv 4$

故予以接受。

11.12.2 ElGamal的其他形式

设系统公开的参数 p 及 $GF(p)$的本原元素 g 已知。

发送信一方随机选取随机密钥 r，r 由收信方和发送方均秘密保存，信息 m 和 p 互素，令

$$W \equiv g^m \bmod p$$

令 Y 为密文，满足$(Y,P)=1$，从下列同解方程中解出 V。

$$Y \equiv rW + mV \bmod (p-1)$$

密文(Y,W,V)送给收信方，收信方计算

$$A \equiv (g^r)^W W^V = g^{Y-XV}(g^X)^V \bmod p$$
$$\equiv g^Y \bmod p$$

故明文 m：

$$m \equiv v^{-1}(Y - rW) \bmod (p-1)$$

例 11.4 已知 $p=11$，$g=7$ 为 $GF(11)$的本原元素，$(p,g)=(11,7)$为系统公开的参数，$r=5$ 由通信双方保存的密钥。

设 A 欲送 $m=3$，$\gcd(m,p)=\gcd(3,11)=1$，设 $Y=7$

$$Y \equiv rW + mV \equiv 5g^m + 3V \bmod 10, \quad W \equiv g^m \equiv 7^3 \bmod 11 \equiv 2$$
$$7 \equiv 5 \cdot 7^3 + 3V \bmod 10 \equiv 5$$
$$3V \equiv 7 - 5 \cdot 2 \equiv -3 \bmod 10$$
$$3V \equiv 7 \bmod 10, 3^{-1} \equiv 7 \bmod 10$$
$$V \equiv 3^{-1} \cdot 7 \equiv 7 \cdot 7 \bmod 10 \equiv 9$$

故密文：$(Y,W,V)=(7,2,9)$。

收信方收到(7,2,9)密文后，计算

$$A \equiv (g^r)^W W^V \bmod p$$
$$\equiv (7^5)^2 \cdot 2^9 \bmod 11$$
$$\equiv (10)^2 \cdot (2)^9 \bmod 11$$
$$\equiv 100 \cdot 512 \bmod 11$$
$$\equiv 1 \cdot 6 \equiv 6$$

另一方面 $$g^Y \bmod p \equiv 7^7 \bmod 11 \equiv 6$$

认证得到验证。

明文 m 满足：

$$Y \equiv rW + mV \bmod (p-1)$$

$$7 = 5 \cdot 2 + 9m \bmod 10$$

$$9m \equiv 7 - 10 \equiv -3 \equiv 7 \bmod 10$$

$$m \equiv 9^{-1} \cdot 7$$

$$9^{-1} \equiv 9 \bmod 10$$

$$m \equiv 9 \cdot 7 \equiv 63 \equiv 3 \bmod 10$$

11.13 DSS

DSS 是 Digital Signature Standard 的缩写，其中 $H(m)$ 是指定为上面介绍的 SHA。

系统有一公钥：供集体使用：

p：素数 p：$2^{L-1} \leqslant p < 2^{L}$，$512 \leqslant L \leqslant 1024$，$L$ 是 64 的倍数。

q：素数 $q \mid (p-1)$，$2^{159} < q < 2^{160}$。

$g = h^{(p-1)/q} \bmod p$，$1 < h < (p-1)$，$h^{(p-1)/q} \bmod p > 1$，即 $g^{q} \equiv 1 \bmod p$。

用户私钥：x，$0 < x < q$。

用户公钥：$y \equiv g^{x} \bmod p$。

k：随机数 $0 < k < q$。

签名(r,s)：

$$r \equiv (g^{k} \bmod p) \bmod q$$

$$s \equiv [k^{-1}(H(m) + xr)] \bmod q$$

验证：

$$w \equiv (s')^{-1} \bmod q$$

$$u_1 \equiv [H(m')w] \bmod q$$

$$u_2 \equiv (r')w \bmod q$$

$$v = [(g^{u_1} y^{u_2}) \bmod p] \bmod q$$

$$v = r'$$

m'，r'，s' 是 m，r，s 收到的内容，如图 11.22 和图 11.23 所示。

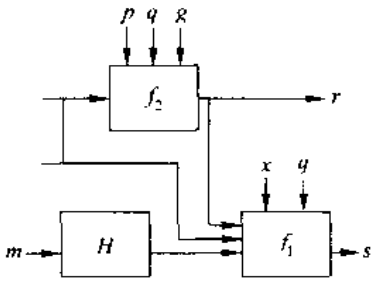


图 11.22

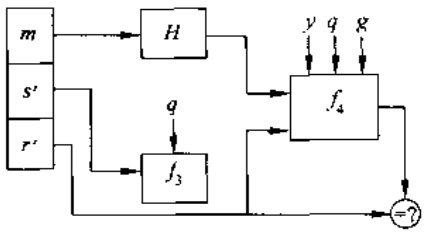


图 11.23

$$
\begin{aligned}
s &= f_1(H(m),k,x,r,q)\\
&\equiv k^{-1}(H(m)+xr) \bmod q\\
r &= f_2(k,p,q,g) \equiv (g^k \bmod p) \bmod q\\
w &\equiv f_3(s',q) = (s')^{-1} \bmod q\\
v &= f_4(y,q,g,H(m'),w,r')\\
&= ((g^{(H(m)w) \bmod q} y^{r'w \bmod q} \bmod p)) \bmod p
\end{aligned}
$$

引理 11.1 t 是任意整数

若 $$g \equiv h^{(p-1)/q} \bmod p$$

则 $$g^t \bmod p = g^{t \bmod q} \bmod p$$

证明 由 Fermat 定理，由于 h 关于 p 是互素，故

$$h^{p-1} \bmod p \equiv 1$$

故对于任意非负整数 n，有

$$
\begin{aligned}
g^{nq} \bmod p &\equiv (h^{(p-1)/q} \bmod p)^{nq} \bmod p\\
&\equiv h^{((p-1)/q)nq} \bmod p\\
&\equiv h^{(p-1)n} \bmod p\\
&\equiv ((h^{p-1} \bmod p)^n) \bmod p
\end{aligned}
$$

故对于非负整数 n 和 z 有

$$
\begin{aligned}
g^{nq+z} \bmod p &\equiv (g^{nq} g^z) \bmod p\\
&\equiv ((g^{nq} \bmod p)(g^z \bmod p)) \bmod p\\
&\equiv g^z \bmod p
\end{aligned}
$$

任意非负整数 t 可表以 $t = nq + z$，这里 n 和 z 为非负整数，且 $0 \leqslant z < q$，故

$$z \equiv t \bmod q$$

引理 11.2 对于非负整数 a 和 b

$$g^{(a \bmod q + b \bmod q) \bmod q} \bmod p \equiv g^{(a+b) \bmod q} \bmod p$$

证明

$$
\begin{aligned}
g^{(a \bmod q + b \bmod q) \bmod q} \bmod p &\equiv g^{(a \bmod q + b \bmod q) \bmod q} \bmod p\\
&\equiv g^{(a+b) \bmod q} \bmod p
\end{aligned}
$$

引理 11.3

$$y^{(rw) \bmod q} \bmod p \equiv g^{(xrw) \bmod q} \bmod p$$

证明

$$
\begin{aligned}
g^{(rw) \bmod q} \bmod p &\equiv (g^x \bmod p)^{(rw) \bmod q} \bmod p \equiv g^{x((rw) \bmod q)} \bmod p\\
&\equiv g^{(x((rw) \bmod q)) \bmod q} \bmod p \equiv g^{(xrw) \bmod q} \bmod p
\end{aligned}
$$

第 12 章　Kerberos 认证系统和 X.509 标准

12.1　概　　论

在一个开放的分布式环境下，一个用户在一客户端要接入网上的某一服务器。由于服务器仅对拥有权限的用户服务。工作站本身很难正确地识别该用户是否合法。以下 3 种攻击方式要特别注意：

① 某一用户已接入一工作站，企图冒充其他用户利用该工作站请求服务。

② 某一用户对来往信息进行窃听，然后重放它，达到扰乱和破坏的目的。

③ 用户更改工作站的网络地址，从改变了地址的工作站冒充别的工作站，请求服务。

通过以上办法，一非授权的用户可能接入某一服务器取得服务，对非授权的信息进行存取。

Kerberos 认识服务系统便是针对以上问题的。它是利用集中的认证取代分散的认证，以减轻服务器的负担。Kerberos 认证服务系统十分复杂，不容易理解它的用意，下面我们采用步步深入的办法。

在讨论正题以前先介绍下面几个符号：

C：　客户。

AS：　认证服务器。

V：　服务器。

ID_C：　客户名称(身份)。

ID_V：　服务器名称。

P_C：　客户上用户的口令。

AD_C：　客户的网址。

k_V：　AS 和 V 共享的密钥。

认证的最简单模式：

S1. $C \rightarrow AS$：(ID_C, P_C, ID_V)。

S2. $AS \rightarrow C$：$Ticket = E_{k_V}\{ID_C, AD_C, ID_V\}$。

S3. $C \rightarrow V$：$\{ID_C, Ticket\}$。

用户登录一工作站，然后对服务器 V 提出申请，即将自己的身份和口令告诉 AS，AS 搜索数据库，若用户的身份和他的口令匹配，且符合权限要求，AS 即产生一证书——Ticket，它的内容包含有$\{ID_C, AD_C, ID_V\}$，并用 AS 和 V 共享的密钥加密，并将它送给用户 C，用户 C 收到后将它送给 V。C 不可能对证书作任何改动。V 收到后解密，若解密得到的 ID_C 和证书一齐送来的证书一致，则接受 C 的申请。

证书 Ticket 用密钥 k_V 加密以防止更改和伪造。ID_C 包含在证书中说明：证书的合法拥有者是 C。AD_C 附在证书上是防止攻击者截获证书，在别的工作站将它重发，达到

非法接入服务器的目的。也就是说要证书发来的地址必须和证书内容的地址一致才有效。

上面的协议显然是解决一些问题，不过也暴露了一些严重的弱点。例如口令用明文形式传输，不安全。特别是传输频繁时，容易被窃听者截获，用来作假。而且一份证书仅用一次，势必导致反复申请服务，多次执行上面过程。

为此引进 TGS 服务器，TGS 是 Ticker-Granting Server 的缩写，意即发放证书的服务器。

申请服务的步骤如下：

S1. $C \rightarrow AS: (ID_C, ID_{tgs})$。

S2. $AS \rightarrow C: (E_{k_i}(Ticket_{tgs}))$。

$Ticket_{tgs} = E_{k_{tgs}}(ID_C, AD_C, ID_{tgs}, TS_1, LT_1)$。

S3. $C \rightarrow TGS: (ID_C, AD_C, Ticket_{tgs})$。

S4. $TGS \rightarrow C: (Ticket_V)$。

$Ticket_V = E_{k_V}(ID_C, AD_C, ID_V, TS_2, LT_2)$。

S5. $C \rightarrow V: (ID_C, Ticket_V)$。

这个协议由 TGS 发证书给客户 C，C 若需要其他服务，C 可由 $Ticket_{tgs}$ 来为自己确认。TGS 可发另一证书，C 可不必每次申请。LT_1 是证书的有效时间，LT_2 也是证书的有效时间，例如 5 小时。LT 是 LifeTime 的缩写。但必须防止攻击者窃取了证书加以假冒应用。比如攻击者掌握了证书后加以重发，非法取得服务器的服务。

证书用只有 AS 和 TGS 掌握的密钥加密，送 C 时又用只有 C 掌握的密钥 k_C 加密。目的在于只有正确的用户才能恢复 $Ticket_{tgs}$。

上面的协议比前面开始介绍的协议有许多进步，但仍然存在漏洞。首先有效时期和 $Ticket_{tgs}$ 在一起。若有效时间比较短，则用户必须反复用口令。攻击者有比较多的机会重发。一攻击者在网络上窃听到并录下证书，他假冒合法用户的网络地址向 TGS 送去证书，骗取 $Ticket_V$。

攻击者掌握了 $Ticket_V$ 也可以非法利用它。

另一方面协议还缺乏服务器向用户认证自己身份的功能，否则攻击者可以假冒服务器，利用它获取信息，起到阻挠破坏作用。

总之，必须增加双向的确认。首先 TGS 和服务器必须确认持证者确实是证书合法拥有者，另一方面服务器也必须让用户确信自己的身份。

12.2 Kerberos 协议的第 4 版本

下面是第 4 版本的 Kerberos 协议：

S1. $C \rightarrow AS: (ID_C, ID_{tgs}, TS_1)$

S2. $AS \rightarrow C: (E_{k_i}(k_{C,tgs}, ID_{tgs}, TS_2, LT_2, Ticket_{tgs}))$

$Ticket_{tgs} = E_{k_{tgs}}(k_{C,tgs}, ID_C, AD_C, ID_{tgs}, TS_2, LT_2)$

S3. $C \rightarrow TGS: (ID_V, Ticket_{tgs}, Authenticator_C)$

$Ticket_V = E_{k_V}(k_{C,V}, ID_C, AD_C, ID_V, TS_4, LT_4)$。

$Authenticator_C = E_{k_{C,tgs}}(ID_C, AD_C, TS_3)$。

S4. $TGS \rightarrow C: (E_{k_{C,tgs}}(k_{C,V}, ID_V, TS_4, Ticket_V)$。

S5. $C \rightarrow V: (Ticket_V, Authenticator_C)$。

$Authenticator_C = E_{k_{C,V}}(ID_C, AD_C, TS_5)$。

S6. $V \rightarrow C: E_{k_{C,V}}(TS_5 + 1)$。

步骤 S1 是 C 向 AS 提出申请，要求接入 TGS，AS 作出的反应是送去的信息包括会话密钥 $k_{C,tgs}$，$k_{C,tgs}$ 是由 AS 随机产生，并用客户 C 存在于 AS 的密钥 k_C 加密。$k_{C,tgs}$ 为用于客户 TGS 的会话密钥。由于它被用户 C 的密钥加密，故只有客户 C 才能读到它，同一$k_{C,tgs}$ 被用 k_{tgs} 加密，也只能由 TGS 读到它，这样会话密钥 $k_{C,tgs}$ 被安全地发给 C 和 TGS 双方。

步骤 S1 里还含有时间邮戳 TS_1，让 AS 知道它是适时的。

步骤 S2 里还有属于 $Ticket_{tgs}$ 的内容，用 k_C 加密，目的在于让 C 确信信息确实是 TGS 发出的。从而知道 $Ticket_{tgs}$ 的有效期。

有了会话密钥 $k_{C,tgs}$ 和 $Ticket_{tgs}$，C 开始向 TGS 送去(ID_C, $Ticket_{tgs}$, $Authenticator_C$)。

$Authenticator_C$ 包含客户的 ID_C、网络地址 AD_C 和时间邮戳 TS_3，它和 $Ticket_{tgs}$ 不同，$Ticket_{tgs}$ 可以重用，而 $Authenticator_C$ 有效时间非常短，一次有效。TGS 对 $Ticket_{tgs}$ 解密得 $k_{C,tgs}$。

$Ticket_{tgs}$ 实际上说明 $k_{C,tgs}$ 只有 C 能用它。TGS 用它对 $Authenticator_C$ 解密可得 ID_C，AD_C，IS_3，和输入信息的客户身份与网络地址进行校验。如完全一致，则 TGS 确信送证书的确是证书合法的拥有者。可见 $Ticket_{tgs}$ 是发放会话密钥，而 $Authenticator_C$ 才是客户的身份的认证。它生命期极短。攻击者窃取它加以重用的可能性被克服了。

TGS 作出 S4 的反应，内容用 $k_{C,tgs}$，由 C 和 TGS 共同掌握的密钥加密，内容包含($k_{C,V}$, ID_V, TS_4, $Ticket_V$)。

客户 C 现在拥有可重复用的证书 $Ticket_V$。

步骤 S5 中 C 向 V 出示证书 $Ticket_V$ 和 $Authenticator_C$。服务器对 $Ticket_V$ 解密得 $k_{C,V}$，再用它对 $Authenticator_C$ 解密。判断证书的合法拥有者是否和发送来证书的身份、地址都是一致。如若一致，对客户的身份的认证才算完成。

步骤 S6 是 V 对 C 的自我认证。而且使 C 确信不是旧的重发。

步骤 S5 和 S6 使得客户 C 和服务器 V 用的会话密钥 $k_{C,V}$ 分发任务完成。$k_{C,V}$ 可用于以后被此间的信息加密或分发新的会话密钥。

现将 Kerberos 协议每一步的各元素及其作用叙述如下：

S1. 客户向 AS 申请接入 TGS。

ID_C：客户告诉 AS 自己的身份。

ID_{tgs}：客户告诉 AS 要接入 TGS 的名字。

S2. AS 回复客户 C 以证书 $Ticket_{tgs}$。

E_{k_C}：客户 C 存放在 AS 的密钥。用 k_C 加密以保护送去信息。

$k_{C,tgs}$：由 AS 随机产生给客户 C 和 TGS 用的会话密钥。使客户 C 和 TGS 间无

需永久性的密钥。

ID_{tgs}：证书 $Ticket_{tgs}$ 的一个内容，说明这证书是发给 TGS 的。

TS_2：证书的发放时间。

LT_2：证书的有效时间。

$Ticket_{tgs}$：为使 C 接入 TGS 的证书。

S3. 请求发放服务器使用证书。

ID_V：告诉 TGS 要接入的服务器的名称。

$Ticket_{tgs}$：使 TGS 确信客户是经 AS 确认。

$Authenticator_C$：由客户产生，为使证书生效。

S4. TGS 发给服务证书。

$E_{k_{C,tgs}}$：加密密钥由客户和 TGS 共有，以保护信息。

ID_V：说明证书是为服务器发的。

TS_4：通知客户证书发放的时间。

$Ticket_V$：客户用它接入服务器 V。

$Ticket_{tgs}$：客户可反复用它而不要重新输入口令。

$E_{k_{tgs}}$：证书是用由 AS 和 TGS 共同掌握的密钥加密，以防止干扰破坏。

$k_{C,tgs}$：$Authenticator_C$ 用来加密，用以确认证书。

ID_C：证书的合法拥有者。

AD_C：防止攻击在其他工作站上应用证书。

TS_2：告诉 TGS 证书发出的时间。

LT_2：通知 TGS 证书的存放时间，防止重发。

$Authenticator_C$：使 TGS 确信送来证书者正是证书的合法拥有者，有效时间极短，以防重发。

$E_{k_{C,tgs}}$：$Authenticator_C$ 用只有 C 和 TGS 掌握的密钥 $k_{C,tgs}$ 加密，防止干扰破坏、修改。

ID_C：必须 ID_C 相同，使对证书提供确认。

AD_C：必须与发来的工作站地址相同，使对证书进行确认。

TS_2：告诉 TGS $Authenticator_C$ 生成的时间。

S5. 客户请求服务

$Ticket_V$：使服务器确信用户已经 AS 确认。

$Authenticator_C$：由客户产生使 $Ticket_V$ 生效。

S6. 服务器使客户确认服务器的身份。

$E_{k_{C,V}}$：使客户相信信息来自 V。

TS+1：使客户相信并非重发。

$Ticket_V$：客户可重复使用，接入同一服务器时可防止重新向 TGS 请求新的证书。

E_{k_V}：证书由只有 TGS 和服务器掌握的密钥加密。

$k_{C,V}$：用来对 $Authenticator_C$ 进行加密，可对证书进行认证。

ID_C：证书的合法拥有者，防止假冒。

AD_C：证书的合法拥有者的网络地址，防止从不同的工作站假冒客户的攻击。

TS_4：通知服务器 V 证书发出的时间。

LT_4：防止证书泄露后被重发。

$Authenticator_C$：使服务确信证书合法拥有者即送书的客户。

$E_{K_{C,V}}$：$Authenticator_C$ 用只有客户 C 和服务器 V 掌握的密钥加密，防止破坏。

ID_C：证书的合法拥有者，防止假冒，必须与证书的 ID_C 一致。

AD_C：证书的合法拥有者的网络地址，对证书进行确认。

12.3 Kerberos 协议的第 5 版本

Kerberos 第 5 版本不同于第 4 版本之点在于第 4 版本有如下两方面的不足和缺憾。

环境方面的问题：

① 第 4 版本用的是数据加密标准 DES，包括加密方式都存在不足之处，第 5 版本可采用其他加密算法，但加以标明，密钥也标明形式和长度，可允许同一个密钥用于不同的算法，已知的加密算法也允许作不同的变形。

② 第 4 版本依赖于 Internet 协议，以网址为例，其他地址就不适用。第 5 版本则不加限制，只要标明形式和长度。

③ 第 4 版本信息的字节安排顺序由自己选择，标明最小有效数字在低位，或最大有效数字在低位，即标明小端或大端，第 5 版本则对信息的结构加以确定。即用抽象文法表示法与基本编码法来定义所有的信息结构，使字节的顺序明确。

④ 第 4 版本证书有效时间用 8 个比特表示，单位是分，故最大有效时间仅 $2^8 \times 5 = 1280$ 分，约 21 小时，有时这限度是不够的。第 5 版本对证书的期限不设限制，只标明开始时间和结束时间。

⑤ 第 4 版本证书发给客户，不允许在其他主机上使用，也不允许其他客户使用。第 5 版本允许客户申请服务的服务器用他的名义申请其他必要的服务。

⑥ 第 4 版本对内部 n 个部门，设置了 n^2 个 Kerberos-Kerberos 关系，第 5 版本支持较简单的关系。

技术方面的问题：

① 第 4 版本发证书给客户时用的里外双重加密是不必要的，而且浪费计算。

② 每一证书都有一会话密钥，客户用它来对 Authenticator 加密，该会话密钥客户与服务器间可继续使用。故存在一种危险，攻击者截取并重发旧的信息，第 5 版本客户和服务器可商议一会话子密钥，只用于一次通信，客户的新存取将使用新的会话子密钥。即终端的每一次存取都将为下一次产生另一新的会话子密钥。

③ 口令攻击一直是个严重的问题，攻击者若掌握了口令将连续使用它，从 Kerberos 取得身份确认证书。第 5 版本采取预确认机制，使得口令攻击更难得逞。

在讨论第 5 版本之前，先引进 Kerberos 领域的概念。一个 Kerberos 环境包含一个 Kerberos 服务器，若干客户，以及一些应用服务器，Kerberos 服务器将用户的 ID 及口令

储存在它的数据库里。所有的客户在 Kerberos 服务器都进行了注册。Kerberos 服务器和其他应用服务器都有通信密钥。应用服务器在 Kerberos 服务都进行了注册。这样的环境称为一个 Kerberos 领域或 Realm。

网络上有许多这样 Realm 构成。而且用户可在自己所在的 Realm 对其他 Kerberos 领域提出申请服务。

Kerberos 服务器和别的 Kerberos 服务器有一密钥，彼此互相注册。这样 Kerberos 服务器依靠别的 Realm 的 Kerberos 服务器对用户的确认。Kerberos 服务器包含 AS 和 TGS 两部分，即确认服务器服务证书发放服务器。

用户若需要其他 Realm 的应用服务器的服务，可对当地的 TGS 取得存取权，然后请求一对远处 TGS 的资格证书。这样用户可以对其他 Realm 的 TGS 请求所需服务的资格证书。其过程如下：

① $C \rightarrow AS: \{ID_C, ID_{tgs}, TS_1\}$

② $AS \rightarrow C: \{E_{k_C}(k_{C,tgs}, ID_{tgs}, TS_2, LT_2, Ticket_{tgs})\}$

③ $C \rightarrow TGS: \{ID_{tgs}, Ticket_{tgs}, Authenticator_C\}$

④ $TGS \rightarrow C: \{E_{k_{C,tgs}}(k_{C,tgs}, ID_{tgs}, TS_4, Ticket_{tgs})\}$

⑤ $C \rightarrow TGS': \{ID_V, Ticket_{tgs}, Authenticator_C\}$

⑥ $TGS \rightarrow C: \{E_{k_{C,tgs}}(k_{C,V}, ID_V, TS_5, Ticket_V)\}$

⑦ $C \rightarrow V': \{Ticket_V, Authenticator_C\}$

这里 TGS′，V′ 分别表示其他领域的 TGS 及应用服务器 V。

第 5 版本的 Kerberos 协议，如图 12.1 所示。

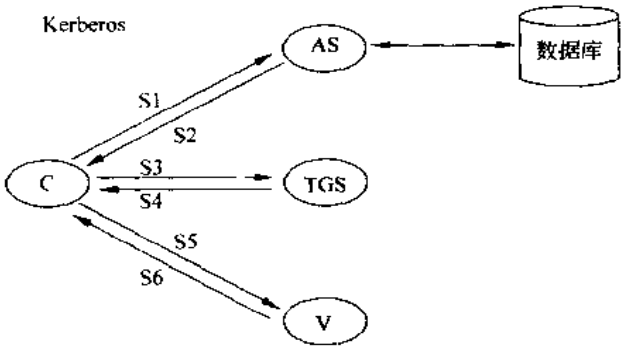


图 12.1

① $C \rightarrow AS: \{Options, ID_C, Realm_C, ID_{tgs}, Times, Nonce_1\}$

② $AS \rightarrow C: \{Realm_C, ID_C, Ticket_{tgs}, E_{k_C}(k_{C,tgs}, Times, Nonce_1, Realm_{tgs}, ID_{tgs})\}$

$$Ticket_{tgs} = E_{k_{tgs}}(Flags, k_{C,tgs}, Realm_C, ID_C, AD_C, Times)$$

③ $C \rightarrow TGS: \{Options, ID_V, Times, Nonce_2, Ticket_{tgs}, Authenticator_C\}$

$$Authenticator_C = E_{k_{C,tgs}}(ID_C, Realm_C, TS_1)$$

④ $TGS \rightarrow C: \{Realm_C, ID_C, Ticket_V, E_{k_{C,tgs}}(k_{C,V}, Times, Nonce_2, Realm_V, ID_V)\}$

$$Ticket_V = E_{k_V}(Flags, k_{C,V}, Realm_C, ID_C, AD_C, Times)$$

⑤ C→V：{Options，$Ticket_c$，$Authenticator_c$}

⑥ V→C：{$E_{k_{c,v}}(ID_C)$，$Realm_C$，TS_2，Subkey，$Seq^{\#}$}

$Authenticator_c = \{E_{k_{c,v}}(ID_C, Realm_C, TS_2, Subkey, Seq^{\#}\}$。

下面对 n 个符号作说明如下：

- $Realm_C$　表示客户 C 所在的 Kerberos 领域。
- Option　请求发给的证书应给予的 flags 标识。
- Times　客户请求发给的证书的有效期限。包括起、止时间或重新更新的时间。
- Nonce　一个随机数，用以说明发给的信息是新的，而不是重发。
- Subkey　客户选的用于会话的加密密钥。
- $Seq^{\#}$　服务器送信息给客户附加的序号，以防止重发攻击。
- Flags　第 5 版 Kerberos 引进 flags 以支持功能扩展，有
  - INITIAL 标志　表明证书是 AS 发的，不是 TGS 发的。客户向 TGS 申请服务证书时，他先向 TGS 出示由 AS 获得的身份证书。第 4 版本这是取得服务证书的惟一途径。第 5 版本准备了其他的功能，客户可以直接从 AS 取得服务证书。它的用处在于服务器（为更换口令的服务器）可能要知道客户的口令最近已被测试过。
  - PRE-AUTHEN 标志　表明是前期认证，即要求 AS 给予证书 Preauthentication 或称之为预先认证，第 5 版本还没有给它确定的格式。用户欲获得证书，预先给 AS 送去身份认证，内容包括版本号和时间邮戳。用基于客户的口令的密钥加密。AS 解密，若预先认证的时间邮戳在允许时间范围内，则发给证书，否则不发给证书。

还有一种可能在预先认证中包含基于用户口令的口令产生器，故口令不断被改变。这又可以避免口令攻击，至少减轻口令攻击的可能性。

- HW-AUTHEN，HW 是 Hardware 的缩写，即使用硬件的初始认证。
- RENEWABLE，即告诉 TGS 要求该证书可在以后可以换取替代证书的标志。证书的有效期很长，证书可能被盗或被攻击者在相当的时间内被盗用，将有效时间缩小，可减少这样的攻击。但要重新申请证书。折衷的办法是更新证书。带有 RENEW-ABLE 标志的证书有两个终止时间。一是该证书的终止时间，另一个是最后允许终止时间，客户有可能将它送到 TGS 更换一个新的终止时间，只要新的终止时间在最后允许时间范围之内。TGS 可以发给新的证书，它的好处在于假如报告证书失窃，可以拒绝将新的证书给盗窃者。
- MAY-POSTOATE。具有标志 MAY-POSTDATE 的证书，客户可用此证书向 TGS 申请具有 POSTDATED 标志的证书和 INVALID 的证书，此后客户可以使后延的证书有效，这种机制有益于一组运行的时间较长而有周期的任务，客户可以一次获得若干个证书，客户可以避免重复用一份认证证书去换取另一个服务证书。
- POSTDADED　指出证书具延长期限，服务器可查 authtime 以确定最早的确认时间。

- INVALID　证书失效，必须在使用前由 KDC 重新确认。
- PROXIABLE　具有此标志的证书允许服务器为一个以客户名义的代理客户服务，即告诉 TGS 为不同网址发一张新证书。
- PROXY　指出这张证书是一个代理人。
- FORWARDABLE　带有此标志的证书，TGS 可发给申请者一带有 FORWARD 标志及不同网络地址的身份认证证书。这证书可送到远端的 TGS，这种机制使得客户得以对其他领域的服务器进行访问，无需每个 Kerberos 保存其他领域服务器的密钥。比如领域可以层次结构化，于是客户可以爬上到树的公共顶点，再下到目标领域。走一步可以将身份认证证书出示给沿途中下一个 TGS。
- FORWARDED　指出这张证书是被转送的，或是根据某个确认过程而核发的。确认过程与被转送的身份确认证书有关。

12.4　公钥的管理

随着网络技术，特别是 Internet 的深入普及，电子商务发展非常迅猛。网上交易对要求保密、信息的完整性，不可抵赖性及身份验证等，都十分迫切。

为了适应电子商务发展的需要，特建立交易双方信赖的第三方给双方发证书，这个机构叫做 CA，它是 Certificate Authority 的缩写。顾名思义是发证书的权威机构。每个用户都有各自的公钥，这公钥是由 CA 签发证书。将来公钥也将和 E-mail 地址，电话号码一样成为每个人的有关信息。每个用户的证书储存在网络附近的数据库里，供检索查找，无需保护。电子商务就建立这样的一种公钥密码系统的平台上。实际上通信用的是对称密码。公钥密码的平台和对称密码构成一混合密码系统。

CA 由 RA 和 CP 两部分组成。RA 是 Registration Authority 的缩写，即审计授权的意思。CP 是 Certificate Processor 的缩写，即证书操作的意思。

RA 负责对证书申请者的资格审查，并决定是否同意发给证书。并承当由此引起错误的一切后果。它应由能承担此责任的机构担任。

CP 则为已被授权的申请者制作、发放及管理证书。承担因操作错误所产生的一切后果，包括失密和为没有被授权的客户发放证书等。

12.4.1　X.509 标准

X.509 规定了认证的框架，它的认证协议是基于公钥认证的应用，它之所以重要是由于证书结构和认证协议用途很广。它应用公钥密码系统和数字签名。不规定什么公钥密码算法，但推荐 RSA。规定需要用到 Hash 函数，但也不指定具体的方法，有一定灵活性。

X.509 的核心是对每一用户的公钥的认证。认证是由一些认证权威机构 CA 完成。

证书的格式包含以下几部分(见图 12.2)：

(1) 版本　用来区分 X.509 不同年份的版本。

(2) 序号　由 CA 对每一证书所给予的一种特殊号码。

(3) 数字签名算法标识符　即用来给证书作数字签名的方法及有关参数。

图 12.2

(4) 发证者名称　即签发证书的CA名称。

(5) 有效期　包含证书开始与终了的时间。

(6) 证书负责人　证书给予的用户姓名，也就是确认该用户的公钥。

(7) 用户用的公钥　包括算法的标识符及相关参数。

(8) 证书发放者的标识符。

(9) 用户的标识符。

(10) 扩充　这是第3版本增加的。

(11) 签名　对证书的其他各部分的确认。包含它们的Hash函数，但是采用CA的

密钥所作的数字签名，包括签名算法的标识符。

12.4.2 证书的管理

若 A 欲与 B 保密通信，他首先应从数据库里取出 B 的证书。

若 A 与 B 的证书由同一个 CA 发放，由对方证书容易对双方的公钥进行验证。因为它都掌握发证的 CA 的公钥。可利用它对数字签名验证。因为签名是用发证 CA 的秘密密钥签名的。

若 A 与 B 的证书由不同的 CA 签发，则情况比较复杂。这里存在一签证树，如图12.3 所示。

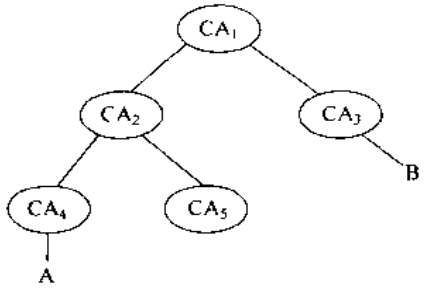


图 12.3

签证树的构造是这样的。

A 和 B 的证书分别由 CA_4 和 CA_3 签发。

CA_4、CA_2、CA_3 的证书分别由它们的上一 CA，即分别由 CA_2、CA_1 签发。

但 CA_2 的证书不仅由上一 CA_1 签发，还由它的下一层 CA_4、CA_5 签发。同样 CA_1 也有 CA_2 签发的证书。

由于 A 掌握 CA_4 的公钥，CA_2 有由 CA_4 签发的证书，所以 A 可利用 CA_4 的公钥对 CA_2 的公钥进行认证。

同理 CA_1 也有 CA_2 签发的证书，A 可以利用他掌握的 CA_2 的公钥对 CA_1 的公钥进行认证。进而利用掌握的经过认证的 CA_1 的公钥对 CA_3 的公钥进行认证。再利用 CA_3 的公钥对 B 的公钥进行认证。到此 A 对 B 的公钥认证完毕。从而保证通信的安全。

通常采用下面的符号表示 CA 对 A 签发的证书。

$$CA\ll A\gg=CA\{V,SN,AI,CA,T_A,A,A_P\}$$

其中 V 为版本，SN 为序号，AI 表算法标识符，CA 为发证书的机构，T_A 为有效期，A 为证书的拥有者，A_P 为公钥的资料。

假定 A 取得 X_1 的证书，B 获得 X_2 的证书，这里假定 X_1 和 X_2 都是 CA 的标识符。A 无法知道 X_2 的公钥，所以 B 的证书对 A 是无用的。A 可以读 B 的证书，但不能确认他的签名。下面步骤可使得 A 获得 B 的公钥。

(1) 由数据库的记录中查到 X_2 由 X_1 签发的证书。则 A 可以由证书中得到 X_2 的公钥，并且通过 X_1 的签名对它进行确认。

(2) A 回过头到数据库查出 X_2 签发给 B 的证书，故可得到 B 的公钥，并得到确认。

这样 A 通过证书链获得 B 的公钥，用 X.509 的可表示为

$$X_1\ll X_2\gg X_2\ll B\gg$$

同样 B 也可以通过反向的链获得 A 的公钥：

$$X_2\ll X_1\gg X_1\ll A\gg$$

这套方法可推到一般

$$X_1\ll X_2\gg X_2\ll X_3\gg\cdots X_N\ll B\gg$$

举例如下：

图 12.4 是一例子，圆圈表示 CA 的级间关系，矩形里的内容表示 CA 附近数据库里的证书。A 可通过数据库找证书得到确认 B 的路径。

$$X\ll W\gg W\ll V\gg V\ll Y\gg Y\ll Z\gg Z\ll B\gg$$

A 可通过这路径取得 B 公钥并进行认证。然后利用 B 的公钥与 B 通信。

B 也可以通过下面的路径取得 A 的公钥。

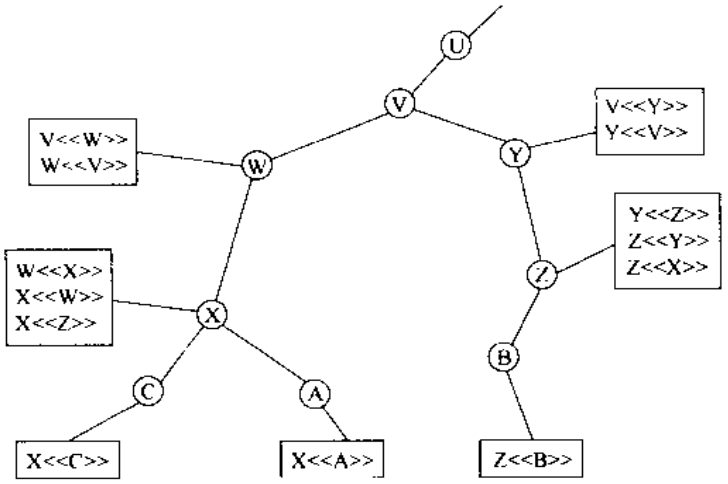


图 12.4

证书的取消

当证书有效期超过时，必须在限期前签发新的证书。当遇到下面几种情况时，可对证书进行作废处理。

(1) 用户的秘密密钥已肯定泄露。

(2) 用户已不再需要证书。

(3) CA 用来签署证书的密钥已泄露。

CA 必须对未到期被取消的证书存有一份清单。这份清单连同对它的签名一律交数据库保存，并公布。用户欲验证某一证书前，必先与取消了的证书清单作校对，以防作假。证书取消牵涉的问题十分复杂。

认证的过程

X.509 包含有 3 种不同的认证过程。可用于各种应用。3 种过程都用的是公钥的数字签名。假定双方都已掌握对方的公钥。

(1) 单向认证。即从一方设为 A 向另一方传送并证明。

① 信息是由 A 发送的。

② 信息是送给 B 的。

③ 信息不是重发，而且完整性可靠。

如图 12.5 所示 A 向 B 送信息：其中 t_A 是时间邮戳，r_A 是随机数 B 的身份。用意在于防止攻击者的延误和重发。以上信息(data)用 A 的秘密

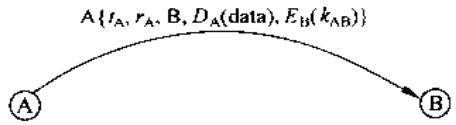


图 12.5

密钥签名。B 可利用 A 的公钥以证实其完整性并确认。k_{AB}是会话密钥，用 B 的公钥加密。

(2) 双向认证(如图 12.6 所示)。

B 向 A 回应的信息，和 A 向 B 传送的信息类似。

(3) 三向认证，如图 12.7 所示。

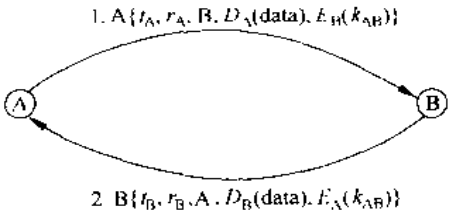


图 12.6

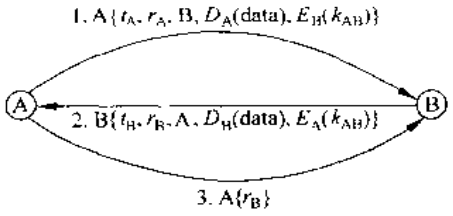


图 12.7

A 向 B 送去 A{r_B}，以防止重发攻击，避免用时间邮戳。

第 13 章　密码的差分分析法基础

13.1　引　　论

从 AES 候选算法可以看到 Feistel 网络还是比较普遍地被采用。DES 就采用这种方法。它将使抗攻击比较弱的变换经过迭代了 n 遍，使之抗攻击能力得到加强。迭代过程是以前轮的输出作为这一轮的输入。每一轮用的子密钥是由密钥通过子密钥产生算法产生的，每轮的变换基于查表、位的置换、算术运算，以及异或等运算。

以 DES 为例，将明文分成左、右相等两部分：L 和 R，利用右半部分 R 经过 f 函数对左半部分作变换，实际上它是可以恢复的。可见每轮主要的作用来自 f 函数。f 函数的灵魂是 S 盒。

在 DES 出现之时，Diffie 和 Hellman 分析后提出，若构造一穷举搜索用的专用机，用上百万个芯片，每秒钟可搜索 10^{12} 个密钥，密钥空间有 $2^{56}(\approx 7.2\times 10^{16})$ 个密钥，可在 10^5 秒，差不多是一天时间内搜索完毕，代价是 2000 万美元。

当然二十多年来，芯片的速度翻了几个量级，价格也大幅度下降。所以 DES 的密钥长度太短已成为它致命弱点。不过除 Diffie 和 Hellman 用的是穷举的强行攻击法外，在这以前对 DES 的真正攻击算法还没出现。

差分分析法是由 Eli Biham 和 Adi Shamir 于 1991 年提出的，所谓差分分析法指的是研究对一对明文对作异或运算、分析它对相应的一对密文对的影响的一种方法，从而确定可能的密钥，并找出概率最大的一个。它比穷举搜索稍为有效一些，无论如何算是一种新的密码分析法。本章结合 DES 介绍微分分析法的基础理论。

13.2　若干符号和定义

1. 本章中通常用十六进制表示，例如 10_x 即十六进制的 10，实际上是十进制的 16。

2. P 表示明文，P 和 P^* 为一对明文对，用 P' 表示 P 和 P^* 的异或，即 $P'=P\oplus P^*$。P_L 和 P_R 为 P 的左、右两部分，即

$$P=(P_L,P_R)$$

3. 密文用 T 表示，T 和 T^* 为一对密文，同样有

$$T'=T\oplus T^*$$

$$T=(T_L,T_R)$$

4. $X'=X\oplus X^*$ 称为 X 和 X^* 的异或，或 X 和 X^* 的 XOR。两个输入的异或，称之为输入的 XOR；一对输出的异或称之为输出的 XOR。

5. f 函数的输入输出在不同的轮次通过不同的英文字母表示，例如英文字母 $a,b,c,\cdots$，表示 f 在不同轮次的输入；$A,B,C,\cdots$ 分别表示 f 函数在不同轮次的输出，如图

13.1 所示。

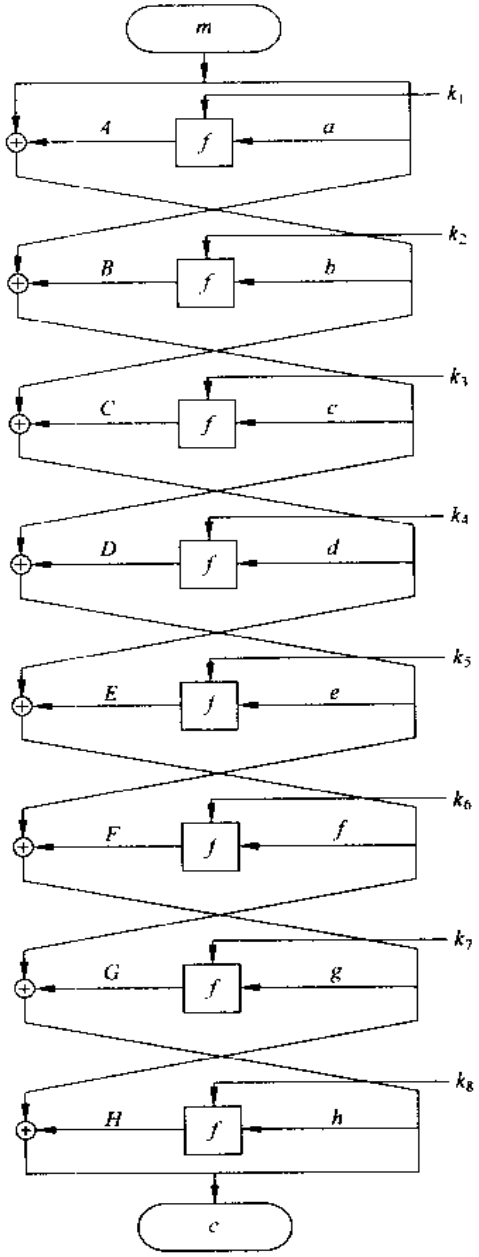


图 13.1

6. 子密钥，第 i 轮的子密钥表以 k_i。

7. S_{i_I} 表示第 i 个 S 盒(S_i)的输入；S_{i_O} 表示第 i 个 S 盒(S_i)的输出；S_{i_K} 表示子密钥 k_i 对应于 S_i 盒的 6 比特部分；S_{i_E} 表示通过 E 变换输出中对应于 S_i 盒的 6 比特部分。$S_{i_{Ea}}$ 表示第 a 轮的 S_{i_E}，同理说明 $S_{i_{Ia}}$，$S_{i_{OA}}$，$S_{i_{EA}}$ 等(如图 13.2 所示)，不一一赘述。

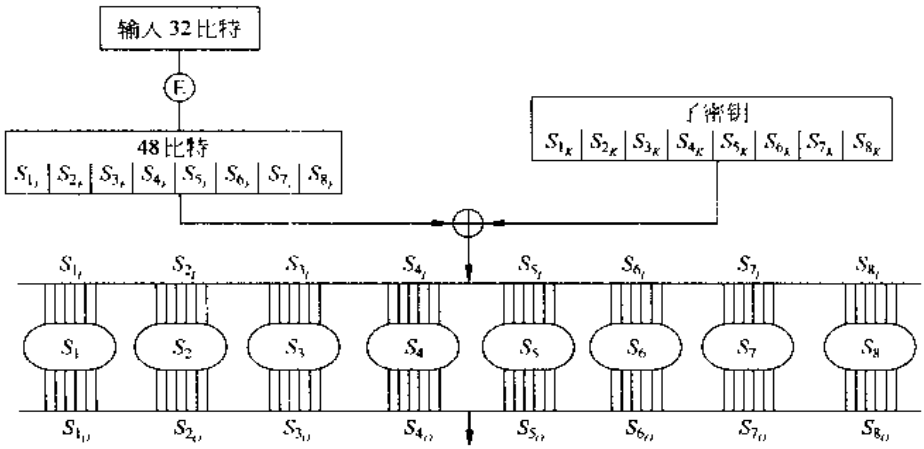


图 13.2

13.3 若干结论

(1) $E(X)\oplus E(X')=E(X\oplus X')$。

(2) $(X\oplus K)\oplus(X'\oplus K)=X\oplus X'$。

(3) $P(X)\oplus P(X')=P(X\oplus X')$。

(4) $(X\oplus Y)\oplus(X'\oplus Y')=(X\oplus X')\oplus(Y\oplus Y')$。

(5) S 盒是非线性的，由一对输入的 XOR，不能掌握对应的一对输出的 XOR，但要特别指出的是在输入相同时例外，任何情况下此时输出的一对也必相同。而且在任一对输入的 XOR 已知时，并不是所有的输出 XOR 都可能出现，而且输出的 XOR 也不是均匀分布的。有的值出现的概率要大一些。

(6) S 盒的设计满足以下关系：

① 所有的 S 盒的输出都不是输入的线性变换或仿射变换。

② 每一 S 盒的输入改变一比特，则输出至少改变两比特。

③ $S(X)$和 $S(X\oplus 0\ 0\ 1\ 1\ 0\ 0)$至少有两比特的差异。

④ $S(X)\neq S(X\oplus 1\ 1\ e\ f\ 0\ 0)$，$e$、$f$ 任意。

⑤ 某一位保持固定后，S 盒的输出中使 1 和 0 的数目之差达到最少。

13.4 XOR

DES 的任一 S 盒可能有 64×64 的输入对，每一对对应一输入的 XOR 和一输出的 XOR。实际上只有 64×16 个可能，每一个可能是多次重复，重复的次数并非一样。

表 13.1 第一列有 64 行从 0 到 37_x，也就是从 000000 到 111111，共 64 种输入状态。第一行有从 0 到 F，即从 0000 到 1111，共 16 种输出状态。

第 I_h 行，第 O_j 列表上的数表示输入的 XOR 状态为 I_h，而输出 XOR 状态为 O_j 的

数目。

比如 30_x 行，4_x 列对应的数为 12，即输入 XOR 为 011000 而输出 XOR 为 0100 的共 4 对。

表 13.1 每行的元素总和为 64，以 30_x 行 4_x 列的数 16，说明输入 XOR 为 30_x 而输出的 XOR 为 4_x 的概率为：

$$p = \frac{16}{64} = \frac{1}{4}$$

表 13.1 S_5 盒的差分分布

输入 XOR	输出 XOR															
	0_x	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
0_x	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1_x	0	0	0	6	0	2	4	4	0	10	12	4	10	6	2	4
2_x	0	0	0	8	0	4	4	4	0	6	8	6	12	6	4	2
3_x	14	4	2	2	10	6	4	2	6	4	4	0	2	2	2	0
4_x	0	0	0	6	0	10	10	6	0	4	6	4	2	8	6	2
5_x	4	8	6	2	2	4	4	2	0	4	4	0	12	2	4	6
6_x	0	4	2	4	8	2	6	2	8	4	4	2	4	2	0	12
7_x	2	4	10	4	0	4	8	4	2	4	8	2	2	2	4	4
8_x	0	0	0	12	0	8	8	4	0	6	2	8	8	2	2	4
9_x	10	2	4	0	2	4	6	0	2	2	8	0	10	0	2	12
A_x	0	8	6	2	2	8	6	0	6	4	6	0	4	0	2	10
B_x	2	4	0	10	2	2	4	0	2	6	2	6	6	4	2	12
C_x	0	0	0	8	0	6	6	0	0	6	6	4	6	6	14	2
D_x	6	6	4	8	4	8	2	6	0	6	4	6	0	2	0	2
E_x	0	4	8	8	6	6	4	0	6	6	4	0	0	4	0	8
F_x	2	0	2	4	4	6	4	2	4	8	2	2	2	6	8	8
10_x	0	0	0	0	0	0	2	14	0	6	6	12	4	6	8	6
11_x	6	8	2	4	6	4	8	6	4	0	6	6	0	4	0	0
12_x	0	8	4	2	6	6	4	6	6	4	2	6	6	0	4	0
13_x	2	4	4	6	2	0	4	6	2	0	6	8	4	6	4	6
14_x	0	8	8	0	10	0	4	2	8	2	2	4	4	8	4	0
15_x	0	4	6	4	2	2	4	10	6	2	0	10	0	4	6	4
16_x	0	8	10	8	0	2	2	6	10	2	0	2	0	6	2	6
17_x	4	4	6	0	10	6	0	2	4	4	4	6	6	6	2	0
18_x	0	6	6	0	8	4	2	2	2	4	6	8	6	6	2	2
19_x	2	6	2	4	0	8	4	6	10	4	0	4	2	8	4	0
$1A_x$	0	6	4	0	4	6	6	6	6	2	2	0	4	4	6	8
$1B_x$	4	4	2	4	10	6	6	4	6	2	2	4	2	2	4	2
$1C_x$	0	10	10	6	6	0	0	12	6	4	0	0	2	4	4	0
$1D_x$	4	2	4	0	8	0	0	2	10	0	2	6	6	6	14	0
$1E_x$	0	2	6	0	14	2	0	0	6	4	10	8	2	2	6	2
$1F_x$	2	4	10	6	2	2	2	8	6	8	0	0	0	4	6	4

续表

输入 XOR	输出 XOR															
	0_x	1_x	2_x	3_x	4	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
20_x	0	0	0	10	0	12	8	2	0	6	4	4	4	2	0	12
21_x	0	4	2	4	4	8	10	0	4	4	10	0	4	0	2	8
22_x	10	4	6	2	2	8	2	2	2	2	6	0	4	0	4	10
23_x	0	4	4	8	0	2	6	0	6	6	2	10	2	4	0	10
24_x	12	0	0	2	2	2	2	0	14	14	2	0	2	6	2	4
25_x	6	4	4	12	4	4	4	10	2	2	2	0	4	2	2	2
26_x	0	0	4	10	10	10	2	4	0	4	6	4	4	4	2	0
27_x	10	4	2	0	2	4	2	0	4	8	0	4	8	8	4	4
28_x	12	2	2	8	2	6	12	0	0	2	6	0	4	0	6	2
29_x	4	2	2	10	0	2	4	0	0	14	10	2	4	6	0	4
$2A_x$	4	2	4	6	0	2	8	2	2	14	2	6	2	6	2	2
$2B_x$	12	2	2	2	4	6	6	2	0	2	6	2	6	0	8	4
$2C_x$	4	2	2	4	0	2	10	4	2	2	4	8	8	4	2	6
$2D_x$	6	2	6	2	8	4	4	4	2	4	6	0	8	2	0	6
$2E_x$	6	6	2	2	0	2	4	6	4	0	6	2	12	2	6	4
$2F_x$	2	2	2	2	2	6	8	8	2	4	4	6	8	2	4	2
30_x	0	4	6	0	12	6	2	2	8	2	4	4	6	2	2	4
31_x	4	8	2	10	2	2	2	2	6	0	0	2	2	4	10	8
32_x	4	2	6	4	4	2	2	4	6	6	4	8	2	2	8	0
33_x	4	4	6	2	10	8	4	2	4	0	2	2	4	6	2	4
34_x	0	8	16	6	2	0	0	12	6	0	0	0	0	8	0	6
35_x	2	2	4	0	8	0	0	0	14	4	6	8	0	2	14	0
36_x	2	6	2	2	8	0	2	2	4	2	6	8	6	4	10	0
37_x	2	2	12	4	2	4	4	10	4	4	2	6	0	2	2	4
38_x	0	6	2	2	2	0	2	2	4	6	4	4	4	6	10	10
39_x	6	2	2	4	12	6	4	8	4	0	2	4	2	4	4	0
$3A_x$	6	4	6	4	6	8	0	6	2	2	6	2	2	6	4	0
$3B_x$	2	6	4	0	0	2	4	6	4	6	8	6	4	4	6	2
$3C_x$	0	10	4	0	12	0	4	2	6	0	4	12	4	4	2	0
$3D_x$	0	8	6	2	2	6	0	8	4	4	0	4	0	12	4	4
$3E_x$	4	8	2	2	2	4	4	14	4	2	0	2	0	8	4	4
$3F_x$	4	8	4	2	4	0	2	4	4	2	4	8	8	6	2	2

定义 13.1 设 X 和 Y 是两个数，分别表示 S 盒输入的 XOR 和输出的 XOR，若存在 S 盒，其输入的 XOR 为 X，而 S 盒的输出的 XOR 等于 Y，则称 X 可通过 S 盒导出 Y，表示为 $X \rightarrow Y$，否则表示为 $X \nrightarrow Y$。

例 13.1 S_1 盒输入的 XOR $S'_{1_j}=34_x$ 有 8 种可能的输出 XOR，而其他 8 个都不可能，从表 13.1 可知，8 个可能输出的 XOR 为

$$1,2,3,4,7,8,D_x,F_x$$

所以　　$34_x \rightarrow 1, 34_x \rightarrow 2, \cdots, 34_x \rightarrow D_x, 34_x \rightarrow F_x$

但　　$34_x \nrightarrow 0, 34_x \nrightarrow 5_x, 34_x \nrightarrow 6_x, \cdots, 34_x \nrightarrow E$

显然有 $X \rightarrow Y$ 的概率 p 等于 $X \rightarrow Y$ 的数目 h 除以 64，即 $p=h/64$。

例如 $34_x \rightarrow 2_x$ 对于 S_1 盒查表 13.1 可知 16 对，故 $34_x \rightarrow 2_x$ 的概率为 $16/64=1/4$。$34_x \rightarrow 4_x$ 占 2 对，故为 1/32。

从 S 盒的差分分布表中可知：有 70%～80%的元素非零，即对于输入的 XOR X 和输出的 XOR Y，$X \rightarrow Y$ 占 70%到 80%，$X \nrightarrow Y$ 只有 20%到 30%。

上面已得 $34_x \rightarrow 4_x$ 的值为 2，即只有两对的输入输出满足它们的 XOR 有 $34_x \rightarrow 4x$。而且这样的对是对偶的。即若有一对是$(S_{1_i}, S_{1_i'})$，则有另一对为$(S_{1_i'}, S_{1_i})$。这一对为$(13_x, 27_x)$。

$$13_x = (010011)_2 \xrightarrow{S_1} 0110$$

$$27_x = (100111)_2 \xrightarrow{S_1} 0010$$

所以 $110100 \xrightarrow{S_1} 0100$ 即

$$34_x \rightarrow 4_x$$

即 010011 作为 S_1 盒的输入，查 S_1 盒的第 1 行第 9 列得数 6，即输出为 0110。

同样 S_1 盒的输入为 $27_x=(100111)_x$，S_1 盒的第 3 行第 3 列得数 2，故输出为 0010。

$$(010011) \oplus (100111) = 110100 = 34_x$$

$$(0110) \oplus (0010) = 0100$$

这就得出　　$34_x \rightarrow 4_x$

表 13.2 给出输入的 XOR 为 34_x 和对应的输出 XOR 的 64 种状态：

表 13.2　可能的输入和输出

输出⊕	可能的输入								
1	03	0F	1E	1F	2A	2B	37	3B	
2	04	05	0E	11	12	14	1A	1B	20
	25	26	2E	2F	30	31	3A		
3	01	02	15	21	35	36			
4	13	27							
7	00	08	0D	17	18	1D	23	29	2C
	34	39	3C						
8	09	0C	19	2D	38	3D			
D	06	10	16	1C	22	24	28	32	
F	07	0A	0B	33	3E	3F			

再以 $34_x \rightarrow 1_x$ 为例，它有 8 个可能输入，彼此成对。

a. $03_x=(000011)_2 \xrightarrow{S_1} 1111_2$；

$37_x=(110111)_2 \xrightarrow{S_1} 1110$；

$(03_x) \oplus (37_x)=34_x \longrightarrow 0001$。

b. $0F_x=(001111)_2 \xrightarrow{S_1} 0001$；

$3B_x=(111011)_2 \xrightarrow{S_1} 0000$；

$(0F_x)\oplus(3B_x)=34_x \longrightarrow 0001$。

c. $1E_x=(011110)_2 \xrightarrow{S_1} 0111$；

$2A_x=(101010)_2 \xrightarrow{S_1} 0110$；

$(1E_x)\oplus(2A_x)=34 \longrightarrow 0001$。

d. $1F_x=(011111)_2 \xrightarrow{S_1} 1000$；

$2B_x=(101011)_2 \xrightarrow{S_1} 1001$；

$(1F_x)\oplus(2B_x)=34 \longrightarrow 0001$。

例 13.2 已知 $S_{1_k}=1_x, S'_{1_E}=35_x, S'_{1_O}=D_x$，求密钥 S_{1_k}。

输入的 XOR 为 $S'_{1_E}=S'_{1_I}=34_x$，使 $S'_{1_O}=D_x$，先不管 S_{1_k} 等是什么，查表 13.1 可知有 8 种可能，这 8 种可能使 S_{1_k} 也有 8 种可能，由于 $S_{1_k}=S_{1_E}\oplus S_{1_I}$，表 13.2。表上每行有两双相同的输入，但次序相反，每行导出两个密钥，每对一个密钥：

$$S_E\oplus S_I,\quad S_E\oplus S_I^*$$

对于 S_1：$34_x \rightarrow D_x$（输入对为$(1_x, 35_x)$）有

S_1 盒输入	可能的密钥
06,32	07,33
10,24	11,35
16,22	17,23
1C,28	1D,29

对于 S_1：$34_x \rightarrow 3$（输入对为(21,15)）有

S_1 盒输入	可能的密钥
01,35	20,14
02,36	23,17
15,21	34,00

以 $34_x \rightarrow D_x$ 输入对为$(1_x, 35_x)$为例，详细讨论如下：

a. $S_{1_E}=000001$

$S_{1_I}=000110$　$S_{1_k}=S_{1_I}\oplus S_{1_E}, S_{1_I}=06_x, S_{1_E}=1_x$

所以　$S_{1_k}=000001\oplus 000110=000111=07_x$

b. $S_{1_I}=110010$　$S_{1_I}=32_x$

所以　$S_{1_k}=000001\oplus 110010=110011=33_x$

c. $S_{1_f}=010000$

所以 $S_{1_k}=000001\oplus 010000=010001=11_x$

d. $S_{1_f}=100100$

所以 $S_{1_k}=000001\oplus 100100=100101=25_x$

e. $S_{1_f}=010110$

所以 $S_{1_k}=000001\oplus 010110=010111=17_x$

f. $S_{1_f}=100010$

所以 $S_{1_k}=000001\oplus 100010=100011=23_x$

g. $S_{1_f}=011100$

所以 $S_{1_k}=000001\oplus 011100=011101=1D_x$

h. $S_{1_f}=101000$

所以 $S_{1_k}=101001=29_x$

若增加条件 $S_{1_F}=100001$

$S_{1_F}^*=010101$

如果 $S_{1_C}^*=3_x$，则有 $34_x\rightarrow 3_x$，由表 13.1 可知共有 6 种输入的可能，即 01,02,15,21,35,36。

如若第 3 轮的 S_1 盒的输入对为

$$S_{1_{BC}}=1_x, S_{1_{EC}}^*=35_x$$

而

$$S_{1'_{CE}}=D_x=(1101)_2$$

问题转换为前面的例子。

13.5 轮特性的概念

和一对加密密切关联的有密文的 XOR、明文的 XOR、每轮输入的 XOR、每轮输出的 XOR，这些构成了 n 轮的特性。

令明文的 XOR 为 Ω_P，密文的 XOR 为 Ω_T，特性还包含一个概率，即随机选择的明文 XOR 有特性中确定的各轮的 XOR 和密文 XOR 的概率。

例 13.3 如图 13.3 所示。

例 13.3 是一轮概率为 1 的特性，不论 L' 是什么。

例 13.4 本例是一轮特性，有 7 个 S 盒的输入特性为 0，只有一个 S 盒非零，使输入的 XOR 导致输出的 XOR 达到概率最大。

因为输入有许多位可以进入相邻的 S 盒，这是由于 E 膨胀作用导致的，我们相信这些位的 XOR 为 0。只有两个隐蔽的位进入每一个 S 盒。这些位的 XOR 非零。这个概率对 S_1 最大的是 $\frac{14}{64}$。故一轮有概率为 $\frac{14}{64}$ 的特性(见表 13.1)。

S_1：$0C_x\rightarrow E_x$，概率为 $\frac{14}{64}$

$S_2,\cdots,S_8$：$00_x\rightarrow 0_x$，概率为 1，如图 13.4 所示。

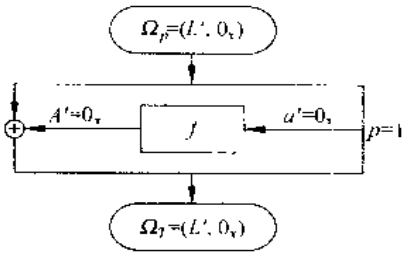


图 13.3

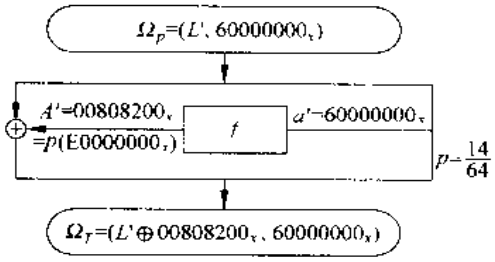


图 13.4

$a'=60000000$，经过膨胀 E，使得进入 S_1 的输入为 $001100=0C_x$，从表 13.1 可知，以输出 XOR 为 E_x 概率最大达到 $\frac{14}{64}$，$E0000000$ 再经过置换 P 得 $P(E0000000)=00808200_x$。

下面是两轮迭代的例子，如图 13.5 所示。概率为 $\frac{14}{64}$。

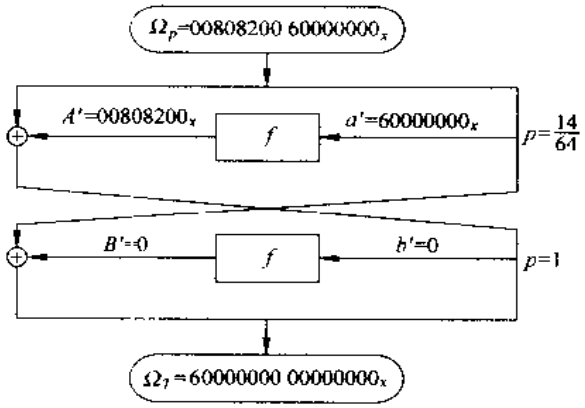


图 13.5

13.6 轮特性讨论的继续

13.5 节引入了轮特性的概念，下面给出完整的定义。

定义 13.2 n 轮特性是下面的三元组

$$\Omega=(\Omega_P,\Omega_A,\Omega_T)$$

其中 Ω_P 和 Ω_T 是 m 比特的数，m 是加密系统的分组规模。Ω_A 是 n 个元素的序列

$$\Omega_A=(\Lambda_1,\Lambda_2,\cdots,\Lambda_n)$$

而且

$$\Lambda_I=(\lambda_I^1,\lambda_I^0)\quad I=1,2,\cdots,n$$

λ_I^1 和 λ_I^0 是 $\frac{m}{2}$ 比特的数。

满足下面的条件

$$\lambda_I^1 = \Omega_P \text{ 的右半部}$$
$$\lambda_O^1 = (\Omega_P \text{ 的左半部}) \oplus \lambda_O^1$$
$$\lambda_I^n = \Omega_T \text{ 的右半部}$$
$$\lambda_I^{n-1} = (\Omega_T \text{ 的左半部}) \oplus \lambda_O^n$$
$$\lambda_O^i = \lambda_I^{i-1} \oplus \lambda_I^{i+1}$$
$$i = 2,3,\cdots,n-1$$

定义 13.3 称 n 轮特性 $\Omega=(\Omega_P,\Omega_A,\Omega_T)$ 和独立密钥 k 是正确的一对，若 $P'=\Omega_P$，且对于第 i 轮加密迭代，利用独立密钥 k 使输入 XOR 等于 λ_I^i，F 函数的输出 XOR 等于 λ_O^i，$i=1,2,\cdots,n$。

否则 k 和 Ω 便是错误的一对，以后简称为正确配对和错误配对。这里 P^1 是明文的异或。

定义 13.4 一 n 轮的特性 $\Omega^1=(\Omega_P^1,\Omega_A^1,\Omega_T^1)$ 和一 m 轮的特性 $\Omega^2=(\Omega_P^2,\Omega_A^2,\Omega_T^2)$，称 Ω^1 可以接续以 Ω^2，若 Ω_T^1 等于 Ω_P^2 左右两半部的交换。

Ω 接续以 Ω_2 得特性

$$\Omega = (\Omega_P^1,\Omega_A,\Omega_T^2)$$

其中 Ω_A 是 Ω_A^1 接续以 Ω_A^2 的结果。

定义 13.5 若函数 f 以概率 P_i^a 使 $\lambda_I^i \to \lambda_O^i$，则称特性 Ω 第 i 轮的概率为 P_i^a。

定义 13.6 n 轮的特性 Ω 有概率 P^a，且

$$P^a = \prod_{i=1}^{n} P_i^a$$

例 13.5 13.5 节的例子可推广到 3 轮如图 13.6 所示。

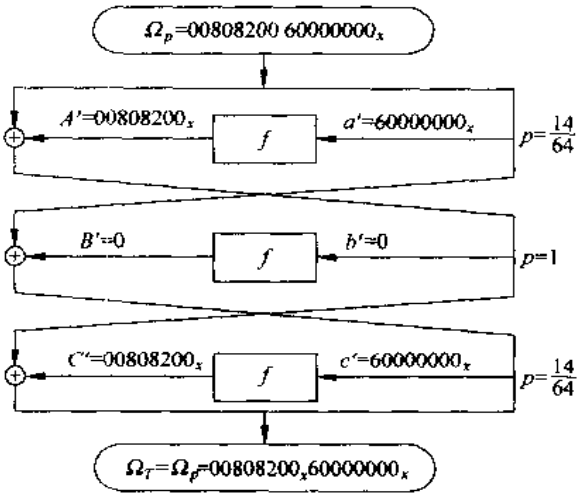


图 13.6

概率

$$P^a = \left(\frac{14}{64}\right)^2 \approx 0.05$$

若对图 13.7 延至 4 轮，$d'=b'\oplus c'=c'=A'$，可见明文若有 5 位的不同，将以 0.05 的概率导致第 4 轮的输入只有 3 位的差别，由于 E 的作用可能有 5 个 S 盒，其输入的 XOR 非零。

3 轮迭代中间一轮输入为零的结构，对 5 轮也有类似对称的结果，如图 13.7 所示。

定义 13.7 特性 $\Omega=(\Omega_P,\Omega_A,\Omega_T)$，若 Ω_P 左右两对半交换便得 Ω_T 时，称之为重复的特性。显然，重复特性可以自身连接任意次。

最重要的重复特性例子如图 13.8 所示。

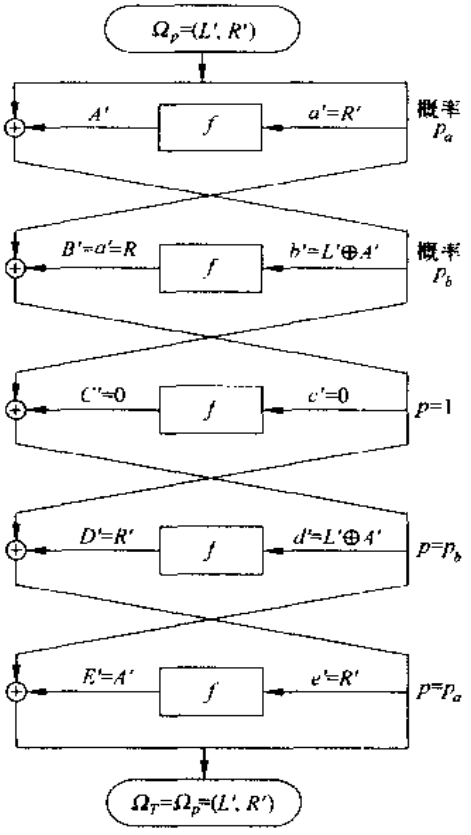


图 13.7

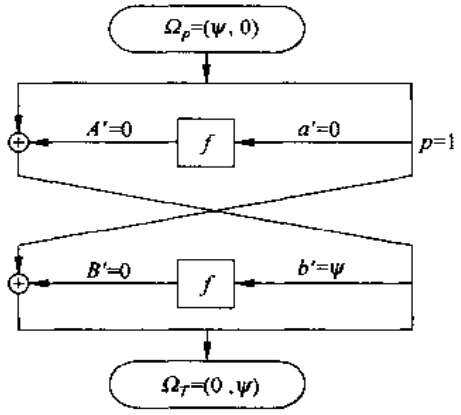


图 13.8

最优的这个特性有概率约 $\frac{1}{234}$，由此构造 5 轮的特性，概率约为

$$\left(\frac{1}{234}\right)^2\approx\frac{1}{55000}$$

13.7 已知明文攻击及 4 轮 DES 的差分分析举例

差分分析法直到现在为止都是用于为选择明文攻击，明文对可以随机选择使之满足明文 XOR 条件，不像其他的选择明文攻击，差分分析法很容易转化为已知明文攻击。

假定差分分析需要 m 对的选择明文，我们已知 $2^{32}\sqrt{2m}$ 个随机的明文及其对应的密文。

n 个密文可以形成 $C(n,2)=\frac{1}{2}n(n-1)$ 个明文对。

$$\frac{1}{2}n(n-1)<\frac{1}{2}n^2 \qquad N=2^{32}\sqrt{2m}$$

所以

$$\frac{1}{2}N^2=\frac{(2^{32}\sqrt{2m})^2}{2}=2^{64}m$$

即可能有 $2^{64}m$ 对明文对，每一对可以有一明文的 XOR，但由于分组为 64 比特，所以只能有 2^{64} 可能的明文 XOR，因

$$\frac{2^{64}m}{2^{64}}=m$$

即每个明文 XOR 将产生 m 对。

4 轮 DES(见图 13.9)的差分分析举例如下。

先看一轮的例子，如图 13.10 所示。

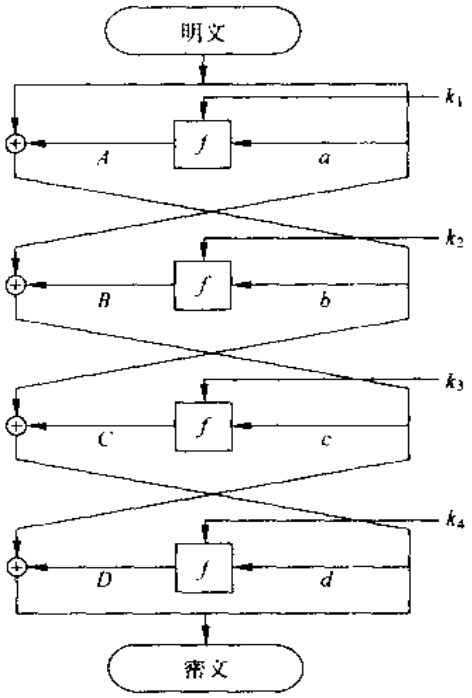


图 13.9

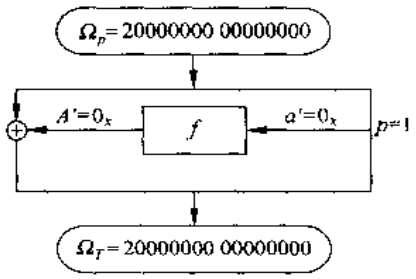


图 13.10

若加上第 2 轮，则有

$$b'=20000000$$

在第 1 轮中由于 $a'=0_x$ 推出 $A'=0_x$，而且概率为 1。两个明文的惟一一位的不同，在第 2 轮开始起到重要作用于 S_1 盒。由于 S_1 盒有一位的不同，至少有两位输出可能不同。这

两位的不同位，第 3 轮可进入 3 个 S 盒，每一个 S 盒的输入有一个比特的差异。所以第 3 轮（$C'=a'\oplus B'=B'$）的输出大致有 6 位差异。由于 $d'=b'\oplus C'$，所以第 4 轮的输入大致有 7 比特的差异，膨胀 E 后将大约达到 11 比特。如此的雪崩现象和第 4 轮所有 S 盒的输入都有不同，这里 T 是密文输出。十分相像。由于输入的 XOR 为零，故从 S_2 到 S_8 的输出 B' 的 XOR 也必须为 0，但 $a'\oplus B'=c'=D'\oplus T'_L$，所以

$$D'=a'\oplus B'\oplus T'_L$$

这里 T 是密文输出。

若已知 T 和 T^* 是一对密文对，由于 $d=T_R$ 的 d 和 d^* 已知，因 a'、T'_L，以及 B' 中的 28 比特已知，对应的 D' 中的 28 比特也可以知道，这 28 比特是 $S_2\sim S_8$ 的输出。所以我们掌握 S_{Ed}，S^*_{Ed} 和 S'_{Od} 在第 4 轮的 7 个 S 盒的值。

已知 4 对加密的明、密文对，对第 4 轮的 7 个 S 盒，利用分别计算的办法，对 S_{kd} 所有可能的 64 个值进行检测

$$S(S_{Ed}\oplus S_{kd})\oplus S(S^*_{Ed}\oplus S_{kd})=S'_{Od}$$

正确的密钥应是使得对任何一对上式都正确。

这样，我们找出最后一轮子密钥 k_4，$7\cdot 6=42$ 比特，如若子密钥是由密钥通过子密钥产生办法而得到的，则从 56 比特的密钥中找到了其中 42 位，还有 14 比特依然不清，可以通过 2^{14} 种可能的尝试，试图找回正确的密钥，即通过试图解密，正确的密钥能正确的解密。

13.8 差分分析法对 DES 的攻击

差分密码分析是一种选择明文攻击法，下面给出它对 DES 进行分析的结果。详细讨论从略。

轮数	选择明文数	分析复杂性
8	2^{14}	2^{9}
9	2^{24}	2^{32}
10	2^{24}	2^{15}
11	2^{31}	2^{42}
12	2^{31}	2^{21}
13	2^{39}	2^{32}
14	2^{39}	2^{29}
15	2^{47}	2^{37}
16	2^{47}	2^{47}

从上面分析可见，差分密码分析法更多的是理论上的价值，因为它需要的时间和数据都很大。

S_1 盒的差分分布表

输入 XOR	输出 XOR															
	0_x	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
0_x	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1_x	0	0	0	6	0	2	4	4	0	10	12	4	10	6	2	4
2_x	0	0	0	8	0	4	4	4	0	6	8	6	12	6	4	2
3_x	14	4	2	2	10	6	4	2	6	4	4	0	2	2	2	0
4_x	0	0	0	6	0	10	10	6	0	4	6	4	2	8	6	2
5_x	4	8	6	2	2	4	4	2	0	4	4	0	12	2	4	6
6_x	0	4	2	4	8	2	6	2	8	4	4	2	4	2	0	12
7_x	2	4	10	4	0	4	8	4	2	4	8	2	2	2	4	4
8_x	0	0	0	12	0	8	8	4	0	6	2	8	8	2	2	4
9_x	10	2	4	0	2	4	6	0	2	2	8	0	10	0	2	12
A_x	0	8	6	2	2	8	6	0	6	4	6	0	4	0	2	10
B_x	2	4	0	10	2	2	4	0	2	6	2	6	6	4	2	12
C_x	0	0	0	8	0	6	6	0	0	6	6	4	6	6	14	2
D_x	6	6	4	8	4	8	2	6	0	6	4	6	0	2	0	2
E_x	0	4	8	8	6	6	4	0	6	6	4	0	0	4	0	8
F_x	2	0	2	4	4	6	4	2	4	8	2	2	2	6	8	8
10_x	0	0	0	0	0	0	2	14	0	6	6	12	4	6	8	6
11_x	6	8	2	4	6	4	8	6	4	0	6	6	0	4	0	0
12_x	0	8	4	2	6	6	4	6	6	4	2	6	6	0	4	0
13_x	2	4	4	6	2	0	4	6	2	0	6	8	4	6	4	6
14_x	0	8	8	0	10	0	4	2	8	2	2	4	4	8	4	0
15_x	0	4	6	4	2	2	4	10	6	2	0	10	0	4	6	4
16_x	0	8	10	8	0	2	2	6	10	2	0	2	0	6	2	6
17_x	4	4	6	0	10	6	0	2	4	4	4	6	6	6	2	0
18_x	0	6	6	0	8	4	2	2	2	4	6	8	6	6	2	2
19_x	2	6	2	4	0	8	4	6	10	4	0	4	2	8	4	0
$1A_x$	0	6	4	0	4	6	6	6	6	2	2	0	4	4	6	8
$1B_x$	4	4	2	4	10	6	6	4	6	2	2	4	2	2	4	2
$1C_x$	0	10	10	6	6	0	0	12	6	4	0	0	2	4	4	0
$1D_x$	4	2	4	0	8	0	0	2	10	0	2	6	6	6	14	0
$1E_x$	0	2	6	0	14	2	0	0	6	4	10	8	2	2	6	2
$1F_x$	2	4	10	6	2	2	2	8	6	8	0	0	0	4	6	4
20_x	0	0	0	10	0	12	8	2	0	6	4	4	4	2	0	12
21_x	0	4	2	4	4	8	10	0	4	4	10	0	4	0	2	8
22_x	10	4	6	2	2	8	2	2	2	2	6	0	4	0	4	10
23_x	0	4	4	8	0	2	6	0	6	6	2	10	2	4	0	10
24_x	12	0	0	2	2	2	2	0	14	14	2	0	2	6	2	4
25_x	6	4	4	12	4	4	4	10	2	2	2	0	4	2	2	2
26_x	0	0	4	10	10	10	2	4	0	4	6	4	4	4	2	0
27_x	10	4	2	0	2	4	2	0	4	8	0	4	8	8	4	4

续表

输入 XOR	输出 XOR															
	0_x	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
28_x	12	2	2	8	2	6	12	0	0	2	6	0	4	0	6	2
29_x	4	2	2	10	0	2	4	0	0	14	10	2	4	6	0	4
$2A_x$	4	2	4	6	0	2	8	2	2	14	2	6	2	6	2	2
$2B_x$	12	2	2	2	4	6	6	2	0	2	6	2	6	0	8	4
$2C_x$	4	2	2	4	0	2	10	4	2	2	4	8	8	4	2	6
$2D_x$	6	2	6	2	8	4	4	4	2	4	6	0	8	2	0	6
$2E_x$	6	6	2	2	0	2	4	6	4	0	6	2	12	2	6	4
$2F_x$	2	2	2	2	2	6	8	8	2	4	4	6	8	2	4	2
30_x	0	4	6	0	12	6	2	2	8	2	4	4	6	2	2	4
31_x	4	8	2	10	2	2	2	2	6	0	0	2	2	4	10	8
32_x	4	2	6	4	4	2	2	4	6	6	4	8	2	2	8	0
33_x	4	4	6	2	10	8	4	2	4	0	2	2	4	6	2	4
34_x	0	8	16	6	2	0	0	12	6	0	0	0	0	8	0	6
35_x	2	2	4	0	8	0	0	0	14	4	6	8	0	2	14	0
36_x	2	6	2	2	8	0	2	2	4	2	6	8	6	4	10	0
37_x	2	2	12	4	2	4	4	10	4	4	2	6	0	2	2	4
38_x	0	6	2	2	2	0	2	2	4	6	4	4	4	6	10	10
39_x	6	2	2	4	12	6	4	8	4	0	2	4	2	4	4	0
$3A_x$	6	4	6	4	6	8	0	6	2	2	6	2	2	6	4	0
$3B_x$	2	6	4	0	0	2	4	6	4	6	8	6	4	4	6	2
$3C_x$	0	10	4	0	12	0	4	2	6	0	4	12	4	4	2	0
$3D_x$	0	8	6	2	2	6	0	8	4	4	0	4	0	12	4	4
$3E_x$	4	8	2	2	2	4	4	14	4	2	0	2	0	8	4	4
$3F_x$	4	8	4	2	1	0	2	4	4	2	4	8	8	6	2	2

S_2 盒的差分分布表

输入 XOR	输出 XOR															
	0_x	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
0_x	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1_x	0	0	0	4	0	2	6	4	0	14	8	6	8	4	6	2
2_x	0	0	0	2	0	4	6	4	0	0	4	6	10	10	12	6
3_x	4	8	4	8	4	6	4	2	4	2	2	4	6	2	0	4
4_x	0	0	0	0	0	6	0	14	0	6	10	4	10	6	4	4
5_x	2	0	4	8	2	4	6	6	2	0	8	4	2	4	10	2
6_x	0	12	6	4	6	4	6	2	2	10	2	8	2	0	0	0
7_x	4	6	6	4	2	4	4	2	6	4	2	4	4	6	0	6
8_x	0	0	0	4	0	4	0	8	0	10	16	6	6	0	6	4
9_x	14	2	4	10	2	8	2	6	2	4	0	0	2	2	2	4

续表

输入 XOR	输出 XOR															
	0_x	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
A_x	0	6	6	2	10	4	10	2	6	2	2	4	2	2	4	2
B_x	6	2	2	0	2	4	6	2	10	2	0	6	6	4	4	8
C_x	0	0	0	4	0	14	0	10	0	6	2	4	4	8	6	6
D_x	6	2	6	2	10	2	0	4	0	10	4	2	8	2	2	4
E_x	0	6	12	8	0	4	2	0	8	2	4	4	6	2	0	6
F_x	0	8	2	0	6	6	8	2	4	4	4	6	8	0	4	2
10_x	0	0	0	8	0	4	10	2	0	2	8	10	0	10	6	4
11_x	6	6	4	6	4	0	6	4	8	2	10	2	2	4	0	0
12_x	0	6	2	6	2	4	12	4	6	4	0	4	4	6	2	2
13_x	4	0	4	0	8	6	6	0	0	2	0	6	4	8	2	14
14_x	0	6	6	4	10	0	2	12	6	2	2	2	4	4	2	2
15_x	6	8	2	0	8	2	0	2	2	2	2	2	2	14	10	2
16_x	0	8	6	4	2	2	4	2	6	4	6	2	6	0	6	6
17_x	6	4	8	6	4	4	0	4	6	2	4	4	4	2	4	2
18_x	0	6	4	6	10	4	0	2	4	8	0	0	4	8	2	6
19_x	2	4	6	4	4	2	4	2	6	4	6	8	0	6	4	2
$1A_x$	0	6	8	4	2	4	2	2	8	2	2	6	2	4	4	8
$1B_x$	0	6	4	4	0	12	6	4	2	2	2	4	4	2	10	2
$1C_x$	0	4	6	6	12	0	4	0	10	2	6	2	0	0	10	2
$1D_x$	0	6	2	2	6	0	4	16	4	4	2	0	0	4	6	8
$1E_x$	0	4	8	2	10	6	6	0	8	4	0	2	4	4	0	6
$1F_x$	4	2	6	6	2	2	2	4	8	6	10	6	4	0	0	2
20_x	0	0	0	2	0	12	10	4	0	0	0	2	14	2	8	10
21_x	0	4	6	8	2	10	4	2	2	6	4	2	6	2	0	6
22_x	4	12	8	4	2	2	0	0	2	8	8	6	0	6	0	2
23_x	8	2	0	2	8	4	2	6	4	8	2	2	6	4	2	4
24_x	10	4	0	0	0	4	0	2	6	8	6	10	8	0	2	4
25_x	6	0	12	2	8	6	10	0	0	8	2	6	0	0	2	2
26_x	2	2	4	4	2	2	10	14	2	0	4	2	2	4	6	4
27_x	6	0	0	2	6	4	2	4	4	4	8	4	8	0	6	6
28_x	8	0	8	2	4	12	2	0	2	6	2	0	6	2	0	10
29_x	0	2	4	10	2	8	6	4	0	10	0	2	10	0	2	4
$2A_x$	4	0	4	8	6	2	4	4	6	6	2	6	2	2	4	4
$2B_x$	2	2	6	4	0	2	2	6	2	8	8	4	4	4	8	2
$2C_x$	10	6	8	6	0	6	4	4	4	2	4	4	0	0	2	4
$2D_x$	2	2	2	4	0	0	0	2	8	4	4	6	10	2	14	4
$2E_x$	2	4	0	2	10	4	2	0	2	2	6	2	8	8	10	2
$2F_x$	12	4	6	8	2	6	2	8	0	4	0	2	0	8	2	0
30_x	0	4	0	2	4	4	8	6	10	6	2	12	0	0	0	6
31_x	0	10	2	0	6	2	10	2	6	0	2	0	6	6	4	8

续表

输入 XOR	输出 XOR															
	0_x	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
32_x	8	4	6	0	6	4	4	8	4	6	8	0	2	2	2	0
33_x	2	2	6	10	2	0	0	6	4	4	12	8	4	2	2	0
34_x	0	12	6	4	6	0	1	4	4	0	4	6	4	2	4	4
35_x	0	12	4	6	2	4	4	0	10	0	0	8	0	8	0	6
36_x	8	2	4	0	4	0	4	2	0	8	4	2	6	16	2	2
37_x	6	2	2	2	6	6	4	8	2	2	6	2	2	2	4	8
38_x	0	8	8	10	6	2	2	0	4	0	4	2	4	0	4	10
39_x	0	2	0	0	8	0	10	4	10	0	8	4	4	4	4	6
$3A_x$	4	0	2	8	1	2	2	2	4	8	2	0	4	10	10	2
$3B_x$	10	4	4	2	8	2	2	6	4	4	4	2	0	2	2	2
$3C_x$	0	2	6	2	8	4	6	0	10	2	2	4	4	10	4	0
$3D_x$	0	16	10	2	4	2	4	2	8	0	0	8	0	6	2	0
$3E_x$	4	4	0	10	2	4	2	14	4	2	6	6	0	0	6	0
$3F_x$	4	0	0	2	0	8	2	4	0	2	4	4	4	14	10	6

S_3 盒的差分分布表

输入 XOR	输出 XOR															
	0_x	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
0_x	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1_x	0	0	0	2	0	4	2	12	0	14	0	4	8	2	6	10
2_x	0	0	0	2	0	2	0	8	0	4	12	10	4	6	8	8
3_x	8	6	10	4	8	6	0	6	4	4	0	0	0	4	2	2
4_x	0	0	0	4	0	2	4	2	0	12	8	4	6	8	10	4
5_x	6	2	4	8	6	10	6	2	2	8	2	0	2	0	4	2
6_x	0	10	6	6	10	0	4	12	2	4	0	0	6	4	0	0
7_x	2	0	0	4	4	4	4	2	10	4	4	8	4	4	4	6
8_x	0	0	0	10	0	4	4	6	0	6	6	6	6	0	8	8
9_x	10	2	0	2	10	4	6	2	0	6	0	4	6	2	4	6
A_x	0	10	6	0	14	6	4	0	4	6	6	0	4	0	2	2
B_x	2	6	2	10	2	2	4	0	4	2	6	0	2	8	14	0
C_x	0	0	0	8	0	12	12	4	0	8	0	4	2	10	2	2
D_x	8	2	8	0	0	4	2	0	2	8	14	2	6	2	4	2
E_x	0	4	4	2	4	2	4	4	10	4	4	4	4	4	2	8
F_x	4	6	4	6	2	2	4	8	6	2	6	2	0	6	2	4
10_x	0	0	0	4	0	12	4	8	0	4	2	6	2	14	0	8
11_x	8	2	2	6	4	0	2	0	8	4	12	2	10	0	2	2
12_x	0	2	8	2	4	8	0	8	8	0	2	2	4	2	14	0
13_x	4	4	12	0	2	2	2	10	2	2	2	2	4	4	4	8
14_x	0	6	4	4	6	4	6	2	8	6	6	2	2	0	0	8

续表

输入 XOR	输出 XOR															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
15	4	8	2	8	2	4	8	0	4	2	2	2	2	6	8	2
16	0	6	10	2	8	4	2	0	2	2	2	8	4	6	4	4
17	0	6	6	0	6	2	4	4	6	2	2	10	6	8	2	0
18	0	8	4	6	6	0	6	2	4	0	4	2	10	0	6	6
19	4	2	4	8	1	2	10	2	2	2	6	8	2	6	0	2
1A	0	8	6	4	4	0	6	4	4	8	0	10	2	2	2	4
1B	4	10	2	0	2	4	2	4	8	2	2	8	4	2	8	2
1C	0	6	8	8	4	2	8	0	12	0	10	0	4	0	2	0
1D	0	2	0	6	2	8	4	6	2	0	4	2	4	10	0	14
1E	0	4	8	2	4	6	0	4	10	0	2	6	4	8	4	2
1F	0	6	8	0	10	6	4	6	4	2	2	10	4	0	0	2
20	0	0	0	0	0	4	4	8	0	2	2	4	10	16	12	2
21	10	8	8	0	8	4	2	4	0	6	6	6	0	0	2	0
22	12	6	4	4	2	4	10	2	0	4	4	2	4	4	0	2
23	2	2	0	6	0	2	4	0	4	12	4	2	6	4	8	8
24	4	8	2	12	6	4	2	10	2	2	2	4	2	0	4	0
25	6	0	2	0	8	2	0	2	8	8	2	2	4	4	10	6
26	6	2	0	4	4	0	4	0	4	2	14	0	8	10	0	6
27	0	2	4	16	8	6	6	6	0	2	4	4	0	2	2	2
28	6	2	10	0	6	4	0	4	4	2	4	8	2	2	8	2
29	0	2	8	4	0	4	0	6	4	10	4	8	4	4	4	2
2A	2	6	0	4	2	4	4	6	4	8	4	4	4	2	4	6
2B	10	2	6	6	4	4	8	0	4	2	2	0	2	4	4	6
2C	10	4	6	2	4	2	2	2	4	10	4	4	0	2	6	2
2D	4	2	4	4	4	2	4	16	2	0	0	4	4	2	6	6
2E	4	0	2	10	0	6	10	4	2	6	6	2	2	0	2	8
2F	8	2	0	0	4	4	4	2	6	4	6	2	4	8	4	6
30	0	10	8	6	2	0	4	2	10	4	4	6	2	0	6	0
31	2	6	2	0	4	2	8	8	2	2	2	0	2	12	6	6
32	2	0	4	8	2	8	4	4	8	4	2	8	6	2	0	2
33	4	4	6	8	6	6	0	2	2	2	6	4	12	0	0	2
34	0	6	2	2	16	2	2	2	12	2	4	0	4	2	0	8
35	4	6	0	10	8	0	2	2	6	0	0	6	2	10	2	6
36	4	4	4	4	0	6	6	4	4	4	4	4	0	6	2	8
37	4	8	2	4	2	2	6	0	2	4	8	4	10	0	6	2
38	0	8	12	0	2	2	6	6	2	10	2	2	0	8	0	4
39	2	6	4	0	6	4	6	4	8	0	4	4	2	4	8	2
3A	6	0	2	2	4	6	4	4	4	2	2	6	12	2	6	2
3B	2	2	6	0	0	10	4	8	4	2	4	8	4	4	0	6
3C	0	2	4	2	12	2	0	6	2	0	2	8	4	6	4	10
3D	4	6	8	6	2	2	2	2	10	2	6	6	2	4	2	0
3E	8	6	4	4	2	10	2	0	2	2	4	2	4	2	10	2
3F	2	6	4	0	0	10	8	2	2	8	6	4	6	2	0	4

S_4 盒的差分分布表

输入 XOR	输出 XOR															
	0_x	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
0_x	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1_x	0	0	0	0	0	16	16	0	0	16	16	0	0	0	0	0
2_x	0	0	0	8	0	4	4	8	0	4	4	8	8	8	8	0
3_x	8	6	2	0	2	4	8	2	6	0	4	6	0	6	2	8
4_x	0	0	0	8	0	0	12	4	0	12	0	4	8	4	4	8
5_x	4	2	2	8	2	12	0	2	2	0	12	2	8	2	2	4
6_x	0	8	8	4	8	8	0	0	8	0	8	0	4	0	0	8
7_x	4	2	6	4	6	0	16	6	2	0	0	2	4	2	6	4
8_x	0	0	0	4	0	8	4	8	0	4	8	8	4	8	8	0
9_x	8	4	4	4	4	0	8	4	4	0	0	4	4	4	4	8
A_x	0	6	6	0	6	4	4	6	6	4	4	6	0	6	6	0
B_x	0	12	0	8	0	0	0	0	12	0	0	12	8	12	0	0
C_x	0	0	0	4	0	8	4	8	0	4	8	8	4	8	8	0
D_x	8	4	4	4	4	0	0	4	4	8	0	4	4	4	4	8
E_x	0	6	6	4	6	0	4	6	6	4	0	6	4	6	6	0
F_x	0	6	6	4	6	4	0	6	6	0	4	6	4	6	6	0
10_x	0	0	0	0	0	8	12	4	0	12	8	4	0	4	4	8
11_x	4	2	2	16	2	4	0	2	2	0	4	2	16	2	2	4
12_x	0	0	0	8	0	4	4	8	0	4	4	8	8	8	8	0
13_x	8	2	6	0	6	4	0	6	2	8	4	2	0	2	6	8
14_x	0	8	8	0	8	0	8	0	8	8	0	0	0	0	0	16
15_x	8	4	4	0	4	8	0	4	4	0	8	4	0	4	4	8
16_x	0	8	8	4	8	8	0	0	8	0	8	0	4	0	0	8
17_x	4	6	2	4	2	0	0	2	6	16	0	6	4	6	2	4
18_x	0	8	8	8	8	4	0	0	8	0	4	0	8	0	0	8
19_x	4	4	4	0	4	4	16	4	4	0	4	4	0	4	4	4
$1A_x$	0	6	6	4	6	0	4	6	6	4	0	6	4	6	6	0
$1B_x$	0	6	6	4	6	4	0	6	6	0	4	6	4	6	6	0
$1C_x$	0	8	8	8	8	4	0	0	8	0	4	0	8	0	0	8
$1D_x$	4	4	4	0	4	4	0	4	4	16	4	4	0	4	4	4
$1E_x$	0	6	6	0	6	4	4	6	6	4	4	6	0	6	6	0
$1F_x$	0	0	12	8	12	0	0	12	0	0	0	0	8	0	12	0
20_x	0	0	0	8	0	0	0	12	0	0	0	12	8	12	12	0
21_x	0	4	8	0	8	4	8	8	4	0	4	4	0	4	8	0
22_x	8	2	2	0	2	4	8	6	2	8	4	6	0	6	6	0
23_x	4	6	2	8	2	4	0	2	6	0	4	6	8	6	2	4
24_x	0	6	6	4	6	4	0	6	6	0	4	6	4	6	6	0
25_x	0	8	4	4	4	0	0	4	8	8	0	8	4	8	4	0
26_x	0	6	6	0	6	4	8	2	6	8	4	2	0	2	2	8
27_x	4	6	2	8	2	4	0	2	6	0	4	6	8	6	2	4

续表

输入 XOR	输出 XOR															
	0_x	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
28_x	16	4	4	0	4	4	4	4	4	4	4	4	0	4	4	0
29_x	0	6	2	8	2	4	0	2	6	8	4	6	8	6	2	0
$2A_x$	0	2	2	16	2	4	4	2	2	4	4	2	16	2	2	0
$2B_x$	8	0	4	0	4	8	16	4	0	0	8	0	0	0	4	8
$2C_x$	8	4	4	4	4	0	8	4	4	8	0	4	4	4	4	0
$2D_x$	4	2	6	4	6	8	0	6	2	0	8	2	4	2	6	4
$2E_x$	16	0	0	0	0	16	0	0	0	0	16	0	0	0	0	16
$2F_x$	16	0	0	0	0	0	16	0	0	16	0	0	0	0	0	16
30_x	0	6	6	4	6	4	0	6	6	0	4	6	4	6	6	0
31_x	0	8	4	4	4	0	0	4	8	8	0	8	4	8	4	0
32_x	16	6	6	4	6	0	4	2	6	4	0	2	4	2	2	0
33_x	0	2	6	4	6	8	8	6	2	0	8	2	4	2	6	0
34_x	0	12	12	8	12	0	0	0	12	0	0	0	8	0	0	0
35_x	0	4	8	0	8	4	8	8	4	0	4	4	0	4	8	0
36_x	0	2	2	4	2	0	4	6	2	4	0	6	4	6	6	16
37_x	0	2	6	4	6	8	8	6	2	0	8	2	4	2	6	0
38_x	0	4	4	0	4	4	4	4	4	4	4	4	0	4	4	16
39_x	0	6	2	8	2	4	0	2	6	8	4	6	8	6	2	0
$3A_x$	0	4	4	0	4	8	8	4	4	8	8	4	0	4	4	0
$3B_x$	16	4	4	0	4	0	0	4	4	0	0	4	0	4	4	16
$3C_x$	0	4	4	4	4	0	8	4	4	8	0	4	4	4	4	8
$3D_x$	4	2	6	4	6	8	0	6	2	0	8	2	4	2	6	4
$3E_x$	0	2	2	8	2	12	4	2	2	4	12	2	8	2	2	0
$3F_x$	8	4	0	8	0	0	0	0	4	16	0	4	8	4	0	8

S_5 盒的差分分布表

输入 XOR	输出 XOR															
	0_x	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
0_x	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1_x	0	0	0	4	0	10	8	6	0	4	2	2	12	10	2	4
2_x	0	0	0	4	0	10	6	4	0	6	4	2	4	8	10	6
3_x	8	2	4	6	4	4	2	2	6	8	6	4	4	0	2	2
4_x	0	0	0	8	0	4	10	6	0	6	6	4	8	6	0	6
5_x	12	2	0	4	0	4	8	2	4	0	16	2	0	2	0	8
6_x	0	8	4	6	4	6	2	2	4	4	6	0	6	0	2	10
7_x	2	0	4	8	4	2	6	6	2	8	6	2	2	0	6	6
8_x	0	0	0	2	0	8	10	4	0	4	10	4	8	4	4	6
9_x	8	6	0	4	0	6	6	2	2	10	2	8	6	2	0	2
A_x	0	6	8	6	0	8	0	0	8	10	4	2	8	0	0	4

续表

输入 XOR	输出 XOR															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
B	4	2	2	4	8	10	6	4	2	6	2	2	6	2	2	2
C	0	0	0	10	0	2	10	2	0	6	10	6	6	6	2	4
D	10	4	2	2	0	6	16	0	0	2	10	2	2	4	0	4
E	0	6	4	8	4	6	10	2	4	4	4	2	4	0	2	4
F	4	4	0	8	0	2	0	2	8	2	4	2	8	4	4	12
10	0	0	0	0	0	4	4	12	0	2	8	10	4	6	12	2
11	6	6	10	10	4	0	2	6	2	4	0	6	2	4	2	0
12	0	2	4	2	10	4	0	10	8	6	0	6	0	6	6	0
13	0	0	6	2	8	0	0	4	4	6	2	8	2	8	10	4
14	0	12	2	6	4	0	4	4	8	4	4	4	6	2	4	0
15	4	8	0	2	8	0	2	4	2	2	4	2	4	8	8	6
16	0	6	10	2	14	0	2	2	4	4	0	6	0	4	6	4
17	0	6	8	4	8	4	0	2	8	4	0	2	2	8	6	2
18	0	10	8	0	6	4	0	4	4	4	6	4	4	4	0	6
19	0	4	6	2	4	4	2	6	4	2	2	4	12	2	10	0
1A	0	2	16	2	12	2	0	6	4	0	0	4	0	4	4	8
1B	2	8	12	0	0	2	2	6	8	4	0	6	0	0	8	6
1C	0	10	2	6	6	6	6	4	8	2	0	4	4	4	2	0
1D	4	6	2	0	8	2	4	6	6	0	8	6	2	4	2	4
1E	0	2	6	2	4	0	0	2	12	2	2	6	2	10	10	4
1F	0	6	8	4	8	8	0	6	6	2	0	6	0	6	2	2
20	0	0	0	8	0	8	2	6	0	4	4	4	6	6	8	8
21	0	0	0	6	6	2	6	4	6	10	14	4	0	0	4	2
22	14	4	0	10	0	2	12	2	2	2	10	2	0	0	2	2
23	2	0	0	4	2	2	10	4	0	8	8	2	6	8	0	8
24	6	2	8	4	4	4	6	2	2	6	6	2	6	2	2	2
25	6	0	0	8	2	8	2	6	6	4	2	2	4	2	6	6
26	12	0	0	4	0	4	4	4	0	8	4	0	12	8	0	4
27	12	2	0	2	0	12	2	2	4	4	8	4	8	2	2	0
28	2	8	4	6	2	4	6	0	6	6	4	0	2	2	2	10
29	6	4	6	8	8	4	6	2	0	0	2	2	10	0	2	4
2A	4	4	0	2	2	4	6	2	0	0	6	4	10	4	4	12
2B	4	6	2	6	0	0	12	2	0	4	12	2	6	4	0	4
2C	8	6	2	6	4	8	6	0	4	4	0	2	6	0	6	2
2D	4	4	0	4	0	6	4	2	4	12	0	4	4	6	4	6
2E	6	0	2	4	0	6	6	4	2	10	6	10	6	2	0	0
2F	10	4	0	2	2	6	10	2	0	2	2	4	6	2	2	10
30	0	4	8	4	6	4	0	6	10	4	2	4	2	6	4	0
31	0	6	6	4	10	2	0	0	4	4	0	0	4	6	12	6
32	4	6	0	2	6	4	6	0	6	0	4	6	4	10	6	0

续表

输入 XOR	输出 XOR															
	0_x	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
33_x	8	10	0	14	8	0	0	8	2	0	2	4	0	4	4	0
34_x	0	4	4	2	14	4	0	8	6	8	2	2	0	4	6	0
35_x	0	4	16	0	8	4	0	4	4	4	0	8	0	4	4	4
36_x	4	4	4	6	2	2	2	12	2	4	4	8	2	4	4	0
37_x	4	2	2	2	4	2	0	8	2	2	2	12	6	2	8	6
38_x	0	4	8	4	12	0	0	8	10	2	0	0	0	4	2	10
39_x	0	8	12	0	2	2	2	2	12	4	0	8	0	4	4	4
$3A_x$	0	14	4	0	4	6	0	0	6	2	10	8	0	0	4	6
$3B_x$	0	2	2	2	4	4	8	6	8	2	2	2	6	14	2	0
$3C_x$	0	0	10	2	6	0	0	2	6	2	2	10	2	4	10	8
$3D_x$	0	6	12	2	4	8	0	8	8	2	2	0	2	2	4	4
$3E_x$	4	4	10	0	2	4	8	8	2	2	0	2	6	8	4	0
$3F_x$	8	6	6	0	4	2	2	4	4	2	8	6	2	4	6	0

S_4 盒的差分分布表

输入 XOR	输出 XOR															
	0_x	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
0_x	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1_x	0	0	0	6	0	2	6	2	0	4	2	4	6	16	14	2
2_x	0	0	0	2	0	10	6	10	0	2	4	8	6	6	8	2
3_x	0	8	0	8	0	6	4	6	4	4	4	12	2	4	2	0
4_x	0	0	0	8	0	0	8	0	0	6	8	10	2	4	10	8
5_x	10	2	4	4	4	8	8	4	2	2	0	4	0	8	0	4
6_x	0	8	4	4	8	4	2	2	12	0	2	6	6	2	2	2
7_x	6	6	4	0	2	10	2	2	2	2	6	6	8	0	6	2
8_x	0	0	0	6	0	2	16	4	0	2	6	2	4	12	6	4
9_x	10	4	2	6	0	2	6	2	4	0	8	6	4	4	2	4
A_x	0	14	4	4	0	2	2	2	10	4	4	4	6	4	2	2
B_x	4	6	2	0	2	2	12	8	2	2	2	6	8	2	0	6
C_x	0	0	0	12	0	10	4	6	0	8	4	4	2	12	2	0
D_x	12	0	2	10	6	4	4	2	4	2	6	0	2	6	0	4
E_x	0	6	4	0	4	4	10	8	6	2	4	6	2	0	6	2
F_x	2	2	2	2	6	2	6	2	10	4	8	2	6	4	4	2
10_x	0	0	0	8	0	8	0	12	0	4	2	6	8	4	6	6
11_x	6	2	6	4	6	2	6	4	6	6	4	2	4	0	6	0
12_x	0	8	4	2	0	4	2	0	4	10	6	2	8	6	4	4
13_x	6	6	12	0	12	2	0	6	6	2	0	4	0	2	4	2
14_x	0	4	6	2	8	6	0	2	6	10	4	0	2	4	6	4
15_x	2	2	6	6	4	4	2	6	2	6	8	4	4	0	4	4

续表

输入 XOR	输出 XOR															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
16	0	4	14	6	8	4	2	6	2	0	2	0	4	2	0	10
17	2	6	8	0	0	2	0	2	2	6	0	8	8	2	12	6
18	0	4	6	6	8	4	2	2	6	4	6	4	2	4	2	4
19	2	6	0	2	4	4	4	6	4	8	6	4	2	2	6	4
1A	0	6	6	0	8	2	4	6	4	2	4	6	2	0	4	10
1B	0	4	10	2	4	4	2	6	6	6	2	2	6	6	2	2
1C	0	0	8	2	12	2	6	2	8	6	6	2	4	0	4	2
1D	2	4	0	6	8	6	0	2	6	8	6	0	2	1	0	10
1E	0	10	8	2	8	2	0	2	6	4	2	4	6	4	2	4
1F	0	6	6	8	6	4	2	4	4	2	2	0	2	4	2	12
20	0	0	0	0	0	6	6	4	0	4	8	8	4	6	10	8
21	2	8	6	8	4	4	6	6	8	4	0	4	0	2	2	0
22	16	2	4	6	2	4	2	0	6	4	8	2	0	2	2	4
23	0	4	0	4	4	6	10	4	2	2	6	2	4	6	6	4
24	10	8	0	6	12	6	10	4	8	0	0	0	0	0	0	0
25	0	2	4	2	0	4	4	0	4	0	10	10	4	10	6	4
26	2	2	0	12	2	2	6	2	4	4	8	0	6	6	8	0
27	8	4	0	8	2	4	2	4	0	6	2	4	4	8	2	6
28	6	8	4	6	0	4	2	2	4	8	2	6	4	2	2	4
29	2	4	4	0	8	8	6	8	6	4	0	4	4	4	2	0
2A	6	0	0	6	6	4	6	8	2	4	0	2	2	4	6	8
2B	12	0	4	0	0	4	2	2	2	6	10	6	10	2	4	0
2C	4	2	6	0	0	6	8	6	4	2	2	8	4	6	4	2
2D	6	2	2	6	6	4	4	2	6	2	4	8	4	2	4	2
2E	4	6	2	4	2	4	4	2	4	2	4	6	4	10	4	2
2F	10	0	4	8	0	6	6	2	0	4	4	2	6	2	2	8
30	0	12	8	2	0	6	0	0	6	6	0	2	8	2	6	6
31	2	6	10	4	2	2	2	4	6	0	2	6	0	2	4	12
32	4	2	2	8	10	8	8	6	0	2	2	4	4	2	2	0
33	4	2	2	2	6	0	4	0	10	6	6	4	0	4	8	6
34	0	4	4	2	6	4	0	4	6	2	6	4	2	8	0	12
35	6	12	4	2	4	2	2	4	8	2	2	0	6	4	4	2
36	0	2	2	4	4	4	4	0	2	10	12	4	0	10	4	2
37	10	2	2	6	14	2	2	6	2	0	4	6	2	0	4	2
38	0	4	14	0	8	2	0	4	4	4	2	0	8	2	4	8
39	2	4	8	0	6	2	0	6	2	6	4	2	8	6	2	6
3A	8	4	0	4	6	2	0	4	6	8	6	0	6	0	4	6
3B	0	4	6	6	2	2	2	14	0	12	0	4	2	2	8	0
3C	0	6	16	0	2	2	2	8	4	2	0	12	6	2	2	0
3D	0	6	2	2	2	6	8	2	4	2	6	2	6	2	4	10
3E	4	2	2	4	4	0	6	10	4	2	4	6	6	2	6	2
3F	0	4	6	6	4	8	4	0	4	8	4	0	4	8	2	2

S_7 盒的差分分布表

输入 XOR	输出 XOR															
	0_x	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
0_x	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1_x	0	0	0	2	0	4	4	14	0	12	4	6	2	6	6	4
2_x	0	0	0	0	0	12	2	2	0	4	0	4	8	12	6	14
3_x	8	2	12	2	6	8	6	0	6	4	4	2	2	0	0	2
4_x	0	0	0	8	0	4	4	8	0	8	8	12	2	6	2	2
5_x	6	0	0	2	8	0	8	4	0	2	6	0	10	6	6	6
6_x	0	2	12	0	8	4	8	2	4	4	4	2	6	0	6	2
7_x	4	6	4	12	0	4	2	0	0	14	2	6	4	0	0	6
8_x	0	0	0	8	0	0	6	10	0	4	12	4	6	6	0	8
9_x	10	8	4	8	6	2	2	0	2	6	8	2	0	6	0	0
A_x	0	10	6	2	12	2	4	0	4	4	6	4	4	0	0	6
B_x	0	2	2	2	4	8	6	4	4	0	4	2	6	4	2	14
C_x	0	0	0	4	0	4	8	4	0	2	6	0	14	12	8	2
D_x	6	6	2	4	2	6	4	6	6	4	8	8	0	2	0	0
E_x	0	12	10	10	0	2	4	2	8	6	4	2	0	0	2	2
F_x	2	0	0	0	6	8	8	0	6	2	4	6	8	0	6	8
10_x	0	0	0	4	0	2	8	6	0	6	4	10	8	4	8	4
11_x	6	10	10	4	4	2	0	4	4	0	2	8	4	2	2	2
12_x	0	0	8	8	2	8	2	8	6	4	2	8	0	0	8	0
13_x	4	4	2	2	8	6	0	2	2	2	0	4	6	8	14	0
14_x	0	8	6	2	8	8	2	6	4	2	0	2	8	6	0	2
15_x	4	4	8	2	4	0	4	10	8	2	4	4	4	2	0	4
16_x	0	6	10	2	2	2	2	4	10	8	2	2	0	4	10	0
17_x	8	2	4	2	6	4	0	6	4	4	2	2	0	4	8	8
18_x	0	16	2	2	6	0	6	0	6	2	8	0	6	0	2	8
19_x	0	8	0	2	4	4	10	4	8	0	6	4	2	6	2	4
$1A_x$	0	2	4	8	12	4	0	6	4	4	0	2	0	6	4	8
$1B_x$	0	6	2	6	4	2	4	4	6	4	8	4	2	0	10	2
$1C_x$	0	8	4	4	2	6	6	6	6	4	6	8	0	2	0	2
$1D_x$	4	4	4	0	0	2	4	2	4	2	2	4	10	10	8	4
$1E_x$	0	0	2	2	12	6	2	0	12	2	2	4	2	6	8	4
$1F_x$	2	2	10	14	2	4	2	4	4	6	0	2	4	8	0	0
20_x	0	0	0	14	0	8	4	2	0	4	2	8	2	6	0	14
21_x	4	2	6	2	12	2	4	0	6	4	10	2	4	2	2	2
22_x	10	6	0	2	4	4	10	0	4	0	12	2	8	0	0	2
23_x	0	6	2	2	2	4	6	10	0	4	8	2	2	6	0	10
24_x	4	2	0	6	8	2	6	0	8	2	2	0	8	2	12	2
25_x	2	0	2	16	2	4	6	4	6	8	2	4	0	6	0	2
26_x	6	10	0	10	0	6	4	4	2	2	4	6	2	4	2	2
27_x	4	0	2	0	2	2	14	0	4	6	6	2	12	2	4	4

续表

输入 XOR	输出 XOR															
	0_x	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
28_x	14	4	6	4	4	6	2	0	6	6	2	2	4	0	2	2
29_x	2	2	0	2	0	8	4	2	4	6	4	4	6	4	12	4
$2A_x$	2	4	0	0	0	2	8	12	0	8	2	4	8	4	4	6
$2B_x$	16	6	2	4	6	10	2	2	2	2	2	2	4	2	2	0
$2C_x$	2	6	6	8	2	2	0	6	0	8	4	2	2	6	8	2
$2D_x$	6	2	4	2	8	8	2	8	2	4	4	0	2	0	8	4
$2E_x$	2	4	8	0	2	2	2	4	0	2	8	4	14	6	0	6
$2F_x$	2	2	2	8	0	2	2	6	4	6	8	8	6	2	0	6
30_x	0	6	8	2	8	4	4	0	10	4	4	6	0	0	2	6
31_x	0	8	4	0	6	2	2	6	6	0	0	2	6	4	8	10
32_x	2	4	0	0	6	4	10	6	6	4	6	2	4	6	2	2
33_x	0	16	6	8	2	0	2	2	4	2	8	4	0	4	6	0
34_x	0	4	14	8	2	2	2	4	16	2	2	2	0	2	0	4
35_x	0	6	0	0	10	8	2	2	6	0	0	8	6	4	4	8
36_x	2	0	2	2	4	6	4	4	2	2	4	2	4	16	10	0
37_x	6	6	6	8	4	2	4	4	4	0	6	8	2	4	0	0
38_x	0	2	2	2	8	8	0	2	2	2	0	6	6	4	10	10
39_x	4	4	16	8	0	6	4	2	4	4	2	6	0	2	2	0
$3A_x$	16	6	4	0	2	0	2	6	0	4	8	10	0	0	4	2
$3B_x$	2	0	0	2	0	4	4	4	2	6	2	6	6	12	12	2
$3C_x$	0	0	8	0	12	8	2	6	6	4	0	2	2	4	6	4
$3D_x$	2	4	12	2	2	2	0	4	6	10	2	6	4	2	0	6
$3E_x$	4	6	6	6	2	0	4	8	2	10	4	6	0	4	2	0
$3F_x$	14	0	0	0	8	0	6	8	4	2	0	0	4	8	4	6

S_8 盒的差分分布表

输入 XOR	输出 XOR															
	0_x	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
0_x	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1_x	0	0	0	6	0	16	10	0	0	0	6	0	14	6	2	4
2_x	0	0	0	8	0	10	4	2	0	10	2	4	8	8	6	2
3_x	6	0	2	8	2	6	4	0	6	6	6	2	2	0	8	6
4_x	0	0	0	2	0	4	6	12	0	6	8	4	10	4	8	0
5_x	4	10	6	0	0	2	6	0	4	10	4	6	8	2	0	2
6_x	0	0	10	4	6	4	4	8	2	6	4	2	4	2	2	6
7_x	6	2	8	2	8	10	6	6	4	2	0	4	0	0	0	6
8_x	0	0	0	4	0	6	4	2	0	8	6	10	8	2	2	12
9_x	8	4	0	6	0	4	4	6	2	4	6	2	12	2	0	4
A_x	0	0	16	4	6	6	4	0	4	6	4	2	2	0	0	10

续表

输入 XOR	输出 XOR															
	0_x	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
B_x	2	8	0	6	2	6	0	4	4	10	0	2	10	2	6	2
C_x	0	0	0	2	0	10	10	6	0	6	6	6	2	6	10	0
D_x	6	0	4	10	2	0	8	6	2	2	6	10	2	2	2	2
E_x	0	0	6	8	1	8	0	2	10	6	2	4	6	2	4	2
F_x	8	0	4	2	2	4	2	2	2	6	4	6	0	2	14	6
10_x	0	0	0	4	0	0	8	12	0	0	8	8	2	10	6	6
11_x	0	6	4	6	2	2	6	6	4	6	4	6	0	4	4	4
12_x	0	4	0	8	6	2	8	4	2	4	4	6	2	4	10	0
13_x	4	2	2	6	8	6	2	2	14	2	2	4	2	2	2	4
14_x	0	16	4	2	6	0	2	6	4	0	4	6	4	6	4	0
15_x	0	10	6	0	6	0	2	8	2	2	0	8	2	6	6	6
16_x	0	12	6	4	6	0	0	0	8	6	6	2	2	6	4	2
17_x	0	6	8	0	6	2	4	6	6	0	2	6	4	4	2	8
18_x	0	12	2	2	8	0	8	0	10	4	4	2	4	2	0	6
19_x	6	4	8	0	8	0	4	2	0	0	12	2	4	6	2	6
$1A_x$	0	4	6	2	8	8	0	4	8	0	0	0	6	2	0	16
$1B_x$	2	4	8	10	2	4	2	8	2	4	8	2	0	2	4	2
$1C_x$	0	12	6	4	6	4	2	2	6	0	4	4	2	10	2	0
$1D_x$	8	6	0	0	10	0	0	8	10	4	2	2	2	8	4	0
$1E_x$	0	4	8	6	8	2	4	4	10	2	2	4	2	0	6	2
$1F_x$	4	2	4	2	6	2	4	0	2	6	2	2	2	16	8	2
20_x	0	0	0	16	0	4	0	0	0	14	6	4	2	0	4	14
21_x	0	0	2	10	2	8	10	0	0	6	6	0	10	2	2	6
22_x	8	0	6	0	6	4	10	2	0	6	8	0	4	4	2	4
23_x	4	8	0	6	0	4	8	6	2	2	10	4	8	0	0	2
24_x	4	0	4	8	4	6	2	4	8	6	2	0	0	4	4	8
25_x	0	4	6	8	2	8	8	0	4	2	4	4	2	2	6	4
26_x	2	6	0	6	4	4	4	6	6	0	4	4	10	4	2	2
27_x	6	6	0	0	2	2	6	2	4	4	6	10	2	6	2	6
28_x	10	2	6	2	4	12	12	0	2	2	4	0	0	0	2	6
29_x	4	0	0	14	2	10	4	2	8	6	4	0	4	2	2	2
$2A_x$	8	8	0	2	0	2	4	0	2	6	8	14	2	8	0	0
$2B_x$	2	2	0	0	4	2	10	4	6	2	4	0	6	4	8	10
$2C_x$	2	6	6	2	4	6	2	0	2	6	4	0	6	4	10	4
$2D_x$	8	0	4	4	6	2	0	0	6	8	2	4	6	4	4	6
$2E_x$	6	2	2	4	2	2	6	12	4	0	4	2	8	8	0	2
$2F_x$	8	12	4	6	6	4	2	2	2	2	4	2	2	4	0	4
30_x	0	4	6	2	10	2	2	2	4	8	0	0	8	4	6	6
31_x	4	6	8	0	4	6	0	4	4	6	10	2	2	4	4	0
32_x	6	6	6	2	4	6	0	2	0	6	8	2	2	6	6	2

附录 A　DES 程 序

1. <DESProcess. h>

```
//
//////////////////////////////////////////////////////////////////////

#if ! defined(AFX_DESPROCESS_H__6FD6DF17_CFC4_415A_B247_6209049569BA__INCLUDED_)
#define AFX_DESPROCESS_H__6FD6DF17_CFC4_415A_B247_6209049569BA__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define maxlen 60000
#include <iostream. h>
#include "string. h"
#include "malloc. h"
#include "stdio. h"

/* IP */
static int ip[64]={
        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 };

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

```
/* E矩阵 */
static int e[48] = {
        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
};
//S盒
static int sbox[8][64]={
        /* S1 */
        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,

        /* S2 */
        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,

        /* S3 */
        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,

        /* S4 */
        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,

        /* S5 */
        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,

          * S6 */
```

```
        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,

        /* S7 */
        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,

        /* S8 */
        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
};
/* P */
static int p[32]={
        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
};
static int pc1[56]={
        57, 49, 41, 33, 25, 17, 9,
        1, 58, 50, 42, 34, 26, 18,
        10, 2, 59, 51, 43, 35, 27,
        19, 11, 3, 60, 52, 44, 36,
        63, 55, 47, 39, 31, 23, 15,
        7, 62, 54, 46, 38, 30, 22,
        14, 6, 61, 53, 45, 37, 29,
        21, 13, 5, 28, 20, 12, 4
};
/* PC-1 */
static int ls[16]={
        1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};

/* PC-2 */
static int pc2[48]={
```

```
        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
};
struct bits {
    bool bit[64] ;
    int totalbit;
};
struct chars{
    char ch[8];
    int totalchar;
};
struct bitarray
{
    bits bitarr[16] ;
};
class DESProcess
{public:
    DESProcess();
    virtual ~DESProcess();

    chartobits (chars ch, bits * bts) ;
    bitstochar (bits bts, chars * ch) ;

    bitmaping (int bitmapno, bits inbts, bits * outbts) ;

    InitIP  (bits bts64, bits * Lbts32, bits * Rbts32) ;
    FinalIP (bits Lbts32, bits Rbts32, bits * bts64) ;

    leftshift (bits * bit28, int times) ;
    pc1tran (bits bts64, bits * Cbts28, bits * Dbts28) ;
    pc2tran (bits Cbts28, bits Dbts28, bits * Kbts48) ;
    keygen (bits key, bitarray * subkey) ;

    etran (bits bts32, bits * bts48) ;
    bitToSBoxPos (bits bts48, int boxno, int * row, int * col) ;
    sboxtran (bits bts48, bits * bts32) ;
    ptran (bits bts32, bits * fout) ;
```

```
    firan (bits inbts32, bits keyi, bits * fRes);

    exclusiveOR (bits bts1, bits bts2, bits * xorRes) ;
    desblock (chars intext8, chars key, bool encode, chars * outtext8) ;

    desCoding (char mfile[], char key[], char cfile[], bool codedir) ;

    desEncode (char mfile[], char key[], char cfile[]) ;
    desDecode (char mfile[], char key[], char cfile[]) ;

    dispfile (char fn[]) ;
    float bitdiff (char mfile[], char cfile[], float * bitdiff8byte) ;
};

#endif // ! defined(AFX_DESPROCESS_H__6FD6DF17_CFC4_415A_B247_6209049569BA__INCLUDED_)
```

2. <DESProcess.cpp>

```
// DESProcess.cpp: implementation of the DESProcess class.
//////////////////////////////////////////////////////////////////////
#include "DESProcess.h"
//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

DESProcess::DESProcess()
{
}

DESProcess::~DESProcess()
{
}

DESProcess::chartobits (chars ch, bits * bts)
{
    int cnt1,cnt2,pos ;
    for (cnt1=0 ; cnt1 <ch.totalchar;cnt1++)
    {
        pos=int(ch.ch[cnt1]) ;
        pos = pos <0 ? pos +256 : pos ;
        pos =pos + 28 ;
        for (cnt2=7 ; cnt2>=0; cnt2--)
        {
            bts->bit[cnt1 * ch.totalchar+ cnt2] = (pos % 2!=0) ? true:false;
```

```
                pos/=2;
            }
        }
        bts->totalbit = ch.totalchar * 8 ;
    }
    DESProcess::bitstochar (bits bts, chars *ch)
    {
        int cnt1,cnt2, pos, power ;
        for (cnt1=0 ; cnt1 <8 ; cnt1 ++)
        {
            pos = 0 ;
            power = 128 ;
            for (cnt2=0 ; cnt2 <8 ; cnt2++)
            {
                if (bts.bit[cnt1 * 8+cnt2]) pos+=power;
                power /= 2 ;
            }
            ch->ch[cnt1] = char(pos-28) ;
        }
        ch->totalchar = bts.totalbit / 8 ;
    }

    DESProcess::bitmaping (int bitmapno, bits inbts, bits *outbts)
    {
        int cnt, mapsize;
        switch (bitmapno)
        {
        case 1:     mapsize = 64  ; break ;
        case 2:     mapsize = 64  ; break ;
        case 3:     mapsize = 48  ; break ;
        case 4:     mapsize = 32  ; break ;
        case 5:     mapsize = 56  ; break ;
        case 6:     mapsize = 48  ; break ;
        }
        outbts->totalbit =mapsize ;
        for (cnt =0 ; cnt<mapsize; cnt++)
        {
            switch (bitmapno)
            {
            case 1:     outbts->bit[cnt] = inbts.bit[ip[cnt]-1] ;  break;
            case 2:     outbts->bit[cnt] = inbts.bit[fp[cnt]-1] ;  break;
            case 3:     outbts->bit[cnt] = inbts.bit[e[cnt]-1] ;   break;
            case 4:     outbts->bit[cnt] = inbts.bit[p[cnt]-1] ;   break;
            case 5:     outbts->bit[cnt] = inbts.bit[pc1[cnt]-1] ; break;
```

```
            case 6:     outbts->bit[cnt] = inbts.bit[pc2[cnt]-1] ; break;
            }
    }
}

DESProcess::InitIP  (bits bts64, bits *Lbts32, bits *Rbts32)
{
    int cnt ;
    bits tempbts;

    bitmaping(1,bts64,&tempbts) ;
    for (cnt = 0 ; cnt <32 ; cnt++)
    {
        Lbts32->bit[cnt]=tempbts.bit [cnt];
        Rbts32->bit[cnt]=tempbts.bit [cnt+32];
    }
    Lbts32->totalbit = 32 ;
    Rbts32->totalbit = 32 ;
}

DESProcess::FinalIP (bits Lbts32, bits Rbts32, bits *bts64)
{
    int cnt ;
    bits tempbts;
    for (cnt = 0;cnt <32 ; cnt++)
    {
        tempbts.bit[cnt] = Lbts32.bit[cnt];
        tempbts.bit[cnt+32]= Rbts32.bit[cnt];
    }
    tempbts.totalbit = 64 ;
    bitmaping(2,tempbts,bts64) ;
}

DESProcess::leftshift (bits *bit28, int times)
{
    bool lastbit ;
    int cnt1,cnt2;
    for (cnt1=0 ; cnt1<times ; cnt1++)
    {
        lastbit = bit28->bit[0] ;
        for (cnt2 =0 ; cnt2<bit28->totalbit-1 ; cnt2++)
            bit28->bit[cnt2] =bit28->bit[cnt2+1] ;
        bit28->bit[bit28->totalbit-1]=lastbit ;
    }
```

```
}

DESProcess::pc1tran (bits bts64, bits *Cbts28, bits *Dbts28)
{
    int cnt ;
    bits tempbts;

    bitmaping(5,bts64,&tempbts) ;
    for (cnt = 0 ;cnt <28; cnt++ )
    {
        Cbts28->bit[cnt]=tempbts.bit [cnt];
        Dbts28->bit[cnt]=tempbts.bit [cnt+28];
    }
    Cbts28->totalbit = 28 ;
    Dbts28->totalbit = 28 ;
}

DESProcess::pc2tran (bits Cbts28, bits Dbts28, bits *Kbts48)
{
    int cnt ;
    bits tempbts;
    for (cnt = 0;cnt <28 ; cnt++)
    {
        tempbts.bit[cnt] = Cbts28.bit[cnt];
        tempbts.bit[cnt+28]= Dbts28.bit[cnt];
    }
    tempbts.totalbit = 56 ;
    bitmaping(6,tempbts,Kbts48) ;
}

DESProcess::keygen (bits key, bitarray *subkey)
{
    bits Cbts28,Dbts28,Kbts48 ;
    int cnt ;

    pc1tran (key, &Cbts28, &Dbts28) ;
    for (cnt=0; cnt <16 ; cnt++ )
    {
        leftshift(&Cbts28,ls[cnt]);
        leftshift(&Dbts28,ls[cnt]);
        pc2tran(Cbts28,Dbts28,&Kbts48) ;
        subkey->bitarr[cnt]=Kbts48 ;
    }
}
```

```
DESProcess::etran (bits bts32, bits * bts48)
{
    bitmaping(3,bts32,bts48) ;
}

DESProcess::bitToSBoxPos (bits bts48, int boxno, int * row, int * col)
{
    * row = (bts48.bit[boxno * 6]? 2:0)+(bts48.bit[boxno * 6+5]? 1:0) ;
    * col = (bts48.bit[boxno * 6+1]? 8:0)+(bts48.bit[boxno * 6+2]? 4:0) +
            (bts48.bit[boxno * 6+3]? 2:0)+(bts48.bit[boxno * 6+4]? 1:0) ;
}

DESProcess::sboxtran (bits bts48, bits * bts32)
{
    int cnt1,cnt2, row, col, boxout ;

    for (cnt1=0 ; cnt1<8 ; cnt1++)
    {
        bitToSBoxPos (bts48, cnt1, &row, &col) ;
        boxout = sbox[cnt1][row * 16+col] ;
        for (cnt2 =3; cnt2>=0 ; cnt2--)
        {
            bts32->bit[cnt1 * 4+cnt2] = (boxout % 2 !=0)? true:false ;
            boxout/=2;
        }
    }
    bts32->totalbit =32 ;
}

DESProcess::ptran (bits bts32, bits * fout)
{
    bitmaping(4,bts32,fout) ;
}

DESProcess::ftran (bits inbts32, bits keyi, bits * fRes)
{
    bits bts48, bts32, xorRes48;

    etran (inbts32, &bts48) ;
    exclusiveOR(bts48, keyi, &xorRes48) ;
    sboxtran (xorRes48, &bts32) ;
    ptran (bts32, fRes) ;
}
```

```
DESProcess::exclusiveOR (bits bts1, bits bts2, bits *xorRes)
{
    int cnt ;

    xorRes->totalbit = bts1.totalbit ;
    for (cnt=0; cnt < bts1.totalbit;cnt++)
        xorRes->bit[cnt] = bts1.bit[cnt] ^ bts2.bit[cnt] ;
}

DESProcess::desblock (chars intext8, chars key, bool encode, chars *outtext8)
{
    int cnt, keypos ;
    bits inbts, Lbts, Rbts, oldLbts ;
    bits keybit,outbts,fRes ;
    bitarray subkey ;

    chartobits (intext8, &inbts);
    chartobits (key, &keybit);
    keygen (keybit,&subkey) ;

    InitIP  (inbts, &Lbts, &Rbts) ;
    for (cnt = 0 ; cnt<16 ; cnt++)
    {
        oldLbts = Lbts ;
        Lbts = Rbts ;
        keypos = encode ? cnt : (15-cnt) ;
        ftran(Rbts,subkey.bitarr[keypos],&fRes) ;
        exclusiveOR (oldLbts,fRes, &Rbts);
    }
        FinalIP (Rbts, Lbts, &outbts) ;
        bitstochar (outbts,outtext8) ;
}

DESProcess::desEncode (char mfile[], char key[], char cfile[])
{
    desCoding (mfile, key, cfile, true) ;
}

DESProcess::desDecode (char mfile[], char key[], char cfile[])
{
    desCoding (mfile, key,cfile, false) ;
}

DESProcess::desCoding (char mfile[], char key[], char cfile[], bool codedir)
```

```
{
    int cnt1,cnt2;
    chars inch, inkey. outch ;
    FILE * mfp ;
    FILE * cfp ;

    for (cnt1=0 ; cnt1<8; cnt1++)
        inkey. ch [cnt1]= key[cnt1];
    inkey. totalchar = 8 ;

    if ((mfp=fopen(mfile,"r"))==NULL)
    {cout<< "Cannot open the message file" << endl ;}
    else
    {
    cfp=fopen(cfile,"w") ;

    cnt1=0 ;
    inch. ch [cnt1]=fgetc(mfp) ;
    inch. totalchar =8 ;
    cnt2 =0 ;
    while (! feof(mfp))
    {
        cnt1++ ;
        cnt2++ ;
        inch. ch[cnt1]=fgetc(mfp) ;
        if (cnt1==7)
        {
            desblock (inch, inkey, codedir, &outch) ;
            for (cnt1=0; cnt1<8; cnt1++)
                fputc(outch. ch[cnt1],cfp);
            cnt1 = -1 ;
        }
    }
    cnt2 = cnt2%8 ;
    if (cnt2!=0)
    {
        for (cnt1=0;cnt1<(8-cnt2); cnt1++)
            inch. ch[cnt1+cnt2]=' ' ;
        inch. totalchar = 8;
        desblock (inch, inkey, codedir, &outch) ;
            for (cnt1=0; cnt1<8; cnt1++)
                fputc(outch. ch[cnt1],cfp);
    }
    fclose (mfp) ;
```

```
        fclose (cfp) ;
        }
    }

    DESProcess::dispfile (char fn[])
    {
        FILE * fp ;
        char ch ;
        if ((fp=fopen(fn,"r"))==NULL)
        {cout<< "Cannot open the file to display" << endl ;}
        else
        {
              while (! feof(fp))
              {
                    ch = fgetc(fp) ;
                    cout << ch ;
              }
              cout << endl ;
        }
    }

    float DESProcess::bitdiff (char mfile[], char cfile[], float * bitdiff8byte)
    {
        FILE * mfp, * cfp ;
        int ttch=0, xorRes, ttbitdiff=0;
        char mch, cch ;
        float bdiff=0 ;

        if ((mfp=fopen(mfile,"r"))==NULL)
        {cout<< "Cannot open the file to compare" << endl ;}
        if ((cfp=fopen(cfile,"r"))==NULL)
        {cout<< "Cannot open the file to compare" << endl ;}
        else
        {
              while (! feof(mfp)& ! feof(cfp))
              {
                ttch ++ ;
                mch=fgetc (mfp) ;
                cch=fgetc (cfp) ;
                xorRes = mch ^ cch ;
                ttbitdiff += (xorRes& 1) +(xorRes& 2)/2+(xorRes& 4)/4+(xorRes& 8)/8+
                (xorRes& 16)/16 +(xorRes& 32)/32+(xorRes& 64)/64+(xorRes& 128)/128;
              }
              bdiff = float(ttbitdiff) * 100/float(ttch * 8) ;
```

```
    }
    * bitdiff8byte = float(ttbitdiff) * 100/float(8 * 8) ;
    return bdiff ;
}
```

3. <destest. cpp>

```
#include "DESProcess. h"

void main (void)
{
    DESProcess testdes;
    char deskey1[]="security";
    char deskey2[]="tecurity";
    float bitdiff8byte=0 ;

    cout << "<<DES Algorithm Test>>" << endl ;
    cout << "The original message = "<<endl;testdes. dispfile ("c:\m1. txt") ;
    cout << "The DES key = " << deskey1<< endl;

    testdes. desEncode ("c:\m1. txt", deskey1, "c:\c1. txt") ;
    cout << "The Encode message = "<<endl;testdes. dispfile ("c:\c1. txt") ;

    testdes. desDecode ("c:\c1. txt", deskey1, "c:\mo1. txt") ;
    cout << "The Decode message = " <<endl ;testdes. dispfile ("c:\mo1. txt") ;

    cout << "(1) Change one bit in message to see the Encode message's changing below(use the
same key):" <<endl ;
    testdes. desEncode ("c:\m2. txt", deskey1, "c:\c2. txt") ;
    cout << "The Encode message = "<<endl ; testdes. dispfile ("c:\c2. txt") ;

    testdes. desDecode ("c:\c2. txt", deskey1, "c:\mo2. txt") ;
    cout << "The Decode message = " <<endl;    testdes. dispfile ("c:\mo2. txt") ;
    cout<< "The bit difference between two crypto message is " ;
    cout << testdes. bitdiff ("c:\c1. txt", "c:\c2. txt", &bitdiff8byte) << " %"<<endl ;
    cout << "The bit difference per 64 bits is " << bitdiff8byte << " %"<< endl ;
    cout << endl ;

    cout << "(2) Change one bit in DES key to see the Encode message's changing below;" <<endl ;
    cout << "The new DES key = " << deskey2 << endl ;
    testdes. desEncode ("c:\m1. txt", deskey2, "c:\c2. txt") ;
    cout << "The Encode message = "<<endl ;testdes. dispfile ("c:\c2. txt") ;
```

```
    testdes.desDecode ("c:\c2.txt", deskey2, "c:\mo2.txt") ;
    cout << "The Decode message = " <<endl; testdes.dispfile ("c:\mo2.txt") ;
    cout<< "The bit difference between two crypto message is " ;
    cout << testdes.bitdiff ("c:\c1.txt", "c:\c2.txt", &bitdiff8byte) << " %"<<endl ;
}
```

附录 B　IDEA 程 序

```
/ ***************************
  * 功能：完成 IDEA 块加密算法 *
  ***************************/
#include<stdio. h>
#include<process. h>
#include<stdib. h>
#include<conio. h>

#define maxim 65537
#define fuyi 65536
#define one 65535
#define round 8

void cip(unsigned IN[4], unsigned Z[7][10]);
void key(unsigned uskey[9], unsigned Z[7][10]);
void de_key(unsigned intZ[7][10], unsigned int DK[7][10]);
unsigned int inv(unsigned int xin);
unsigned int mul(unsigned int a, unsigned int b);

void main()
{
  int i,j,k,x;
  unsigned int Z[7][10],DK[7][10],XX[5],TT[5],YY[5];
  unsigned int uskey[9];
FILE * fpout, * fpin;
printf("\n Input Key");
for (i=1;i<=8;i++)
  scanf("%6u",&uskey[i]);
for (i=0;i<9;i++)
  uskey[i]=100+i * 3;
key (uskey,Z)/ * 产生加密子密钥 Z[i][r] * /
key (uskey,Z);
de_key (Z,DK); / * 计算解密子密钥 DK[i][r] * /
if ((fpin=(fopen("ekey. tex","w"))==NULL
  {
    printf("cannot open file!");
    exit(EXIT_FAILURE);
  }
```

```
for (i=0;i<7;i++)
  {
for (j=0;j<10;j++)
  fprintf(fpin,"%6u",z[i][j]);
fprintf(fpin,"\n");
}
fclose(fpin);

/* XX[1..3]中为明文 */
for (i=0;i<4;i++) XX[i]=2*i+101;
clrscr();
printf("Ming wen %6u %6u %6u %6u \n",XX[0],XX[1],XX[2],XX[3]);
if ((fpin=(fopen("ideaming.txt","w")))==NULL)
  {
    printf("cannot open file!");
    exit(EXIT_FAILURE);
  }
fprintf(fpin,"%6u %6u %6u %6u \n",XX[0],XX[1],XX[2],XX[3]);
fclose(fpin);
for(i=1;i<=30000;i++)
  cip(XX,YY,Z); /* 用密钥 Z 加密 XX 中的明文并存放在 YY 中 */
printf("\n\n Mingwen %6u %6u %6u %6u \n",YY[0],YY[1],YY[2],YY[3]);
if((fpout=fopen("ideamiwn.tex","w"))==NULL)
  {
    printf("cannot open file!");
    exit(EXIT_FAILURE);
  }
fprintf(fpout,"%6u %6u %6u %6u \n",YY[0],YY[1],YY[2],YY[3]);
fclose(fpout);
for (i=1;i<=30000;i++)
  cip (YY,TT,DK); /* encipher YY to TT with Key DK */
printf("\n\jie Mi  %6u %6u %6u %6u  \n",TT[0],TT[1],TT[2],TT[3]);
if ((fpout=fopen ("dideaout.txt","w"))==NULL)
  {
    printf("cannot open file!");
    exit(EXIT_FAILURE);
  }
fprintf(fpout,"%6u %6u %6u %6u  \n",TT[1],TT[2],TT[3],TT[4]);
fclose(fpout);
}
/* 此函数执行 IDEA 算法中的加密过程 */
void cip(unsigned int IN[4],unsigned int OUT[4],unsigned int Z[7][10])
{
unsigned int r,x1,x2,x3,x4,kk,t1,t2,a;
```

```
x1=IN[0];x2=IN[1];x3=IN[2];x4=IN[3];
for (r=1;r<=8;r++)
  {
  /* 对 64 位的块进行分组运算 */
  x1=mul(x1,Z[1][r]);x4=mul(x4,Z[4][r]);
  x2=(x2+Z[2][r])&one;x3=(x3+Z[3][r])&one;
  /* MA 结构的函数 */
  kk=mul(Z[5][r],(x1^x3));
  t1=mul(Z[6][r],(kk+(x2^x4))&one;
  /* 随机变换 PI */
  x1=x1^t1;x4=x4^t2;a=x2^t2;x2=x3^t1;x3=a;
  }
  /* 输出转换 */
  OUT[0]=mul(x1,Z[1][round+1]);
  OUT[3]=mul(x4,Z[4][round+1]);
  OUT[1]=(x3+Z[2][round+1])&one;
  OUT[2]=(x2+Z[3][round+1])&one;
}
/* 用高低算法实现乘法运算 */
unsigned int mul(unsigned int a,unsigned intb)
{
  long int p;
  long unsigned q;
  if (a==0) p=maxim-b;
    else if (b==0) p=maxim-a;
          else
            { q=(unsigned long)a*(unsigned long)b;
              p=(q&one)-(q>>16);
              if (p<=0) p=p+maxim;
              {
  return (unsigned) (p&one);
}
/* 通过 Euclidean gcd 算法计算 xin 的倒数 */
unsigned inv(unsigned xin)
{
long n1,n2,q,r,b1,b2,t;
if (xin==0)
  b2=0;
    else
    {n1=maxim;n2=xin;b2=1;b1=0;
     do {
          r=(n1%n2);q=(n1-r)/n2;
          if(r==0)
          if(b2<0)b2=maxim+b2;
```

```
                else
                  {n1=n2;n2=r;
                   t=b2;
                   b2=b1-q*b2;b1=t;
                   }
                }while (r!=0);
            }
          return(unsigned long int)b2;
          }

          /* 产生加密子密钥 Z */
          void key(unsigned int uskey[9],unsigned int Z[7][10])
          {
          unsigned int S[54];
          int i,j,r;

          for (i=1;i<9;i++)
              S[i-1]=uskey[i];
        /* shifts */
        for (i=8;i<54;i++)
        {
          if ((i+2)%8==0) /* 对于 S[14],S[22],…进行计算 */
             S[i]=((S[i-7]<<9)^(S[i-14]>>7))&one;
            else if ((i+1)%8==0) /* 对于 S[15],S[23],…进行计算 */
                S[i]=((S[i-15]<<9)^(S[i-14]>>7))&one;
              else
                S[i]=((S[i-7]<<9)^(S[i-6]>>7))&one;
        }
        /* 取得子密钥 */
        for (r=1;r<=round-1;r++)
          for (j=1;j<7;j++)
            Z[j][r]=S[6*(r-1)+j-1];
        }

        /* 计算解密子密钥 DK */
        void de_key(unsigned int Z[7][10],unsigned int DK[7][10])
        {
        int j;
        for (j=1;j<=round+1;j++)
          { DK[1][round-j+2]=inv(Z[1][j]);
            DK[4][round-j+2]=inv(Z[4][j]);
            if (i==1|j==round+1)
            {
            DK[2][round-j+2]=(fuyi-Z[2][j])&one;
```

```
        DK[3][round-j+2]=(fuyi-Z[3][j])&one;
      }
    else
      {
        DK[2][round-j+2]=(fuyi-Z[3][j])&one;
        DK[3][round-j+2]=(fuyi-Z[2][j])&one;
      }
    }
  for (j=1;j<=round+1;j++)
      {
        DK[5][round+1-j]=Z[5][j]);
        DK[6][round+1-j]=Z[6][j]);
      }
  }
```

附录 C　AES 程序

```
#include "../std_defs.h"

static char *alg_name[] = {"rijndael", "rijndael.c", "rijndael"};

char **cipher_name()
{
    return alg_name;
}

#define LARGE_TABLES

ulbyte pow_tab[256];
ulbyte log_tab[256];
ulbyte sbx_tab[256];
ulbyte isb_tab[256];
u4byte rco_tab[10];
u4byte ft_tab[4][256];
u4byte it_tab[4][256];

#ifdef LARGE_TABLES
  u4byte fl_tab[4][256];
  u4byte il_tab[4][256];
#endif

u4byte tab_gen = 0;

u4byte k_len;
u4byte e_key[64];
u4byte d_key[64];

#define ff_mult(a,b) (a && b ? pow_tab[(log_tab[a] + log_tab[b]) % 255] : 0)

#define f_rn(bo, bi, n, k) \
    bo[n] = ft_tab[0][byte(bi[n],0)] ^ \
            ft_tab[1][byte(bi[(n + 1)& 3],1)] ^ \
            ft_tab[2][byte(bi[(n + 2)& 3],2)] ^ \
            ft_tab[3][byte(bi[(n + 3)& 3],3)] ^ *(k + n)
```

```
#define i_rn(bo, bi, n, k) \
    bo[n] = it_tab[0][byte(bi[n],0)] ^ \
            it_tab[1][byte(bi[(n + 3)& 3],1)] ^ \
            it_tab[2][byte(bi[(n + 2)& 3],2)] ^ \
            it_tab[3][byte(bi[(n - 1)& 3],3)] ^ *(k + n)

#ifdef LARGE_TABLES

#define ls_box(x) \
    (fl_tab[0][byte(x, 0)] ^ \
      fl_tab[1][byte(x, 1)] ^ \
      fl_tab[2][byte(x, 2)] ^ \
      fl_tab[3][byte(x, 3)] )

#define f_rl(bo, bi, n, k) \
    bo[n] = fl_tab[0][byte(bi[n],0)] ^ \
            fl_tab[1][byte(bi[(n + 1)& 3],1)] ^ \
            fl_tab[2][byte(bi[(n + 2)& 3],2)] ^ \
            fl_tab[3][byte(bi[(n + 3)& 3],3)] ^ *(k + n)

#define i_rl(bo, bi, n, k) \
    bo[n] = il_tab[0][byte(bi[n],0)] ^ \
            il_tab[1][byte(bi[(n + 3)& 3],1)] ^ \
            il_tab[2][byte(bi[(n + 2)& 3],2)] ^ \
            il_tab[3][byte(bi[(n + 1)& 3],3)] ^ *(k + n)

#else

#define ls_box(x) \
    ((u4byte)sbx_tab[byte(x, 0)] << 0) ^ \
    ((u4byte)sbx_tab[byte(x, 1)] << 8) ^ \
    ((u4byte)sbx_tab[byte(x, 2)] << 16) ^ \
    ((u4byte)sbx_tab[byte(x, 3)] << 24)

#define f_rl(bo, bi, n, k) \
    bo[n] = (u4byte)sbx_tab[byte(bi[n],0)] ^ \
        rotl(((u4byte)sbx_tab[byte(bi[(n + 1)& 3],1)]), 8) ^ \
        rotl(((u4byte)sbx_tab[byte(bi[(n + 2)& 3],2)]), 16) ^ \
        rotl(((u4byte)sbx_tab[byte(bi[(n + 3)& 3],3)]), 24) ^ *(k + n)

#define i_rl(bo, bi, n, k) \
    bo[n] = (u4byte)isb_tab[byte(bi[n],0)] ^ \
        rotl(((u4byte)isb_tab[byte(bi[(n + 3)& 3],1)]), 8) ^ \
        rotl(((u4byte)isb_tab[byte(bi[(n - 2)& 3],2)]), 16) ^ \
```

```
        rotl(((u4byte)isb_tab[byte(bi[(n + 1)& 3],3)]), 24) ^ *(k + n)

#endif

void gen_tabs(void)
{ u4byte i, t;
    ulbyte p, q;

    /* log and power tables for GF(2**8) finite field with */
    /* 0x11b as modular polynomial -- the simplest prmitive */
    /* root is 0x11, used here to generate the tables */

    for(i = 0,p = 1; i < 256; ++i)
    {
        pow_tab[i] = (ulbyte)p; log_tab[p] = (ulbyte)i;

        p = p ^ (p << 1) ^ (p& 0x80 ? 0x01b : 0);
    }

    log_tab[1] = 0; p = 1;

    for(i = 0; i < 10; ++i)
    {
        rco_tab[i] = p;

        p = (p << 1) ^ (p& 0x80 ? 0x1b : 0);
    }

    for(i = 0; i < 256; ++i)
    {
        p = (i ? pow_tab[255 - log_tab[i]] : 0); q = p;
        q = (q >> 7) | (q << 1); p ^= q;
        q = (q >> 7) | (q << 1); p ^= q;
        q = (q >> 7) | (q << 1); p ^= q;
        q = (q >> 7) | (q << 1); p ^= q ^ 0x63;
        sbx_tab[i] = (ulbyte)p; isb_tab[p] = (ulbyte)i;
    }

    for(i = 0; i < 256; ++i)
    {
        p = sbx_tab[i];

#ifdef LARGE_TABLES
```

```
        t = p; fl_tab[0][i] = t;
        fl_tab[1][i] = rotl(t, 8);
        fl_tab[2][i] = rotl(t, 16);
        fl_tab[3][i] = rotl(t, 24);
#endif
        t = ((u4byte)ff_mult(2, p)) |
            ((u4byte)p << 8) |
            ((u4byte)p << 16) |
            ((u4byte)ff_mult(3, p) << 24);

        ft_tab[0][i] = t;
        ft_tab[1][i] = rotl(t, 8);
        ft_tab[2][i] = rotl(t, 16);
        ft_tab[3][i] = rotl(t, 24);

        p = isb_tab[i];

#ifdef LARGE_TABLES

        t = p; il_tab[0][i] = t;
        il_tab[1][i] = rotl(t, 8);
        il_tab[2][i] = rotl(t, 16);
        il_tab[3][i] = rotl(t, 24);
#endif
        t = ((u4byte)ff_mult(14, p)) |
            ((u4byte)ff_mult(9, p) << 8) |
            ((u4byte)ff_mult(13, p) << 16) |
            ((u4byte)ff_mult(11, p) << 24);

        it_tab[0][i] = t;
        it_tab[1][i] = rotl(t, 8);
        it_tab[2][i] = rotl(t, 16);
        it_tab[3][i] = rotl(t, 24);
    }

    tab_gen = 1;
};

#define star_x(x) (((x)& 0x7f7f7f7f) << 1) ^ ((((x)& 0x80808080) >> 7) * 0x1b)

#define imix_col(y,x) \
    u = star_x(x); \
    v = star_x(u); \
    w = star_x(v); \
```

```
    t = w ^ (x); \
    (y) = u ^ v ^ w; \
    (y) ^= rotr(u ^ t, 8) ^ \
           rotr(v ^ t, 16) ^ \
           rotr(t,24)

/* initialise the key schedule from the user supplied key */

#define loop4(i) \
{ t = ls_box(rotr(t, 8)) ^ rco_tab[i]; \
    t ^= e_key[4 * i]; e_key[4 * i + 4] = t; \
    t ^= e_key[4 * i + 1]; e_key[4 * i + 5] = t; \
    t ^= e_key[4 * i + 2]; e_key[4 * i + 6] = t; \
    t ^= e_key[4 * i + 3]; e_key[4 * i + 7] = t; \
}

#define loop6(i) \
{ t = ls_box(rotr(t, 8)) ^ rco_tab[i]; \
    t ^= e_key[6 * i]; e_key[6 * i + 6] = t; \
    t ^= e_key[6 * i + 1]; e_key[6 * i + 7] = t; \
    t ^= e_key[6 * i + 2]; e_key[6 * i + 8] = t; \
    t ^= e_key[6 * i + 3]; e_key[6 * i + 9] = t; \
    t ^= e_key[6 * i + 4]; e_key[6 * i + 10] = t; \
    t ^= e_key[6 * i + 5]; e_key[6 * i + 11] = t; \
}

#define loop8(i) \
{ t = ls_box(rotr(t, 8)) ^ rco_tab[i]; \
    t ^= e_key[8 * i]; e_key[8 * i + 8] = t; \
    t ^= e_key[8 * i + 1]; e_key[8 * i + 9] = t; \
    t ^= e_key[8 * i + 2]; e_key[8 * i + 10] = t; \
    t = e_key[8 * i + 3]; e_key[8 * i + 11] = t; \
    t = e_key[8 * i + 4] ^ ls_box(t); \
    e_key[8 * i + 12] = t; \
    t ^= e_key[8 * i + 5]; e_key[8 * i + 13] = t; \
    t ^= e_key[8 * i + 6]; e_key[8 * i + 14] = t; \
    t ^= e_key[8 * i + 7]; e_key[8 * i + 15] = t; \
}

u4byte *set_key(const u4byte in_key[], const u4byte key_len)
{ u4byte i, t, u, v, w;

    if(! tab_gen)
```

```
        gen_tabs();

    k_len = (key_len + 31) / 32;

    e_key[0] = in_key[0]; e_key[1] = in_key[1];
    e_key[2] = in_key[2]; e_key[3] = in_key[3];

    switch(k_len)
    {
        case 4: t = e_key[3];
                for(i = 0; i < 10; ++i)
                    loop4(i);
                break;

        case 6: e_key[4] = in_key[4]; t = e_key[5] = in_key[5];
                for(i = 0; i < 8; ++i)
                    loop6(i);
                break;

        case 8: e_key[4] = in_key[4]; e_key[5] = in_key[5];
                e_key[6] = in_key[6]; t = e_key[7] = in_key[7];
                for(i = 0; i < 7; ++i)
                    loop8(i);
                break;
    }

    d_key[0] = e_key[0]; d_key[1] = e_key[1];
    d_key[2] = e_key[2]; d_key[3] = e_key[3];

    for(i = 4; i < 4 * k_len + 24; ++i)
    {
        imix_col(d_key[i], e_key[i]);
    }

    return e_key;
};

#define f_nround(bo, bi, k) \
    f_rn(bo, bi, 0, k); \
    f_rn(bo, bi, 1, k); \
    f_rn(bo, bi, 2, k); \
    f_rn(bo, bi, 3, k); \
    k += 4
```

```
#define f_lround(bo, bi, k) \
    f_rl(bo, bi, 0, k); \
    f_rl(bo, bi, 1, k); \
    f_rl(bo, bi, 2, k); \
    f_rl(bo, bi, 3, k)

void encrypt(const u4byte in_blk[4], u4byte out_blk[4])
{ u4byte b0[4], b1[4], *kp;

    b0[0] = in_blk[0] ^ e_key[0]; b0[1] = in_blk[1] ^ e_key[1];
    b0[2] = in_blk[2] ^ e_key[2]; b0[3] = in_blk[3] ^ e_key[3];

    kp = e_key + 4;

    if(k_len > 6)
    {
        f_nround(b1, b0, kp); f_nround(b0, b1, kp);
    }

    if(k_len > 4)
    {
        f_nround(b1, b0, kp); f_nround(b0, b1, kp);
    }

    f_nround(b1, b0, kp); f_nround(b0, b1, kp);
    f_nround(b1, b0, kp); f_nround(b0, b1, kp);
    f_nround(b1, b0, kp); f_nround(b0, b1, kp);
    f_nround(b1, b0, kp); f_nround(b0, b1, kp);
    f_nround(b1, b0, kp); f_lround(b0, b1, kp);

    out_blk[0] = b0[0]; out_blk[1] = b0[1];
    out_blk[2] = b0[2]; out_blk[3] = b0[3];
};

/* 解密 */

#define i_nround(bo, bi, k) \
    i_rn(bo, bi, 0, k); \
    i_rn(bo, bi, 1, k); \
    i_rn(bo, bi, 2, k); \
    i_rn(bo, bi, 3, k); \
    k -= 4

#define i_lround(bo, bi, k) \
```

```
    i_rl(bo, bi, 0, k); \
    i_rl(bo, bi, 1, k); \
    i_rl(bo, bi, 2, k); \
    i_rl(bo, bi, 3, k)

void decrypt(const u4byte in_blk[4], u4byte out_blk[4])
{ u4byte b0[4], b1[4], *kp;

    b0[0] = in_blk[0] ^ e_key[4 * k_len + 24]; b0[1] = in_blk[1] ^ e_key[4 * k_len + 25];
    b0[2] = in_blk[2] ^ e_key[4 * k_len + 26]; b0[3] = in_blk[3] ^ e_key[4 * k_len + 27];

    kp = d_key + 4 * (k_len + 5);

    if(k_len > 6)
    {
        i_nround(b1, b0, kp); i_nround(b0, b1, kp);
    }

    if(k_len > 4)
    {
        i_nround(b1, b0, kp); i_nround(b0, b1, kp);
    }

    i_nround(b1, b0, kp); i_nround(b0, b1, kp);
    i_nround(b1, b0, kp); i_nround(b0, b1, kp);
    i_nround(b1, b0, kp); i_nround(b0, b1, kp);
    i_nround(b1, b0, kp); i_nround(b0, b1, kp);
    i_nround(b1, b0, kp); i_tround(b0, b1, kp);

    out_blk[0] = b0[0]; out_blk[1] = b0[1];
    out_blk[2] = b0[2]; out_blk[3] = b0[3];
};
```

附录 D　RSA 程 序

BigInt. h

```
# ifndef CLASS_BIGINT
# define CLASS_BIGINT
# define TOTALBITS 64
# define TOTALBOXS 66
# define FULL      0xFFFFFFFFFFFFFFFF
# define ZERO      0x0

# define MAXDEC   0x3B9ACA00
# define MAXDEC2  0x0DE0B6B3A7640000
# define LenMAXHEX 16
# define LenMAXWRD 8
# define LenMAXDEC 9
# define LenZ         15

class BigInt
{
public:
    BigInt();
    virtual~BigInt();
    unsigned_int64 iVal[TOTALBOXS];    // value
    unsigned short ildx;               // idx of value

public:
    void setZero();
    void setUpperBlock(const short& which);
    void loadDEC(const String itext);
    void loadHEX(const String itext);
    void loadSTR(const String itext);
    String outputDEC();
    String outputHEX();
    String outputSTR();
    BigInt& operator<<=(const short& iBit);
    BigInt& operator>>=(const short& iBit);
    BigInt& operator=(const BigInt& p1);
    BigInt& operator=(const unsigned_int64& p1);
    friend BigInt operator+(const BigInt& p1, const BigInt& p2);
    friend BigInt operator-(const BigInt& p1,const unsigned_int64& p2);
```

```
    friend BigInt operator-(const BigInt& p1, const BigInt& p2);
    friend BigInt operator*(const BigInt& p1, const BigInt& p2);
    friend BigInt operator*(const BigInt& p1, const unsigned_int64& p2);
    friend BigInt operator/(const BigInt& p1, const BigInt& p2);
    friend BigInt operator%(const BigInt& p1, const BigInt& p2);
    friend inline bool operator>(const BigInt& p1, const BigInt& p2);
    friend inline bool operator--(const BigInt& p1, const BigInt& p2);
    friend inline bool operator!=(const BigInt& p1, const BigInt& p2);
    BigInt& operator ++();
    BigInt& operator --();
    short Get8BitBlock(const short& unUseHeader, const short& BitIndex);
    unsigned char Get4BitBlock(const short& which);
    unsigned char Get8BitBlock(const short& which);
    unsigned short Get12BitBlock(const short& which);
    unsigned char Get4BitBlockZero();
    unsigned_int64 First64Bits();
    unsigned_int64 First32Bits();
    void ShiftAdd(const unsigned_int64 p1);
};

unsigned char hdigit(unsigned char c);
bool BIisZero(const BigInt& x);
int BIcompare(const BigInt& p1,const BigInt& p2);
int BIcompare(const BigInt& p1,const unsigned_int64& p2);
void BIswap(BigInt& p1,BigInt& p2);
BigInt BIdivision(const BigInt& p1, const BigInt& p2, BigInt& pr);
BigInt BImodmul(const BigInt& p1, const BigInt& p2, const BigInt& pp);
BigInt BImodmul2(const BigInt& p1, const BigInt& p2, const BigInt& pb);
BigInt BImodexp(const BigInt& p1, const BigInt& p2, const BigInt& pp);
BigInt BIgcd(const BigInt& p1,const BigInt& p2);
BigInt BIinverse(const BigInt& p1, const BigInt& p2);     // 求可逆数计算
BigInt BImodpow2(const BigInt& p1, const BigInt& pp, const short& d);
// 建立预处理表
void MakePreTable(const BigInt& p1, const BigInt& pp);
// 两个大整数模乘(预处理)
BigInt BImodmulPre(const BigInt& p2, const BigInt& pp);
// 大整数(预处理)
BigInt BImodexpPre(const BigInt& p1, const BigInt& p2, const BigInt& pp);
BigInt BImodexpPre2(const BigInt& p1, const BigInt& p2, const BigInt& pp);
void BImul(const unsigned_int64& p1,const unsigned_int64& p2,unsigned_int64& high, unsigned
_int64& low);
void MakeThemClose(const BigInt& p1, BigInt& p2);
short FindZeroBit(const unsigned_int64& p1);
void MakeModBaseTable(const BigInt& pp);
```

```
// 建立预处理表(乘法积)
void MakeModMultTable(const BigInt& pp);
// Montgomery 乘法
BigInt MontMult(const BigInt& p1, const BigInt& p2, const BigInt& pp);
// Montgomery 模幂
BigInt MontExpo(const BigInt& p1,const BigInt& p2,const BigInt& pp);

#endif
```

BigInt.cpp

```
// 定义有关大整数本身的形态,以及四则运算、模、模乘、模幂和改进函数
// 以一个 Object 为处理单元,Object 与 Object 进行运算
// Sample:BigInt a, b, c;
//        c=a+b;(加法)
//
//

#include<vcl.h>
#include<stdio.h>
#include<math.h>
#include<dstring.h>
#include"BigInt.h"
// 预处理表格(用于模指数运算)
BigInt PreTable[16];
BigInt pre_base[16];
// 预处理表格(用于模乘运算)
BigInt TableModBase[36];
BigInt TableModMult[16];
//   unsigned char HexArray[]={ 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
//                              0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D,0x0E,0x0F};
unsigned char HArray[]={'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
unsigned_int64 LMASK[]={ 0x0L, 0x8000000000000000L, 0xC000000000000000L,
                         0xE000000000000000L, 0xF000000000000000L};
unsigned_int64 RMASK[]={0x0L, 0x0000000000000001L, 0x0000000000000003L,
                        0x0000000000000007L, 0x000000000000000FL};

//  a--f(61-66)01100001-01100110
unsigned char hdigit(unsigned char c)
{
    unsigned char hi, lo;
    hi=c >> 4;
    lo=c& 0x0f;
    if( hi== 0x04 ) return( lo + 0x09);
    if( hi== 0x06 ) return( lo + 0x09);
```

```
    return lo;
}
BigInt::BigInt()
{
    iIdx=0;
    for(int i=0; i<=TOTALBOXS; i--)
        iVal[i]=0;
}

BigInt::~BigInt()
{
}

void BigInt::setZero(void)
{
    iIdx=0;
    for(int i=0; i<=TOTALBOXS; i++)
        iVal[i]=0;
}

void BigInt::loadDEC(const String itext)
{
    int seg, bgn, end, ttl;
    char lstr[9];
    BigInt a, b;
    short k;

    ttl=itext.Length();
    seg=(itext.Length()-1)/LenMAXDEC;
    bgn=1;
    b.iIdx=0;
    for(int i=seg;i>=0;i--)
    {
        end=ttl%LenMAXDEC;
        if(end==0)end=LenMAXDEC;
        // Change to HEX
            for(int j=0;j<9;j++)
                lstr[j]='0';
            for(int j=1; j<=end; j++)
                lstr[9-j]=itext[bgn+end-j];
        b.iVal[0]=atol(lstr);
        if(bgn==1)
            a=b;
        else
```

```
            {
                // MAX DEC 1 000 000 000
                a=a * MAXDEC;
                a=a+b;
            }
            bgn=bgn+end;
            ttl=ttl-end;
        }
        iIdx=a.iIdx;
        for(int i=0; i<=iIdx; i++)
            iVal[i]=a.iVal[i];

    }

    void BigInt::loadHEX(const String itext)
    {
        int seg, bgn, end, ttl;
        unsigned_int64 tmp;

        ttl=itext.Length();
        seg=(itext.Length()-1)/LenMAXHEX;
        bgn=1;
        for(int i=seg; i>=0;i--)
        {
            end=ttl% LenMAXHEX;
            if(end== 0) end=LenMAXHEX;
            tmp= 0;
            for(int j=1;j<=end;j++)
            {
                tmp<<=4;
                tmp=tmp|hdigit(itext[bgn]);
                bgn++;
            }
            iVal[i]=tmp;
            ttl=ttl-end;
        }
        iIdx=seg;

    }

    void BigInt::loadSTR(const String itext)
    {
        int seg,bgn,end,ttl;
        unsigned_int64 tmp;
```

```
    ttl=itext.Length();
    seg=(itext.Length()-1)/LenMAXWRD;
    bgn=1;
    for(int i=seg; i>=0; i--)
    {
        end=ttl%LenMAXWRD;
        if(end==0) end=LenMAXWRD;
        tmp=0;
        for(int j=1; j<=end; j++)
        {
            tmp<<=8;
            tmp=tmp|(unsigned char)itext[bgn];
            bgn++;
        }
        iVal[i]=tmp;
        ttl=ttl-end;
    }
    iIdx=seg;
}

int BIcompare(const BigInt& p1,const BigInt& p2)
{
    short i,j;

    i=p1.iIdx;
    j=p2.iIdx;
    if(i>j)
        return 1;
    else if(i<j)
        return -1;
    else
    {
        while(i>=0)
        {
            if(p1.iVal[i]>p2.iVal[i])
                return 1;
            else if(p1.iVal[i]<p2.iVal[i])
                return -1;
            i--;
        }
        return 0;
    }
}
```

```
int BIcompare(const BigInt& p1,const unsigned_int64& p2)
{
    if(p1.iIdx>0)
        return 1;
    else
    {
        if(p1.iVal[0]> p2)
            return 1;
        else if(p1.iVal[0]<p2)
            return -1;
        else
            return 0;
    }
}

BigInt& BigInt::operator++()
{
    unsigned_int64 tmp, carry;
    int i;

    i=0;
    carry=1;
    while( carry==1)
    {
        tmp=iVal[i]+ carry;
        if(tmp<iVal[i])
            carry=1;
        else
            carry=0;
        iVal[i]=tmp;
        i++;
    }
    if(iVal[iIdx-1] !=0)
        iIdx++;
    return * this;
}

BigInt& BigInt::operator--()
{
    unsigned_int64 tmp,carry;
    int i;

    i=0;
    carry=1;
```

```
    while(carry==1)
    {
        if(iVal[i]==0x0)
        {
            iVal[i]=FULL;
            carry=1;
        }
        else
        {
            iVal[i]--;
            carry=0;
        }
    }
    if( iVal[iIdx]==0)
        iIdx--;
    return *this;
}

void BigInt::ShiftAdd(const unsigned_int64 p1)
{
    int i;
    unsigned_int64 carry;
    iVal[iIdx+1]=iVal[iIdx]>>60;

    for (int i=iIdx;i>0;i--)
        iVal[i]=(iVal[i]<<4)|(iVal[i-1]>>60);

    iVal[0]=(iVal[0]<<4);
    carry=p1;
    i=0;
    while(carry !=0)
    {
        iVal[i]=iVal[i]+carry;
        if(iVal[i]<carry)
            carry=1;
        else
            carry=0;
        i++;
    }
    if(iVal[iIdx+1] !=0)
        iIdx++;

}
```

```
BigInt& BigInt::operator<<=(const short& iBit)
{
// LM=0x80000000, 0xC0000000, 0xE0000000, 0xF0000000,....,0xFFFF0000;
// 0000 is 0000, F000 is FFFF
    unsigned __int64 b1,b2,LM;
    short i, shift, ishift;

    if((iIdx==0)&&(iVal[0]==0))
        return *this;

    // left shift over TOTALBITS bits
    ishift=iBit;
    while(ishift>=TOTALBITS)
    {
        for (i=iIdx; i>=0; i--)
            iVal[i+1]=iVal[i];
        iVal[0]=0;
        iIdx++;
        ishift-=TOTALBITS;
    }
    if(ishift==0)
        return *this;

    // left shift less than TOTALBITS bits
    b1=0x0;
    shift=TOTALBITS-ishift;
    LM=FULL<<Shift;
    for(i=0; i<=iIdx; i++)
    {
        b2=iVal[i]&LM;
        iVal[i]=(iVal[i]<<ishift)|b1;
        if(b2==0x0)
          b1=0x0;
        else
          b1=b2>>shift;
    }
    if(iIdx!=TOTALBOXS-1)
    {
        if(b1!=0x0)
        {
            iIdx++;
            iVal[iIdx]=b1;
        }
    }
```

```
    else
    {
        for(i=TOTALBOXS-1;i>=0;i--)
            if( iVal[i] !=0)
            {
                iIdx=i;
                break;
            }
        if(( i==-1)&&( iIdx != 0))
            iIdx=0;
    }
    return * this;
}

BigInt& BigInt::operator>>=(const short& iBit)
{
// RM=0x0+0001, 0x0+0003, 0x0+0007, 0x0+000F,..., 0xFFFFF+};
    unsigned_int64 b1,b2,RM;
    short i, shift, ishift;

    if(( iIdx==0)&( iVal[0]==0))
        return * this;

    ishift=iBit;
    while( ishift>= TOTALBITS)
    {
        for (i=0; i<iIdx; i++ )
            iVal[i]=iVal[i+1];
        iVal[iIdx]=0;
        iIdx--;
        ishift-=TOTALBITS;
    }
    if( ishift==0)
        return * this;

    b1=0x0;
    shift=TOTALBITS-ishift;
    RM=FULL>> Shift;
    for(i=iIdx;i>=0;i--)
    {
        b2=iVal[i]& RM;
        iVal[i]=iVal[i]>>ishift |b1;
        if(b2==0x0)
          b1=0x0;
```

```
            else
                b1=b2<< shift;
        }
        if(( iVal[iIdx]==0)&&( iIdx>0))
            iIdx--;
        return *this;
    }

    String BigInt::outputDEC(void)
    {
        BigInt a, b, c;
        int i,j;
        String aa;

        aa="";
        a.iIdx=iIdx;
        for( int i=iIdx; i>=0;i--)
            a.iVal[i]=iVal[i];
        b.iIdx=0;
        b.iVal[0]=MAXDEC2;
        iIdx=0;
        while( BIcompare(a, MAXDEC2)> 0)
        {
            a=BIdivision(a,b,c);
            iVal[iIdx++]=c.iVal[0];
        }
        iVal[iIdx]=a.iVal[0];
        // output the value
        for( int i=iIdx; i>=0; i--)
            aa=aa+ IntToStr((_int64)iVal[i]);
        return aa;
    }

    String BigInt::outputHEX(void)
    {
    //    unsigned char cs[640];
        unsigned char cs[8192];

        String aa;
        unsigned short k,l;

        for(int i=0;i<8192;i++) cs[i]='\0';
        l=0;
        for(int i=iIdx; i>=0; i--) {
```

```
        for(int j=60;j>=0;j-=4)
            k=(iVal[i]>>j)&0x0F;
            cs[l++]=HArray[k];
        }
    }
//    cs[l]='\0';
    aa.sprintf(cs);
    return aa;
}

String BigInt::outputSTR(void)
{
    unsigned char cs[8192];
    int i, j, k;
    String aa;

    for(int i=0;i<8192;i++) cs[i]='\0';
    j=0;

    cs[j]=(iVal[iIdx]>>56)&0xff;
    if(cs[j]!='\0')j++;
    cs[j]=(iVal[iIdx]>>48)&0xff;
    if(cs[j]!='\0')j++;
    cs[j]=(iVal[iIdx]>>40)&0xff;
    if(cs[j]!='\0')j++;
    cs[j]=(iVal[iIdx]>>32)&0xff;
    if(cs[j]!='\0')j++;
    cs[j]=(iVal[iIdx]>>24)&0xff;
    if(cs[j]!='\0')j++;
    cs[j]=(iVal[iIdx]>>16)&0xff;
    if(cs[j]!='\0')j++;
    cs[j]=(iVal[iIdx]>>8)&0xff;
    if(cs[j]!='\0')j++;
    cs[j]=(iVal[iIdx]&0xff;
    if(cs[j]!='\0')j++;
    for (i=iIdx-1;i>=0;i--)
    {
        for( k=56; k>=0; k-=8)
            cs[j++]=(iVal[i]>>k)&0xff;
    }
//    cs[j]='\0';
    aa.sprintf(cs);
    //    sprintf(cs);
    return aa;
```

```
}

short BigInt::Get8BitBlock(const short& unUseHeader, const short& BitIndex)
{
    // BitIndex must over ZERO!
    // Header over Zero!!!
    short Header, ots;
    unsigned_int64 m,m2;

    ots= -1;
    Header=unUseHeader+4 * BitIndex;
    while( Header> 0)
    {
        Header-= 64;
        ots++;
    }
    Header= -Header;

    m=iVal[iIdx-ots]>>Header;
    if (Header>60)
    {
        Header=64-Header;
        m=m& (iVal[iIdx]<<Header);
    }
    return m& 0x0F;
}

unsigned char BigInt::Get4BitBlock(const short& which)
{
    short iwhich, idx;
    idx=which>> 4;
    iwhich=(which& 0xF)<< 2;
    return(iVal[idx]>> iwhich )& 0x0F );
}

unsigned char BigInt::Get8BitBlock(const short& which)
{
    // -15-14-13-12-11-10-9-8-7-6-5-4-3-2-1-0-
    // Total 16's 4- bits Block in 64bit Integer

    short iwhich, idx;
    unsigned short m;

    idx=which>> 4;
```

```
    iwhich=(which & 0xF)<< 2;
    if(iwhich !=0)
    {
        m=iVal[idx]>> (iwhich- 4 );
    }
    else
    {
        m=( iVal[idx]& 0xF)<< 4;
        if (idx> 0 )
            m=m|(iVal[idx -1]>>60);
    }
    return (m& 0xFF);
}

unsigned short BigInt::Get12BitBlock(const short& which)
{
    short iwhich, idx;
    unsigned short m;

    idx=which >> 4;
    iwhich=(which & 0xF)<< 2;
    if( iwhich==0)
    {
        m=(iVal[idx]& 0xF )<< 8;
        if (idx>0 )
            m=m|( iVal[idx-1]>> 56 );
    }
    else if( iwhich== 4)
    {
        m=( iVal[idx]& 0xFF)<< 4;
        if (idx>0 )
            m=m|( iVal[idx-1]>> 60);
    }
    else
    {
        m=iVal[idx]>> (iwhich-8 );
    }
    return( m& 0x0FFF);
}
unsigned char BigInt::Get4BitBlockZero()
{
    // -15-14-13-12-11-10-9-8-7-6-5-4-3-2-1-0-
    // Total 16's 4-bits Block in 64bit Integer
```

```
    unsigned char iwhich;
    unsigned_int64 m;

    if( iVal[iIdx]> 0xFFFFFFFF)
    {
        m=0x0FFFFFFFFFFFFFFF;
        iwhich=0;
    }
    else
    {
        m=0x0FFFFFFF;
        iwhich=8;
    }
    while(iVal[iIdx]<m)
    {
        m>>=4;
        iwhich++;
    }
    return iwhich;
}

unsigned_int64 BigInt::First64Bits()
{
    unsigned_int64 m;
    int i=0;

    m=iVal[iIdx];
    if( iIdx==0)
        return m;

    while( m& LMASK[2] != LMASK[2])
    {
        m<<1;
        i++;
    }
    return(m|iVal[iIdx-1]>>(64-i));
}

unsigned_int64 BigInt::First32Bits()
{
    unsigned_int64 m;
    int i=0;

    m=iVal[iIdx];
```

```
    if( iIdx==0)
    {
        if( m> 0xFFFFFFFF)
            return(m>>32);
        else
            return m;
    }
    while( m& LMASK[2] != LMASK[2])
    {
        m<<1;
        i++;
    }
    return(( m|iVal[iIdx-1] >>( 64-i)) >> 32);

}

void BigInt::setUpperBlock(const short& which)
{
    unsigned_int64 m;
    short iwhich;

    for(int i; i<=iIdx; i++)
        iVal[i]=0;
    m=0x1000000000000000;
    if(which> 0)
    {
        iwhich=(which-1) << 2;
        iVal[iIdx]=m>> iwhich;
    }
    else
    {
        iIdx++;
        iVal[iIdx]=1;
    }
}

bool BIisZero(const BigInt& x)
{   // if the value is equal to zero return TRUE, else FALSE
    if(x.iIdx !=0)      return false;
    if(x.iVal[0] !=0)return false;
    return true;
}

BigInt& BigInt::operator=(const BigInt& p1)
```

```
{   // operator=
    iIdx=p1.iIdx;
    for(int i=0; i<TOTALBOXS; i++)
        iVal[i]=p1.iVal[i];
    return *this;
}

BigInt& BigInt::operator=(const unsigned_int64& p1)
{
    iIdx=0;
    iVal[0]=p1;
    return *this;
}

inline bool operator>(const BigInt& p1,const BigInt& p2)
{   // operator>
    short stop;

    stop=p1.iIdx;
    if (stop>p2.iIdx)
        return true;
    else if( stop<p2.iIdx)
        return false;
    else
    {
        while( stop>=0)
        {
            if (p1.iVal[stop]>p2.iVal[stop])
                return true;
            else if(p1.iVal[stop]<p2.iVal[stop])
                return false;
            stop--;
        }
        return false;
    }
}

inline bool operator==(const BigInt& p1, const BigInt& p2)
{   // operator==
    if(p1.iIdx!=p2.iIdx)
        return false;
    for(int i=0; i<=p1.iIdx; i++)
    {
        if(p1.iVal[i]!=p2.iVal[i])
```

```
            return false;
    }
    return true;
}

inline bool operator !=(const BigInt& p1, const BigInt& p2)
{   // operator==
    if( p1.iIdx != p2.iIdx)
        return true;
    for(int i=0; i<=p1.iIdx;i++)
    {
        if(p1.iVal[i] != p2.iVal[i])
            return true;
    }
    return false;
}

BigInt operator+(const BigInt& p1,const unsigned_int64& p2)
{
    int     i;
    BigInt result;
    unsigned_int64 carry;

    i=0;
    result.iIdx=p1.iIdx;
    carry=p2;
    while(i<=p1.iIdx)
    {
        result.iVal[i]=p1.iVal[i]+ carry;
        if( result.iVal[i]< carry)
            carry=1;
        else
            carry=0;
        i++;
    }

    if (result.iVal[result.iIdx+1] !=0 )
        result.iIdx++;
    return result;

}

BigInt operator+(const BigInt& p1, const BigInt& p2)
{   // operator+
```

```
    int     i;
    short stop;
    BigInt result;
    unsigned_int64 carry;

    if(BIisZero(p1))
        return p2;
    if( BlisZero(p2))
        return p1;

    result. setZero();
    stop=p1. iIdx;
    if( stop< p2. iIdx)stop=p2. iIdx;
    result. iIdx=stop;

    carry=0;
    for(i=0; i<=stop; i++)
    {
        result. iVal[i]=p1. iVal[i]+ p2. iVal[i]+ carry;
        if (result. iVal[i]<p2. iVal[i])
            carry=1;
        else
            carry=0;
    }
    if(carry==1)
    {
        stop++;
        result. iVal[stop]=1;
        result. iIdx=stop;
    }
    return result;
}

BigInt operator-(const BigInt& p1, const BigInt& p2)
{  // operator-
    int i;
    BigInt result;
    unsigned_int64 carry;

    if(BlisZero(p2))
        return p1;
    else
    {
        carry=0;
```

```
        for( i=0; i<=p1.iIdx; i++)
        {
            if(p1.iVal[i]>=p2.iVal[i])
            {
                result.iVal[i]=p1.iVal[i]-p2.iVal[i];
                if(( result.iVal[i]== 0x0)&&( carry==0x1))
                    result.iVal[i]=FULL;
                else
                {
                    result.iVal[i]  =carry;
                    carry=0x0;
                }
            }
            else
            {
                result.iVal[i]=FULL-p2.iVal[i]+ p1.iVal[i]+ RMASK[1]-carry;
                carry=RMASK[1];
            }
        }
        result.iIdx= p1.iIdx;
        while(( result.iIdx> 0)&&( result.iVal[result.iIdx]==0))
            result.iIdx--;
    }
    return result;
}

BigInt operator *(const BigInt& p1,const unsigned_int64&. p2)
{   // operator *
    // result=p1 * p2.

    BigInt result;
    unsigned_int64 carry, dhigh, dlow;
    int i;

    if( BIisZero(p1))
        return result;
    if(p2==0)
        return result;

    result.iIdx=p1.iIdx+1;
    carry=0;
    for(i=0;i<=p1.iIdx; i++)
    {
            if( result.ival[i]<carry)
```

```
                carry=1;
            else
                carry=0;
            BImul (p1.iVal[i],p2, dhigh, dlow);
            result.iVal[i]+= dlow;
            if(result.iVal[i]<dlow)
                carry ++;
            carry+=dhigh;
        result.iVal[i+1]+=carry;
    }
    while( result.iVal[result.iIdx]==0)
        result.iIdx--;
    return result;
}

BigInt operator *(const BigInt& p1, const BigInt& p2)
{   // operator *
    // result=p1 * p2.

    BigInt result;
    unsigned_int64 carry,dhigh,dlow;
    int i, j, k;

    //   carry= dhigh
    if(BIisZero(p1))
        return result;
    if( BIisZero(p2))
        return result;

    result.iIdx=p1.iIdx+p2.iIdx+1;
    if( result.iIdx> TOTALBOXS)
        result.iIdx=TOTALBOXS;
    for( i=0; i<=p2.iIdx; i++)
    {
        carry=0;
        if(p2.iVal[i] != 0)
        {
            for(j=0;j<=p1.iIdx;j++)
            {
                BImul(p1.iVal[j],p2.iVal[i],dhigh,dlow);
                result.iVal[i+j]+=carry;
                if (result.iVal[i+j]<carry)
                    carry=1;
                else
```

```
                carry=0;
            result.iVal[i-j]+=dlow;
            if(result.iVal[i-j]<dlow)
                carry++;
            carry += dhigh;
        }
    }
    result.iVal[i-p1.iIdx+1]+= carry;
}
while(result.iVal[result.iIdx]==0)
    result.iIdx--;
return result;
}

// 两大整数相除(参考 BIdivision)
BigInt operator/(const BigInt& p1, const BigInt& p2)
{  // operator/
   // result=a/b...d
   BigInt d;
   return BIdivision(p1,p2,d);

}

BigInt operator %(const BigInt& p1, const BigInt& p2)
{  // operator%
   //C= p1 /p2+ result
   BigInt a, mb;
   int test;

   a=p1;
   if( BIcompare(a, p2)>0)
   {
      // Get a number(mb) that is largest multiple of (p2) less then(a).
      mb=p2;
      // Make mb close to a quickly.
      Make ThemClose(a,mb);
      while( BIcompare(a, mb)>0)
          mb<<=1;
      while( BIcompare(a, p2)>0)
      {
         while( BIcompare(a,mb)<0)
             mb>>=1;
         a=a-mb;
         }
      }
      if( BIcompare(a, p2)== 0)
          a.setZero();
```

```
    return a;
}

void BIswap(BigInt& p1, BigInt& p2)
{
    BigInt mp;
    tmp= p1;
    p1=p2;
    p2=tmp;
}

BigInt BIdivision(const BigInt& p1, const BigInt& p2, BigInt& pr)
{
    BigInt q, r,pb;
    int test;

    test= BIcompare(p1,p2);
    if( test> 0)
    {
        r=p1;
        // Get a number pb that is the largest multiple of (p2) less that (p1)
        pb=p2;
        MakeThemClose(r, pb);
        while( BIcompare(r, pb)>0) pb<<= 1;
        while( BIcompare(r,pb)<0) pb >>= 1;
        while(( BIcompare(p2, r)<=0) ||( BIcompare(p2,pb)<=0))
        {
            if( BIcompare(r,pb)>=0)
            {
                r=r-pb;
                q<<= 1;
                q++;
            }
            else
            {
                q<<=1;
            }
            pb>>= 1;
        }
        pr=r;
    }
    else if( test==0)
    {
        q++;
        pr.setZero();
    }
    else
```

```
    {
        q.setZero();
        pr=p1;
    }
    return q;
}

//  建立预处理表(底数)
void MakeModBaseTable(const BigInt& pp)
{
    TableModBase[0]=pp;
    for(int i=1;i<36;i++)
    {
        TableModBase[i]=TableModBase[i-1];
        TableModBase[i]<<=-2;
    }
}

//  建立预处理表(乘法积)
void MakeModMultTable(const BigInt& pp)
{
    TableModMult[0].setZero();
    TableModMult[1]=pp;
    for(int i=2; i<16; i++)
    {
        TableModMult[i]=TableModMult[i-1]+pp;
        if(BIcompare(TableModMult[i],TableModBase[0])>=0)
            TableModMult[i]=TableModMult[i]-TableModBase[0];
    }
}

//  两个大整数模乘
BigInt BImodmul(const BigInt& p1, const BigInt& p2, const BigInt& pp)
{   // operator * %
    // result=( p1 * p2)% pp.

    BigInt a, b, c, result;
    short i,j, kbit, kbit2;

    if(BIisZero(p1))
        return result;
    if(BIisZero(p2))
        return result;

    if(TableModBase[0] !=pp)
        MakeModBaseTable(pp);
```

```
    // Suppose p1, p2< pp
        a=p1 %pp;
        b=p2%pp;
        kbit2 = FindZeroBit(pp.iVal[pp.iIdx]);
        for(i= b.iIdx; i>=0; i--)
        {
            result<<= 64;
            if(b.iVal[i] != 0)
            {
                c= a * b.iVal[i];
                result=result+c;
            }
            if(BIcompare(result,pp)>=0)
            {
                kbit=kbit2- FindZeroBit(result.iVal[result.iIdx]);
                if(result.iIdx>pp.iIdx ) kbit +=64;
                kbit>>= 1;
                for( j=kbit; j>=0; j--)
                {
                    while(BIcompare(result,TableModBase[j])>=0)
                        result=result- TableModBase[j];
                }
            }
        }
        return result;

    }
    //  两个大整数模乘(预处理方法)
    BigInt BImodmul2(const BigInt& p1, const BigInt& p2, const BigInt& pb)
    {   // operator * %
        // result=(p1 * p2)% pp.

        BigInt pre_a[16], a, b, pb2, pb4,pb8,result;
        unsigned_int64 ma1, ma2;
        int i, j, k, test;

        if(BIisZero(p1))
            return result;
        if(BIisZero(p2))
            return result;

        a= p1;
        b= p2;

        test=BIcompare(a,pb);
        if(test> 0)
            a=a%pb;
```

```
else if(test==0)
    return result;

test= BIcompare(b,pb);
if(test> 0)
    b= b%pb;
else if(test== 0)
    return result;

//  pre-calculation. a[0000], a[0001], a[0010],…, a[1111]
pre_a[0].setZero();
pre_a[1]=a;
for( i=2; i<16; i++)
{
    j= i-1;
    pre_a[i]=pre_a[j]+a;
    if(BIcompare(pre_a[i],pb)> 0)
        pre_a[i]= pre_a[i]-pb;
}
pb2= pb;
pb2<<= 1;
pb4= pb2;
pb4<<= 1;
pb8= pb4;
pb8<<= 1;
for( i=b.iIdx; i>=0; i--)
{
    ma1= LMASK[4];
    for(j=LenZ;j>=0;j--)
    {
        result<<= 4;
        ma2= b.iVal[i]& ma1;
        if( ma2 != 0)
        {
            k=j<<2;
            ma2>>= k;
            result= result+pre_a[ma2& 0x0f];
        }
        while(BIcompare(result, pb8)> 0)
            result= result-pb8;
        while(BIcompare(result,pb4)> 0)
            result= result-pb4;
        while(BIcompare(result,pb2)> 0)
            result= result-pb2;
        while(BIcompare(result, pb)> 0)
            result= result-pb;
        ma1>>= 4;
    }
}
```

```
    return result;
}

//  大整数模幂
BigInt BImodexp(const BigInt& p1, const BigInt& p2, const BigInt& pb)
{  //   operator -%
   // result=( p1 - p2)%pp.
   BigInt a, b, result;

   if(BIisZero(p1))
       return result;
   if(BIisZero(p2))
   {
       result++;
       return result;
   }
   a=p1%pb;
   b=p2%pb;
   result++;
   do{
      if(( b.iVal[0]& RMASK[1])==RMASK[1])
          result=BImodmul(a, result, pb);
      b>>= 1;
      if(!BIisZero(b))
          a=BImodmul(a,a,pb);
   } while(! BIisZero(b));
   return result;
}

BigInt BIgcd(const BigInt& p1, const BigInt& p2)
{   // gcd
    // result=gcd(p1,p2)
    BigInt a, b;

    a=p1;
    b=p2;
    while(!BIisZero(b))
    {
        a=a%b;
        BIswap(a, b);
    }
    return a;
}

//  求逆运算
BigInt BIinverse(const BigInt& p1, const BigInt& p2)
{
   // 1 * p1=1 mod p2
   // i=1/p1
   // i:range 1,...,n-1
```

```
    BigInt y,g0,g1,g2,v0,v1,v2;
    g0 = p2;
    g1 = p1;
    v1 = 1;
    while(!BIisZero(g1))
    {
        y=BIdivision(g0,g1,g2);
        g0=g1;
        g1=g2;
        v2=y*v1;
        while(BIcompare(v0,v2)<0)
        v0=v0+p2;
        v0=v0-v2;
        BIswap(v0,v1);
    }
    return v0;
}

BigInt BImodpow2(const BigInt& p1,const BigInt& pp,const short& d)
{
    //result=p1^(2^n)mod pp;
    BigInt result;
    short i;

    result=p1;
    i=d;
    while(i>0)
    {
       result=BImodmul(result,result,pp);
    i--;
    }
    return result;
}

//  两个大整数模乘(预处理)
BigInt BImodmulPre(const BigInt& p2,const BigInt& pb)
{  //operator * %
   //result=(p1 * p2) % pp.

   BigInt b, pb2, pb4,pb8,result;
   unsigned_int64 ma1,ma2;
   int i,j,k;

   b=p2;
```

```
    pb2=pb;
    pb2<<=1;
    pb4=pb2;
    pb4<<=1;
    bp8   pb4;
    pb8<<=1;
    for(i=p2.iIdx;i>=0;i--)
    {
        ma1=LMASK[4];
        for(j=LenZ;j>=0;j--)
        {
            result<<=4;
            ma2=b.iVal[i]& ma1;
            if(ma2 !=0)
            {
                k=j<<2;
                ma2>>=k;
                result=result+pre_base[ma2& 0x0f];
            }
            while(BIcompare(result,pb8)>0)
                result=result-pb8;
            while(BIcompare(result,pb4)>0)
                result=result-pb4;
            while(BIcompare(result,pb2)>0)
                result=result-pb2;
            while(BIcompare(result,pb)>0)
                result=result-pb;
            ma1>>=4;
        }
    }
    return result;
}

void MakePreTable(const BigInt& p1,const BigInt& pp)
{
    int i,j;
    PreTable[0].setZero();
    PreTable[0]   ;
    PreTable[1]=p1;

    //pre calculation. a[0000],a[0001],a[0010],...,a[1111]
    for(i=2;i<16;i++)
        PreTable[i]=BImodmul(PreTable[i-1],p1,pp);
}
```

```
//  模幂(预处理)
BigInt BImodexpPre2(const BigInt& p1,const BigInt& p2,const BigInt& pb)
{
    BigInt a, b, result;
    unsigned_int64 tmp;
    short iz, Lt, count, kZero;

    if(BlisZero(p1))
        return result;
    if(BlisZero(p2))
    {
        result++;
        return result;
    }

    a=p1 % pb;
    b=p2 % pb;
    result++;
    MakePreTable(a,pb);

    Lt=b.iIdx;
    count=TOTALBITS*(Lt+1);

    // find total Zero bit to shift.
    // Make Sure the FIRST BIT must be 1.
    kZero=FindZeroBit(b.iVal[Lt]);
    b<<=kZero;
    count-=kZero;

    do {
        tmp=b.iVal[Lt] & LMASK[4];
        if(count>3)
        {
            tmp>>=60;
            b<<=4;
            count-=4;
        }
        else
        {
            tmp>>=(TOTALBITS-count);
            b<<=count;
            count-=count;
        }
```

```
            result=BImodmul(result,PreTable[tmp& 0x0f],pb);
            iz=0;
            if(count !=0)
            {
                while(((b.iVal[Lt_& LMASK[1])!=LMASK[1])&&(count>0))
                {
                    b<<=1;
                    count--;
                    iz++;
                }
                if(count>3)
                    iz+=4;
                else
                    iz+=count;
                result=BImodpow2(result,pb,iz);
            }
        }while(count !=0);
        return result;
    }

    //  单元乘法
    void BImul(const unsigned_int64& p1,const unsigned_int64& p2,unsigned_int64& high,
    unsigned_int64& low)
    {
        //(aH+aL)*(bH+bL)=aH*bH+aH*bL+aL*bH+aL*bL
        //                          -----    --HIGH--------
        //                                    -----  ----LOW--    -----
        unsigned_int64 aH,aL,bH,bL;
        unsigned_int64 m,m1,m2;

        aH=p1>>32;
        aL=p1 & 0xFFFFFFFF;
        bH=p2>>32;
        bL=p2 & 0xFFFFFFFF;

        high=aH*bH;
        low=aL*bL;

        m1=aH*bL;
        m2=aL*bH;

        high+=(m1>>32);
        high+=(m2>>32);
        m=(m1& 0xFFFFFFFF)+(m2& 0xFFFFFFFF);
```

```
    high += (m>>32);
    m=m<<32;
    low+=m;
    if(low<m)high++;

}

void MakeThemClose(const BigInt& p1,BigInt& p2)
{
    short shift, Sa, Sb;

    shift=TOTALBITS*(p1.iIdx-p2.iIdx);
    if(shift !=0){
        Sa=FindZeroBit(p1.iVal[p1.iIdx]);
        Sb=FindZeroBit(p2.iVal[p2.iIdx]);
        if(Sb>Sa)
            shift=shift+Sb-Sa;
        else
            shift=shift+Sa-Sb;
        shift=shift-1;
        p2<<=shift;
    }
}

short FindZeroBit(const unsigned_int64& p1)
{
    unsigned_int64 cc, t;
    short Fbit=3;

    t=p1;

    cc=FULL>>4;
    while(t<cc)
    {
        cc>>=4;
        Fbit+=4;
    }
    cc=FULL>>Fbit;
    while((cc & t) !=t)
    {
       Fbit--;
       cc=FULL>>Fbit;
    }
    return Fbit;
```

```
}

//  Montgomery 乘法
BigInt MontMult(const BigInt& p1,const BigInt& p2,const BigInt& pp)
{
    BigInt m,base,result;
    BigInt pre_b[16],pre_p[16];
    unsigned short mp, u;
    int bblock;

    base= 0x10;
    base.ildx= 0;

    //Pre
    pre_b[0].setZero();
    pre_b[1]=p2;
    pre_p[0].setZero();
    pre_p[1]=pp;
    for(int i=2;i<16;i++)
    {
        pre_b[i]=pre_b[i-1]+p2;
        pre_p[i]=pre_p[i-1]+pp;
    }

    m=BInverse(pp,base);
    mp=0x10-m.iVal[0];

    bblock=16*pp.ildx+15;
    bblock+=pp.Get4BitBlockZero();

    result.setZero();
    for(int j=0;j<=bblock;j++)
    {
        u=(result.Get4BitBlock(0)+p1.Get4BitBlock(j)*p2.Get4BitBlock(0))*mp;
        u=u & 0xF;
        result=result+pre_b[p1.Get4BitBlock(j)];
        result=result+pre_p[u];
        result>>=4;
    }
    if(BIcompare(result,pp)>=0)
        result=result-pp;
    return result;
```

```
}

//  Montgomery 模幂
BigInt MontExpo(const BigInt& p1,const BigInt& p2,const BigInt& pp)
{
    BigInt zb,zR,zxp,result;
    int bblock,kbit;
    unsigned_int64 m;
    short k,l;

    bblock = pp.Get4BitBlockZero();
    zR = pp;
    zR.setUpperBlock(bblock);

    zb = BImodmul(zR,zR,pp);
    zxp = MontMult(p1,zb,pp);
    result = zR % pp;

    kbit = 63 - FindZeroBit(p2.iVal[p2.iIdx]);
    kbit += 64 * p2.iIdx;

    for(int j = kbit;j >= 0;j--)
    {
        k = j % 64;
        l = kbit/64;
        m = p2.iVal[l] >> k;
        result = MontMult(result,result,pp);
        if((m & 0x01) == 0x01)
            result = MontMult(result,zxp,pp);
    }

    zxp.setZero();
    zxp.iVal[0] = 1;
    result = MontMult(result,zxp,pp);
    return result;
}

//  大整数模幂(预处理)
BigInt BImodexpPre(const BigInt& p1,const BigInt& p2,const BigInt& pb)
{
    //求 p1 的 p2 次方模 pp; p1^p2 mod pb
    //4 bits block.

    BigInt result;
```

```
//  unsigned_int64 tmp;
    int bblock,kbit,k,l,b,j,jj;
    unsigned m,m2;

    if(BIisZero(p1))
        return result;
    if(BIisZero(p2))
    {
       result++;
       return result;
    }

    result++;
    MakePreTable(p1,pb);

    kbit=63 - FindZeroBit(p2.iVal[p2.iIdx]);
    kbit+=64 * p2.iIdx;

    for(j=kbit;j>=0;j--)
    {
        k=j & 0x3F;
        l=j>>6;
        m=p2.iVal[l]>>k;
        if((m & 0x1)==0x1)
        {
            m2=m & 0x1;
            b=1;
            jj=j-1;
            while((b<4)&(jj>=0))
            {
                k=jj & 0x3F;
                l=jj>>6;
                m2<<=1;
                m=p2.iVal[l]>>k;
                m2=m2|(m & 0x1);
                b++;
                jj--;
            }
            j=j-b+1;
            while(b>0)
            {
                result=BImodmul(result,result,pb);
                b--;
            }
```

```
            result=BImodmul(result,PreTable[m2& 0x0f],pb);
        }
        else
            result=BImodmul(result,result,pb);
    }

    return result;

}
```

CRSA.h

```
// 定义一个CRSA类,来简化及隐藏RSA运算过程的函数调用,
// 加密时,只要提供E,N,调用Encrypt(M)即得出C
// 解密时,只要提供D,N,调用Encrypt(C)即得出M

#ifndef CLASS_CRSA
#define CLASS_CRSA
#include "BigInt.h"
class CRSA
{
public:
    CRSA();
    virtual~CRSA();
    BigInt PrimeP;
    BigInt PrimeQ;
    BigInt ValuePQ;
    BigInt Valuefn;
    BigInt PubKey;
    BigInt PriKey;
    String OpStr;

public:
    void init();
    void setValueDEC(const String textN,const String textE);
    void setValueDEC(const String textP,const String textQ,const String textE);
    void setValueDEC(const String textP,const String textQ,const String textE,const String
textD);
    void setValueHEX(const String textN,const String textE);
    void setValueHEX(const String textP,const String textQ,const String textE);
    void setValueHEX(const String textP,const String textQ,const String textE,const String
textD);

    String getPrivateKey(const char iType);
    String getPublicN(const char iType);
```

```
//  加密运算
    void Encrypt(const String pData);
//  解密运算(中国剩余定理)
    void Decrypt(const String pData);
//  加密运算(普通方法)
    String oEncrypt(const String pData);
//  解密运算(普通方法)
    String oDecrypt(const String pData);
//  加密运算(预处理)
    String pEncrypt(const String pData);
//  解密运算(预处理)
    String pDecrypt(const String pData);
//  加密运算(Montgomery)
    String mEncrypt(const String pData);
//  解密运算(Montgomery)
    String mDecrypt(const String pData);
};

#endif
```

CRSA.cpp

```
#include<vcl.h>
#include<stdio.h>
#include<math.h>
#include<dstring.h>
#include"CRSA.h"

CRSA::CRSA()
{
}

CRSA::~CRSA()
{
}

void CRSA::Init(void)
{
    OpStr="";
    PrimeP.setZero();
    PrimeQ.setZero();
    ValuePQ.setZero();
    Valuefn.setZero();
    PubKey.setZero();
```

```
    PriKey.setZero();
}

void CRSA::setValueDEC(const String textN,const String textD)
{
    ValuePQ.loadDEC(textN);
    PriKey.loadDEC(textD);
}

void CRSA::setValueDEC(const String textP,const String textQ,const String textE)
{
    BigInt a,b;

    PrimeP.loadDEC(textP);
    PrimeQ.loadDEC(textQ);
    PubKey.loadDEC(textE);

    ValuePQ=PrimeP*PrimeQ;
    a=PrimeP;
    a--;
    b=PrimeQ;
    b--;
    Valuefn=a*b;
    PriKey=Blinverse(PubKey,Valuefn);
}

void CRSA::setValueDEC(const String textP,const String textQ,const String textE,const String
textD)
{
    BigInt a,b;

    PrimeP.loadDEC(textP);
    PrimeQ.loadDEC(textQ);
    PubKey.loadDEC(textE);
    PriKey.loadDEC(textD);

    ValuePQ=PrimeP*PrimeQ;
    a=PrimeP;
    a--;
    b=PrimeQ;
    b--;
    Valuefn=a*b;

}
```

```
void CRSA::setValueHEX(const String textN,const String textD)
{
    ValuePQ.loadHEX(textN);
    PriKey.loadHEX(textD);
}

void CRSA::setValueHEX(const String textP,const String textQ,const String textE)
{
    BigInt a,b;

    PrimeP.loadHEX(textP);
    PrimeQ.loadHEX(textQ);
    PubKey.loadHEX(textE);

    ValuePQ=PrimeP*PrimeQ;
    a=PrimeP;
    a--;
    b=PrimeQ;
    b--;
    Valuefn=a*b;
    PriKey=Blinverse(PubKey,Valuefn);
}

void CRSA::setValueHEX(const String textP,const String textQ,const String textE,const String
textD)
{
    BigInt a,b;

    PrimeP.loadHEX(textP);
    PrimeQ.loadHEX(textQ);
    PubKey.loadHEX(textE);
    PriKey.loadHEX(textD);

    ValuePQ=PrimeP*PrimeQ;
    a=PrimeP;
    a--;
    b=PrimeQ;
    b--;
    Valuefn=a*b;

}
//输出D
String CRSA::getPrivateKey(const char iType)
```

```
{
    if(iType=='d')
        return PriKey.outputDEC();
    else if(iType=='h')
        return PriKey.outputHEX();
    else
        return PriKey.outputSTR();

{

String CRSA::getPublicN(const char iType)
{
    if(iType=='d')
        return ValuePQ.outputDEC();
    else if(iType=='h')
        return ValuePQ.outputHEX();
    else
        return ValuePQ.outputSTR();

}

void CRSA::Encrypt(const String pData)
{
    BigInt iBase,result;

    iBase.setZero();
    iBase.loadSTR(pData);
    result=BImodexp(iBase,PubKey,ValuePQ);
    OpStr=result.outputHEX();

}

//  加密运算(Montgomery)
String CRSA::mEncrypt(const String pData)
{
    BigInt iBase,result;

    iBase.setZero();
    iBase.loadSTR(pData);
    result=MontExpo(iBase,PubKey,ValuePQ);
    return result.outputHEX();

}
```

```
//  加密运算(预处理)
String CRSA::pEncrypt(const String pData)
{
    BigInt iBase,result;

    iBase.setZero();
    iBase.loadSTR(pData);
    result=BImodexpPre(iBase,PubKey,ValuePQ);
    return result.outputHEX();

}

//  加密运算(普通方法)
String CRSA::oEncrypt(const String pData)
{
    BigInt iBase,result;

    iBase.  setZero();
    iBase.loadSTR(pData);
    result=BImodexp(iBase,PubKey,ValuePQ);
    return result.outputHEX();
}

void CRSA::Decrypt(const String pData)
{
    int i;
    BigInt iBase,result;
    BigInt a,b,m1,m2,y1,y2,dp,dq,s;

    y1=BIinverse(PrimeQ,PrimeP);
    y2=BIinverse(PrimeP,PrimeQ);

    s=PrimeP;
    s--;
    dp=PriKey % s;

    s=PrimeQ;
    s--;
    dq=PriKey % s;

    iBase.setZero();
    iBase.loadHEX(pData);

    result.setZero();
    //Apply chinese remainder theorem
    s=iBase % PrimeP;
```

```
    m1 = BImodexp(s, dp, PrimeP);
    s = iBase % PrimeQ;
    m2 = BImodexp(s, dq, PrimeQ);
    s = (PrimeQ * y1) * m1;
    result = s;
    s = (PrimeP * y2) * m2;
    result = result + s;

    result = result % ValuePQ;
    OpStr = result.outputSTR();

}
//  解密运算(预处理)
String CRSA::pDecrypt(const String pData)
{
    BigInt iBase, result;

    iBase.setZero();
    iBase.setZero();
    iBase.loadHEX(pData);
    result = BImodexpPre(iBase, PriKey, ValuePQ);
    return result.outputSTR();

}

//  解密运算(普通方法)
String CRSA::oDecrypt(const String pData)
{
    BigInt iBase result;
    iBase.setZero();
    iBase.loadHEX(pData);
    result = BImodexp(iBase, PriKey, ValuePQ);
    return result.outputSTR();
}

//  解密运算(Montgomery)
String CRSA::mDecrypt(const String pData)
{
    BigInt iBase, result;
    iBase.setZero();
    iBase.loadHEX(pData);
    result = MontExpo(iBase, PriKey, ValuePQ);
    return result.outputSTR();
}
```

附录 E　大素数生成程序

```
EUCLID11.CPP
FUNCTION:寻找 x 和 y 最大公因数 g 和 ax— by=g, 1<=a<=y and 0<=b<= x.
mpuint 是 一个没有符号的整数类型
#include "euclid11.h"
void Euclid11eanAlgorithm(const mpuint &x, const mpuint &y, mpuint &a,
  mpuint &b, mpuint &g)                    * 定义 euclid11eanalgorithm 函数 * /
{
  unsigned length= x.length;               ' * 将 x.length 存入 unsigned length * /
  if (y.length>length)
    length=y.length;
  if (x<=y)                                ' * 判断 x 是否小于或等于 y * /
  {
    mpuint q(length), r(length);
    mpuint::Divide(y, x, q, r);
    if (r==0)                              / * 判断 r 是否等于 0 * /
    {
      a=1;                                 / * a 置 1 * /
      b=0;                                 / * b 置 0 * /
      g=x;                                 / * g 置 x * /
    }
    else
    {
      mpuint ap(length);
      Euclid11eanAlgorithm(x, r, ap, b, g);
      a=b;
      a * =q;
      a+=ap;                               / * 最后 a=ap+b * q * /

    }
  }
  else
  {
    mpuint ap(length), bp(length);
    Euclid11eanAlgorithm(y, x, bp, ap, g);
    a=y;
    a-=ap;                                 / * 最后 a=y  ap * /
    b=x;
    b-=bp;                                 / * 最后 b=x—bp * /
```

```
  }
}

Greatest CommonDivison: 寻找 x 和 y 最大公因数 g
void GreatestCommonDivisor(const mpuint &x, const mpuint &y, mpuint &g)
{                                          /* 定义 GreatestCommonDivisor 函数 */
  if (x<=y)                                /* 判断 x 是否小于或等于 y */
  {
    mpuint r(y);
    r %= x;                                /* 对 r 和 x 作模运算 */
    if (r==0)                              /* 判断 r 是否等于 0 */
      g=x;
    else
      GreatestCommonDivisor(x, r, g);
  }
  else
    GreatestCommonDivisor(y, x, g);
}
EUCLID11.H
#ifndef H__EUCLID11
#define H__EUCLID11
#include "mpuint.h"
typedef mpuint UITYPE;
void Euclid11eanAlgorithm(const UITYPE &x, const UITYPE &y, UITYPE &a,
  UITYPE &b, UITYPE &g);
void GreatestCommonDivisor(const UITYPE &x, const UITYPE &y, UITYPE &g);
#endif

MPUINT.CPP
#include "mpuint.h"
mpuint::mpuint(unsigned len)
{
  length=len;                              /* 将 len 存入 length */
  value=new unsigned short[len];           /* 将 new unsigned short 变量存入 value */
}
mpuint::mpuint(const mpuint &n)
{
  length=n.length;
  value=new unsigned short[length];
  unsigned i;
  for (i=0; i< length; i++)                /* 由 i=0,作 i 小于 length 次循环 */
    value[i]=n.value[i];                   /* 将 n.value 变量存入 value 变量 */
}
mpuint::~mpuint()
```

```
{
  delete [] value;
}
void mpuint::operator=(const mpuint &n)
{
  unsigned i;
  for (i=0; i< length && i< n.length; i++)
    value[i]=n.value[i];
  for (; i< length; i--)
    value[i]=0;                          /* 置 0 于 value 变量 */
  for (; i< n.length; i++)
  {
    if (n.value[i] !=0)
      numeric_overflow();
  }
}
void mpuint::operator=(unsigned short n)
{
  value[0]=n;                            /* 置 n 于 value[0]变量 */
  unsigned i;
  for (i=1; i< length; i++)              /* 由 i=1,作 i 小于 length 次循环 */
    value[i]=0;                          /* 置 0 于 value 变量 */
}
void mpuint::operator+=(const mpuint &n)
{
  unsigned i;
  unsigned long carry=0;
  for (i=0; i< length; i++)              /* 由 i=0,作 i 小于 length 次循环 */
  {
    unsigned long sum=value[i]+(i< n.length ? n.value[i] : 0)+carry;
    value[i]=sum;                        /* 将 sum 值置入 value 变量中 */
    carry=sum >> 16;
  }
  if (carry !=0)                         /* carry 作非运算 */
    numeric_overflow();
  for (; i< n.length; i++)
  {
    if (n.value[i] !=0)
      numeric_overflow();
  }
}
void mpuint::operator+=(unsigned short n)
{
  value[0]+=n;
```

```
  if (value[0]< n)                    /* 判断 value[0]变量是否小于 n 值 */
  {
    unsigned i;
    for (i=1; i< length; i++)         /* 由 i=1,作 i<length 次循环 */
    {
      if (++value[i] !=0)
        return;
    }
    numeric_overflow();
  }
}

void mpuint::operator-=(const mpuint &n)
{
  unsigned i;
  unsigned long borrow=0;
  for (i=0; i< length; i++)
  {
    unsigned long subtrahend=(i< n.length ? n.value[i] : 0)+borrow;
    borrow=(unsigned long)(value[i])< subtrahend;
    value[i]-=subtrahend;
  }
  if (borrow !=0)
    numeric_overflow();
  for (; i< n.length; i++)
  {
    if (n.value[i] !=0)
      numeric_overflow();
  }
}
void mpuint::operator-=(unsigned short n)
{
  if (value[0] >=n)                   /* 判断 value[0]变量是否大于 n */
    value[0]-=n;                      /* value[0]变量减 n 存入 value[0]变量 */
  else
  {
    value[0]-=n;
    for (unsigned i=1; i< length; i++)
    {
      if (--value[0] !=0xFFFF)
        return;
    }
    numeric_overflow();
  }
```

```
}
void mpuint::operator *=(const mpuint &n)
{
  unsigned i;
  unsigned short *multiplier=new unsigned short[length];
  for (i=0; i< length; i++)
  {
    multiplier[i]=value[i];
    value[i]=0;
  }
  for (i=0; i< length; i++)
  {
    unsigned j;
    for (j=0; j< n.length; j--)        /*由 j=0,作 j<n.length 次循环*/
    {
      unsigned long product=multiplier[i] * n.value[j];   /*定义函数*/
      unsigned k=i+j;
      while (product !=0)
      {
        if (k >=length)                /*判断 k 是否大于或等于 length*/
          numeric_overflow();
        product+=value[k];
        value[k]=product;
        product >>=16;
        k++;                           /*k 累加*/
      }
    }
  }
}
void mpuint::operator *=(unsigned short n)
{
  unsigned i;
  unsigned long product=0;
  for (i=0; i< length; i++)            /*由 i=0,作 i<length 次循环*/
  {
    product+=n * value[i];
    value[i]=product;
    product >>=16;
  }
  if (product !=0)
    numeric_overflow();
}
unsigned short mpuint::remainder(unsigned short n)   /*定义函数*/
{
```

```
  unsigned i=length;
  unsigned rem=0;
  while (i--!=0)
  {
    unsigned long dividend=(unsigned long) rem<< 16 .  /*定义函数*/
      (unsigned long) value[i];
    value[i]=dividend / n;
    rem=dividend % n;
  }
  return rem;
}
void mpuint::operator /=(unsigned short n)
{
  (void) remainder(n);
}
void mpuint::operator %=(unsigned short n)
{
  * this=remainder(n);
}
int mpuint::Compare(const mpuint &n) const     /*定义函数*/
{
  unsigned i;
  if (length>n.length)
  {
    for (i=length-1; i >=n.length; i--)     /*由 i=length 减 1,作 i=n.length 次循环*/
    {
      if (value[i] !=0)
        return 1;
    }
  }
  else if (n.length>length)
  {
    for (i=n.length-1; i >=length; i--)     /*由 i=n.length 减 1,作 i=length 次循环*/
    {
      if (n.value[i] !=0)
        return-1;
    }
  }
  else
    i=length-1;
  while (true)
  {
    if (value[i]>n.value[i])                /*判断 value 变量是否大于 n.value 变量/
      return 1;
```

```
        if (value[i]< n.value[i])                    /*判断 value 变量是否小于 n.value 变量/
          return-1;
        if (i==0)                                    /*判断 i 是否等于 0*/
          return 0;
        i--;
      }
    }
    int mpuint::Compare(unsigned short n) const      /*定义函数*/
    {
      unsigned i;
      for (i=length-1; i>=1; i--)
      {
        if (value[i] !=0)
          return 1;
      }
      return value[0]>n? 1 : value[0]< n? -1:0;
    }
    bool mpuint::IsZero(void) const
    {
      unsigned i;
      for (i=0; i< length; i++)
      {
        if (value[i] !=0)
          return false;
      }
      return true;
    }
    char * mpuint::edit(char * s) const
    {
      mpuint n(*this);
      unsigned i=0;
      do
        s[i++]=n.remainder(10)+'0';
      while (!n.IsZero());
      s[i]=0;
      unsigned j;
      for (j=0;--i>j; j++)
      {
        char c=s[i];
        s[i]=s[j];
        s[j]=c;
      }
      return s;
    }
```

```
bool mpuint::scan(const char * &s)
{
  const char * t=s;
  bool found=false;
  while ( * t==' ' || * t=='\t')
    t++;
  * this=0;
  while ('0'<= * t && * t<='9')
  {
    found=true;
    * this * =10;
    * this+=(unsigned short) ( * t++-'0');
  }
  s=t;
  return found;
}
void mpuint::shift(unsigned bit)
{
  for (unsigned i=0; i< length; i++)
  {
    unsigned long x=value[i]<< 1 | bit;
    value[i]=x;
    bit=x >> 16;                                    /* x右移16位*/
  }
  if (bit !=0)
    numeric_overflow();
}
void mpuint::Divide(const mpuint &dividend, const mpuint &divisor, mpuint &quotient,
  mpuint &remainder)                                /*定义变量*/
{
  if (divisor.IsZero())
    numeric_overflow();
  remainder=0;                                      /*remainder置0*/
  quotient=0;                                       /*quotient置0*/
  unsigned i=dividend.length;
  while (i--!=0)
  {
    unsigned bit=16;
    while (bit--!=0)
    {
      remainder.shift(dividend.value[i] >> bit & 1);
      if (divisor<=remainder)                       /*判断divisor是否等于remainder*/
      {
        quotient.shift(1);
```

```
            remainder -= divisor;
          }
          else
            quotient.shift(false);
        }
      }
    }
    void mpuint::operator /=(const mpuint& n)
    {
      mpuint remainder(n.length);
      mpuint dividend(*this);
      Divide(dividend, n, *this, remainder);
    }
    void mpuint::operator %=(const mpuint& n)
    {
      mpuint quotient(length);
      mpuint dividend(*this);
      Divide(dividend, n, quotient, *this);
    }
    void mpuint::Power(const mpuint &base, const mpuint &exponent,
      const mpuint &modulus, mpuint &result)
    {
      mpuint r(2*base.length+1);                    /*调用函数*/
      r=1;                                          /*r置1*/
      bool one=true;
      unsigned i=exponent.length;
      while (i--!=0)
      {
        unsigned bit=1<<15;
        do
        {
          if (!one)
          {
            mpuint n(r);
            r*=n;
            r%=modulus;
          }
          if (exponent.value[i] & bit)
          {
            r*=base;
            r%=modulus;
            one=false;
          }
          bit>>=1;                                  /*右移1位*/
```

```
    } while (bit !=0);
  }
  result=r;
}
void mpuint::dump() const
{
  unsigned i;
  for (i=0; i< length; i++)
    printf(" %x", value[i]);
  putchar('\n');
}
```

MPUINT. H

```
#ifndef H_MPUINT
#define H_MPUINT
extern void numeric_overflow(void);
class mpuint
{
  private:
    unsigned short remainder(unsigned short);
    void shift(unsigned);
  public:
    unsigned short * value;
    bool IsZero(void) const;
    int Compare(const mpuint &) const;
    int Compare(unsigned short) const;
    unsigned length;
    mpuint(unsigned);
    mpuint(const mpuint &);
    ~mpuint();
    void operator=(const mpuint &);
    void operator=(unsigned short);
    void operator+=(const mpuint &);
    void operator+=(unsigned short);
    void operator-=(const mpuint &);
    void operator-=(unsigned short);
    void operator*=(const mpuint &);
    void operator*=(unsigned short);
    void operator /=(const mpuint &);
    void operator /=(unsigned short);
    void operator %=(const mpuint &);
    void operator %=(unsigned short);
    static void Divide(const mpuint &, const mpuint &, mpuint &, mpuint &);
    char * edit(char *) const;
    bool scan(const char * &);
```

```
    void dump() const;
    bool mpuint::operator==(const mpuint &n) const {return Compare(n)==0;}
    bool mpuint::operator !=(const mpuint &n) const {return Compare(n) !=0;}
    bool mpuint::operator >(const mpuint &n) const {return Compare(n) >0;}
    bool mpuint::operator >=(const mpuint &n) const {return Compare(n) >=0;}
    bool mpuint::operator<(const mpuint &n) const {return Compare(n)<0;}
    bool mpuint::operator<=(const mpuint &n) const {return Compare(n)<=0;}
    bool mpuint::operator==(unsigned short n) const {return Compare(n)==0;}
    bool mpuint::operator !=(unsigned short n) const {return Compare(n) !=0;}
    bool mpuint::operator >(unsigned short n) const {return Compare(n) >0;}
    bool mpuint::operator >=(unsigned short n) const {return Compare(n) >=0;}
    bool mpuint::operator<(unsigned short n) const {return Compare(n)<0;}
    bool mpuint::operator<=(unsigned short n) const {return Compare(n)<=0;}
    static void Power(const mpuint &, const mpuint &, const mpuint &,
      mpuint &);
};

#endif
```

RANDOM.CPP

运用 IDEA 密码变换产生随机数

```
#include
#include "mpuint.h"
#include "random.h"
#include "idea.h"                                    /*调用函数*/
static int RandomKey(void)                           /*定义函数*/
{
  void Random(mpuint &x)
{
  cprintf("Please type %d random time\r\n", x.length*2);   /*输入时间*/
  cprintf("Please type %d random date\r\n", x.length*2);   /*输入日期*/
  while (kbhit() !=0)
    RandomKey();
    Idea( d x.length^2)                              /*调用函数*/
    Idea( n y.length~2)
  for (unsigned i=0; i< x.length; i++)
    x.value[i]=RandomKey()<< 8 | RandomKey();
    y.value[I]=RandomKey()<< 8 | RandomKey()
  cprintf("\r\nThank you\r\n");
}
```

FILE NAME:RANDOM.H

```
#ifndef H_RANDOM
```

```
#define H_RANDOM
#include "mpuint.h"
#include "idea.h"

void Random(mpuint &x);
#endif
```

FILE NAME:PRM.CPP

```
#include
#include "PRM.h"
#include "euclid11.h"
#include "random.h"
// 用 100 次几率式的测试方法判断一个大数是否素数
static bool IsPrime(const mpuint &p)
{
  mpuint pminus1(p);
  pminus1--=1;
  unsigned count=101;
  while (--count !=0)
  {
    mpuint r(p.length);                          /*调用函数*/
    mpuint x(p.length);
    {
      for (unsigned i=0; i< x.length; i++)       /*由 i=0,作 i<x.length 次循环)
        x.value[i]=rand()<<8 | rand();
    }
    x %=p;
    if (x !=0)
    {
      mpuint::Power(x, pminus1, p, r);
      if (r !=1)
        return false;
    }
  }
  return true;
}

// 产生强素数
void strongprime(slp,spn)                        /*定义函数*/
unsigned int slp;
unsigned long * spn;
{
  LINT pt,ps,pr,prs,n_tmp;
  unsigned long * psp,preg;
```

```
int spcnt,spget,tplen,tmp1,tmp2;                    /*定义整数*/
for(spcnt=2;spcnt<=LENGTH;spcnt++) n_tmp[spcnt]=0x0;
n_tmp[0]=1;n_tmp[1]=1;                              /*n_tmp[0]变量和 n_tmp[1]变量置 1*/
tplen=(int) (log((double) slp)/log(2.0));
spcnt=(slp-tplen)/2;
level_prime(spcnt,ps);
fprintf(fp," %d   p1\n",count);
for(tmp1=0;tmp1<=LENGTH;tmp1++) fprintf(fp,"%1x",ps[tmp1]);
fprintf(fp,"\n");
spcnt=slp-tplen-spcnt;
level_prime(spcnt,pr);
fprintf(fp,"%d      p2\n",count);
for(tmp1=0;tmp1<=LENGTH;tmp1++) fprintf(fp,"%1x",pr[tmp1]);
fprintf(fp,"\n");
inverse(ps,pr,pt);
shiftleft(pt);
multiply(ps,pt,pt);
sub(pt,n_tmp);
multiply(pr,ps,prs);
shiftleft(prs);
for(spcnt=1;spcnt<=LENGTH;spcnt++) pr[spcnt]=0;
tmp1=slp%32;
if(tmp1==0) {pr[0]=slp/32;tmp2=pr[0];pr[tmp2]=LMASK;}
  else { pr[0]=slp/32+1;tmp2=pr[0];
      pr[tmp2]=RMASK;pr[tmp2]<<=(tmp1-1);}
      division(pr,prs,ps,n_tmp);
      multiply(ps,prs,pr);
      if(compare(pt,n_tmp)<0) {add(pt,pr);add(pt,prs);}
      else add(pt,pr);
      spcnt=0;spget=0;
      while(spget==0)
      {
        spget=primetest(pt);
        spcnt++;
        if(spget==0) add(pt,prs);
        }
        count=spcnt;
        psp=spn;
        for(spcnt=0;spcnt<=LENGTH;spcnt++,psp++) *psp=pt[spcnt];
        }

GeneratePrime;                                      /*功能是产生一个大素数*/
static void GeneratePrime(mpuint &p)                /*定义函数*/
{
```

```
  Random(p);                                        /*调用函数*/
  p.value[p.length-1] |=0x8000;
  p.value[0] |=1;
  while (! IsPrime(p))
    p+=2;
}
void GenerateKeys(mpuint &d, mpuint &e, mpuint &n)
{
  mpuint p(d.length/2);
  GeneratePrime(p);
  mpuint q(d.length/2);
  GeneratePrime(q);
  mpuint pp(p);
  pp-=1;                                            /*运算*/
  mpuint qq(q);
  qq-=1;
  mpuint pq(d.length);
  pq=pp;
  pq*=qq;
  n=p;
  n*=q;
  Random(d);
  d %=pq;
  mpuint temp(d.length);
  mpuint g(d.length);
  while (true)
  {
    EuclidlleanAlgorithm(d, pq, e, temp, g);
    if (g==1)
      break;
    d+=1;
  }
}
PRM.H
#include "mpuint.h"
void GenerateKeys(mpuint &d, mpuint &e, mpuint &n);
strongprime(mpunit..mpuint)
#endif

idea.cpp
#include "idea.h"
static uint16 inv(uint16 x)
{
   uint16 t0,t1;
```

```
    uint16 q,y;
    if (x<=1)
        return x;
    t1=(uint16)(0x10001l/x);
    y=(uint16)(0x10001l%x);
    if (y==1)
     return low16(1-t1);
    t0=1;
    do
    {
        q=x/y;
        x=x%y;
        t0+=q*t1;
        if (x==1)
     return t0;
        q=y/x;
        y=y%x;
        t1+=q*t0;
    } while (y!=1);
    return low16(1-t1);
}
static void en_key_idea(word16 *userkey, word16 *Z)
{
    int i,j;
                                                    /* shifts */
    for (j=0;j<8;j++)
        Z[j]=*userkey++;
    for (i=0;j<KEYLEN;j++)
    {
        i++;
        Z[i+7]=((Z[i&7]<<9) | (Z[i+1 &7] >> 7));
        Z+=i&8;
        i&=7;
    }
}
static void de_key_idea(IDEAkey Z,IDEAkey DK)
{
    int j;
    uint16 t1,t2,t3;
    IDEAkey T;
    word16 *p=T+KEYLEN;
    t1=inv(*Z++);
    t2=-*Z++;
    t3=-*Z++;
```

```
        *--p=inv(*Z++);
        *--p=t3;
        *--p=t2;
        *--p=t1;
    for (j=1;j<ROUNDS;j++)
    {
        t1=*Z++;
        *--p=*Z++;
        *--p=t1;
        t1=inv(*Z++);
        t2=-*Z++;
        t3=-*Z++;
        *--p=inv(*Z++);
        *--p=t2;
        *--p=t3;
        *--p=t1;
    }
    t1=*Z++;
    *--p=*Z++;
    *--p=t1;
    t1=inv(*Z++);
    t2=-*Z++;
    t3=-*Z++;
    *--p=inv(*Z++);
    *--p=t3;
    *--p=t2;
    *--p=t1;
    for(j=0,p=T;j<KEYLEN;j++)
    {
        *DK++=*p;
        *p++=0;
    }
}
uint16 mul(uint16 a, uint16 b)
{
    word32 p;
    if (a)
    {
        if (b)
        {
      p=(word32)a*b;
      b=(uint16)(low16(p));
      a=(uint16)(p>>16);
      return b-a+(b<a);
```

```
        }
        else
        {
       return 1-a;
        }
    }
    else
        return 1-b;
}
#define MUL(x,y) (x=mul(low16(x),y))
#define CONST
static void cipher_idea(word16 in[4],word16 out[4],register CONST IDEAkey Z)
{
    register uint16 x1,x2,x3,x4,t1,t2;
    int r=ROUNDS;                                    /*运算*/
    x1=*in++; x2=*in++;
    x3=*in++; x4=*in;
    do
    {
        MUL(x1,*Z++);
        x2+=*Z++;
        x3+=*Z++;
        MUL(x4,*Z++);
        t2=x1^x3;
        MUL(t2,*Z++);
        t1=t2+(x2^x4);
        MUL(t1,*Z++);
        t2=t1+t2;
        x1^=t1;
        x4^=t2;
        t2^=x2;
        x2=x3^t1;
        x3=t2;
    } while (--r);
    MUL(x1,*Z++);
    *out++=x1;
    *out++=(x3+*Z++);
    *out++=(x2+*Z++);
    MUL(x4,*Z);
    *out=x4;
}
char read_char_from_file(FILE *fp)
{
    char temp=0;
```

```
    if ((fread(&temp,sizeof(char),1,fp))!=1)
        end_of_file=1;
    return (temp);
}

word16 read_word16_from_file(FILE * fp)
{
    word16 temp=0;

    if ((fread(&temp,sizeof(word16),1,fp))!=1)
        end_of_file=1;
    return temp;
}
void write_char_to_file(char data,FILE * fp)
{
    if ((fwrite(&data,sizeof(char),1,fp))!=1)
    {
        printf("Fatal Error writing output file!!! \n");
        exit(-1);
    }
}
void write_word16_to_file(word16 data,FILE * fp)
{
    if ((fwrite(&data,sizeof(word16),1,fp))!=1)
    {
        printf("Fatal Error writing output file!!! \n");
        exit(-1);
    }
}
void cipher_file(FILE * in,FILE * out,word16 * key)
{
    word16 input[4],output[4];
    IDEAkey Z;
    int x,y;
    int count=0;
    long length;
    int temp;
    en_key_idea(key,Z);
    end_of_file=0;
    length=filelength(fileno(in));
    fwrite(&length,sizeof(long),1,out);
    while (! end_of_file)
    {
        x=0;
```

```
        while (x<4)
        {
      input[x]=((word16)(read_char_from_file(in)<<8));
      if (! end_of_file)
      {
          temp=read_char_from_file(in);
          if (temp<0) temp+=256;
          input[x]=input[x]|temp;
          x++;
      }
      if (end_of_file)
      {
          while (x<4) input[x++]=0;
          break;
      }
        }
        cipher_idea(input,output,Z);
        for (y=0;y<x;y++)
        {
      if (noisy) if (count++%256==0) printf(".");
      write_word16_to_file(output[y],out);
        }
    }
}
void decipher_file(FILE * in,FILE * out,word16 * key)
{
    word16 input[4],output[4];
    int x,y;
    IDEAkey Z,DK;
    int count=0;
    long length=0;
    en_key_idea(key,Z);
    de_key_idea(Z,DK);
    end_of_file=0;
    fread(&length,sizeof(long),1,in);
    while (! end_of_file)
    {
        x=0;
        while (x<4)
        {
      input[x]=read_word16_from_file(in);
      if (end_of_file)
          break;
      x++;
```

```
        }
        cipher_idea(input,output,DK);
        for (y=0;y<x;y++)
        {
      if (noisy) if (count++%256==0) printf(".");
      if (length--->0)
          write_char_to_file(((char)(output[y]>>8)),out);
      if (length--->0)
          write_char_to_file(((char)(output[y]&255)),out);
        }
    }
}
void swap_files_and_clean_up(char * file)
{
    long fsize,count;
    FILE * fp;
    char temp[100];
    if (overwrite)
    {
        if ((fp=fopen(file,"r+b"))==NULL)
        {
      printf("\nError overwriting old file, security compromised.\n");
        }
        else
        {
      fseek(fp,01,SEEK_END);
      fsize=ftell(fp);
      fseek(fp,01,SEEK_SET);
      for (count=0;count<fsize;count++)
          fputc('0',fp);
      fclose(fp);
        }
        if ((remove(file))!=0)
        {
      printf("\nERROR removing old file<%s>\n",file);
      printf("encoded data remains in temporary file<%s>\n",tempfilename);
      exit(-1);
        }
    }
    else
    {
        strcpy(temp,file);
        file=strtok(temp,".");
        strcat(file,".enc");
```

```
    }
    if ((rename(tempfilename,file))!=0)
    {
        printf("\nERROR renaming temporary file<%s>!! \n",tempfilename);
        printf("Data is safely processed and stored in that file. \n");
        exit(-1);
    }
}
#define KBYTES 1024

void getuserkeyfromargv(word16 * key,char * arg)
{
    int x;
    for (x=0;x<strlen(arg) && x<8;x++)
    {
        if (x==0) key[x]=arg[x]<<8;
        else key[x]=((arg[x]<<8)|(key[x-1]>>8));
    }
    if (strlen(arg)>8) printf ("\nONLY first * 8 * characters of key used!!! \n");
    if (x<8) while (x<8) key[x++]=0;
}
main(int argc, char * * argv)
{
    word16 userkey[8];
    char filename[100];
    FILE * fp, * temp;
    int to_or_from;
    noisy=1;
    overwrite=0;
    if (argc!=4)
    {
        printf("Usage: idea filename. ext [e|d[w]] key\n");
        printf("e=encode    d=decode    file\n");
        printf(" 64 bit \n");
        exit(-1);
    }
    else
    {
        strncpy(filename,argv[1],99);
        filename[99]='\0';
        if (argv[2][0]=='e') to_or_from=1;
        else if (argv[2][0]=='d') to_or_from=0;
        else
        {
```

```
        printf("Usage: idea filename.ext [e|d[w]] key\n");
        printf(" e=encrypt d=decrypt w=overwrite file\n");
        printf(" 64 bit !!! \n");
        exit(-1);
         }
        if (argv[2][1]=='w') overwrite=1;
        getuserkeyfromargv(userkey,argv[3]);
    }
    if ((fp=fopen(filename,"r+b"))==NULL)
    {
        printf("\nError opening File %s\n",filename);
        exit (-1);
    }
    if ((temp=fopen(tempfilename,"w+b"))==NULL)
    {
        printf("\nError opening temporary file\n");
        exit(-1);
    }
    if (to_or_from==1)
    {
        printf("\nEncoding file %s    ",filename);
        cipher_file(fp,temp,userkey);
    }
    else
    {
        printf("\nDecoding file %s    ",filename);
        decipher_file(fp,temp,userkey);
    }
    fclose (fp);
    fclose(temp);
    swap_files_and_clean_up(filename);
    return 0;
```

idea.h

```
#ifndef _IDEA_DOT_H
#define _IDEA_DOT_H
#include<stdio.h>
#include<time.h>
#include<process.h>
#include<io.h>
#include<string.h>
#include<conio.h>
#define IDEAKEYSIZE 16
#define IDEABLOCKSIZE 8
```

```
#define word16 unsigned int
#define word32 unsigned long int
#define ROUNDS      8
#define KEYLEN      (6 * ROUNDS+4)
#define tempfilename "tempfile.? nc"
int end_of_file,noisy,overwrite;                    /* 全面变量 */
#define low16(x) ((x) & 0xffff)
typedef unsigned int uint16;
typedef word16 IDEAkey[KEYLEN];
void en_key_idea(word16 userkey[8],IDEAkey Z);
void de_key_idea(IDEAkey Z、IDEAkey DK);
void cipher_idea(word16 in[4],word16 out[4],IDEAkey Z);
uint16 inv(uint16 x);
uint16 mul(uint16 a,uint16 b);
char read_char_from_file(FILE * fp);
word16 read_word16_from_file(FILE * fp);
void write_char_to_file(char data,FILE * fp);
void write_word16_to_file(word16 data,FILE * fp);
void cipher_file(FILE * in,FILE * out,word16 * key);
void decipher_file(FILE * in,FILE * out,word16 * key);
void swap_files_and_clean_up(char * file);
#endif
```

参考文献

1. Cryptography and Network Security: Principles and Practice, Second Edition. William Stallings Prentice Hall, InC. 1999
2. Bruce Schneier. Applied Cryptography: Protocols, Algorithms, and Source Code in C, Second Edition. John Wiley & Sons, Inc. 1996
3. Arto Salomaa. Public-Key Cryptography. Springer-Verlag, 1990
4. Man Young Rhee. Cryptography and Secure Communications. McGraw-Hill Book Co. 1994
5. Wanyne Patterson. Mathematical Cryptology for Computer Scientists and Mathematicians. Rowman & Dittlefield, 1987