序

W.迪菲(Whitfield Diffie)

密码学文献有一个奇妙的发展历程，当然，密而不宣总是扮演主要角色。第一次世界大战前，重要的密码学进展很少出现在公开文献中，但该领域却和其它专业学科一样向前发展。直到 1918 年，二十世纪最有影响的密码分析文章之一——William F. Friedman 的专题论文《重合指数及其在密码学中的应用》作为私立的“河岸（Riverbank）实验室”的一份研究报告问世了[577]，其实，这篇著作涉及的工作是在战时完成的。同年，加州奥克兰的 Edward H.Hebern 申请了第一个转轮机专利[710]，这种装置在差不多 50 年里被指定为美军的主要密码设备。

然而，第一次世界大战后，情况开始变化，完全处于秘密工作状态的美国陆军和海军的机要部门开始在密码学方面取得根本性的进展。在 30 年代和 40 年代，有几篇基础性的文章出现在公开的文献中，有关该领域的几篇论文也发表了，只不过这些论文的内容离当时真正的技术水平相去甚远，战争结束时，情况急转直下，公开的文献几乎殆尽。只有一个突出的例外，那就是仙农(Claude Shannon)的文章《保密系统的通信理论》[1432]出现在 1949 年《贝尔系统技术杂志》上，它类似于 Friedman1918 年的文章，也是战时工作的产物。这篇文章在第二次世界大战结束后即被解密，可能是由于失误。

从 1949 年到 1967 年，密码学文献近乎空白。在 1967 年，一部与众不同的著作——David Kahn 的《破译者》[794]——出现了，它没有任何新的技术思想，但却对以往的密码学历史作了相当完整的记述，包括提及政府仍然认为是秘密的一些事情。《破译者》的意义不仅在于它涉及到的相当广泛的领域，而且在于它使成千上万原本不知道密码学的人了解密码学。新的密码学文章慢慢地开始源源不断地被编写出来了。

大约在同一时期，早期为空军研制敌我识别装置的 Horst Feistel 在位于纽约约克镇高地的 IBM Watson 实验室里花费了毕生精力致力于密码学的研究。在那里他开始着手美国数据加密标准（DES）的研究，到 70 年代初期，IBM 发表了 Feistel 和他的同事在这个课题方面的几篇技术报告[1482，1484，552]。

这就是我于 1972 年底涉足密码学领域时的情形，当时密码学的文献还不丰富，但却也包括一些非常有价值的东西。

密码学提出了一个一般的学科领域都难以遇到的难题：即它需要密码学和密码分析学紧密结合互为促进。这是由于缺乏实际通信检验的实情所致。提出一个表面上看似不可破的系统并不难。许多学究式的设计就非常复杂，以至于密码分析家不知从何入手，分析这些设计中的漏洞远比原先设计它们更难。结果是，那些能强劲推动学术研究的竞争过程在密码学中并没起多大作用。

当 Martin Hellman 和我在 1975 年提出公开密钥密码学[496]时，我们的一种间接贡献是引入了一个看来不易解决的难题。现在一个有抱负的密码体制设计者能够提出被认为是很聪明的一些东西——这些东西比只是把有意义的正文变成无意义的乱语更有用。结果研究密码学的人数、召开的会议、发表的论著和文章都惊人地增加了。

我在接受 Donald E.Fink 奖（该奖是奖给在 IEEE 杂志上发表过最好文章的人，我和 Hellman 在 1980 年共同获得该奖）发表演讲时，告诉听众我在写作“保密性与鉴别”一文时，有一种经历——我相信这种经历，即使在那些参加 IEEE 授奖会的著名学者们当中也是罕见的：我写的那篇文章，并非我的研究结果而是我想要研究的课题，因为在我首次沉迷于密码学的时候，这类文章根本就找不到。如果那时我可以走进斯坦福书店，挑选现代密码学的书籍，我也许能在多年前就了解这个领域了。但是在 1972 年秋季，我能找到的资料仅仅是几篇经典论文和一些难理解的技术报告而已。

当代的研究人员再也没有这样的问题了。现在的问题是要在大量的文章和书籍中选择从何处入手。研究人员如此，那些仅仅想利用密码学的程序员和工程师又会怎样呢？这些人会转向哪里呢？直到今天，在能够设计出通俗文章中所描述的那类密码实用程序之前，花费大量时间去寻找，并研究那些文献仍是很有必要的。

Bruce Schneier 的《应用密码学》正好填补了这个空白的。Schneier 从通信保密性的目的和达到目的所用的基本程序实例入手，对 20 年来公开研究的全部成果作了全景式的概括。书名开门见山：从首次叫某人进行保密会话的世俗目的，到数字货币和以密码方式进行保密选举的可能性，到处你都可以发现应用密码学的用处。

Schneier 不满足于这本书仅仅涉及真实世界（因为此书叙述了直至代码的全部过程），他还叙述了发展密码学和应用密码学的那些领域，讨论了从国际密码研究协会直到国家安全局这样的一些机构。

在 70 年代后期和 80 年代初，当公众在密码学方面的兴趣显示出来时，国家安全局（NSA）即美国官方密码机构曾多次试图平息它。第一次是一名长期在 NSA 工作的雇员的一封信，据说这封信是这个雇员自己写的，此雇员自认是如此，表面上看来亦是如此。这封信是发给 IEEE 的，它警告密码资料的出版违反了国际武器交易条例（ITAR）。然而这种观点并没有被条例本身所支持，条例明显不包括已发表的资料。但这封信却为密码学的公开实践和 1977 年的信息论专题研讨会做了许多意想不到的宣传。

一个更为严重的事态发生在 1980 年，当时 NSA 发现，美国教育委员会在出版物审查方面说服国会对密码学领域的出版物进行合法地控制，结果与 NSA 的愿望大相经庭，形成了密码学论文自愿送审的程序；要求研究人员在论文发表之前需就发表出去是否有害国家利益征询 NSA 的意见。

随着 80 年代的到来，NSA 将重点更多的集中在实际应用上，而不是密码学的研究中。现有的法律授权 NSA 通过国务院控制密码设备的出口。随着商务活动的日益国际化和世界市场上美国份额的减退，国内外市场上需要单一产品的压力增加了。这种单一产品受到出口控制，于是 NSA 不仅对出口什么，而且也对在美国出售什么都施加了相当大的影响。

密码学的公开使用面临一种新的挑战，政府建议在可防止涂改的芯片上用一种秘密算法代替广为人知且随处可得的数据加密标准（DES），这些芯片将含有政府监控所需的编纂机制。这种“密钥托管”计划的弊病是它潜在地损害了个人隐私权，并且以前的软件加密不得不以高价增用硬件来实现，迄今，密钥托管产品正值熊市，但这种方案却已经引起了广泛的批评，特别是那些独立的密码学家怨声载道。然而，人们看到的更多是编程技术的未来而不是政治，并且还加倍地努力向世界提供更强的密码，这种密码能够实现对公众的监视。

从出口控制法律涉及第一修正案的意见来看， 1980 年发生了大倒退，当时“联邦注册”公布了对 ITAR 的修正，其中提到：“……增加的条款清楚地说明，技术数据出口的规定并不干预第一修正案中个人的权利”，但事实上第一修正案和出口控制法的紧张关系还未消除，

最近由 RSA 数据安全公司召开的一次会议清楚地表明了这一点，从出口控制办公室来的 NSA 的代表表达了意见：发表密码程序的人从法律上说是处在“灰色领域”。如果真是这样的话，本书第一版业已曝光，内容也处在“灰色领域”中了。本书自身的出口申请已经得到军需品控制委员会当局在出版物条款下的认可，但是，装在磁盘上的程序的出口申请却遭到拒绝。

NSA 的策略从试图控制密码研究到紧紧抓住密码产品的开发和应用的改变，可能是由于认识到即便世界上所有最好的密码学论文都不能保护哪怕是一比特的信息。如果置之高阁，本书也许不比以前的书和文章更好，但若置于程序员编写密码的工作站旁时，这本书无疑是最好的。

Whitfield Diffie 于
加州 Mountain View

前　言

世界上有两种密码：一种是防止你的小妹妹看你的文件；另一种是防止当局者阅读你的文件资料。这本书写的是后一种情况。

如果把一封信锁在保险柜中，把保险柜藏在纽约的某个地方…，然后告诉你去看这封信。这并不是安全，而是隐藏。相反，如果把一封信锁在保险柜中，然后把保险柜及其设计规范和许多同样的保险柜给你，以便你和世界上最好的开保险柜的专家能够研究锁的装置。而你还是无法打开保险柜去读这封信，这样才是安全的。

许多年来，这种密码学是军队独家专有的领域。美国国家安全局以及前苏联、英国、法国、以色列及其它国家的安全机构已将大量的财力投入到加密自己的通信，同时又千方百计地去破译别人的通信的残酷游戏之中，面对这些政府，个人既无专门知识又无足够财力保护自己的秘密。

在过去 20 年里，公开的密码学研究爆炸性地增长。从二次世界大战以来，当普通公民还在长期使用经典密码时，计算机密码学成为世界军事的独占领域。今天，最新的计算机密码学已应用到军事当局的高墙之外，现在非专业人员都可以利用密码技术去阻止最强大的敌人，包括军方的安全机构。

平头百姓真的需要这种保密性吗？是的，他们可能正策划一次政治运动，讨论税收或正干一件非法的事情；他们也可能正设计一件新产品，讨论一种市场策略，或计划接管竞争对手的生意，或者，他们可能生活在一个不尊重个人隐私权的国家，也可能做一些他们自己认为并非违法实际却是非法的事情。不管理由是什么，他的数据和通信都是私人的、秘密的，与他人无关。

这本书正好在混乱的年代发表。1994 年，克林顿当局核准了托管加密标准（包括 Clipper 芯片和 Fortezza 卡），并将数字电话法案签署成为法律。这两个行政令企图确保政府实施电子监控的能力。

一些危险的 Orwellian 假设在作祟：即政府有权侦听私人通信，个人对政府保守秘密是错误的，如果可能，法律总有能力强制实施法院授权的监控，但是，这是公民第一次被强迫采取积极措施，以使他们自己能被监控。这两个行政令并不是政府在某个模糊范围内的简单倡议，而是一种先发制人的单方面尝试，旨在侵占以前属于人民的权力。

Clipper 和数字电话不保护隐私，它强迫个人无条件地相信政府将尊重他们的隐私。非法窃听小马丁・路德・金电话的执法机构，同样也能容易地窃听用 Clipper 保护的电话。最近，地方警察机关在好些管区内都有因非法窃听而被控有罪或被提出民事诉讼的，这些地方包括马里兰、康涅狄格、佛蒙特、佐治亚、密苏里和内华达。为了随时方便警察局的工作而配置这种技术是很糟糕的想法。

这儿给我们的教训是采用法律手段并不能充分保护我们自己，我们需要用数学来保护自己。加密太重要了，不能让给政府独享。

本书为你提供了一些可用来保护自己隐私的工具。提供密码产品可能宣布为非法，但提供有关的信息绝不会犯法。

怎样读这本书？

我写《应用密码学》一书是为了在真实介绍密码学的同时给出全面的参考文献。我尽量在不损失正确性的情况下保持文本的可读性。这本书不想成为一本数学书。虽然我无意给出任何错误信息，但匆忙中理论难免有失严谨。对形式方法感兴趣的人，可以参考大量的学术文献。

第一章介绍了密码学，定义了许多术语，简要讨论了计算机出现前密码学的情况。

第一篇（第二~六章）描述密码学的各种协议：人们能用密码学做什么。协议范围从简单（一人向另一人发送加密消息）到复杂（在电话上抛掷硬币）再到深奥的（秘密的和匿名的数字货币交易）。这些协议中有些一目了然，有些却十分奇异。密码术能够解决大多数人绝没有认识到的许多问题。

第二篇（第 7~10 章）讨论密码技术。对密码学的大多数基本应用来说，这一部分的四章都是很重要的。第七章和第八章讨论密钥：密钥应选多长才能保密，怎样产生、存储密钥，怎样处理密钥等等。密钥管理是密码学最困难的一部分，经常是保密系统的一个致命弱点；第九章讨论了使用密码算法的不同方法；第十章给出了与算法有关的细节：怎样选择、实现和使用算法。

第三篇（第 9~23 章）列出了多个算法。第 11 章提供了数学背景，如果你对公开密钥算法感兴趣，这一章是需要了解的。如果你只想实现 DES（或类似的东西），你可以跳过这一章；第 12 章讨论 DES：DES 算法、它的历史、它的安全性和它的一些变形；第 13、14、15 章讨论其它的分组算法。如果你需要比 DES 更保密的算法，请阅读 IDEA 和三重 DES 算法这节。如果你想阅读一系列比 DES 算法更安全的算法，就请读完整章；第 16、17 章讨论序列密码算法；第 18 章集中讨论单向 hash 函数；虽然讨论了好些单向 hash 函数，但 MD5 和 SHA 是最通用的；第 19 章讨论公开密钥加密算法。第 20 章讨论了公开密钥数字签名算法；第 21 章讨论了公开密钥鉴别算法；第 22 章讨论了公开密钥密钥交换算法。几种重要的公开密钥算法分别是 RSA、DSA、Fiat-Shamir 和 Diffie-hellman 算法；第 23 章有更深奥的公开密钥算法和协议。这一章的数学知识是非常复杂的，请系好你的安全带。

第四篇（第 24~25 章）转向密码学的真实世界。第 24 章讨论这些算法和协议的一些实际实现；第 25 章接触到围绕密码学的一些政治问题，这些章节并不全面。

此外，书中还包括在第三篇中讨论的 10 个算法的源代码清单，由于篇幅的限制，我不可能涉及所有的源代码，况且，密码的源代码不能出口（非常奇怪的是，国务院允许本书的第一版和源代码出口，但不允许含有同样源代码的计算机磁盘的出口）。配套的源代码盘包括的源代码比本书中列出的要多得多；这也许是除军事机构以外的最大的密码源代码集。我只能发送源代码盘给住在美国和加拿大的美国和加拿大公民，但我希望有一天这种情况会改变。如果你对这本书的实现或密码算法均感兴趣的话，设法得到这个磁盘。详细情况请看本书的最后一页。

对这本书的一种批评，是它的广博性代替了可读性。这是对的，但我想给可能是偶然在学术文献或产品中需要一个算法的人提供一个参考。对于那些对教材更感兴趣的人，我只能抱歉。密码学领域正日趋热门，这是第一次把这么多资料收集在一本书中。即使这样，还是有许多东西限于篇幅舍弃了，我尽量保留了那些我认为是重要的、有实用价值的或者是有趣

的专题，如果我对某一专题讨论不深，我会给出深入讨论这些专题的参考文献。

笔者在写作过程中已尽力查出和根除书中的错误，但我相信不可能消除所有的错误。第二版肯定比第一版的错误少得多。谬误表可以从我这里得到，并且它被定期发往 Usenet 的新闻组 sci.crypt。如果读者发现错误，请通知我，我将向发现每个错误的第一个人免费寄送一张源码盘，以示感谢。

致谢

为本书提供帮助的人真是数不胜数，他们都值得提及。我谨感谢 Don Alvarez, Ross Anderson, Dave Balenson, Karl Barrus, Steve Bellovin, Dan Bernstein, Eli Biham, Joan Boyar, Karen Cooper, Whit Diffie, Joan Feigenbaum, Phil Karn, Neal Kobitz, Xuejia Lai, Tom Leranth, Mike Markowitz, Ralph Merkle, Bill Patton, Peter Pearson, Charles Pfleegar, Ken Pizzini, Bart Preneel, Mark Riordan, Joachim Schurman 和 Marc Schwartz 感谢他们对本书第一版的审阅和校订。感谢 Marc Vauclair 将它译成法文。感谢 Abe Abraham, Ross Anderson, Dave Banisar, Steve Bellovin, Eli Biham, Matt Bishop, Matt Blaze, Gary Carter, Jan Camenisch, Claude Crepeau, Joan Daemen, Jorge Davila, Ed Dawson, Whit Diffie, Carl Ellison, Joan Feigenbaum, Niels Ferguson, Matt Franklin, Rosario Gennaro, Dieter Gollmann, Mark Goresky, Richard Graveman, Stuart Haber, Jingman He, Bob Hogue, Kenneth Iversen, Markus Jakobsson, Burt Kaliski, Phil Karn, John Kelsey, John Kennedy, Lars Knudsen, Paul Kocher, John Ladwig, Xuejia Lai, Arjen Lenstra, Paul Leyland, Mike Markowitz, Jim Massey, Bruce McNair, William Hugh Murray, Roger Needham, Clif Neuman, Kaisa Nyberg, Luke O'Connor, Peter Pearson, Rene Peralta, Bart Preneel, Yisrael Radai, Matt Robshaw, Michael Roe, Phil Rogaway, Avi Rubin, Paul Rubin, Selwyn Russell, Kazue Sako, Mahmoud Salmasizadeh, Markus Stadler, Dmitry Titov, Jimmy Upton, Marc Vauclair, Serge Vaudenay, Gideon Yuval, Glen Zorn 和其它不愿透露姓名的政府官员们对第 2 版的审阅和校订。感谢 Lawrie Brown, Leisa Condie, Joan Daemen, Peter Gutmann, Alan Insley, Chris Johnston, John Kelsey, Xuejia Lai, Bill Leininger, Mike Markowitz, Richard Outerbridge, Peter Pearson, Ken Pizzini, Colin Plumb, RSA Data Security, Inc.,Michael Wood 和 Phil Zimmermann 提供的源代码。感谢 Paul MacNerland 为第一版制图；Karen Cooper 为第二版排版；Beth Friedman 为第二版核对；Carol Kennedy 为第二版做索引；sci.crypt 和 Cypherpunks 邮件列表的读者们在主意、答问和谬误方面的支持；Randy Seuss 提供的 Internet 接入；Jeff Duntemann 和 Jon Erickson 给我鼓励、支持、交谈和友情和食粮让我动手写作；还要感谢 AT&T 公司开除我，使这一切成为可能。没有这些人的帮助，仅靠我个人是不可能写出好书的。

B.施奈尔

作者简介

BRUCE SCHNEIER 是 Counterpane Systems 公司的总裁，该公司位于伊利诺依州的 Oak 公园，是一个密码学和计算机安全方面的专业咨询公司。Bruce 还是《电子邮件安全》(E-Mail Security <John Wiley & Sons, 1995>)和《保护您的计算机》(Protect Your Macintosh<Peachpit Press, 1994>)两书的作著。在主要的密码学杂志上已发表数十篇论文。是 Dr.Dobb's 杂志责任编辑之一，负责“算法幽经”栏目。同时担任《计算机和通信安全评论》（Computer and Communication Security Reviews）的编辑。Bruce 是“国际密码研究协会”的理事会成员、“电子隐私信息中心”的顾问团成员和“新安全范例工作组”(New Security Paradigms Workshop)程序委员会成员。此外，他还经常开办密码学、计算机安全和隐私保护方面的学术讲座。

目　录

第一章　基础知识

1.1　专业术语

发送者和接收者

假设发送者想发送消息给接收者，且想安全地发送信息：她想确信偷听者不能阅读发送的消息。

消息和加密

消息被称为明文。用某种方法伪装消息以隐藏它的内容的过程称为加密，加了密的消息称为密文，而把密文转变为明文的过程称为解密。图 1.1 表明了这个过程。

（如果你遵循 ISO 7498-2 标准，那就用到术语“译成密码（encipher）”和“解译密码（decipher）”。某些文化似乎认为术语“加密(encrypt)”和“解密(decrypt)”令人生厌，如同陈年腐尸。）

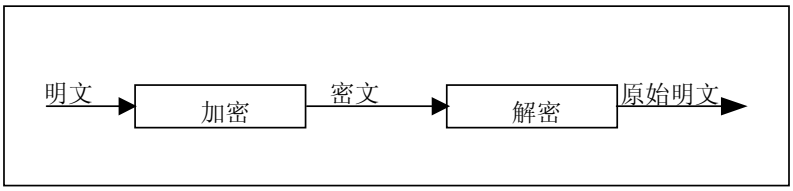

图 1.1　加密和解密

使消息保密的技术和科学叫做**密码编码学**，从事此行的叫密码编码者，**密码分析者**是从事密码分析的专业人员，**密码分析学**就是破译密文的科学和技术，即揭穿伪装。作为数学的一个分支的密码学包括密码编码学和密码分析学两者，精于此道的人称为密码学家，现代的密码学家通常也是理论数学家。

明文用 M（消息）或 P（明文）表示，它可能是比特流（文本文件、位图、数字化的语音流或数字化的视频图像）。至于涉及到计算机，P 是简单地二进制数据（除了这一章节外，这本书本身只涉及二进制数据和计算机密码学）。明文可被传送或存储，无论在哪种情况，M 指待加密的消息。

密文用 C 表示，它也是二进制数据，有时和 M 一样大，有时稍大（通过压缩和加密的结合，C 有可能比 P 小些。然而，单单加密通常达不到这一点）。加密函数 E 作用于 M 得到密文 C，用数学表示为：

$$E（M）=C.$$

相反地，解密函数 D 作用于 C 产生 M

$$D（C）=M.$$

先加密后再解密消息，原始的明文将恢复出来，下面的等式必须成立：

$$D（E（M））=M$$

鉴别、完整性和抗抵赖

除了提供机密性外，密码学通常有其它的作用：.

－鉴别

消息的接收者应该能够确认消息的来源；入侵者不可能伪装成他人。

－完整性

消息的接收者应该能够验证在传送过程中消息没有被修改；入侵者不可能用假消息代替合法消息。

－抗抵赖

发送者事后不可能虚假地否认他发送的消息。

这些功能是通过计算机进行社会交流，至关重要的需求，就象面对面交流一样。某人是否就是他说的人；某人的身份证明文件（驾驶执照、医学学历或者护照）是否有效；声称从某人那里来的文件是否确实从那个人那里来的；这些事情都是通过鉴别、完整性和抗抵赖来实现的。

算法和密钥

密码算法也叫密码，是用于加密和解密的数学函数。（通常情况下，有两个相关的函数：一个用作加密，另一个用作解密）

如果算法的保密性是基于保持算法的秘密，这种算法称为受限制的算法。受限制的算法具有历史意义，但按现在的标准，它们的保密性已远远不够。大的或经常变换的用户组织不能使用它们，因为每有一个用户离开这个组织，其它的用户就必须改换另外不同的算法。如果有人无意暴露了这个秘密，所有人都必须改变他们的算法。

更糟的是，受限制的密码算法不可能进行质量控制或标准化。每个用户组织必须有他们自己的唯一算法。这样的组织不可能采用流行的硬件或软件产品。但窃听者却可以买到这些流行产品并学习算法，于是用户不得不自己编写算法并予以实现，如果这个组织中没有好的密码学家，那么他们就无法知道他们是否拥有安全的算法。

尽管有这些主要缺陷，受限制的算法对低密级的应用来说还是很流行的，用户或者没有认识到或者不在乎他们系统中内在的问题。

现代密码学用密钥解决了这个问题，密钥用 K 表示。K 可以是很多数值里的任意值。密钥 K 的可能值的范围叫做密钥空间。加密和解密运算都使用这个密钥（即运算都依赖于密钥，并用 K 作为下标表示），这样，加/解密函数现在变成：

$$E_K（M）=C$$

$$D_K（C）=M.$$

这些函数具有下面的特性（见图 1.2）：

$$D_K（E_K（M））=M.$$

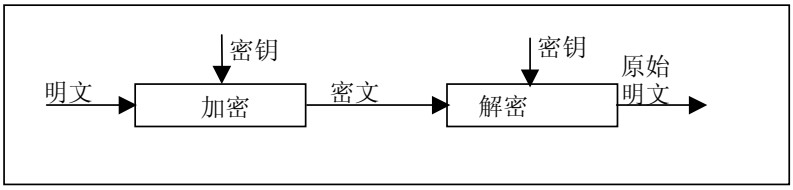

图 1.2　使用一个密钥的加/解密

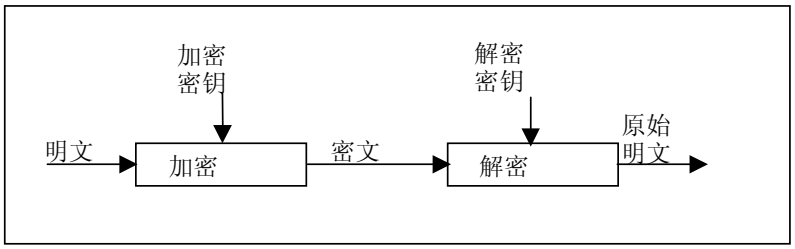

图 1.3　使用两个密钥的加/解密

有些算法使用不同的加密密钥和解密密钥（见图 1.3），也就是说加密密钥 K1 与相应的解密密钥 K2 不同，在这种情况下：

$$E_{K1}（M）=C$$

$$D_{K2}（C）=M$$

$$D_{K2}\ （E_{K1}（M））=M$$

所有这些算法的安全性都基于密钥的安全性；而不是基于算法的细节的安全性。这就意味着算法可以公开，也可以被分析，可以大量生产使用算法的产品，即使偷听者知道你的算法也没有关系；如果他不知道你使用的具体密钥，他就不可能阅读你的消息。

密码系统由算法、以及所有可能的明文、密文和密钥组成的。

对称算法

基于密钥的算法通常有两类：对称算法和公开密钥算法。

对称算法有时又叫传统密码算法，就是加密密钥能够从解密密钥中推算出来，反过来也成立。在大多数对称算法中，加/解密密钥是相同的。这些算法也叫秘密密钥算法或单密钥算法，它要求发送者和接收者在安全通信之前，商定一个密钥。对称算法的安全性依赖于密钥，泄漏密钥就意味着任何人都能对消息进行加/解密。只要通信需要保密，密钥就必须保密。

对称算法的加密和解密表示为：

$$E_K（M）=C$$

$$D_K（C）=M$$

对称算法可分为两类。一次只对明文中的单个比特（有时对字节）运算的算法称为序列算法或序列密码。另一类算法是对明文的一组比特亚行运算，这些比特组称为分组，相应的算法称为分组算法或分组密码。现代计算机密码算法的典型分组长度为 64 比特——这个长度大到足以防止分析破译，但又小到足以方便使用（在计算机出现前，算法普遍地每次只对明文的一个字符运算，可认为是序列密码对字符序列的运算）。

公开密钥算法

公开密钥算法（也叫非对称算法）是这样设计的：用作加密的密钥不同于用作解密的密钥，而且解密密钥不能根据加密密钥计算出来（至少在合理假定的长时间内）。之所以叫做公开密钥算法，是因为加密密钥能够公开，即陌生者能用加密密钥加密信息，但只有用相应的解密密钥才能解密信息。在这些系统中，加密密钥叫做公开密钥（简称公钥），解密密钥叫做私人密钥（简称私钥）。私人密钥有时也叫秘密密钥。为了避免与对称算法混淆，此处不用秘密密钥这个名字。

用公开密钥 K 加密表示为

$$E_K（M）=C.$$

虽然公开密钥和私人密钥是不同的，但用相应的私人密钥解密可表示为：

$$D_K（C）=M$$

有时消息用私人密钥加密而用公开密钥解密，这用于数字签名（见 2.6 节），尽管可能产生混淆，但这些运算可分别表示为：

$$E_K（M）=C$$

$$D_K（C）=M$$

密码分析

密码编码学的主要目的是保持明文（或密钥，或明文和密钥）的秘密以防止偷听者（也叫对手、攻击者、截取者、入侵者、敌手或干脆称为敌人）知晓。这里假设偷听者完全能够接获收发者之间的通信。

密码分析学是在不知道密钥的情况下。恢复出明文的科学。成功的密码分析能恢复出消息的明文或密钥。密码分析也可以发现密码体制的弱点，最终得到上述结果（密钥通过非密码分析方式的丢失叫做泄露。）

对密码进行分析的尝试称为攻击。荷兰人 A.Kerckhoffs 最早在 19 世纪阐明密码分析的一个基本假设，这个假设就是秘密必须全寓于密钥中[794]。Kerckhoffs 假设密码分析者已有密码算法及其实现的全部详细资料（当然，可以假设中央情报局(CIA)不会把密码算法告诉摩萨德(Mossad)（译注：以色列的情报组织），但 Mossad 也许会通过什么方法推出来）。在实际的密码分析中并不总是有这些详细信息的——应该如此假设。如果其他人不能破译算法，即便了解算法如何工作也是徒然，如果连算法的知识都没有，那就肯定不可能破译它。

常用的密码分析攻击有四类，当然，每一类都假设密码分析者知道所用的加密算法的全部知识：

(1) 唯密文攻击。密码分析者有一些消息的密文，这些消息都用同一加密算法加密。密码分析者的任务是恢复尽可能多的明文，或者最好是能推算出加密消息的密钥来，以便可采用相同的密钥解出其他被加密的消息。

已知：$C_1=E_K（P_1）$，$C_2=E_K（P_2）$，……，$C_I=E_K（P_i）$

推导出：P_1，P_2，…，P_i；K 或者找出一个算法从 $C_{i+1}= E_K（P_{i+1}）$ 推出 P_{i+1}。

(2) 已知明文攻击。密码分析者不仅可得到一些消息的密文，而且也知道这些消息的明

文。分析者的任务就是用加密信息推出用来加密的密钥或导出一个算法，此算法可以对用同一密钥加密的任何新的消息进行解密。

已知：P_1，$C_1=E_k（P_1）$，P_2，$C_2=E_k（P_2）$，…，P_i，$C_i=E_k（P_i）$，

推导出：密钥 k，或从 $C_{i+1}= E_k（P_{i+1}）$ 推出 P_{i+1} 的算法。

（3）选择明文攻击。分析者不仅可得到一些消息的密文和相应的明文，而且他们也可选择被加密的明文。这比已知明文攻击更有效。因为密码分析者能选择特定的明文块去加密，那些块可能产生更多关于密钥的信息，分析者的任务是推出用来加密消息的密钥或导出一个算法，此算法可以对用同一密钥加密的任何新的消息进行解密。

已知：P_1，$C_1=E_k（P_1）$，P_2，$C_2=E_k（P_2）$，…，P_i，$C_i=E_k（P_i）$

其中 P_1，P_2，…，P_i 是由密码分析者选择的。

推导出：密钥 k，或从 $C_{i+1}= E_k（P_{i+1}）$ 推出 P_{i+1} 的算法。

（4）自适应选择明文攻击。这是选择明文攻击的特殊情况。密码分析者不仅能选择被加密的明文，而且也能基于以前加密的结果修正这个选择。在选择明文攻击中，密码分析者还可以选择一大块被加了密的明文。而在自适应选择密文攻击中，他可选取较小的明文块，然后再基于第一块的结果选择另一明文块，以此类推。

另外还有至少三类其它的密码分析攻击。

（5）选择密文攻击。密码分析者能选择不同的被加密的密文，并可得到对应的解密的明文，例如密码分析者存取一个防窜改的自动解密盒，密码分析者的任务是推出密钥。

已知：C_1，$P_1=D_k（C_1）$，C_2，$P_2=D_k（C_2）$，…，C_i，$P_i=D_k（C_i）$，

推导出： k。

这种攻击主要用于公开密钥体制，这将在 19.3 节中讨论。选择密文攻击有时也可有效地用于对称算法（有时选择明文攻击和选择密文攻击一起称作选择文本攻击。）

（6）选择密钥攻击。这种攻击并不表示密码分析者能够选择密钥，它只表示密码分析者具有不同密钥之间的关系的有关知识。这种方法有点奇特和晦涩，不是很实际，将在 12.4 节讨论。

（7）软磨硬泡(Rubber-hose)攻击。密码分析者威胁、勒索，或者折磨某人，直到他给出密钥为止。行贿有时称为购买密钥攻击。这些是非常有效的攻击，并且经常是破译算法的最好途径。

已知明文攻击和选择明文攻击比你想象的更常见。密码分析者得到加了密的明文消息或贿赂某人去加密所选择的消息，这种事情时有所闻。如果你给某大使一则消息，也可能发现该消息已加密了，并被送回他的国家去研究。此时你会去贿赂某人；密码分析者也许知道，许多消息有标准的开头和结尾。加密的源码特别脆弱，这是因为有规律地出现关键字， 如出现：#define，struct，else，return 等。加了密的可执行代码也有同样问题，如：调用函数、循环结构等等。已知明文攻击（甚至选择明文攻击）在二战中已被成功地用来破译德国和日本的密码。David Kahn 的书中有此类攻击的历史例子[794，795，796]。

不要忘记 Kerckhoffs 的假设：如果你的新的密码系统的强度依赖于攻击者不知道算法的内部机理，你注定会失败。如果你相信保持算法的内部秘密比让研究团体公开分析它更能改进你的密码系统的安全性，那你就错了。如果你认为别人不能反汇编你的代码和逆向设计你

的算法，那你就太天真了（1994 年 RC4 算法就发生了这种情况—见 17.1 节）。最好的算法是那些已经公开的，并经过世界上最好的密码分析家们多年的攻击，但还是不能破译的算法（国家安全局对外保持他们的算法的秘密，但他们有世界上最好的密码分析家在内部工作，你却没有。另外，他们互相讨论他们的算法，通过执著的审查发现他们工作中的弱点）。

密码分析者不是总能知道算法的。例如在二战中美国人破译日本人的外交密码——紫密（PURPLE）[794]就是例子，而且美国人一直在做这种事。如果算法用于商业安全程序中，那么拆开这个程序，把算法恢复出来只是时间和金钱问题；如果算法用于军队的通讯系统中，购买（或窃取）这种设备，进行逆向工程恢复算法也只是简单的时间和金钱的问题。

那些因为自己不能破译某个算法就草率地声称有一个不可破译的密码的人要么是天才，要么是笨蛋，不幸的是后者居多。千万要提一味吹嘘算法的优点，但又拒绝公开的人，相信他们的算法就像相信骗人的包医百病的灵丹妙药一样。

好的密码分析家总会坚持审查，以图把不好的算法从好的算法中剔除出去。

算法的安全性

根据被破译的难易程度，不同的密码算法具有不同的安全等级。如果破译算法的代价大于加密数据的价值，那么你可能是安全的。如果破译算法所需的时间比加密数据保密的时间更长，那么你可能是安全的；如果用单密钥加密的数据量比破译算法需要的数据量少得多，那么你可能是安全的。

我说“可能”是因为在密码分析中总有新的突破。另一方面，大多数数据随着时间的推移，其价值会越来越小，而数据的价值总是比突破保护它的安全性的代价更小，这点是很重要的。

Lars Knudsen 把破译算法分为不同的类别，安全性的递减顺序[858]为：

1．**全部破译**。密码分析者找出密钥 K，这样 D_K（C）=P。
2．**全盘推导**。密码分析者找到一个代替算法 A，在不知道密钥 K 的情况下，等价于 D_K（C）=P。
3．**实例（或局部）推导**。密码分析者从截获的密文中找出明文。
4．**信息推导**。密码分析者获得一些有关密钥或明文的信息。这些信息可能是密钥的几个比特、有关明文格式的信息等等。

如果不论密码分析者有多少密文，都没有足够的信息恢复出明文，那么这个算法就是无条件保密的，事实上，只有一次一密乱码本（参看 1.5 节），才是不可破的（给出无限多的资源仍然不可破）。所有其它的密码系统在唯密文攻击中都是可破的，只要简单地一个接一个地去试每种可能的密钥，并且检查所得明文是否有意义，这种方法叫做蛮力攻击（见 7.1 节）。

密码学更关心在计算上不可破译的密码系统。如果一个算法用（现在或将来）可得到的资源都不能破译，这个算法则被认为在计算上是安全的（有时叫做强的）。准确地说，“可

用资源”就是公开数据的分析整理。

你可以用不同方式衡量攻击方法的复杂性（见 11.1 节）：

1．数据复杂性。用作攻击输入所需的数据量。

2．处理复杂性。完成攻击所需要的时间，这个经常叫做工作因素。

3．存储需求。进行攻击所需要的存储量。

作为一个法则，攻击的复杂性取这三个因数的最小化，有些攻击包括这三种复杂性的折中：存储需求越大，攻击可能越快。

复杂性用数量级来表示。如果算法的处理复杂性是 2^{128}，那么破译这个算法也需要 2^{128} 次运算（这些运算可能是非常复杂和耗时的。）。假设你有足够的计算速度去完成每秒钟一百万次运算，并且用 100 万个并行处理器完成这个任务，那么仍需花费 10^{19} 年以上才能找出密钥，那是宇宙年龄的 10 亿倍。

当攻击的复杂性是常数时（除非一些密码分析者发现更好的密码分析攻击），就只取决于计算能力了。在过去的半个世纪中，我们已看到计算能力的显著提高，并且没有理由认为这种趋势不会继续。许多密码分析攻击用并行处理机是非常理想的：这个任务可分成亿万个子任务，且处理之间不需相互作用。一种算法在现有技术条件下不可破译就简单地宣称该算法是安全的，这未免有些冒险。好的密码系统应设计成能抵御未来许多年后计算能力的发展。

过去的术语

历史上，将处理语言单元的密码系统称为密本：字、短语、句子等。例如，单词“OCELOT”可能是整个短语“TURN LEFT 90 DEGREES，”的密文，单词“LOLLIPOP”可能是“TURN RIGHT 90 DEGREES”的密文，而字“BENT EAR”可能是“HOWITZER”的密文。这种类型的密本在本书里没有讨论。密本在特殊环境中才有用，而密码在任何情况下都有用；密码本中若没有“ANTEATERS“这一条，那么你就不能提它。但你可以用密码来指代任何东西。

1.2 隐写术

隐写术是将秘密消息隐藏在其它消息中，这样，真正存在的秘密被隐藏了。通常发送者写一篇无伤大雅的消息，然后在同一张纸中隐藏秘密消息。历史上的隐写方式有隐形墨水，用小针在选择的字符上刺小的针眼，在手写的字符之间留下细微差别，在打印字符上用铅笔作记号、除了几个字符外，大部分字符用格子盖起来等等。

最近，人们在图象中隐藏秘密消息，用图象的每个字节的最不重要的比特代替消息比特。图象并没有怎么改变—大多数图象标准规定的颜色等级比人类眼睛能够觉察得到的要多得多——秘密消息却能够在接收端剥离出来。用这种方法可在 1024 x 1024 灰色刻度图片中存储 64K 字节的消息。能做此类把戏的公开程序已有好几种。

Peter Wayner 的模拟函数也能使消息隐匿，这类函数能修改消息，使它的统计外形与一些其它东西相似：如纽约时报的题录部分、莎士比亚的戏剧、Internet 网上的新闻组[1584，

1585]。这类隐写术愚弄不了普通人，但却可以愚弄那些为特定的消息而有目的地扫描 Internet 的大型计算机。

1.3 代替密码和换位密码

在计算机出现前，密码学由基于字符的密码算法构成。不同的密码算法是字符之间互相代换或者是互相之间换位，好的密码算法是结合这两种方法，每次进行多次运算。

现在事情变得复杂多了，但原理还是没变。重要的变化是算法对比特而不是对字母进行变换，实际上这只是字母表长度上的改变，从 26 个元素变为 2 个元素。大多数好的密码算法仍然是代替和换位的元素组合。

代替密码

代替密码就是明文中每一个字符被替换成密文中的另外一个字符。接收者对密文进行逆替换就恢复出明文来。

在经典密码学中，有四种类型的代替密码——

（1）简单代替密码，或单字母密码：就是明文的一个字符用相应的一个密文字符代替。报纸中的密报就是简单的代替密码。

（2）多名码代替密码：它与简单代替密码系统相似，唯一的不同是单个字符明文可以映射成密文的几个字符之一，例如 A 可能对应于 5、13、25 或 56，“B”可能对应于 7、19、31 或 42，等等。

（3）字母代替密码：字符块被成组加密，例如“ABA”可能对应于“RTQ”，ABB 可能对应于“SLL”等。

（4）多表代替密码：由多个简单的代替密码构成，例如，可能有 5 个被使用的不同的简单代替密码，单独的一个字符用来改变明文的每个字符的位置。

著名的凯撒密码就是一种简单的代替密码，它的每一个明文字符都由其右边第 3 个（模 26）字符代替（A 由 D 代替，B 由 E 代替……W 由 Z 代替…X 由 A 代替，Y 由 B 代替，Z 由 C 代替）。它实际上更简单，因为密文字符是明文字符的环移，并且不是任意置换。

ROT13 是建在 UNIX 系统上的简单的加密程序，它也是简单的代替密码。在这种密码中，A 被 N 代替，B 被 O 代替等等，每一个字母是环移 13 所对应的字母。

用 ROT13 加密文件两遍便恢复出原始的文件：

$$P=ROT13(ROT13(P))$$

ROT13 并非为保密而设计的，它经常用在互联网 Vsenet 电子邮件中隐藏特定的内容，以避免泄露一个难题的解答等等。

简单代替密码是很容易破译的，因为它没有把明文的不同字母的出现频率掩盖起来。在好的密码分析者重构明文之前，所有的密文都由 25 个英文字母组成[1434]，破译这类密码的算法可以在[578，587，1600，78，1475，1236，880]中找到。好的计算机破译算法见[703]。

多名码代替密码早在 1401 年最早由 Duchy Mantua 公司使用，这些密码比简单代替密码更难破译，但仍不能掩盖明文语言的所有统计特性，用已知明文攻击，破译这种密码非常容易，唯密文攻击要难一些，但在计算机上只需几秒钟[710]。详情见[126]中的叙述。

多字母代替密码是字母成组加密，普莱费尔在 1854 年发明了这种密码。在第一次世界

大战中英国人就采用这种密码[794]。字母成对加密，它的密码分析在[587，1475，880]中讨论。希尔密码是多字母代替密码的另一个例子[732]。有时你会把 Huffman 编码用作密码，这是一种不安全的多字母代替密码。

多表代替密码由 Leon Battista 在 1568 年发明[794]，在美国南北战争期间由联军使用。尽管他们容易破译[819，577，587，794]（特别是在计算机的帮助下），许多商用计算机保密产品都使用这种密码形式[1387，1390，1502]。（怎么破译这个加密方案的细节能够在[135，139]中找到，这个方案用在 Word-Perfect 中）。维吉尼亚密码（第一次在 1586 年发表）和博福特密码均是多表代替密码的例子。

多表代替密码有多个单字母密钥，每一个密钥被用来加密一个明文字母。第一个密钥加密明文的第一个字母，第二个密钥加密明文的第二个字母等等。在所有的密钥用完后，密钥又再循环使用，若有 20 个单个字母密钥，那么每隔 20 个字母的明文都被同一密钥加密，这叫做密码的周期。在经典密码学中，密码周期越长越难破译，使用计算机就能够轻易破译具有很长周期的代替密码。

滚动密钥密码（有时叫书本密码）是多表代替密码的另一个例子，就是用一个文本去加密另一个文本，即使这种密码的周期与文本一样长，它也是很容易被破译的[576，794]。

换位密码

在换位密码中，明文的字母保持相同，但顺序被打乱了。在简单的纵行换位密码中，明文以固定的宽度水平地写在一张图表纸上，密文按垂直方向读出（见图 1.4），解密就是将密文按相同的宽度垂直地写在图表纸上，然后水平地读出明文。

Plaintext: COMPUTERGRAPHICSMAYBESLOWBUTATLEASTITSEXPENSIVE
COMPUTERGR
APHICSMAYB
ESLOWBUTAT
LEASTITSEX
PENSIVE
Ciphertext: CAELPOPSEEMHLANPIOSSUCWTITSBIVEMUTERATSGYAERBTX

图 1.4　纵行换位密码

对这些密码的分析在[587，1475]中讨论。由于密文字符与明文字符相同，对密文的频数分析将揭示出 每个字母和英语有相似的或然值。这给了密码分析者很好的线索，他就能够用各种技术去决定字母的准确顺序，以得到明文。密文通过二次换位密码极大地增强了安全性。甚至有更强的换位密码，但计算机几乎都能破译。

在第一次世界大战中，德国人所用的 ADFGVX 密码就是一种换位密码与简单的代替密码的组合。在那个时代它是一个非常复杂的算法，但被法国密码分析家 George Painvin 所破[794]。

虽然许多现代密码也使用换位，但由于它对存储要求很大，有时还要求消息为某个特定

的长度，因而比较麻烦。代替密码要常用得多。

转轮机

在 20 年代，人们发明各种机械加密设备用来自动处理加密。大多数是基于转轮的概念，机械转轮用线连起来完成通常的密码代替。

转轮机有一个键盘和一系列转轮，它是 Vigenere 密码的一种实现。每个转轮是字母的任意组合，有 26 个位置，并且完成一种简单代替。例如：一个转轮可能被用线连起来以完成用“F”代替“A”，用“U”代替“B”，用“L”代替“C”等等，而且转轮的输出栓连接到相邻的输入栓。

例如，在 4 个转轮的密码机中，第一个转轮可能用“F”代替“A”， 第二个转轮可能用“Y”代替“F”， 第三个转轮可能用“E”代替“Y”， 第四个转轮可能用“C”代替“E”，“C”应该是输出密文。那么当转轮移动后，下一次代替将不同了。

为使机器更安全，可把几种转轮和移动的齿轮结合起来。因为所有转轮以不同的速度移动，n 个转轮的机器的周期是 26^n。，为进一步阻止密码分析，有些转轮机在每个转轮上还有不同的位置号。

最著名的转轮装置是恩尼格马(Enigma)。恩尼格马在第二次世界大战期间由德国人使用。其基本原理由欧洲的 Arthur Scherbius 和 Arvid Gerhard Damn 发明，它由 Arthur Scherbius 在美国申请了专利[1383]，德国人为了战时使用，大大地加强了基本设计。

恩尼格马有三个转轮，从五个转轮中选择。转轮机中有一块稍微改变明文序列的插板，有一个反射轮导致每个转轮对每一个明文字母操作两次。像恩尼格马那样复杂的密码，在第二次世界大战期间都被破译了。波兰密码小组最早破译了德国的恩尼格马，并告诉了英国人。德国人在战争进行过程中修改了他们的密码。英国人继续对新的方案进行分析，他们是如何破译的，请见[794，86，448，498，446，880，1315，1587，690]。有关怎么破译恩尼格马的两个传奇报道在[735，796]中叙述。

进一步的读物

这不是一本经典密码学的书，因此不多讨论这些问题。计算机出现前有两本优秀的密码学著作是[587，1475]；[448]提出了一些密码机的现代密码分析方法。Dorothy Denning 在[456]中讨论了许多密码，而[880]对这些密码作了很多复杂的数学分析。另一本更早的讨论模拟密码的著作见[99]，文献 [579]对这个学科做了很好的回顾。David Kahn 的历史性密码学论著也是非常优秀的[794，795，796]。

1.4 简单异或

异或在 C 语言中是“^”操作，或者用数学表达式 $\oplus$ 表示。它是对比特的标准操作：

$$0 \oplus 0 = 0$$

$$0 \oplus 1 = 1$$

$$1 \oplus 0 = 1$$

1⊕1 = 0

也要注意：

a⊕a = 0

a⊕b⊕b = a

简单异或算法实际上并不复杂，因为它并不比维吉尼亚密码多什么东西。它之所以被包括在这本书中，是因为它在商业软件包中很流行，至少在 MS-DOS 和 Macintosh 世界中是这样[1502，1387]。不幸的是，如果一个软件保密程序宣称它有一个“专有”加密算法（该算法比 DES 更快），其优势在于是下述算法的一个变种。

```
/ * usage crypto Key input-file output-fife * /
void main (（int argc, char argvl ）)
* FILE * fi，* fo；
  int * cp；
  int   c；
  if（（cp = argv[1]）&& *cp != '\0'）{
  if（（fi=fopen（argv[2]，“rb”））!  =NULL）{
     if（（fo=fopen（argv[3]，“wb”））! =NULL）{
       While（（c=getc（fi）!=EOF）{
         if（! * cp）   cp=argv[1]；
         C ^=*（cp + +）；
              Putc（c，fo）；
       }
       fclose（fo）；
     }
     fclose（fi）；
     }
  }
}
```

这是一个对称算法。明文用一个关键字作异或运算以产生密文。因为用同一值去异或两次就恢复出原来的值，所以加密和解密都严格采用同一程序。

$$P \oplus K = C$$

$$C \oplus K = P$$

这种方法没有实际的保密性，这类加密易于破译，甚至没有计算机也能破译[587,1475]，用计算机只需花费几秒钟就可破译。

假设明文是英文，而且假设密钥长度是一个任意小的字节数，下面是它的破译方法——

（1）用重合码计数法找出密钥长度[577]。用密文异或相对其本身的各种字节的位移，统计那些相等的字节数。如果移位是密钥长度的倍数，那么超过 6%的字节将是相等的。如果不是，则只有 0.4%以下的字节将是相等的（假设一随机密钥加密标准的 ASCII 文本；其

他的明文将有不同的数值)。这叫做重合指数。指出密钥长度倍数的最小位移就是密钥的长度。

（2）按那个长度移动密文，并且和自己异或：这样就消除了密钥，留下明文和移动了密钥长度的明文的异或。由于英语每字节有 1.3 比特的实际信息（见 11.1 节），有足够的多余度去测定唯一的解密。

尽管如此，一些软件销售商在兜售这种游戏式算法时，还声称“几乎和 DES 一样保密”，这使人感到震惊。NSA 最终允许美国的数字蜂窝电话产业界使用这个算法（有 160 比特的重复“密钥”）对话音保密。异或或许能防止你的小妹妹读你的文件，但却不能防止密码分析家在几分钟内破译它。

1.5 一次一密乱码本

不管你是否相信它，有一种理想的加密方案，叫做一次一密乱码本，由 Major Joseph Mauborgne 和 AT&T 公司的 Gilbert Vernam 在 1917 年发明的[794]（实际上，一次一密乱码本是门限方案的特殊情况，见 3.7 节)。典型地，一次一密乱码本不外乎是一个大的不重复的真随机密钥字母集，这个密钥字母集被写在几张纸上，并一起粘成一个乱码本。它最初的形式是将一次一密乱码本用于电传打字机。发方用乱码本中的每一密钥字母准确地加密一个明文字符。加密是明文字符和一次一密乱码本密钥字符的模 26 加法。

每个密钥仅对一个消息使用一次。发方对所发的消息加密，然后销毁乱码本中用过的一页或用过的磁带部分。收方有一个同样的乱码本，并依次使用乱码本上的每个密钥去解密密文的每个字符。收方在解密消息后销毁乱码本中用过的一页或用过的磁带部分。新的消息则用乱码本的新的密钥加密。

例如，如果消息是：

ONETIMEPAD ，

而取自乱码本的密钥序列是

TBFRGFARFM ，

那么密文就是

IPKLPSFHGQ ，

因为

O + T mod26 = I

N + B mod26 = P

E + F mod26 = K

等等。

如果偷窃听者不能得到用来加密消息的一次一密乱码本，这个方案是完全保密的。给出的密文消息相当于同样长度的任何可能的明文消息。

由于每一密钥序列都是等概的（记住，密钥是以随机方式产生的），敌方没有任何信息用来对密文进行密码分析，密钥序列也可能是：

POYYAEAAZX

解密出来是：

SALMONEGGS

或密钥序列为：

BXFGBMTMXM

解密出来的明文为：

GREENFLUID

值得重申的是：由于明文消息是等概的，所以密码分析者没有办法确定哪一明文消息是正确的。随机密钥序列异或一非随机的明文消息产生一完全随机的密文消息。再大的计算能力也无能为力。

值得注意的是，密钥字母必须是随机产生的。对这种方案的攻击将是针对用来产生密钥序列的那种方法。使用伪随机数发生器是不值得考虑的，它们通常具有非随机性。如果你采用真随机源（这比第一次出现难得多，见 17.14 节。），它就是安全的。

另一个重要的事情是密钥序列不能重复使用，即使你用多兆字节的乱码本，如果密码分析家有多个密钥重叠的密文，他也能够重构明文。他把每排密文移来移去，并计算每个位置的适配量。如果他们排列正确，则适配的比例会突然升高（准确的百分比与明文的语种有关）。从这一点来说，密码分析是容易的，它类似于重合指数法，只不过用两个“周期”作比较[904]。所以千万别重复使用密钥序列。

一次一密乱码本的想法很容易推广到二进制数据的加密，只需由二进制数字组成的一次一密乱码本代替由字母组成的一次一密乱码，用异或代替一次一密乱码本的明文字符加法就成。为了解密，用同样的一次一密乱码本对密文异或，其他保持不变，保密性也很完善。

这听起来很好，但有几个问题。因为密钥比特必须是随机的，并且绝不能重复使用，密钥序列的长度要等于消息的长度。一次一密乱码本可能对短信息是可行的，但它决不可能在 1.44Mbps 的通信信道上工作。你能在一张 CD-ROM 中存储 650 兆字节的随机二进制数。但有一些问题：首先，你需要准确地复制两份随机数比特，但 CD-ROM 只是对大量的数据来说是经济的；其次，你需要能够销毁已经使用过的比特，而 CD-ROM 没有抹除设备，除非物理毁坏整张盘。数字磁带对这种东西来说是更好的媒体。

即使解决了密钥的分配和存储问题，还需确信发方和收方是完全同步的。如果收方有一比特的偏移（或者一些比特在传送过程中丢失了），消息就变成乱七八糟的东西了。另一方面，如果某些比特在传送中被改变了（没有增减任何比特，更像由于随机噪声引起的），那些改变了的比特就不能正确地解密。再者，一次一密乱码本不提供鉴别。

一次一密乱码本在今天仍有应用场合，主要用于高度机密的低带宽信道。美国和前苏联之间的热线电话（现在还在起作用吗？）据传就是用一次一密乱码本加密的。许多苏联间谍传递的消息也是用一次一密乱码本加密的。到今天这些消息仍是保密的，并将一直保密下去。不管超级计算机工作多久，也不管半个世纪中有多少人，用什么样的方法和技术，具有多大的计算能力，他们都不可能阅读苏联间谍用一次一密乱码本加密的消息（除非他们恰好回到那个年代，并得到加密消息的一次一密乱码本）。

1.6 计算机算法

计算机密码算法有多种，最通用的有三种：

——DES（数据加密标准）是最通用的计算机加密算法。DES 是美国和国际标准，它是对称算法，加密和解密的密钥是相同的。

——RSA（根据它的发明者命名的，即 Rivest，Shamir 和 Adleman）是最流行的公开密钥算法，它能用作加密和数字签名。

——DSA（数字签名算法，用作数字签名标准的一部分）是另一种公开密钥算法，它不能用作加密，只用作数字签名。

这些就是本书所要涉及的题材。

1.7 大数

在整本书中，我用各种大数去描述密码算法中的不同内容。因为很容易忽略这些数和它们的实际意义，所以表 1.1 给出了一些大数的物理模拟量。

表中这些数是估计的数量级，并且是从各种资料中精选得到的，天体物理学中许多大数在 Freeman Dyson 的文章中有解释，该文章发表在《现代物理学评论》中，名为“时间永无止境：开放宇宙中的物理学和生物学”。汽车事故的死亡人数是根据 1993 年交通部统计数据每百万人中有 178 起死亡事故和人均寿命为 75.4 年计算出来的。

表 1.1　大数

物 理 模 拟 量	大　数
每天被闪电杀死的可能性	90 亿（2^{33}）分之一
赢得国家发行彩票头等奖的可能性	4 百万（2^{22}）分之一
赢得国家发行彩票头等奖并且在同一天被闪电杀死的可能性	$1/2^{55}$
每年淹死的可能性	$\frac{1}{59000}\left(2^{-16}\right)$
1993 年在美国交通事故中死亡的可能性	6100（2^{13}）分之一
一生在美国死于交通事故的可能性	88（2^{7}）分之一
到下一个冰川年代的时间	14000（2^{14}）年
到太阳变成新星的时间	10^{9}（2^{30}）年
行星的年龄	10^{9}（2^{30}）年
宇宙的年龄	10^{10}（2^{34}）年
行星中的原子数	10^{51}（2^{170}）
太阳中的原子数	10^{57}（2^{190}）
银河系中的原子数	10^{67}（2^{223}）
宇宙中的原子数（黑粒子除外）	10^{77}（2^{265}）
宇宙的体积	10^{84}（2^{280}）cm^3

如果宇宙是封闭的：

宇宙的生命期	10^{11}（2^{37}）年
	10^{18}（2^{61}）秒

如果宇宙是开放的：

到小弥撒星冷却下来的时间	10^{14}（2^{47}）年
到行星脱离星系的时间	10^{15}（2^{50}）年
到行星脱离银河系的时间	10^{19}（2^{64}）年
到由引力线引起的轨道蜕变的时间	10^{20}（2^{67}）年
到由散播过程引起黑洞湮没的时间	10^{64}（2^{213}）年
到所有物质在 0^0 时都为液体的时间	10^{65}（2^{216}）年
到所有物质都蜕变成铁的时间	$10^{10^{26}}$ 年
到所有物质都收缩为黑洞的时间	$10^{10^{76}}$ 年

第一篇　密码协议

第二章　协议结构模块

2.1　协议介绍

密码学的用途是解决种种难题（实际上，很多人忘记了这也是计算机的主要用途）。密码学解决的各种难题围绕机密性、鉴别、完整性和不诚实的人。你可能了解各种算法和技术，除非它们能够解决某些问题，否则这些东西只是理论而已，这就是为什么我们要先了解协议的原因。

协议是一系列步骤，它包括两方或多方，设计它的目的是要完成一项任务。这个定义很重要：“一系列步骤”意味着协议是从开始到结束的一个序列，每一步必须依次执行，在前一步完成前，后面的步骤都不能执行；“包括两方或多方”意味着完成这个协议至少需要两个人，单独的一个人不能构成协议，当然单独的一个人也可采取一系列步骤去完成一个任务（例如烤蛋糕），但这不是协议（另外一些人必须吃蛋糕才构成协议）；最后，“设计它的目的是要完成一项任务”意味着协议必须做一些事。有些东西看起来像协议，但不完成一个任务，那也不是协议，只是浪费时间而已。

协议还有其他特点：

（1）协议中的每人都必须了解协议，并且预先知道所要完成的所有步骤。

（2）协议中的每人都必须同意遵循它。

（3）协议必须是不模糊的，每一步必须明确定义，并且不会引起误解。

（4）协议必须是完整的，对每种可能的情况必须规定具体的动作。

此书中的协议就安排成一系列步骤，并且协议是按照规定的步骤线形执行的，除非指定它转到其他步骤。每一步至少要做下列两件事中的一件，即由一方或多方计算，或者是在各方中传送信息。

密码协议是使用密码学的协议。参与该协议的伙伴可能是朋友和完全信任的人，或者也可能是敌人和互相完全不信任的人。密码协议包含某种密码算法，但通常，协议的目的不仅仅是为了简单的秘密性。参与协议的各方可能为了计算一个数值想共享它们的秘密部分、共同产生随机系列、确定互相的身份、或者同时签署合同。在协议中使用密码的目的是防止或发现偷听者和欺骗。如果你以前没有见过这些协议，它们会从根本上改变你的思想，相互之间不信任的各方也能够在网络上完成这些协议。一般地，这能够被陈述为：

_不可能完成或知道得比协议中规定的更多。

这看起来很难。接下来的几章讨论了许多协议。在其中的一些协议中，参与者中的一个有可能欺骗其他人。偷听者也可能暗中破坏协议或获悉秘密信息。一些协议之所以失败，是因为设计者对需求不是定义得很完备。其它一些失败是因为协议的设计者分析得不够充分。就像算法一样，证明它不安全比证明它安全容易得多。

协议的目的

在日常生活中，几乎所有的事情都有非正式的协议：电话订货、玩扑克、选举中投票，没有人认真考虑过这些协议，这些协议随着时间的推移而发展，人们都知道怎样使用它们，而且它们也很有效。

越来越多的人通过计算机网络交流，从而代替了面对面的交流。计算机需要正式的协议来完成人们不用考虑就能做的事情。如果你从一个州迁移到另一个州，可能会发现投票亭与你以前使用的完全不同，你会很容易去适应它。但计算机就不那么灵活了。

许多面对面的协议依靠人的现场存在来保证公平和安全。你会交给陌生人一叠现金去为你买食品吗？如果你没有看到他洗牌和发牌，你愿意和他玩扑克吗？如果没有匿名的保证，你会将秘密投票寄给政府吗？

那种假设使用计算机网络的人都是诚实的想法，是天真的。天真的想法还有：假设计算机网络的管理员是诚实的，假设计算机网络的设计者是诚实的。当然，绝大多数人是诚实的，但是不诚实的少数人可能招致很多损害。通过规定协议，可以查出不诚实者企图欺骗的把戏，还可开发挫败这些欺骗者的协议。

除了规定协议的行为外，协议还根据完成某一任务的机理，抽象出完成此任务的过程。不管是 IBM PCs 还是 VAX 机或者传真机，通信协议是相同的。我们能够考查协议，而不用囿于具体的实现上。当我们坚信有一个好的协议时，在从计算机到电话再到智能烘箱的所有事情中，我们都能够实现它。

游戏角色

为了帮助说明协议，我列出了几个人作为助手（参见表 2.1）。Alice 和 Bob 是开始的两个人。他们将完成所有的两人协议。按规定，由 Alice 发起所有协议，Bob 响应。如果协议需要第三或第四人，Carol 和 Dave 将扮演这些角色。由其他人扮演的专门配角，将在后面介绍。

表 2.1　剧中人

Alice	所有协议中的第一个参加者
Bob	所有协议中的第二个参加者
Carol	在三、四方协议中的参加者
Dave	在四方协议中的参加者
Eve	窃听者
Mallory	恶意的主动攻击者
Trent	值得信赖的仲裁者
Walter	监察人：在某些协议中保护 Alice 和 Bob
Peggy	证明人
Victor	验证者

仲裁协议

仲裁者是在完成协议的过程中，值得信任的公正的第三方（参见图 2.1 中的 a），“公正”意味着仲裁者在协议中没有既得利益，对参与协议的任何人也没有特别的利害关系。“值得信任”表示协议中的所有人都接受这一事实，即仲裁者说的都是真实的，他做的是正确的，并且他将完成协议中涉及他的部分。仲裁者能帮助互不信任的双方完成协议。

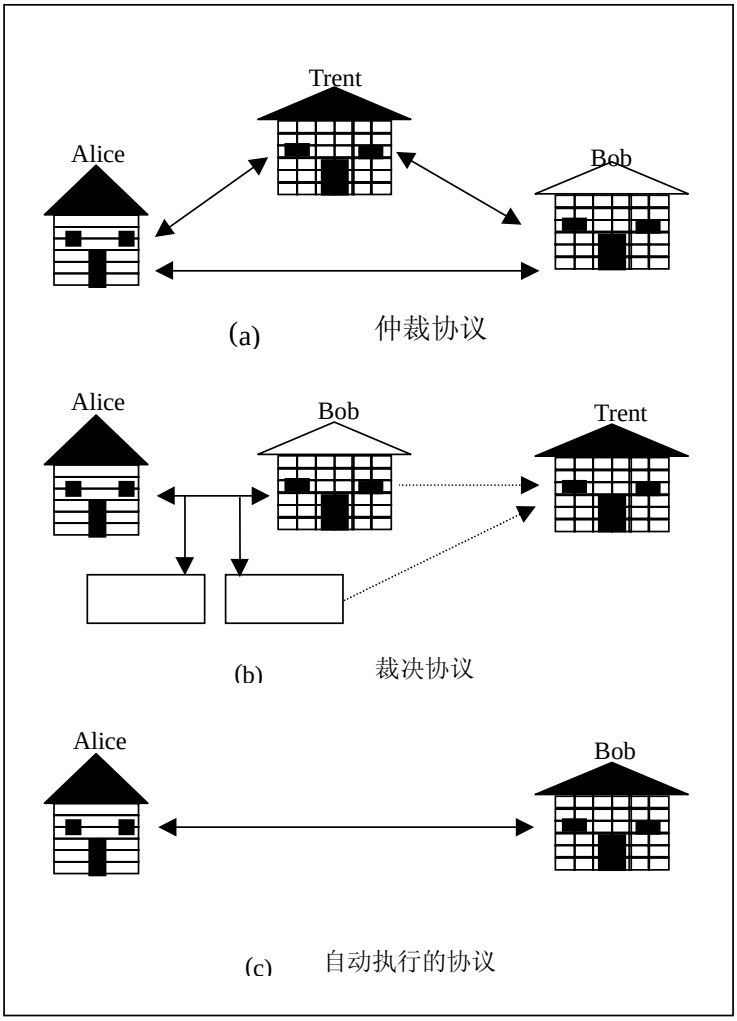

图 2.1　协议类型

在现实社会中，律师经常作为仲裁者。例如，Alice 要卖汽车给不认识的 Bob。Bob 想用支票付帐，但 Alice 不知道支票的真假。在 Alice 将车子转给 Bob 前，她必须查清支票的真伪。同样，Bob 也并不相信 Alice，就像 Alice 不相信 Bob 一样，在没有获得所有权前，也不愿将支票交与 Alice。

这时就需要双方都信任的律师。在律师的帮助下，Alice 和 Bob 能够用下面的协议保证互不欺骗。

（1）Alice 将车的所有权交给律师。

（2）Bob 将支票交给 Alice。

（3）Alice 在银行兑现支票。

（4）在等到支票鉴别无误能够兑现的时间之后，律师将车的所有权交给 Bob。如果在规定的时间内支票不能兑现，Alice 将证据出示给律师，律师将车的所有权和钥匙交还给 Alice。

在这个协议中，Alice 相信律师不会将车的所有权交给 Bob，除非支票已经兑现；如果支票不能兑现，律师会把车的所有权交还给 Alice。而 Bob 相信律师有车的所有权，在支票兑现后，将会把车主权和钥匙交给他。而律师并不关心支票是否兑现，不管在什么情况下，他只做那些他应该做的事，因为不管在哪种情况，他都有报酬。

在这例子中，律师起着担保代理作用。律师也作为遗嘱和合同谈判的仲裁人，还作为各种股票交易中买方和卖方之间的仲裁人。

银行也使用仲裁协议。Bob 能够用保付支票从 Alice 手中购买汽车：

（1）Bob 开一张支票并交到银行。

（2）在验明 Bob 的钱足以支付支票上的数目后，银行将保付支票交与 Bob。

（3）Alice 将车的所有权交给 Bob，Bob 将保付支票交给 Alice。

（4）Alice 兑现支票。

这个协议也是有效的，因为 Alice 相信银行的证明。Alice 相信银行保存有 Bob 的钱给她，不会将她的钱用于蚊虫大量滋生的国家的财政不稳的房地产业务。

公证人是另一种仲裁人，当 Bob 从 Alice 接收到已公证的文件时，他相信 Alice 签署的文件是她自己亲自签署的。如果有必要，公证人可出庭证实这个事实。

仲裁人的概念与人类社会一样悠久。总是有那么一些人——统治者、牧师等等，他们有公平处理事情的权威。在我们的社会中，仲裁者总是有一定社会地位和声望的人。而背叛公众的信任是很危险的事情。例如，视担保为儿戏的律师几乎肯定会被开除出律师界。现实世界里并不总是如此美好的，但它确是理想的。

这种思想可以转化到计算机世界中，但计算机仲裁者有下面几个问题：

（1）如果你知道对方是谁，并能见到他的面，就很容易找到和相信中立的第三方。互相怀疑的双方很可能也怀疑在网络别的什么地方并不露面的仲裁者。

（2）计算机网络必须负担仲裁者的费用。就像我们知道的律师费用，谁想负担那种网络费用呢？

（3）在任何仲裁协议中都有延迟的特性。

（4）仲裁者必须处理每一笔交易。任何一个协议在大范围执行时，仲裁者是潜在的瓶颈。增加仲裁者的数目能缓解这个问题，但费用将会增加。

（6）由于在网络中每人都必须相信仲裁者，对试图破坏网络的人来说，仲裁者便是一个易受攻击的弱点。

尽管如此，仲裁者仍扮演一个角色。在使用可信任的仲裁协议中，这个角色将由 Trent 来扮演。

裁决协议

由于雇用仲裁者代价高昂，仲裁协议可以分成两个低级的子协议。一个是非仲裁子协议，这个子协议是想要完成协议的各方每次都必须执行的；另一个是仲裁子协议，仅在例外的情况下执行的，即有争议的时候才执行，这种特殊的仲裁者叫做裁决人（参见图 2.1 中的 b）。

裁决人也是公正的和可信的第三方。他不像仲裁者，并不直接参与每一个协议。只有为了要确定协议是否被公平地执行，才将他请来。

法官是职业的裁决者。法官不像公证人，仅仅在有争议时才需要他出场，Alice 和 Bob 可以在没有法官的情况下订立合同。除非他们中有一个人把另一人拖到法院，否则法官决不会看到合同。

合同—签字协议可以归纳为下面形式：

非仲裁子协议（每次都执行）：

（1）Alice 和 Bob 谈判合同的条款。

（2）Alice 签署合同。

（3）Bob 签署合同。

裁决子协议（仅在有争议时执行）：

（1）Alice 和 Bob 出现在法官面前。

（2）Alice 提出她的证据。

（3）Bob 也提出他的证据。

（4）法官根据证据裁决。

裁决者和仲裁之间的不同是裁决者（在这本书中用的）并不总是必需的。如果有争议，法官被请来裁决。如果没有争议，就没有必要请法官。

已有计算机裁决协议。这些协议依赖于与协议有关的各方都是诚实的；如果有人怀疑欺骗时，一个中立的第三方能够根据存在的数据正文文本判断是否有人在欺骗。在好的裁决协议中，裁决者还能确定欺骗人的身份。裁决协议是为了发现欺骗，而不是为了阻止欺骗。发现欺骗是起了防止和阻碍欺骗的作用。

自动执行的协议

自动执行的协议是协议中最好的。协议本身就保证了公平性（参看图 2.1 中的 c）。不需要仲裁者来完成协议，也不需要裁决者来解决争端。协议的构成本身不可能发生如何争端。如果协议中的一方试图欺骗，其他各方马上就能发觉并且停止执行协议。无论欺骗方想通过欺骗来得到什么，他都不能如愿以偿。

最好，让每个协议都能自动执行。不幸的是，在所有情形下，没有一个是自动执行的协议。

对协议的攻击

密码攻击可以直接攻击协议中所用的密码算法、用来实现该算法和协议的密码技术、或者攻击协议本身。本书这节仅讨论协议。我们假设密码算法和密码技术是安全的，只关注对协议本身的攻击。

人们可以采用各种方法对协议进行攻击。与协议无关的人能够偷听协议的一部分或全部，这叫做**被动攻击**。因为攻击者不可能影响协议，所有他能做的事是观察协议并试图获取信息。这种攻击相当于在 1.1 节中讨论的唯密文攻击。由于被动攻击难于发现，因此协议应阻止被动攻击而不是发现这种攻击。在这种协议中，偷听者的角色将由 Eve 扮演。

另一种攻击可能改变协议以便对自己有利。他可能假装是其它一些人，在协议中引入新的信息，删掉原有的信息，用另外的信息代替原来的信息，重放旧的信息，破坏通信信道，或者改变存贮在计算机中的信息等。这些叫做**主动攻击**，因为他们具有主动的干预。这种形

式的攻击依赖于网络。

被动攻击试图获取协议中各方的信息。他们收集协议各方所传送的信息，并试图对它们进行密码分析。而主动攻击可能有更多的目的。攻击者可能对获取信息感兴趣，也可能降低系统性能，破坏已有的信息，或者获得非授权的资源存取。

主动攻击严重得多，特别是在那些各方都不必彼此信任的协议中。攻击者不一定都是入侵者，他可能是合法的系统用户，他们可能是系统管理员。甚至有很多主动攻击者，他们都在一起工作，每人都是合法的系统用户。这个恶意的主动攻击者的角色将由 Mallory 扮演。

攻击者也可能是与协议有关的各方中的一方。他可能在协议期间撒谎，或者根本不遵守协议，这类攻击者叫做**骗子**。**被动骗子**遵守协议，但试图获取协议外的其他信息。**主动骗子**在协议的执行中试图通过欺骗来破坏协议。

如果与协议有关的各方中的大多数是主动骗子，就很难保持协议的安全性。但合法用户发觉是否有主动欺骗却是可能的。当然，协议对被动欺骗来说应该是安全的。

2.2 使用对称密码术的通信

通信双方怎样安全地通信呢？当然，他们可以对通信加密。完整的协议比它更复杂，让我们来看看 Alice 发送加密的信息给 Bob 会发生什么情况吧：

（1）Alice 和 Bob 协商用同一密码系统。

（2）Alice 和 Bob 协商同一密钥。

（3）Alice 用加密算法和选取的密钥加密她的明文信息，得到了密文信息。

（4）Alice 发送密文信息给 Bob。

（5）Bob 用同样的算法和密钥解密密文，然后读它。

位于 Alice 和 Bob 之间的窃听者 Eve 监听这个协议，她能做什么呢？如果她听到的是在第（4）步中发送的密文，她必须设法分析密文，这是唯密文的被动攻击法；有很多算法能够阻止 Eve，使她不可能得到问题的解答。

尽管如此，但 Eve 却不笨，她也想窃听步骤（1）和步骤（2），这样她就知道了算法和密钥，她就和 Bob 知道的一样多。当步骤（4）中的信息通过信道传送过来时，她所做的全部工作就是解密密文信息。

好的密码系统的全部安全性只与密钥有关，和算法没有任何关系。这就是为什么密钥管理在密码学中如此重要的原因。有了对称算法，Alice 和 Bob 能够公开地实现步骤（1），但他们必须秘密地完成步骤（2）。在协议执行前、执行过程上和执行后，只要信息必须保持秘密，密钥就必须保持秘密，否则，信息就将不再秘密了。（公开密钥密码学用另一种方法解决了这个问题，将在 2.5 节中讨论。）

主动攻击者 Mallory 可能做其他一些事情，他可能企图破坏在（4）中使用的通信信道，使 Alice 和 Bob 根本不可能通信。他也可能截取 Alice 的信息并用他自己的信息替代它。如果他也知道密钥（通过截取（2）的通信或者破译密码系统），他可能加密自己的信息，然后发送给 Bob，用来代替截取的信息。Bob 没有办法知道接收到的信息不是来自 Alice。如果 Mallory 不知道密钥，他所产生的代替信息，被解密出来是无意义的，Bob 就会认为从 Alice

那里来的信息是网络或者是 Alice 有严重的问题。

Alice 又怎么样呢？她能做什么来破坏这个协议吗？她可以把密钥的副本给 Eve。现在 Eve 可以读 Bob 所发的信息，他还不知道 Eve 已经把他的话重印在《纽约时报》上。虽然问题很严重，但这并不是协议的问题。在协议过程的任何一点都不可能阻止 Alice 把明文的副本交给 Eve。当然 Bob 也可能做 Alice 所做的事。协议假定 Alice 和 Bob 互相信任。

总之，对称密码算法存在下面的问题：

（1）密钥必须秘密地分配，它们比任何加密的信息更有价值，因为知道了密钥意味着知道了所有信息。对于遍及世界的加密系统，这可能是令人沮丧的任务，需经常派信使将密钥传递到目的地。

（2）如果密钥被损害了（被偷窃，猜出来，被逼迫交出来，受贿等等），那么 Eve 就能用该密钥去解密所有传送的信息，也能够假装是几方中的一方，产生虚假信息去愚弄另一方。

（3）假设网络中每对用户使用不同的密钥，那么密钥总数随着用户数的增加迅速增多。N 个用户的网络需要 n（n-1）/2 个密钥。例如，10 个用户互相通信需要 45 个不同的密钥，100 个用户需要 4950 个不同的密钥，这个问题可以通过将用户数量控制在较小数目来减轻，但这并不总是可能的。

2.3 单向函数

单向函数的概念是公开密钥密码的中心。尽管它本身并不是一个协议，但对这本书中所讨论的大多数协议来说却是一个基本结构模块。

单向函数的概念是计算起来相对容易，但求逆却非常困难。也就是说，已知 x，我们很容易计算 f（x）。但已知 f（x），却难于计算出 x。在这里，“难”定义成：即使世界上所有的计算机都用来计算，从 f（x）计算出 x 也要花费数百万年的时间　。

打碎盘子就是一个很好的单向函数的例子。把盘子打碎成数千片碎片是很容易的事情，然而，要把所有这些碎片再拼成为一个完整的盘子，却是非常困难的事情。

这听起来很好，但事实上却不能证实它的真实性。如果严格地按数学定义，我们不能证明单向函数的存在性，同时也还没有实际的证据能够构造出单向函数。即使这样，还是有很多函数看起来和感觉像单向函数：我们能够有效地计算它们，且至今还不知道有什么办法能容易地求出它们的逆。例如，在有限域中 x^2 是很容易计算的，但计算 $x^{1/2}$ 却难得多。在这节的其余部分，假定单向函数存在， 11.2 节将更详细地讨论它。

那么，单向函数有什么好处呢？单向函数不能用作加密。用单向函数加密的信息是毫无用处的，无人能解开它（练习：在盘子上写上信息，然后砸成碎片，把这些碎片给你的朋友，要求你的朋友读这上面的信息，观察你的朋友对单向函数会有多么深刻的印象）。对公开密钥密码，我们还需要一些其它的东西（虽然有单向函数的密码学应用，参看 3.2 节）。

陷门单向函数是有一个秘密陷门的一类特殊单向函数。它在一个方向上易于计算而反方向却难于计算。但是，如果你知道那个秘密，你也能很容易在另一个方向计算这个函数。也就是说， 已知 x，易于计算 f（x），而已知 f（x），却难于计算 x。然而，有一些秘密信息 y，

一旦给出 f（x）和 y，就很容易计算 x。

拆开表是很好的单向陷门函数的例子。很容易把表拆成数百片小片，把这些小片组装成能够工作的表是非常困难的。然而，通过秘密信息（表的装配指令），就很容易把表还原。

2.4 单向 Hash 函数

单向 Hash 函数有很多名字：压缩函数、缩短函数、消息摘要、指纹、密码校验和、信息完整性检验（DIC）、操作检验码（MDC）。不管你怎么叫，它是现代密码学的中心。单向 Hash 函数是许多协议的另一个结构模块。

Hash 函数长期以来一直在计算机科学中使用，无论从数学上或别的角度看，Hash 函数就是把可变输入长度串（叫做预映射，Pre-image）转换成固定长度（经常更短）输出串（叫做 hash 值）的一种函数。简单的 Hash 函数就是对预映射的处理，并且返回由所有输入字节异或组成的一字节。

这儿的关键就是采集预映射的指纹：产生一个值，这个值能够指出候选选预映射是否与真实的预映射有相同的值。因为 Hash 函数是典型的多到一的函数，我们不能用它们来确定两个串一定相同，但我们可用它的来得到准确性的合理保证。

单向 Hash 函数是在一个方向上工作的 Hash 函数，从预映射的值很容易计算其 Hash 值，但要产生一个预映射的值使其 Hash 值等于一个特殊值却是很难的。前面提到的 Hash 函数不是单向函数：已知一个特殊的字节值，要产生一个字节串使它的异或结果等于那个值是很容易的事情。用单向 Hash 函数你不可能那样做。好的 hash 函数也是无冲突的：难于产生两个预映射的值，使他们的 hash 值相同。

Hash 函数是公开的，对处理过程不用保密。单向 hash 函数的安全性是它的单向性。无论怎么看，输出不依赖于输入。预映射的值的单个比特的改变，平均而言，将引起 hash 值中一半的比特改变。已知一个 hash 值，要找到预映射的值，使它的 hash 值等于已知的 hash 值在计算上是不可行的。

可把单向 Hash 函数看作是构成指纹文件的一种方法。如果你想验证某人持有一特定的文件（你同时也持有该文件），但你不想他将文件传给你，那么，就要求他将该文件的单向 Hash 值传送给你，如果他传送的 Hash 值是正确的，那么几乎可以肯定地说他持有那份文件。这是在金融交易中的特殊使用，你不希望在网络的一些地方把提取 100 美圆变成提取 1000 美圆。一般情况下，你应使用不带密钥的单向 Hash 函数，以便任何人都能验证 Hash 值。如果你只想接收者才能验证 Hash 值，那么就读下一节。

消息鉴别码

消息鉴别码（MAC）也叫数据鉴别码（DAC），它是带有秘密密钥的单向 hash 函数（见 18.14 节）。Hash 值是预映射的值和密钥的函数。这在理论上与 hash 函数一样，除非只有拥有密钥的某些人才能验证 hash 值。你可以用 hash 函数或分组加密算法产生 MAC；也有专用于 MAC 的算法。

2.5 使用公开密钥密码术的通信

对称算法可看成保险柜，密钥就是保险柜的号码组合。知道号码组合的人能够打开保险柜，放入文件，再关闭它。持有号码组合的其他人可以打开保险柜，取出文件来，而不知道保险柜号码组合的人就必须去摸索打开保险柜的方法。

1976 年，Whitfield Diffie 和 Martin Hellman 永远改变了密码学的范例（NSA 宣称早在 1966 年就有这种概念的知识，但没有提供证据）。他们提出了**公开密钥密码学**。他们使用两个不同的密钥：一个是公开的，另一个是秘密的。持有公钥的任何人都可加密信息，但却不能解密。只有持有私钥的人才能解密。就好像有人把密码保险柜变成一个信箱，把邮件投进邮箱相当于用公开密钥加密，任何人都可以做，只要打开窗口，把它投进去。取出邮件相当于用私钥解密。一般情况下，打开它是很难的，你需要焊接机和火把。然而，如果你拥有私钥（开信箱的钥匙），就很容易从邮箱中取出邮件。

数学上，这个过程是基于前面讨论过的单向陷门函数。加密是容易的，加密指令就是公开密钥，任何人都能加密信息。解密是困难的，它做得非常困难，以致于不知道这个秘密，即使用 Cray 计算机和几百万年的时间都不能解开这个信息。这个秘密或陷门就是私钥。持有这个秘密，解密就和加密一样容易。

下面描述 Alice 怎样使用公开密钥密码发送信息给 Bob：

（1）Alice 和 Bob 选用一个公开密钥密码系统。

（2）Bob 将他的公钥传送给 Alice。

（3）Alice 用 Bob 的公钥加密她的信息，然后传送给 Bob。

（4）Bob 用他的私钥解密 Alice 的信息。

注意公钥密码是怎样解决对称密码系统的密钥管理问题的。在对称密码系统中， Alice 和 Bob 不得不选取同一密钥。Alice 能够随机选取一个，但她不得不把选取的密钥传给 Bob。她可能事先交给 Bob，但那样做需要有先见之明。她也可以通过秘密信使把密钥送给 Bob，但那样做太费时间。采用公钥密码，就很容易了，不用事先安排，Alice 就能把信息安全地发送给 Bob。整个交换过程一直都在窃听的 Eve，有 Bob 的公钥和用公钥加密的信息，但却不能恢复 Bob 的私钥或者传送的信息。

更一般地说，网络中的用户约定一公钥密码系统，每一用户有自己的公钥和私钥，并且公钥在某些地方的数据库中都是公开的，现在这个协议就更容易了：

（1）Alice 从数据库中得到 Bob 的公钥。

（2）Alice 用 Bob 的公钥加密信息，然后送给 Bob。

（3）Bob 用自己的私钥解密 Alice 发送的信息。

第一个协议中，在 Alice 给 Bob 发送信息前，Bob 必须将他的公钥传送给 Alice，第二个协议更像传统的邮件方式，直到 Bob 想读他的信息时，他才与协议有牵连。

混合密码系统

在讨论把 DES 算法作为标准建议的同时，公开了第一个公开密钥算法。这导致了密码学团体中的政治党派之争。

在实际的世界中，公开密钥算法不会代替对称算法。公开密钥算法不用来加密消息，而用来加密密钥。这样做有两个理由：

1．公钥算法比对称算法慢，对称算法一般比公钥算法快一千倍。是的，计算机变得越来越快，在 15 年后计算机运行公开密钥密码算法的速度比得上现在计算机运行对称密码的速度。但是，带宽需求也在增加，总有比公开密钥密码处理更快的加密数据要求。

2．公开密钥密码系统对选择明文攻击是脆弱的。如果 C=E（P），当 P 是 N 个可能明文集中的一个明文，那么密码分析者只需要加密所有 N 个可能的明文，并能与 C 比较结果（记住，加密密钥是公开的）。用这种方法，他不可能恢复解密密钥，但他能够确定 P。

如果持有少量几个可能加了密的明文消息，那么采用选择明文攻击可能特别有效。例如，如果 P 是比 1 百万美圆少的某个美圆值，密码分析家尝试所有 1 百万个可能的美圆值（可能的加密解决了这个问题，见 23.15 节），即使 P 不很明确，这种攻击也是非常有效的。单是知道密文与某个特殊的明文不相符，就可能是有用的信息。对称密码系统不易受这种攻击，因为密码分析家不可能用未知的密钥来完成加密的尝试。

在大多数实际的实现中，公开密钥密码用来保安全和分发会话密钥。这些会话密钥用在对称算法中，对通信消息进行保密[879]。有时称这种系统为**混合密码系统**。

（1）Bob 将他的公开密钥发给 Alice。

（2）Alice 产生随机会话密钥 K，用 Bob 的公开密钥加密，并把加密的密钥 E_B（K）送给 Bob。

（3）Bob 用他的私钥解密 Alice 的消息，恢复出会话密钥。

$$D_B（E_B（K））= K$$

（4）他们两人用同一会话密钥对他们的通信信息进行加密。

把公开密钥密码用于密钥分配解决了很重要的密钥管理问题。对对称密码而言，数据加密密钥直到使用时才起作用。如果 Eve 得到了密钥，那么她就能够解密用这个密钥加密的消息。在前面的协议中，当需要对通信加密时，才产生会话密钥，不再需要时就销毁，这极大地减少了会话密钥遭到损害的风险。当然，私钥面对泄露是脆弱的，但风险较小，因为只有每次对通信的会话密钥加密时才用它。这在 3.1 节进一步讨论。

Merkle 的难题

Ralph Merkle 发明了第一个公开密钥密码的设计。1974 年他在 Berkeley 的加利福尼亚大学注册了由 Lance Hoffman 教的计算机安全课程。在这个学期初，他的学期论文的题目是处理“不安全的信道上的安全通信”的问题[1064]。Hoffman 不理解 Merkle 的建议，最终 Merkle 放弃了这门课程。尽管连遭失败让人们难以理解其结果，但他仍孜孜不倦地在这个问题上工作着。

Merkle 的技术基于：发送者和接收者解决难题(Puzzles)比窃听者更容易。下面谈谈 Alice 怎样不用首先和 Bob 交换密钥就能把加密消息发给 Bob：

（1）Bob 产生 2^{20} 或大约 1 百万个这种形式的消息：“这是难题数 x，这是秘密密钥数 y”，其中 x 是随机数，y 是随机的秘密密钥。每个消息的 x 和 y 都是不相同的。

采用对称算法，他用不同的 20 比特密钥对每个消息加密，并都发给 Alice。

（2）Alice 随机选择一个消息，通过穷举攻击恢复明文。这个工作量是很大的，但并不是不可能的。

（3）Alice 用她恢复的密钥和有些对称算法加密秘密消息，并把它和 x 一起发给 Bob。

（4）Bob 知道他用哪个秘密密钥 y 对消息 x 加密的，这样他就能解密消息。

Eve 能够破译这个系统，但是她必须做比 Alice 和 Bob 多得多的工作。为了恢复第（3）步的消息，她必须完成对 Bob 在第（1）步中的所有 2^{20} 消息的穷举攻击。这个攻击的复杂性是 2^{40}。X 的值不会对 Eve 有什么帮助。他们在第（1）步中是随机指定的。一般情况下，Eve 花费的努力大约是 Alice 花费的努力的平方。

按照密码的标准，n 到 n^2 没有什么优势，但在某些情况下，这可能足够（复杂）了。如果 Alice 和 Bob 每秒可试 1 万个，这将要花他们每个人 1 分钟去完成他们的步骤，再花另外 1 分钟在 1.544Mb/s 链路上完成从 Alice 到 Bob 的通信难题。如果 Eve 有同样的计算设备，破译这个系统将花费她大约 1 年的时间，其它算法甚至更难破译。

2.6 数字签名

在文件上手写签名长期以来被用作作者身份的证明，或至少同意文件的内容。签名为什么会如此引人注目呢？[1392]

（1）签名是可信的。签名使文件的接收者相信签名者是慎重地在文件上签字的。

（2）签名不可伪造。签名证明是签字者而不是其他人慎重地在文件上签字。

（3）签名不可重用。签名是文件的一部分，不法之徒不可能将签名移到不同的文件上。

（4）签名的文件是不可改变的。在文件签名后，文件不能改变。

（5）签名是不可抵赖的。签名和文件是物理的东西。签名者事后不能声称他没有签过名。

在现实生活中，关于签名的这些陈述没有一个是完全真实的。签名能够被伪造，签名能够从文章中盗用移到另一篇文章中，文件在签名后能够被改变。然而，我们之所以愿意与这些问题纠缠在一起，因为欺骗是困难的，并且还要冒被发现的危险。

我们或许愿在计算机上做这种事情，但还存在一些问题。首先计算机文件易于复制。即使某人的签名难以伪造（例如，手写签名的图形），但是从一个文件到另一个文件剪裁和粘贴有效的签名都是很容易的。这种签名并没有什么意义；其次文件在签名后也易于修改，并且不会留下任何修改的痕迹。

使用对称密码系统和仲裁者的文件签名

Alice 想对数字消息签名，并送给 Bob。在 Trent 和对称密码系统的帮助下，她能做到。

Trent 是一个有权的、值得依赖的仲裁者。他能同时与 Alice 和 Bob（也可以是其他想对数据文件签名的任何人）通信。他和 Alice 共享秘密密钥 K_A，和 Bob 共享另一个不同的秘密密钥 K_B。这些密钥在协议开始前就早已建好，并且为了多次签名可多次重复使用。

（1）Alice 用 K_A 加密她准备发送给 Bob 的信息，并把它传送给 Trent。

（2）Trent 用 K_A 解密信息。

（3）Trent 把这个解密信息和他收到 Alice 信息的声明，一起用 K_B 加密。

（4）Trent 把加密的信息包传给 Bob。

（5）Bob 用 K_B 解密信息包，他就能读 Alice 所发的信息和 Trent 的证书，证明信息来自 Alice。

Trent 怎么知道信息是从 Alice 而不是从其他人冒名顶替者那里来的呢？从信息的加密推断出来。由于只有他和 Alice 共享他们两人的秘密密钥，所以只有 Alice 能用这个密钥加密信息。

这和文件签名一样好吗？让我们看看我们需要的特点：

(1) 这个签名是可信的，Trent 是可信的仲裁者，并且知道消息是从 Alice 那里来的，Trent 的证书对 Bob 起着证明的作用。

(2) 这个签名是不可伪造的。只有 Alice（和 Trent，但每个人都相信他）知道 K_A，因此只有 Alice 才能把用 K_A 加密的信息传给 Trent。如果有人冒充 Alice，Trent 在第（2）步马上就会察觉，并且不会去证明它的可靠性。

（3）这个签名是不能重新使用的。如果 Bob 想把 Trent 的证书附到另一个信息上，Alice 可能就会大叫受骗了。仲裁者（可能是 Trent 或者可存取同一信息的完全不同的仲裁者）就会要求 Bob 同时提供信息和 Alice 加密后的信息，然后仲裁者就用 K_A 加密信息，他 马上就会发现它与 Bob 提供的加密信息不相同。很显然，Bob 由于不知道 K_A，他不可能提供加密信息使它与用 K_A 加密的信息相符。

（4）签名文件是不能改变的。Bob 想在接收后改变文件，Trent 就可用刚才描述的同样办法证明 Bob 的愚蠢行为。

（5）签名是不能抵赖的，即使 Alice 以后声称她没有发信息给 Bob，Trent 的证书会说明不是这样。记住：Trent 是每个人都信任的，他说的都是正确的。

如果 Bob 想把 Alice 签名的文件给 Carol 阅读，他不能把自己的秘密密钥交给她，他还得通过 Trent：

（1）Bob 把信息和 Trent 关于信息是来自 Alice 的声明用 K_B 加密，然后送回给 Trent。

（2）Trent 用 K_B 解密信息包。

（3）Trent 检查他的数据库，并确认原始信息是从 Alice 那里来的。

（4）Trent 用他和 Carol 共享的密钥 K_C 重新加密信息包，把它送给 Carol。

（5）Carol 用 K_C 解密信息包，她就能阅读信息和 Trent 证实信息来自 Alice 的证书。

这些协议是可行的，但对 Trent 来说是非常耗时的。他不得不整天加密、解密信息，在彼此想发送签名文件的每一对人之间充当中间人。他必须备有数据库信息（虽然可以通过把发送者加密的信息的拷贝发送给接收者来避免）。在任何通信系统中，即使他是毫无思想的软件程序，他都是通信瓶颈。

更困难的是产生和保持像 Trent 那样的网络用户都信任的人。Trent 必须是完善无缺的，即使他在 100 万次签名中只犯了一个错误，也将不会有人再信任他。Trent 必须是完全安全的，如果他的秘密密钥数据库泄漏了，或有人能修改他的程序代码，所有人的签名可能是完

全无用的。一些声称是数年前签名的假文件便可能出现，这将引起混乱，政府可能倒台，混乱状态可能盛行。理论上这种协议或许是可行的，但实际上不能很好运转。

数字签名树

Ralph Merkle 提出了一个基于秘密密钥密码的数字签名方案，该方案利用树型结构产生无限多的一次签名[1067，1068]。这个方案的基本思想是在某些公开文档中放入树的根文件，从而鉴别它。根节点对一个信息签名，并鉴别树中的子节点，这些节点的每一个都对信息签名，并对它的子节点鉴别，一直延续下去。

使用公钥密码对文件签名

有几种公钥算法能用作数字签名。在一些算法中，例如 RSA（见 119.3 节），公钥或者私钥都可用作加密。用你的私钥加密文件，你就拥有安全的数字签名。在其它情况下，如 DSA（见 20.1 节），算法便区分开来了——数字签名算法不能用于加密。这种思想首先由 Diffie 和 Hellman[496]提出，并且在其他文章中得到进一步的发展[1282，1382，1024，1283，426]。 文献[1099]对这个领域作了很好的综述。

基本协议是简单的：

（1）Alice 用她的私钥对文件加密，从而对文件签名。

（2）Alice 将签名的文件传给 Bob。

（3）Bob 用 Alice 的公钥解密文件，从而验证签名。

这个协议比以前的算法更好。不需要 Trent 去签名和验证。他只需要证明 Alice 的公钥的确是她的）。甚至协议的双方不需要 Trent 来解决争端；如果 Bob 不能完成第（3）步，那么他知道签名是无效的。

这个协议也满足我们期待的特征：

（1）签名是可信的。当 Bob 用 Alice 的公钥验证信息时，他知道是由 Alice 签名的。

（2）签名是不可伪造的。只有 Alice 知道她的私钥。

（3）签名是不可重用的。签名是文件的函数，并且不可能转换成另外的文件。

（4）被签名的文件是不可改变的。如果文件有任何改变，文件就不可能用 Alice 的公钥验证。

（5）签名是不可抵赖的。Bob 不用 Alice 的帮助就能验证 Alice 的签名。

文件签名和时间标记

实际上，Bob 在某些情况下可以欺骗 Alice。他可能把签名和文件一起重用。如果 Alice 在合同上签名，这种重用不会有什么问题（同一个合同的另一个拷贝是什么？）。但如果 Alice 在一张数字支票上签名，那样做就令人兴奋了。

假若 Alice 交给 Bob＄100 的签名数字支票，Bob 把支票拿到银行去验证签名，然后把钱从 Alice 的帐户上转到自己的帐上。Bob 是一个无耻之徒，他保存了数字支票的副本。过了一星期，他又把数字支票拿到银行（或可能是另一个银行），银行验证数字支票并把钱转到他的帐上。只要 Alice 不去对支票本清帐，Bob 就可以一直干下去。

因此，数字签名经常包括时间标记。对日期和时间的签名附在信息中，并跟信息中的其他部分一起签名。银行将时间标记存贮在数据库中。现在，当 Bob 第二次想支取 Alice 的支

票时，银行就要检查时间标记是否和数据库中的一样。由于银行已经从 Alice 的支票上支付了这一时间标记的支票，于是就叫警察。这样一来 Bob 就要在 Leavenworth 监狱中度过 15 个春秋去研读密码协议的书籍了。

用公钥密码和单向 Hash 函数对文件签名

在实际的实现过程中，采用公钥密码算法对长文件签名效率太低。为了节约时间，数字签名协议经常和单向 Hash 函数一起使用。Alice 并不对整个文件签名，只对文件的 Hash 值签名。在这个协议中，单向 Hash 函数和数字签名算法是事先就协商好了的。

（1）Alice 产生文件的单向 Hash 值。

（2）Alice 用她的私钥对 Hash 加密，凭此表示对文件签名。

（3）Alice 将文件和 Hash 签名送给 Bob。

（4）Bob 用 Alice 发送的文件产生文件的单向 Hash 值，然后用数字签名算法对 hash 值运算，同时用 Alice 的公钥对签名的 Hash 解密。如果签名的 hash 值与自己产生的 Hash 值匹配，签名就是有效的。

计算速度大大地提高了，并且两个不同的文件有相同的 160 比特 Hash 值的概率为 $1/2^{160}$。因此，使用 Hash 函数的签名和文件签名一样安全。如果使用非单向 Hash 函数，可能很容易产生多个文件使它们的 hash 值相同，这样对一特定的文件签名就可复制用于对大量的文件签名。

这个协议还有其它好处。首先，签名和文件可以分开保存。其次，接收者对文件和签名的存贮量要求大大降低了。档案系统可用这类协议来验证文件的存在而不需保存它们的内容。中央数据库只存贮各个文件的 Hash 值，根本不需要看文件。用户将文件的 Hash 值传给数据库，然后数据库对提交的文件加上时间标记并保存。如果以后有人对某文件的存在发生争执，数据库可通过找到文件的 Hash 值来解决争端。这里可能牵连到大量的隐秘：Alice 可能有某文件的版权，但仍保持文件的秘密。只有当她想证明她的版权时，她才不得不把文件公开（参看 4.1 节）.

算法和术语

有许多数字签名算法，它们都是公钥算法，用秘密信息对文件签名，用公开信息去验证。有时签名过程也叫“**用私钥加密**”，验证过程也叫“**用公钥解密**”，这会使人误解，并且仅仅只对 RSA 这个算法而言才是这样，而不同的算法有不同的实现，例如有时使用单向 Hash 函数和时间标记对签名和验证过程进行处理要增加额外的步骤。许多算法可用作数字签名，但不能用作加密。

一般地，提到签名和验证过程通常不包括任何算法的细节。用私钥 K 签名信息表示为：

$$S_K（M）.$$

用相应的公钥验证信息表示为：

$$V_K（M）.$$

在签名时，附在文件上的比特串叫**数字签名**（在上面的例子中，用私钥对文件的单向 Hash 值加密）或者就叫**签名**。信息的接收者用以确认发送者的身份和信息的完整性的整个

协议叫做鉴别。这些协议进一步的细节将在 3.2 节中讨论。

多重签名

Alice 和 Bob 怎么对同一数字文件签名呢？不用单向 Hash 函数，有两种选择。第一种选择是 Alice 和 Bob 分别对文件的副本签名，结果签名的信息是原文的两倍。第二种就是 Alice 首先签名，然后 Bob 对 Alice 的签名再进行签名，这是可行的，但是在不验证 Bob 的签名的情况下就验证 Alice 的签名是不可能的。

采用单向 Hash 函数，多重签名是容易的：

（1）Alice 对文件的 hash 签名。

（2）Bob 对文件的 hash 签名。

（3）Bob 将他的签名交给 Alice。

（4）Alice 把文件、她的签名和 Bob 的签名发给 Carol。

（5）Carol 验证 Alice 和 Bob 的签名。

Alice 和 Bob 能同时或顺序地完成（1）和（2），在（5）Carol 可以只验证其中一人的签名而不用验证另一人的签名。

抗抵赖和数字签名

Alice 有可能用数字签名作欺骗，并且无人能阻止她。她可能对文件签名，然后声称并没有那样做。首先，她按常规对文件签名，然后她以匿名的形式发布她的私钥，故意把私钥丢失在公共场所，或者只要假装做上面两者中的一个。这样，发现该私钥的任何人都可装成是 Alice 对文件签名，于是 Alice 就声明她的签名受到侵害，其他人正在假装她签名云云。她否认对文件的签名和任何其他的用她的私钥签名的文件，这叫做**抵赖**。

采用时间标记可以限制这种欺骗的作用，但 Alice 总可以声称她的密钥在较早的时候就丢失了。如果 Alice 把事情做得好，她可以对文件签名，然后成功地声称并没有对文件签名。这就是为什么我们经常听到把私钥隐藏在防拆的模块中的原因，这样，Alice 就不可能接近和乱用私钥了。

虽然没有办法阻止这种可能的乱用，但可以采取措施保证旧的签名不会失效的（例如，Alice 可能有意丢失她的密钥，以便不用对昨天从 Bob 那里买的旧车付帐，在这个过程中，Alice 使她的银行帐户无效）。数字签名文件的接收者持有签名的时间标记就能解决这个问题[453]。

一般的协议在[28]中给出：

（1）Alice 对信息签名。

（2）Alice 产生一个报头，报头中包含有些鉴别信息。她把报头和签名的信息连接起来，对连接的信息签名，然后把签名的信息发给 Trent。

（3）Trent 验证外面的签名，并确认鉴别信息。他在 Alice 签名信息中增加一个时间标记和鉴别信息。然后对所有的信息签名，并把它发给 Bob 和 Alice。

（4）Bob 验证 Trent 的签名、鉴别信息和 Alice 的签名。

（5）Alice 验证 Trent 发给 Bob 的信息。如果她没有发起这个信息，她很快就会大喊大叫了。

另一个方案是在事后有劳Trent[209]。Bob在接收到签名信息后，他可能把副本发给Trent验证，Trent 能够证实 Alice 的签名的有效性。

数字签名的应用

数字签名最早的建议应用之一是用来对禁止核试验条约的验证[1454，1467]。美国和前苏联（还有人记得苏联吗？）互相允许把地震测试仪放入另一个国家中，以便对核试验进行监控。问题是每个国家需要确信东道国没有篡改从监控国家的地震仪传来的数据。同时，东道主国家需要确信监测器只发送规定的需要监测的信息。

传统的鉴别技术能解决第一个问题，但只有数字签名能同时解决两个问题。东道国一方只能读，但不能窜改从地震测试仪来的数据；而监督国确信数据没有被窜改。

2.7 带加密的数字签名

通过把公钥密码和数字签名结合起来，我们能够产生一个协议，可把数字签名的真实性和加密的安全性合起来。想象你妈妈写的一封信：签名提供了原作者的证明，而信封提供了秘密性 。

（1）Ailce 用她的私钥对信息签名。

$$S_A（M）.$$

（2）Alice 用 Bob 的公钥对签名的信息加密，然后送给 Bob。

$$E_B（S_A（M））.$$

（3）Bob 用他的私钥解密。

$$D_B（E_B（S_A（M）））=S_A（M）.$$

（4）Bob 用 Alice 的公钥验证并且恢复出信息。

$$V_A（S_A（M））=M.$$

加密前签名是很自然的。当 Alice 写一封信，她在信中签名，然后把信装入信封中。如果她把没签名的信放入信封，然后在信封上签名，那么 Bob 可能会担心是否这封信被替换了。如果 Bob 把 Alice 的信和信封给 Carol 看，Carol 可能因信没装对信封而控告 Bob 说谎。

在电子通信中也是这样，加密前签名是一种谨慎的习惯做法[48]。这样做不仅是更安全（敌人不可能从加密信息中把签名移走，然后加上他自己的签名）。而且还有法律的考虑：当他附加他的签名时，如果签名者不能见到被签名的文本，那么签名没有多少法律强制作用[1312]。有一些针对 RSA 签名技术的密码分析攻击（见 19.3 节）。

Alice 没有理由必须把同一个公钥/私钥密钥对用作加密和签名。她可以有两个密钥对：一个用作加密，另一个用作解密。分开使用有它的好处：她能够把她的加密密钥交给警察而不泄露她的签名，一个密钥被托管（见 4.13）而不会影响到其他密钥，并且密钥能够有不同的长度，能够在不同的时间终止使用。

当然，这个协议应该用时间标记来阻止信息的重复使用。时间标记也能阻止其他潜在的危险，例如下面描述的这种情形。

作为收据的重发信息

我们来考虑这个协议附带确认信息的实现情形：每当 Bob 接收到信息，他再把它传送回发方作为接收确认。

（1）Alice 用她的私钥对信息签名，再用 Bob 的公钥加密，然后传给 Bob。

$$E_B（S_A（M））.$$

（2）Bob 用他的私钥对信息解密，并用 Alice 的公钥验证签名，由此验证确是 Alice 对信息签名，并恢复出信息。

$$V_A（D_B（E_B（S_A（M））））=M.$$

（3）Bob 用他的私钥对信息签名，用 Alice 的公钥加密，再把它送回给 Alice。

$$E_A（S_A（M））.$$

（4）Alice 用她的私钥对信息解密，并用 Bob 的公钥对验证 Bob 的签名。如果接收的信息与她传给 Bob 的相同，她就知道 Bob 准确地接收到她所发送的信息。

如果同一算法既用作加密又用作数字签名，就有可能受到攻击[305]。在这些情况中，数字签名操作是加密操作的逆过程：$V_X=E_X$，并且 $S_X=D_X$。

假设 Mallory 是持有自己公钥和私钥的系统合法用户。让我们看看他怎么读 Bob 的邮件。首先他将 Alice 在（1）中送给 Bob 的信息记录下来，在以后的某个时间，他将那个信息送给 Bob，声称信息是从 Mallory 处来的。Bob 认为是从 Mallory 来的合法信息，于是就用私钥解密，然后通过用 Mallory 的公钥解密来验证 Mallory 的签名，那么得到的信息纯粹是乱七八糟的信息：

$$E_M（D_B（E_B（D_A（M））））=E_M（D_A（M））.$$

即使这样，Bob 继续执行协议，并且将收据发给 Mallory。

$$E_M（D_B（E_M（D_A（M））））.$$

现在 Mallory 所要做的就是用他的私钥对信息解密，用 Bob 的公钥加密，再用 Mallory 自己的私钥解密，并用 Alice 的公钥加密。哇！Mallory 就获得了所要的信息 M。

认为 Bob 会自动回送给 Mallory 一个收据不是不合理的。例如：这个协议可能嵌入在通信软件中，并且在接收后自动发送收据。收到乱七八糟的东西也送出确认收据会导致不安全性。如果在发送收据前仔细地检查信息是否能理解，就能避免这个安全问题。

还有一种更强的攻击，就是允许 Mallory 送给 Bob 一个与窃听的信息不同的信息。不要对从其他人那里来的信息随便签名并将结果交给其他人。

阻止重发攻击

由于加密运算与签名/验证运算相同，同时解密运算又与签名运算相同，因此，上述攻击才凑效。安全的协议应该是加密和数字签名操作稍微不同，每次操作使用不同的密钥能做到这一点，每次操作使用不同的算法也能做到；采用时间标记也能做到，它使得输入的信息和输出信息不同。用单向 Hash 函数的数字签名也能解决这个问题（参看 2.6 节）；；

一般说来，下面这个协议是非常安全的，它使用的是公开密钥算法：

（1）Alice 对消息签名。

（2）Alice 用 Bob 的公钥对消息和签名加密（采用和签名算法不同的加密算法），然后

将它传送给 Bob。

（3）Bob 用他的私钥对消息解密。

（4）Bob 验证 Alice 的签名。

对公钥密码的攻击

在所有公钥密码协议中，回避了 Alice 怎么得到 Bob 的公钥这件事。3.1 节将更详细地讨论，但在这儿值得提及。

得到某人的公钥的最容易的方法是从某个地方的安全数据库中得到。这个数据库必须是公开的，以便任何人都可得到其他人的公钥。数字库也必须阻止 Trent 以外的其他人写入数据；否则 Mallory 可能用他选取的任意一个公钥代替 Bob 的，他那样做后，Bob 就不能解读发给他的信息，但 Mallory 却能读。

即使公钥存贮在安全数据库中，Mallory 仍能在传送期间用另外的公钥来代替。为了防止这个问题，Trent 可用他的私钥对每一公钥进行签名。当用这种方式时，Trent 常被称为**密钥鉴证机关**或**密钥分配中心（KDC）**。在实际的实现中，KDC 对由用户名、公钥和其他用户的重要信息组成的一组信息进行签名。被签名的这组信息存贮在 KDC 数据库中。当 Alice 得到 Bob 的密钥时，她验证 KDC 的签名以使自己确信密钥的有效性。

在最后的分析中，对 Mallory 来说并不是不可能，只不过更困难了：Alice 仍将 KDC 的公钥存贮在某个地方，Mallory 只得用自己的公钥代替那个密钥，破坏数据库，然后用自己的密钥代替有效密钥（好像他就是 KDC 一样，用自己的私钥对所有密钥签名），再进行运算，如果 Mallory 要制造更多的麻烦，他甚至连文件签名都可以伪造。密钥交换将在 3.1 节详细讨论

2.8 随机和伪随机序列的产生

为什么在一本关于密码学的书中还不厌其烦地谈论随机数产生呢？随机数产生器已嵌入在大多数编译器中了，产生随机数仅仅是函数调用而已。为什么不用编译器的那种呢？ 不幸的是，那些随机数产生器对密码来说几乎肯定是不安全的，甚至可能不是很随机的。它们中的大多数都是非常差的随机数。

随机序列产生器并不是随机的，因为它们不必要是完全随机的。像计算机游戏，大多数简单应用中只需几个随机数，几乎无人注意到它们，然而密码学对随机数产生器的性质是极其敏感的。用粗劣的随机数产生器，你会得到十分奇妙的相关和奇怪的结果[1231，1238]。如果安全性依赖于你的随机数发生器，那么你得到的最后的东西就是这种奇妙的相关和结果。

随机数产生器不能产生随机序列，它甚至可能产生不了乍看起来像随机序列的数。当然，在计算机上不可能产生真正的随机数。Donald Knuth 引用 John Von Neumann 的原话：“任何人考虑用数学的方法产生随机数肯定是不合情理的”。计算机确是怪兽：数据从一端进入，在内部经过完全可预测的操作，从另一端出来的却是不同的数据；把同一数据在不相干情况下输入进去，两次出来的数据是相同的；把同样的数据送入相同的两个计算机，它们的运算

结果是相同的。计算机只能是一个有限的状态数（一个大数，但无论如何是有限的），并且输出状态总是过去的输入和计算机当前状态的确定的函数。这就是说计算机中的随机序列产生器（至少，在有限状态机中）是周期性的，周期性的任何东西都是可预测的。如果是可预测的，那么它就不可能是随机的。真正的随机序列产生器需要随机输入，计算机不可能提供这种随机输入。

伪随机序列

最好的计算机能产生的是伪随机序列产生器，什么意思呢？许多人试图形式化地定义它，但我不赞成，随机数序列是看起来是随机的序列，序列的周期应足够长，使得实际应用中相当长的有限序列都不是周期性的。就是说如果你需要十亿个随机比特，就不要选择仅在一万六千比特后就重复的序列产生器。这些相对短的非周期性的子序列应尽可能和随机序列没有多少区别。例如：它们应该有大约相同数目的 0 和 1，长度为 1 的游程大约占一半（同一比特序列），长度为 2 的游程占 1/4，长度为 3 的游程占 1/8 等等。它们应该是不可压缩的，0 和 1 游程的分布应该是相同的[643，863，99，1357]。这些性质是根据实验测得的，然后用 chi-square 检验与统计期望值比较。

我们的意思是，如果一序列产生器是伪随机的，它应有下面的性质：

（1） 看起来是随机的，这表明它通过了我们所能找到的所有随机性统计检验（开始[863]中的检验）。

人们在计算机上已经做了许多努力来产生好的伪随机序列，学术文献中有很多讨论伪随机序列产生器和各种随机性检验的，但所有这些产生器是周期的（都不可能例外）。然而周期大于 2^{256} 比特的随机数序列，能够大量得到应用。

这里的关键问题还是那些奇妙的相关性和奇怪的结果。如果你以某种方式使用它们，则每个随机数序列产生器都将产生这些结果。这正是为什么密码分析者用它来对系统进行攻击的原因。

密码学意义上安全的伪随机序列

密码的应用比其他大多数应用对伪随机序列的要求更严格。密码学的随机性并不仅仅意味着统计的随机性，虽然它也是其中的一部分。密码学意义上安全的伪随机数，还必须具有下面的性质：

（2）它是不可预测的。即使给出产生序列的算法或硬件和所有以前产生的比特流的全部知识，也不可能通过计算来预测下一个随机比特应是什么。

密码学意义上安全的伪随机序列应该是不可压缩的……除非你知道密钥。密钥通常是用来设置产生器的初始状态的种子。

像任何密码算法一样，密码学意义上安全的伪随机序列产生器也会受到攻击，就好像加密算法有可能被破译一样，破译密码学意义上安全的伪随机序列产生器也是可能的。密码学讲的都是关于如何使产生器抵抗攻击。

真正的随机序列

现在我们走进哲学家的领域，真有随机数这样的东西吗？随机序列是什么？你怎么知道序列是随机的？“101110100”比“101010101”更随机吗？量子力学告诉我们，在现实世界中有真正的随机性。但是在计算机芯片和有限状态机的确定世界中，这种随机性还能保持吗？

暂且不说哲学。从我们的观点来说，如果一个随机序列产生器具有下面的第三条性质，它就是**真正随机**的：

（3）它不能可靠地重复产生。如果你用完全同样的输入对序列产生器操作两次（至少与人所能做到的最精确的一样），你将得到两个不相关的随机序列。

满足这三条性质的产生器的输出对于一次一密乱码本、密钥的产生和任何其它需要真正随机数序列产生器的密码应用来说都是足够好的。难点在于确定真正的随机数。如果我用一个给定的密钥，用 DES 算法重复地对一个字符串加密，我将得到一个好的，看起来随机的输出。但你仍不可能知道它是否真的随机数，除非你租用美国国家安全局的 DES 破译专家。

第三章　基本协议

3.1　密钥交换

通常的密码技术是用单独的密钥对每一次单独的会话加密，这个密钥称为会话密钥，因为它只在一次特殊的通信中使用。正如 8.5 节讨论的一样，会话密钥只用于通信期间。这个会话密钥怎么到达会话者的手中可能是很复杂的事情。

对称密码的密钥交换

这个协议假设 Alice 和 Bob（网络上的用户）每人和密钥分配中心（KDC）共享一个秘密密钥[1260]，我们的协议中的 Trent 就是 KDC。在协议开始执行前，这些密钥必须在适当的位置（协议忽略了怎么分配这些秘密密钥这个非常实际的问题，只是假设它们在适当的位置，并且 Mallory 不知道它们是什么）。

（1）Alice 呼叫 Trent，并请求一个与 Bob 通信的会话密钥。

（2）Trent 产生一随机会话密钥，并对它的两个副本加密：一个用 Alice 的密钥，另一个用 Bob 的密钥加密。Trent 发送这两个副本给 Alice。

（3）Alice 对她的会话密钥的副本解密。

（4）Alice 将 Bob 的会话密钥副本送给 Bob。

（5）Bob 对他的会话密钥的副本解密。

（6）Alice 和 Bob 用这个会话密钥安全地通信。

这个协议依赖于 Trent 的绝对安全性。Trent 更可能是可信的计算机程序，而不是可信的个人。如果 Mallory 破坏了 Trent，整个网络都会遭受损害。他有 Trent 与每个用户共享的所有秘密密钥；他可以读所有过去和将来的通信业务。他所做的事情就是对通信线路进行搭线窃听，并监视加密的报文业务。

这个系统的另外一个问题是 Trent 可能会成为瓶颈。他必须参与每一次密钥交换，如果 Trent 失败了，这个系统就会被破坏。

公开密钥密码的密钥交换

基础的混合密码体制曾在 2.5 节中讨论过。Alice 和 Bob 使用公开密钥密码协商会话密钥，并用协商的会话密钥加密数据。在一些实际的实现中，Alice 和 Bob 签了名的公开密钥可在数据库中获得。这使得密钥交换协议更容易，即使 Bob 从来没有听说过 Alice，Alice 也能够把信息安全地发送给 Bob。

（1）Alice 从 KDC 得到 Bob 的公开密钥。

（2）Alice 产生随机会话密钥，用 Bob 的公开密钥加密它，然后将它传给 Bob。

（3）Bob 用他的私钥解密 Alice 的信息。

（4）他们两人用同一会话密钥对他们的通信进行加密。

中间人攻击

Eve 除了试图破译公开密钥算法或者尝试对密文作唯密文攻击之外，没有更好的办法。Mallory 比 Eve 更有能力，他不仅能监听 Alice 和 Bob 之间的信息、还能修改信息、删除信息、并能产生全新的信息。当 Mallory 同 Alice 谈话时，他能模仿 Bob，他也能模仿 Alice 同 Bob 谈话。下面要谈的是这种攻击怎样生效的：

（1）Alice 将她的公开密钥传送给 Bob。Mallory 截取了这个密钥并将自己的公开密钥传送给 Bob。

（2）Bob 将他的公开密钥传送给 Alice。Mallory 截取这个密钥，并将自己的公开密钥传送给 Alice。

（3）当 Alice 将用“Bob”的公开密钥加了密的信息传送给 Bob 时，Mallory 截取它。由于信息实际上是用 Mallory 的公开密钥加密的，他就用自己的私钥解密。再用 Bob 的公开密钥对信息重新加密，并将它传送给 Bob。

（4）当 Bob 将用“Alice”的公开密钥加密的信息传送给 Alice 时，Mallory 截取它。由于信息实际上是用他自己的公开密钥加密的，他用他的私钥解密信息，再用 Alice 的公开密钥重新加密，并将它传送给 Alice。

即使 Alice 和 Bob 的公开密钥存储在数据库中，这种攻击也是可行的。Mallory 能够截取 Alice 的数据库查询，并用自己的公开密钥代替 Bob 的公开密钥，对 Bob 他也能做同样的事情，用自己的公开密钥代替 Alice 的公开密钥。他也能秘密地侵入数据库，用他自己的密钥代替 Alice 和 Bob 的密钥。接下来他就简单地等着 Alice 和 Bob 互相谈话，然后截取和修改信息，他成功了！

“中间人攻击”是可行的，因为 Alice 和 Bob 无法验证他们正在作的互相交谈。假设 Mallory 没有导致任何值得注意的网络延迟，他们两人就没有办法知道有人正在他们中间阅读他们自认为是秘密的消息。

联锁协议

由 Ron Rivest 和 Adi Shamir[1327]发明的**联锁协议**是阻止“中间人攻击”的好办法。下面是这个协议怎么工作的：

（1）Alice 将她的公开密钥传送给 Bob。

（2）Bob 将他的公开密钥传送给 Alice。

（3）Alice 用 Bob 的公开密钥加密她的报文，并将加密报文的一半传送给 Bob。

（4）Bob 用 Alice 的公开密钥加密他的报文，并将加密报文的一半传送给 Alice。

（5）Alice 将加了密的另一半报文传送给 Bob。

（6）Bob 将 Alice 的两半报文合在一起，并用他的私钥解密；Bob 将他加了密的另一半报文传送给 Alice。

（7）Alice 将 Bob 两半报文合在一起，并用她的私钥解密。

这里重要的一点是：只有报文的一半，没有另一半，报文是毫无用处的。Bob 只有到步骤（6）步才能读 Alice 的报文，Alice 只有到步骤（7）步才能读 Bob 的报文。有很多办法

实现它：

（1）如果采用分组加密算法，每一分组的一半（例如，每隔一比特）能在每半个报文中发送。

（2）报文的解密依赖于初始矢量（参看 9.3 节），初始矢量可以在报文的另一半中发送。

（3）首先发送的一半报文可能是加密报文的单向 hash 函数（参看 2.4 节），并且加密报文本身可能是另一半。

为了了解这样做是怎样对 Mallory 制造麻烦的，让我们再看看他破坏协议的企图。他仍然能够在第（1）步和第（2）步中用自己的公开密钥代替 Alice 和 Bob 公开密钥。但现在，当他在第（3）步截取 Alice 的一半报文时，他不能用他的私钥对报文解密，然后用 Bob 的公开密钥再加密，他不得不虚构一完全不同的新报文，并将它的一半发送给 Bob。当他在第（4）步截取 Bob 给 Alice 的一半报文时，他有同样的问题，他不能用他的私钥解密，并用 Alice 的公开密钥再加密，他又不得不虚构一完全不同的新报文，并将它的一半发送给 Alice。当他在第（5）步和第（6）步截取到实际报文的另一半时，他再去把他虚构新报文改回来，就太迟了。Alice 和 Bob 之间的会话必会是完全不同的。

Mallory 也可以不用这种办法。如果他非常了解 Alice 和 Bob，他就可以模仿他们之中的一个同另一人通话，他们绝不会想到正受到欺骗。但这样做肯定比坐在他们之间截取和读他们的报文更难。

使用数字签名的密钥交换

在会话密钥交换协议期间采用数字签名也能防止“中间人攻击”。Trent 对 Alice 和 Bob 的公开密钥签名。签名的密钥包括一个已签名的所有权证书。当 Alice 和 Bob 收到密钥时，他们每人都能验证 Trent 的签名。那么，他们就知道公开密钥是哪个人的。密钥的交换就能进行了。

Mallory 会遇到严重的阻力。他不能假冒 Bob 或者 Alice，因为他不知道他们的私钥。他也不能用他的公开密钥代替他们两人的公开密钥，因为当他有由 Trent 签名的证书时，这个证书是为 Mallory 签发的。他所能做的事情就是窃听往来的加密报文，或者破坏通信线路，阻止 Alice 和 Bob 谈话。

这个协议也动用 Trent，但 KDC 遭受损害的风险比第一种协议小。如果 Mallory 危及到 Trent 的安全（侵入 KDC），他所得到的只是 Trent 的私钥。这个密钥使他仅能对新的密钥签名；它不会让他对任何会话密钥解密，或者读取任何报文。为了能够读往来的报文，Mallory 不得不冒充网络上的某个用户，并且欺骗合法用户用他的假的公开密钥加密报文。

Mallory 能够发起那种攻击。持有 Trent 的私钥，他能够产生假的签名密钥去愚弄 Alice 和 Bob。然后 Mallory 就能够在数据库中交换他们真正的签名密钥，或者截取用户向数据库的请求，并用他的假密钥代替。这使他能够发起“中间人攻击”，并读取他人的通信。

这种攻击是可行的，但记住 Mallory 必须能够截取和修改信息。在一些网络中，截取和修改报文比被动地坐在网络旁读取往来的报文更难。在广播信道上，如无线网中，几乎不可能用其他报文来替代某个报文（整个网络可能被堵塞）。在计算机网络中做这种事要容易些，并且随时日的推移变得越来越容易，例如 IP 欺骗、路由攻击等等；主动攻击并不一定表示

有人用数据显示仪抠出数据，且也不限于三字符的代理。

密钥和报文传输

Alice 和 Bob 在交换报文前不需要完成密钥交换协议。在下面的协议中，Alice 在没有任何以前的密钥交换协议的情况下，将报文 M 传送给 Bob：

（1）Alice 产生一随机会话密钥 K，并用 K 加密 M。

$$E_K（M）$$

（2）Alice 从数据库中得到 Bob 的公开密钥。

（3）Alice 用 Bob 的公开密钥加密 K。

$$E_B（K）$$

（4）Alice 将加密的报文和加密的会话密钥传送给 Bob。

$$E_K（M），E_B（K）$$

为了增加安全性，防止“中间人攻击”，Alice 可对传输签名。

（5）Bob 用他的私钥将 Alice 的会话密钥 K 解密。

（6）Bob 用会话密钥将 Alice 的报文解密。

这个混合系统表示，公开密钥密码怎样被经常用于通信系统的。它可以和数字签名、时间标记以及任何其它安全协议组合在一起使用。

密钥和报文广播

没有理由认为 Alice 不会把加了密的报文传送给几个人。在这个例子中，Alice 就把加密报文传送给 Bob、Carol 和 Dave：

（1）Alice 产生一随机会话密钥 K，并用 K 加密报文 M。

$$E_K（M）$$

（2）Alice 从数据库中得到 Bob、Carol 和 Dave 的公开密钥。

（3）Alice 用 Bob 的公开密钥加密 K，用 Carol 的公开密钥加密 K，用 Dave 的公开密钥加密 K。

$$E_B（K），E_C（K），E_D（K）$$

（5）Alice 广播加密的报文和所有加密的密钥，将它传送给要接收它的人。

$$E_B（K），E_C（K），E_D（K），E_K（M）$$

（5）只有 Bob、Carol 和 Dave 能用他们的私钥将 K 解密。

（6）只有 Bob、Carol 和 Dave 能用 K 将 Alice 的报文解密。

这个协议可以在存储转发网络上实现。中央服务器能够将 Alice 的报文，连同特别加密的密钥一起转发给 Bob、Carol 和 Dave。服务器不一定是安全的或者可信的，因为它不可能对任何报文解密。

3.2 鉴别

当 Alice 登录进入计算机（或自动柜员机、电话银行系统、或其它的终端类型）时，计

算机怎么知道她是谁呢？计算机怎么知道她不是其他人伪造 Alice 的身份呢？传统的办法是用通行字来解决这个问题的。Alice 先输入她的通行字，然后计算机确认它是正确的。Alice 和计算机两者都知道这个秘密通行字，当 Alice 每次登录时，计算机都要求 Alice 输入通行字。

利用单向函数的鉴别

Roger Needham 和 Mike Guy 意识到计算机没有必要知道通行字。计算机只需有能力区别有效通行字和无效通行字就成。这种办法很容易用单向函数来实现[1599，526，1274，1121]。计算机存储通行字的单向函数而不是存储通行字。

（1）Alice 将她的通行字传送给计算机。

（2）计算机完成通行字的单向函数计算。

（3）计算机把单向函数的运算结果和它以前存储的值进行比较。

由于计算机不再存储每人的有效通行字表，所以某些人侵入计算机，并偷取通行字的威胁就减少了。由通行字的单向函数生成通行字表是没用的，因为单向函数不可能逆向恢复出通行字。

字典式攻击和 Salt

用单向函数加密的通行字文件还是脆弱的。Mallory 在他的业余时间编制 1，000，000 个最常用的通行字表，他用单向函数对所有 1，000，000 个通行字进行运算，并将结果存储起来。如果每个通行字大约是 8 个字节，运算结果的文件不会超过 8M 字节，几张软盘就能存下。现在 Mallory 偷出加密的通行字文件。把加密的通行字和已加密的可能通行字文件进行比较，再观察哪个能匹配。

这就是字典式攻击，它的成功率令人吃惊（参看 8.1 节）。Salt 是使这种攻击更困难的一种方法。

Salt 是一随机字符串，它与通行字连接在一起，再用单向函数对其运算。然后将 Salt 值和单向函数运算的结果存入主机数据库中。如果可能的 Salt 值的数目足够大的话，它实际上就消除了对常用通行字采用的字典式攻击，因为 Mallory 不得不产生每个可能的 Salt 值的单向 hash 值。这是初始化矢量的简单尝试（参看 9.3 节）。

这儿的关键是确信当 Mallory 试图破译其他人的通行字时，他不得不每次试验字典里的每个通行字的加密，而不是只对可能的通行字进行大量的预先计算。

许多 Salt 是必需的。大多数 Unix 系统仅使用 12 比特的 Salt。即使那样，Daniel Klein 开发了一个猜测通行字的程序，在大约一星期里，经常能破译出一个给定的系统中的 40%的通行字[847，848]（见 8.1 节）。David Feldmeier 和 Philip Karn 编辑了大约 732，000 个常用的通行字表，表中的通行字都和 4096 个可能的 Salt 值中的每个值都有联系。采用这张表，他们估计在一给定系统中，大约能够破译出 30%的通行字[561]。

Salt 不是万灵药，增加 Salt 的比特数不能解决所有问题。Salt 只防止对通行字文件采用的一般的字典式攻击，不能防止对单个通行字的一致攻击。在多个机器上有相同的通行字的人使人难以理解，因为这样做并不比拙劣选用的通行字更好。

SKEY

SKEY 是一种鉴别程序，它依赖于单向函数的安全性。这很容易理解。

为了设置系统，Alice 输入随机数 R，计算机计算 f（R）、f（f（R））、f（f（f（R））等等大约 100 次。调用 x_1 ，x_2 ，x_3 ，。。。，x_{100} 这些数。计算机打印出这些数的列表，Alice 把这些数放入口袋妥善保管，计算机也顺利地在登录数据库中 Alice 的名字后面存储 x_{101} 的值。

当 Alice 第一次登录时，她输入她的名字和 x_{100}，计算机计算 f（x_{100}），并把它和 x_{101} 比较，如果它们匹配，那么证明 Alice 身份是真的。然后，计算机用 x_{101} 代替数据库中的 x_{100}。Alice 将从她的列表中取消 x_{100}。

Alice 每次登录时，都输入她的列表中未取消的最后的数 x_I，计算机计算 f（x_I），并和存储在它的数据库中的 x_{I+1} 比较。因为每个数只被用一次，并且这个函数是单向的，所以 Eve 不可能得到任何有用的信息。同样的，数据库对攻击者也毫无用处。当然，当 Alice 用完了她的列表上面的数后，她必须重新初始化系统。

采用公开密钥密码的鉴别

谨慎而言，第一个协议有严重的安全问题。当 Alice 将她的通行字发给她的主机时，能够进入她的数据通道的任何人都可读取她的通行字。可以通过迂回的传输路径（经四个工业竞争对手、三个国家和两个思想激进的大学）访问她的主机，在信道的任何一点 Eve 都有可能窃听 Alice 的登录序列。如果 Eve 可以存取主机的处理机存储器，那么在主机对通行字 hash 前，Eve 都能够看到通行字。

公开密钥密码能解决这个问题。主机保存每个用户的公开密钥文件，所有用户保存自己的私钥。这里给出一个协议。当登录时，协议按下面进行：

（1）主机发送一个随机字符串给 Alice。

（2）Alice 用她的私钥对此随机字符串加密，并将此字符串他和她的名字一起传送回主机。

（3）主机在它的数据库中查找 Alice 的公开密钥，并用公开密钥解密。

（4）如果解密后的字符串与主机在第一步中发送给 Alice 的字符串匹配，则允许 Alice 访问系统。

没有其他人能访问 Alice 的秘密密钥，因此不可能有任何人冒充 Alice。更重要的是，Alice 决不会在传输线路上将她的私钥发送给主机。窃听这个交互过程中的 Eve，不可能得到任何信息使她能够推导出 Alice 的私钥和冒充 Alice。

私钥既长又难记，它可能是由用户的硬件或通信软件自动处理的。这就需要一个 Alice 信任的智能终端。但主机和通信线路都不必是安全的。

愚蠢的是加密任意字符串——不仅是由不可信的第三方发送的随机数，而且是在任何环境送来的；可以采用类似于 19.3 节中讨论的攻击。安全的身份证明协议采用下面更复杂的形式：

（1）Alice 根据一些随机数和她的私钥进行计算，并将结果传送给主机。

（2）主机将一不同的随机数传送给 Alice。

（3）Alice 根据这些随机数（她产生的和她从主机接收的）和她的私钥进行一些计算，

并将结果传送给主机。

（4）主机用从 Alice 那里接收来的各种数据和 Alice 的公开密钥进行计算，以此来验证 Alice 是否知道自己的私钥。

（5）如果她知道，则她的身份就被证实了。

如果 Alice 不相信主机，就像主机不相信她一样，那么 Alice 将要求主机用同样方式证实其身份。

第（1）步似乎是不必要的和令人费解的，但它被用来阻止对协议的攻击。21.1 节和 21.2 节从数学上描述了几种用作身份证明的算法和协议[935]。

用联锁协议互相鉴别

Alice 和 Bob 是想要互相鉴别的两个用户。他们每人有一个另一人知道的通行字：Alice 通行字是 P_A，Bob 的是 P_B。下面的协议是行不通的：

（1）Alice 和 Bob 的交换公开密钥。

（2）Alice 用 Bob 的公开密钥加密 P_A，并将它传送给 Bob。

（3）Bob 用 Alice 的公开密钥加密 P_B，并发送给 Alice。

（4）Alice 解密她在第（2）步中接受到的信息并验证它是正确的。

（5）Bob 解密他在第（3）步中接受到的信息并验证它是正确的。

Mallory 能够成功发起“中间人攻击”（ 见 3.1 节）。

（1）Alice 和 Bob 交换公开密钥。Mallory 截取这两个报文，他用自己的公开密钥代替 Bob 的，并将它发送给 Alice。然后，他又用他的公开密钥代替 Alice 的，并将它发送给 Bob。

（2）Alice 用“Bob”的公开密钥对 P_A 加密，并发送给 Bob。Mallory 截取这个报文，用他的私钥对 P_A 解密，再用 Bob 的公开密钥加密，并将它发送给 Bob。

（3）Bob 用“Alice”的公开密钥对 P_B 加密，并发送给 Alice。Mallory 截取它，用他的私钥对 P_B 解密。再用 Alice 的公开密钥对它加密，并发送给 Alice。

（4）Alice 对 P_B 解密，并验证它是正确的。

（5）Bob 对 P_A 解密，并验证它是正确的。

从 Alice 和 Bob 处看并没有什么不同，然而 Mallory 知道 P_A 和 P_B。

Donald Davies 和 Wyn Price 描述了怎样采用联锁协议（在 3.1 节中描述）来挫败这种攻击[435]。Steve Bellovin 和 Michael Merritt 讨论了对这个协议进行攻击的各种方法[110]。如果 Alice 是用户，Bob 是主机，Mallory 可以假装是 Bob 和 Alice 一起完成协议的开始几步，然后终止连接。真实的技巧要求 Mallory 通过模拟线路噪声或网络失败来终止连接，但最终的结果是 Mallory 有 Alice 的通行字。然后，他可以和 Bob 连接，完成协议。Mallory 也就有 Bob 的通行字了。

假如用户的通行字比主机的通行字更敏感，那就可以修改这个协议，使 Alice 给出她的通行字之前，让 Bob 先给出他的通行字。这导致了一个更加复杂的攻击，文献[110]中有此描述。

SKID

SKID2 和 SKID3 是为 RACE 的 RIPE 项目开发的对称密码识别协议[1305]（见 25.7 节）。它们都用 MAC 来提供安全性，并且 SKID2 和 SKID3 两个协议都假设 Alice 和 Bob 共享同一秘密密钥 K。

SKID2 允许 Bob 向 Alice 证明他的身份。下面是这个协议：

（1）Alice 选用随机数 R_A（RIPE 文件规定 64 比特的数），并将它发送给 Bob。

（2）Bob 选用随机数 R_B（RIPE 文件规定 64 比特的数），将下面的数发送给 Alice：

$$R_B，H_K（R_A，R_B，B）$$

H_K 是 MAC（RIPE 文件建议的 RIPE-MAC 函数，参看 18.14 节）。B 是 Bob 的名字。

（3）Alice 计算 $H_K（R_A，R_B，B）$，并和她从 Bob 那里接收到的信息比较，如果结果一致，那么 Alice 知道她正与 Bob 通信。

SKID3 提供 Alice 和 Bob 之间的相互鉴别。第（1）步到第（3）步与 SKID2 是一样的，以后的协议按下面进行：

（4）Alice 向 Bob 发送：　　$H_K（R_B，A）$

A 是 Alice 的名字。

（5）Bob 计算 $H_K（R_B，A）$，并将它与从 Alice 那里收到的比较，如果相同，那么 Bob 知道他正与 Alice 通信。

这个协议对中间人攻击来说是不安全的。一般地，中间人攻击能够击败任何不包括某些秘密的协议。

信息鉴别

当 Bob 从 Alice 那里接收信息，他怎么知道信息是可信的呢？如果 Alice 对她的信息签名，就容易了。Alice 的签名足以使任何人都相信信息是可信的。

对称算法提供了一些鉴别。当 Bob 从 Alice 那里接收到用他们的共享密钥加密的信息时，他知道信息是从 Alice 那里来的，没有其他人知道他们的密钥。然而，Bob 没有办法使第三者相信这个事实，Bob 不可能把信息给 Trent 看，并使他相信信息是从 Alice 那里来的。Trent 能够相信信息是从 Alice 或 Bob 那里来的（因为没有其他人共享他们的秘密密钥），但是他没有办法知道信息到底是从谁那里来的。

如果信息没有被加密，Alice 也能使用 MAC。这也使 Bob 相信信息是可信的，但与对称密码的解决方法有同样的问题。

3.3　鉴别和密钥交换

这些协议综合利用密钥交换和鉴别，解决了一般的计算机问题：Alice 和 Bob 分别坐在网络的两端，他们想安全地交谈。Alice 和 Bob 怎么交换秘密密钥呢？他们中的每个人怎么确信他们当时正在同对方交谈而不是同 Mallory 谈话呢？大多数协议假设 Trent 与参与者双方各共享一个不同的秘密密钥，并且所有这些密钥在协议开始前都在适当的位置。在这些协议中使用的符号见表 3.1。

表 3.1　在鉴别和密钥交换协议中使用的符号

A	Alice 的名字
B	Bob 的名字
E_A	用 Trent 和 Alice 共享的密钥加密
E_B	用 Trent 和 Bob 共享的密钥加密
I	索引号
K	随机会话密钥
L	生存期
T_A，T_B	时间标记
R_A，R_B	随机数，分别由 Alice 和 Bob 选择的数。

Wide-Mouth Frog

Wide-Mouth Frog 协议[283，284]可能是最简单的对称密钥管理协议，该协议使用一个可信的服务器。Alice 和 Bob 两人各和 Trent 共享一秘密密钥。这些密钥只作密钥分配用，而不是用作加密用户之间的实际报文。会话密钥只通过两个报文就从 Alice 传送给 Bob：

（1）Alice 将时间标记 T_A 连同 Bob 的名字 B 和随机会话密钥 K 一起，用她和 Trent 共享的密钥对整个报文加密。她将加了密的报文和她的身份 A 一起发送给 Trent：

$$A，E_A（T_A，B，K）$$

（2）Trent 解密从 Alice 来的报文。然后将一个新的时间标记 T_B 连同 Alice 的名字和随机会话密钥一起，用他与 Bob 共享的密钥对整个报文加密，并将它发送给 Bob：

$$E_B（T_B，A，K）$$

这个协议最重要的假设是 Alice 完全有能力产生好的会话密钥。请记住，随机数是不容易产生的，无法相信 Alice 能够做好这件事。

Yahalom

在这个协议中，Alice 和 Bob 两人各与 Trent 共享一秘密密钥[283，284]。

（1）Alice 将她的名字连同随机数 R_A 一起，将它发送给 Bob。

$$A，R_A$$

（2）Bob 将 Alice 的名字、Alice 的随机数、他自己的随机数 R_B 一起用他和 Trent 共享的密钥加密。再将加密的结果和 Bob 的名字一起发送给 Trent。

$$B，E_B（A，R_A，R_B）$$

（3）Trent 产生两个报文，第一个报文由 Bob 的名字、随机会话密钥 K、Alice 的随机数和 Bob 的随机数组成。用他和 Alice 共享的密钥对所有第一个报文加密；第二个报文由 Alice 的名字和随机会话密钥组成，用他和 Bob 共享的密钥加密，然后将这两个报文发送给 Alice。

$$E_A（B，K，R_A，R_B），E_B（A，K）$$

（4）Alice 解密第一个报文，提出 K，并确认 R_A 的值与她在第（1）步时的值一样。Alice 发送两个报文给 Bob。第一个报文是从 Trent 那里接收到的用 Bob 的密钥加密的报文，第二个是用会话密钥加密的 R_B。

E_B（A，K），E_K（R_B）

（5）Bob 用他的密钥解密报文，提取 K，并确认 R_B 与他在第（2）步中的值一样。

结果，Alice 和 Bob 互相确信是正在同对方谈话，而不是跟第三者。这里的新东西是：Bob 是同 Trent 接触的第一人，而 Trent 仅发送给 Alice 一个报文。

Needham 和 Schroeder

这个协议由 Roger Needham 和 Michael Schroeder 发明[1159]，也采用对称密码和 Trent。

（1）Alice 将由她的名字 A，Bob 的名字 B 和随机数 R_A 组成的报文传给 Trent。

（A，B，R_A）

（2）Trent 产生一随机会话密钥 K。他用与 Bob 共享的秘密密钥对随机会话密钥 K 和 Alice 名字组成的报文加密。然后用他和 Alice 共享的秘密密钥对 Alice 的随机值、Bob 的名字、会话密钥 K 和已加密的报文进行加密，最后，将加密的报文传送给 Alice：

E_A（R_A，B，K，E_B（K，A））

（3）Alice 将报文解密并提取 K。她确认 R_A 与她在第（1）步中发送给 Trent 的一样。然后她将 Trent 用 Bob 的密钥加密的报文发送给 Bob。

E_B（K，A）

（4）Bob 对报文解密并提取 K，然后产生另一随机数 R_B。他用 K 加密它并将它发送给 Alice。

E_K（R_B）

（5）Alice 用 K 将报文解密，产生 R_B-1 并用 K 对它加密，然后将报文发回给 Bob。

E_K（R_B-1）

（6）Bob 用 K 对信息解密，并验证它是 R_B-1。

所有这些围绕 R_A、R_B、R_B-1 的麻烦用来防止**重放攻击**。在这种攻击中，Mallory 可能记录旧的报文，在以后再使用它们以达到破坏协议的目的。在第（2）步中 R_A 的出现使 Alice 确信 Trent 的报文是合法的，并且不是以前协议的重放。在第（5）步，当 Alice 成功地解密 R_B，并将 R_B-1 送回给 Bob 之后，Bob 确信 Alice 的报文不是早期协议执行的重放。

这个协议的主要安全漏洞是旧的会话密钥仍有价值。如果 Mallory 可以存取旧的密钥 K，他可以发起一次成功的攻击[461]。他所做的全部工作是记录 Alice 在第（3）步发送给 Bob 的报文。然后，一旦他有 K，他能够假装是 Alice：

（1）Mallory 发送给 Bob 下面的信息：

E_B（K，A）

（2）Bob 提取 K，产生 R_B，并发送给“Alice”：

E_K（R_B）

（3）Mallory 截取此报文，用 K 对它解密，并发送给 Bob：

E_K（R_B-1）

（4）Bob 验证“Alice”的报文是 R_B-1。

到此为止，Mallory 成功地使 Bob 确信他就是 Alice 了。

一个使用时间标记的更强的协议能够击败这种攻击[461，456]。在第（2）步中，一个

时间标记被附到用 Bob 的密钥加密的 Trent 的信息中：E_B（K，A，T）。时间标记需要一个安全的和精确的系统时钟，这对系统本身来说不是一个普通问题。

如果 Trent 与 Alice 共享的密钥 K_A 泄露了，后果是非常严重的。Mallory 能够用它获得同 Bob 交谈的会话密钥（或他想要交谈的其他任何人的会话密钥）。情况甚至更坏，在 Alice 更换她的密钥后 Mallory 还能够继续做这种事情[90]。

Needham 和 Schroeder 试图在他们的协议改进版本中改正这些问题[1160]。他们的新协议基本上与发表在同一杂志的同一期上 Otway-Rees 的协议相同。

Otway-Rees

这个协议也是使用对称密码[1224]。

（1）Alice 产生一报文，此报文包括一个索引号 I、她的名字 A、Bob 的名字 B 和一随机数 R_A，用她和 Trent 共享的密钥对此报文加密，她将索引号、她的名字和 Bob 的名字与她加密的报文一起发送给 Bob：

$$I，A，B，E_A（R_A，I，A，B）$$

（2）Bob 产生一报文，此报文包括一新的随机数 R_B、索引号 I、Alice 的名字 A 和 Bob 的名字 B。用他与 Trent 共享的密钥对此报文加密。他将 Alice 的加密报文、索引号、Alice 的名字、Bob 的名字与他加了密的报文一起发送给 Trent：

$$I，A，B，E_A（R_A，I，A，B），E_B（R_B，I，A，B）$$

（3）Trent 产生一随机会话密钥 K，然后，产生两个报文。一个是用他与 Alice 共享的密钥对 Alice 的随机数和会话密钥加密，另一个是用与 Bob 共享的密钥对 Bob 的随机数和会话密钥加密。他将这两个报文与索引号一起发送给 Bob：

$$I，E_A（R_A，K），E_B（R_B，K）$$

（4）Bob 将用 Alice 的密钥加密的报文连同索引号一起发送给 Alice：

$$I，E_A（R_A，K）$$

（5）Alice 解密报文，恢复出她的密钥和随机数，然后她确认协议中的索引号和随机数都没有改变。

假设所有随机数都匹配，并且按照这种方法索引号没有改变，Alice 和 Bob 现在相互确认对方的身份，他们就有一个秘密密钥用于通信了。

Kerberos

Kerberos 是 Needham-Schroeder 协议的变型，将在 24.5 节中详细讨论它。在基本的 Kerberos 第 5 版本的协议中， Trent 和 Alice、Bob 中的每人各共享一密钥。Alice 想产生一会话密钥用于与 Bob 通信。

（1）Alice 将她的身份 A 和 Bob 的身份 B 发送给 Trent：

$$A，B$$

（2）Trent 产生一报文，该报文由时间标记 T、使用寿命 L、随机会话密钥 K 和 Alice 的身份构成。他用与 Bob 共享的密钥加密报文。然后，他取时间标记、使用寿命、会话密钥和 Bob 的身份，并且用他与 Alice 共享的密钥加密，并把这两个加密报文发给 Alice：

E_A（T，L，K，B），E_B（T，L，K，A）

（3）Alice 用她的身分和时间标记产生报文，并用 K 对它进行加密，将它发送给 Bob。Alice 也将从 Trent 那里来的用 Bob 的密钥加密的报文发送给 Bob：

E_K（A，T），E_B（T，L，K，A）

（4）Bob 用 K 对时间标记加 1 的报文进行加密，并将它发送给 Alice：

E_K（T+1）.

这个协议是可行的，但它假设每个人的时钟都与 Trent 的时钟同步。实际上，这个结果是通过把时钟同步到一个安全的定时服务器的几分钟之内，并在这个时间间隔内检测重放而获得的。

Neuman-Stubblebine

不管是由于系统缺陷还是由于破坏，时钟可能变得不同步。如果时钟不同步，这些协议的大多数可能受到攻击[644]。如果发送者的时钟比接收者的时钟超前，Mallory 能够截取从发送者来的报文，当时间标记变成接收地当前时间时，Mallory 重放报文。这种攻击叫做**隐瞒重放**，并有使人气愤的结果。

这个协议首先在[820]中提出，并在[1162]中改进以图反击这种隐瞒攻击。它是 Yahalom 协议的增强，是一个非常好的协议：

（1） Alice 把她的名字和随机数一起送给 Bob。

A，R_A

（2） Bob 把 Alice 的名字连同她的随机数、和一个时间标记一起，用他与 Trent 共享的密钥加密，并把加密的结果、他的名字和一个新的随机数一起发给 Trent。

B，R_B，E_B（A，R_A，T_B）

（3） Trent 产生随机会话密钥，然后产生两个报文，第一个报文由 Bob 的名字、Alice 的随机数、随机会话密钥和时间标记组成，所有这些报文用他与 Alice 共享的密钥加密；第二个报文由 Alice 的名字、会话密钥和时间标记组成，所有这些报文用他与 Bob 共享的密钥加密。他将这两个报文和 Bob 的随机数一起发给 Alice。

E_A（B，R_A，K，T_B），E_B（A，K，T_B），R_B

（4）Alice 解出用她的密钥加密的报文，提出密钥 K，并确认 R_A 与她在第（1）步中的值相同。Alice 发给 Bob 两个消息，第一个是从 Trent 那里接收的用 Bob 的密钥加密的消息。第二个是用会话密钥 K 加密的 R_B。

E_B（A，K，T_B），E_K（R_B）

（5）Bob 解出用他的密钥加密的消息，提出密钥 K，并确认 T_B 和 R_B 与它们在第（2）步中有相同的值。

假设随机数和时间标记都匹配，Alice 和 Bob 就会相信互相的身份，并共享一个秘密密钥。因为时间标记只是相对于 Bob 的时间，所以不需要同步时钟，Bob 只检查他自己产生的时间标记。

这个协议的好处是：在某些预先确定的时间内，Alice 能够用从 Trent 那里接收的消息与 Bob 作后续的鉴别。假设 Alice 和 Bob 完成了上面的协议和通信，然后终止连接，Alice

和 Bob 也不必依赖 Trent，就能够在 3 步之内重新鉴别。

（1）Alice 将 Trent 在第（3）步发给她的信息和一个新的随机数送给 Bob。

E_B（A，K，T_B），R'_A

(2) Bob 发给 Alice 另一个新的随机数，并且 Alice 的新随机数用他们的会话密钥加密。

R'_B，E_K（R'_A）

（3）Alice 用他们的会话密钥加密 Bob 的新随机数，并把它发给 Bob。

E_K（R'_B）

新随机数防止了重放攻击。

DASS

分布式鉴别安全协议（DASS）是由数字设备公司开发的，它也提供相互鉴别和密钥交换[604，1519，1518]。与前面的协议不同，DASS 同时使用了公开密钥和对称密码。Alice 和 Bob 每人有一个私钥，Trent 有他们公钥签名的副本。

（1）Alice 送一个消息给 Trent，这个消息由 Bob 的名字组成。

B

（2）Trent 把 Bob 的公钥 K_B 发给 Alice，用 Trent 的私钥 T 签名。签名消息包括 Bob 的名字。

S_T（B，K_B）

（3）Alice 验证 Trent 的签名以确认她接收的密钥确实是 Bob 的公钥。她产生一随机会话密钥 K 和一公钥/私钥密钥对 K_P，她用 K 加密时间标记，然后用她的私钥 K_A 对密钥的寿命周期 L、她的名字和 K_P 签名。最后，她用 Bob 的公钥 K 加密，并用 K_P 签名。她将所有这些消息发给 Bob。

E_K（T_A），S_{KA}（L，A，K_P），S_{KP}（E_{KB}（K））

（4）Bob 发送一个消息给 Trent（这可能是另一个 Trent），它由 Alice 的名字组成。

A

（5）Trent 把 Alice 的公钥 K_B 发给 Bob，用 Trent 的私钥 T 签名。签名消息包括 Alice 的名字。

S_T（A，K_A）

（6）Bob 验证 Trent 的签名以确认他接收的密钥确实是 Alice 的公钥。然后他验证 Alice 的签名并恢复出 K_P。他验证签名并用他的私钥恢复 K。然后解密 T_A 以确信这是当前的消息。

（7）如果需要互相鉴别，Bob 用 K 加密一个新的时间标记，并把它送给 Alice。

E_K（T_B）

（8）Alice 用 K 解密 T_B 以确信消息是当前的。

DEC 公司的 SPX 产品是基于 DSSA 鉴别协议的。额外的信息可在[34]中找到。

Denning-Sacco

这个协议也使用公开密钥密码[461]。Trent 保存每个人的公开密钥数据库。

（1）Alice 发送一个有关她和 Bob 的身份消息给 Trent：

A， B

（2）Trent 把用 Trent 的私钥 T 签名的 Bob 的公钥 K_B 发给 Alice。Trent 也把用 Trent 的私钥 T 签名的 Alice 自己的公钥 K_A 发给 Alice。

S_T（B，K_B），S_T（A，K_A）

（3）Alice 向 Bob 传送随机会话密钥、时间标记（都用她自己私钥签名并用 Bob 的公钥加密）和两个签了名的公开密钥。

E_B（S_A（K，T_A）），S_T（B，K_B），S_T（A，K_A）

(4) Bob 用他的私钥解密 Alice 的消息，然后用 Alice 的公钥验证她的签名。他检查以确信时间标记仍有效。

在这里 Alice 和 Bob 两人都有密钥 K，他们能够安全地通信。

这看起来很好，但实际不是这样的。在和 Alice 一起完成协议后，Bob 能够伪装是 Alice[5]。注意：

（1）Bob 把他的名字和 Carol 名字发给 Trent

B，C

（2）Trent 把 Bob 和 Carol 的已签名的公钥发给 Bob。

S_T（B，K_B），S_T（C，K_C）

（3）Bob 将以前从 Alice 那里接收的会话密钥和时间标记的签名用 Carol 的公钥加密，并和 Alice 和 Carol 的证书一起发给 Carol。

E_C（S_A（K，T_A）），S_T（A，K_A），S_T（C，K_C）

（4）Carol 用她的私钥解密 Alice 的消息，然后用 Alice 的公钥验证她的签名。她检查以确信时间标记仍有效。

Carol 现在认为她正在与 Alice 交谈，Bob 成功地欺骗了她。事实上，在时间标记截止前，Bob 可以欺骗网上的任何人。

这个问题容易解决。在第（3）步的加密消息内加上名字：

E_B（S_A（A，B，K，T_A）），S_T（A，K_A），S_T（B，K_B）

因为这一步清楚地表明是 Alice 和 Bob 之间在通信，所以现在 Bob 就不可能对 Carol 重放旧消息。

Woo-Lam

这个协议也使用公开密钥密码[1610，1611]：

（1） Alice 发送一个有关她和 Bob 的身份消息给 Trent。

A， B

（2） Trent 用他的私钥 T 对 Bob 的公钥签名，然后把它发给 Alice。

S_T（K_B）

（3） Alice 验证 Trent 的签名，然后把她的名字和一个随机数用 Bob 的公钥加密，并把它发给 Bob。

E_{KB}（A，R_A）

（4） Bob 把他的名字、Alice 的名字和用 Trent 的公钥 K_T 加密的 Alice 的随机数一起

发给 Trent。

A，B，E_{KT}（R_A）

（5） Trent 把用 Trent 的私钥签名的 Alice 的公钥 K_A 发给 Bob，Trent 用 Trent 的私钥对所有 Alice 的随机数、随机会话密钥、Alice 的名字和 Bob 的名字签名并用 Bob 的公钥加密，并把它也发给 Bob。

S_T（K_A），E_{KB}（S_T（R_A，K，A，B））

（6） Bob 验证 Trent 的签名。然后他将第（5）步中 Trent 的消息的第二部分和一个新随机数一起用 Alice 的公钥加密，并将结果发给 Alice。

E_{KA}（S_T（R_A，K，A，B），R_B）

（7） Alice 验证 Trent 的签名和她的随机数。然后她将第二个随机数用会话密钥 K 加密，并发给 Bob。

E_K（R_B）

（8）Bob 解密他的随机数，并验证它没有改变。

其它协议

已有文献中有许多其它协议。X.509 协议在 24.9 节中讨论，KryptoKnight 在 24.6 节中讨论，加密密钥交换在 22.5 节中讨论。

Kuperee[694]是一个新的公开密钥协议，正在做关于使用**信标**的协议，信标是一个可信的网络节点，它不断地广播只以当时为限的鉴别。

学术上的教训

在前面的协议中，那些被破译的协议和没有被破译的协议都有一些重大的教训：

- 因为设计者试图设计得太精巧，许多协议失败了。他们通过省去重要的部分：名字、随机数等等来优化他们的协议， 但矫枉过正了[43，44]。
- 试图优化绝对是一个可怕的陷阱，并且全部命运依赖于你所做的假设。例如：如果你有识别的时间，你就可以做许许多多你没去做也做不到的事情。
- 选择的协议依赖于底层的通信体系结构。你难道不想使消息的大小和数量最小吗？是团体中的所有人都能够互相交谈呢？还是只有他们中的几个人能够交谈？

类似这些问题导致了协议分析的形式化方法的开发。

3.4 鉴别和密钥交换协议的形式分析

在网络上的一对计算机（和人）之间建立安全的会话密钥问题是如此的重要，以至于引发出许多研究。一些研究着重开发协议， 例如开发 3.1 节、3.2 节、3.3 节中讨论的那些协议。这又导致了更大和更多有趣的问题：鉴别和密钥交换协议的形式分析。在提出似乎是安全的协议的以后岁月里，人们发现了这些协议的缺陷，研究人员想要得到从一开始就能证明协议的安全性各种工具。虽然很多这种工作都能应用到一般的密码协议中去，但是研究的重点却毫无例外地放在鉴别和密钥交换上。

对密码协议的分析有四种基本途径[1045]：

1. 使用规范语言和验证工具建立协议模型和验证协议，它不是特别为密码协议分析设计的。
2. 开发专家系统，协议设计者能够用它来调查研究不同的情况。
3. 用分析知识和信任的逻辑，建立协议族的需求模型。
4. 开发形式化方法，它基于密码系统的代数重写项性质。

关于这四种途径的所有讨论和围绕它们的研究远远超出了本书的范围。文献[1047，1355]对这个题目作了很好的介绍。我只略微谈到这个领域的主要作用。

第一种途径把密码学协议当作任何其他计算机程序对待，并试图证明它的正确性。一些研究者把协议表示为有限状态机[1449，1565]，其他人使用一阶判定微积分[822]，还另有一些人使用规范语言来分析协议[1566]。然而证明正确性与证明安全性不同，并且这个方法对于发现许多缺陷的协议来说是行不通的。虽然这一种途径最早被广泛研究，但这个领域的大多数工作已经转向获得普及的第 3 种途径。

第二种途径是使用专家系统来确定协议是否能达到不合乎需要的状态（例如密钥的泄露）。虽然这种途径能够更好地识别缺陷，但它既不能保证安全性，又不能为开发攻击提供技术。它的好处在于决定协议是否包含已知的缺陷，但不可能发现未知的缺陷。这种途径的例子可在[987，1521]中找到。[1092]讨论了一个由美国军方开发的基于规则的系统，这个系统叫做询问器。

第三种途径到目前为止是最流行的，它是由 Michael Burrows、Martin Abadi 和 Roger Needham 首先发明的。为了进行知识和信任分析，他们开发了一个形式逻辑模型，叫做 BAM **逻辑**[283，284]。BAM 逻辑是分析鉴别协议时用得最广泛的逻辑。它假设鉴别是完整性和新鲜度的函数，并使用逻辑规则来对贯穿协议的那些属性的双方进行跟踪。虽然已经提出了这种途径的许多变化和扩展，但大多数协议设计者仍在引用最初的研究。

BAM 逻辑并不提供安全性证明，它只能推出鉴别。它具有容易使用的简单、明了的逻辑，对于发现缺陷仍然有用。BAM 逻辑中的一些命题有：

Alice 相信 X（Alice 装作好像 X 是正确的）。

Alice 看 X（某些人已经把包含 X 的消息发给 Alice，Alice 可能在解密消息后，能够读和重复 X）。

Alice 说 X（在某一时间，Alice 发送包括命题 X 的消息。不知道的是，消息在多早以前曾被发送过，或是在协议当前运行期间发送的。已经知道，当 Alice 说 X 时，Alice 相信 X。）

X 是新的（在当前运行协议以前，X 在任何时间没有把消息发送出去）等等。

BAM 逻辑也为协议中有关信任理由提供规则。这些规则能够用到协议的逻辑命题，用来证明事情或回答有关协议的问题。例如，消息内涵的规则是：

如果 Alice 相信 Alice 和 Bob 共享秘密密钥 K，Alice 看见用 K 加密的 X，而 Alice 没有用 K 加密 X，那么 Alice 相信 Bob 曾经说过 X。

另一个规则是只以当时为限的验证规则：

如果 Alice 相信 X 只在最近被发送，并且 Bob 曾经说过 X，那么 Alice 就认为 Bob 相信 X。

用 BAM 逻辑进行分析分四步：

（1）采用以前描述的命题，把协议转换为理想化形式。

（2）加上有关协议初始状态的所有假设。

（3）把逻辑公式放到命题中：在每个命题后断言系统的状态。

（4）为了发现协议各方持有的信任，运用逻辑基本原理去断言和假设。

BAM 逻辑的作者“把理想化的协议看作比在文献中发现传统的描述更清楚和更完善的规范…”[283，284]。其它协议没有这种印记，并因为它不可能正确地反映实际的协议而批评这个步骤[1161，1612]。在[221，1557]中有进一步的争论。其它批评试图表明，BAM 逻辑可能推导出关于协议明显错误的特征[1161]，见[285，1509]的辩驳。并且 BAM 逻辑只涉及信任，而与安全性无关[1509]。在[1488，706，1002]中有更多这方面的争论。

尽管有这些批评，BAM 逻辑仍是成功的。它已经在几种协议中发现缺陷，这些协议包括 Needham-Schroeder 和一个早期 CCITT X。509 协议草案[303]。它已经发现很多协议中的冗余，这些协议包括 Yahalom、Needham-Schroeder 和 Kerberos。许多人的文章使用 BAM 逻辑，声称他们协议的安全性[40，1162，73]。

其它逻辑系统也有公布，一些设计成为 BAM 逻辑的扩展[645，586，1556，828]，另一些是基于 BAM 逻辑去改进发现的弱点[1488，1002]。虽然 GNY 有一些缺点[40]，但它是这些当中最为成功的一个[645]。文献[292，474]提出应将概率信任加到 BAM 逻辑中去，以配合成功，在[156，798，288]讨论了其它形式逻辑，[1514]试图把几个逻辑的特点结合起来，[1124，1511]提出了信任能够随时间而改变的逻辑。

密码协议分析的第四种途径是把协议当作一个代数系统模型，表示有关协议参与者了解的状态，然后分析某种状态的可达性。这种途径没有像形式逻辑那样引起更多的注意，但情况正在改变。它首先由 Michael Merritt 使用[1076]，他证明代数模型可用来分析密码协议。其它途径在[473，1508，1530，1531，1532，1510，1612]中描述。

海军研究试验室（NRL）的协议分析器可能是这些技术中最成功的应用[1512，823，1046，1513]；它被用来在各种协议中寻找新的和已知的缺陷[1044，1045，1047]。这台协议分析器定义了下面的行为：

- 接收（Bob，Alice，M，N）。（Bob 在 N 地附近，接收消息 M 作为来自 Alice 的消息。）
- 获悉（Eve，M）。（Eve 获悉 M。）
- 发送（Alice，Bob，Q，M）。（根据查询 Q，Alice 发送 M 给 Bob。）
- 请求（Bob，Alice，Q，N）。（Bob 在 N 地附近，发送 Q 给 Alice。）

从这些行为中，可以确定需求。例如：

- 如果 Bob 在过去某些点接收到从 Alice 来的消息 M，那么 Eve 在过去某些点没有获悉 M。
- 如果 Bob 在他的 N 地附近接收到从 Alice 来的消息 M，Alice 给 Bob 发送 M 作为 Bob 在 N 地附近查询的响应。

为了使用 NRL 协议分析器，必须按以前的结构规定协议。分析有四个步骤：□为诚实的参与者定义传送；□描述对所有诚实和不诚实参与者可得到的操作；□描述基本的协议构造部件；□描述还原规则。这里表示的所有要点是已知的协议要与它的需求相符。采用像 NRL 协议分析器这样的工具，最终会产生一个能够证明是安全的协议。

3.5 多密钥公开密钥密码学

公开密钥密码使用两个密钥，用一个密钥加密的报文能用另一个密钥解密。通常一个密钥是私有的，而另一个是公开的。让我们假设 Alice 有一个密钥，Bob 有另一个，那么 Alice 能够加密报文，并且只有 Bob 能把它解密； 反过来 Bob 能够加密报文，只有 Alice 能读它。

Colin Boyd 推广了这个概念[217]。设想一种具有三个密钥 K_A，K_B，K_C 的公开密钥密码的变体。表 3.2 给出了它的分配。

表 3.2 三个密钥密钥分配

Alice	K_A
Bob	K_B
Carol	K_C
Dave	K_A 和 K_B
Ellen	K_B 和 K_C
Frank	K_A 和 K_C

Alice 可以用 K_A 加密报文以便 Ellen 用 K_B 和 K_C 解密此报文。这样 Bob 和 Carol 可能冲突。Bob 能对报文加密以便 Frank 能读它，Carol 也能加密报文以便 Dave 能读它。Dave 能用 K_A 加密报文以便 Ellen 能读它，由于有 K_A ， Frank 也能读它，或者 Dave 同时用 K_A 和 K_B 加密以便 Carol 能读它。类似地，Ellen 能够加密报文以便 Alice 和 Dave 或者 Frank 能读它。表 3.3 归纳了所有可能的组合，再也没有其它组合。

表 3.3 三个密钥的报文加密

用作加密的密钥	必须用的解密密钥
K_A	K_B 和 K_C
K_B	K_A 和 K_C
K_C	K_A 和 K_B
K_A 和 K_B	K_C
K_A 和 K_C	K_B
K_B 和 K_C	K_A

这可以推广到 n 个密钥，如果密钥的某个子集用来加密报文，那么就需要用其它密钥来解密此报文。

广播报文

假设有 100 个工人在野外作业，你想给他们当中的一些人发送报文，但预先并不知道是该向哪些人发。可以为每个人单独加密报文或者为每种可能的组合都给出密钥。第一种选择

需要增加很多通信量；第二种选择需要更多密钥。

采用多密钥密码方案就容易得多，我们将用三个工人：Alice、Bob 和 Carol。。将 K_A 和 K_B 给 Alice，将 K_B 和 K_C 给 Bob，将 K_C 和 K_A 给 Carol。现在可以同想要通信的任何子集交谈。如果想发送一报文只有 Alice 能读它，就用 K_C 加密此报文。当 Alice 接收到此报文时，她先用 K_A，然后再用 K_B 解密。如果想发送一报文只有 Bob 能读它，就用 K_A 加密；用 K_B 加密时，只有 Carol 才能读它。如果发送一报文使 Alice 和 Bob 都能读它，就用 K_A 和 K_C 对它加密，等等。

这可能不会有什么激动人心的地方，但对于 100 个工人来说，它就非常管用了。单独的报文表示和每个工人共享一个密钥（总共 100 个密钥）和每个报文。每个可能子集的密钥表示共有 2^{100}-2 个不同的密钥（针对全部工人的报文和不对工人的报文除外）。这里的方案只需要一个加密报文和 100 个不同的密钥，这个方案的缺陷是你不得不广播哪个工人的子集能够读报文，否则，每个工人将不得不试所有可能的密钥组合，以寻找正确的一组密钥，甚至只要意向接收者的名字也行得通。至少，要想实现简单，每个人实际上得到大量的密钥数据。

有其它用于报文广播的技术，其中有一些避免了前面的问题，这些将在 22.7 节中讨论。

3.6 秘密分割

设想你已发明了一种新的、特别粘、特别甜的奶油饼的馅，或者你已经制作了一种碎肉夹饼的调味料，那怕它比你的竞争者的更无味。重要的是：你都必须保守秘密。你只能告诉最信赖的雇员各种成分准确的调合，但如果他们中的一个背叛到对手方时怎么办呢？秘密就会泄漏，不久，每个出售黄油的宫殿将做出和你的一样的无味的调味料。

这种情况就要求秘密分割。有各种方法把消息分割成许多碎片[551]。每一片本身并不代表什么，但把这些碎片放到一块，消息就会重现出来。如果消息是一个秘方，每一个雇员有一部分，那么只有他们放在一起才能做出这种调味料。如果任意一雇员辞职带走一部分制法，这个信息本身是毫无用处的。

在两个人之间分割一消息是最简单的共享问题。下面是 Trent 把一消息分割给 Alice 和 Bob 的一个协议：

（1）Trent 产生一随机比特串 R，和消息 M 一样长。

（2）Trent 用 R 异或 M 得到 S：

$$M \oplus R = S$$

（3）Trent 把 R 给 Alice，将 S 给 Bob。

为了重构此消息，Alice 和 Bob 只需一起做一步：

（4）Alice 和 Bob 将他们的消息异或就可得到此消息：

$$R \oplus S = M.$$

如果做得适当，这种技术是绝对安全的。每一部分本身是毫无价值的。实质上，Trent 是用一次一密乱码本加密消息，并将密文给一人，乱码本给另一人。1.5 节讨论过一次一密乱码本，它们具有完全保密性。无论有多大计算能力都不能根据消息碎片之一就确定出此消息来。

把这种方案推广到多人也是容易的。为了在多个人中分割一消息，将此消息与多个随机比特异或成混合物。在下面的例子中，Trent 把信息划分成四部分：

（1）Trent 产生三个随机比特串 R、S、T，每个随机串与消息 M 一样长。

（2）Trent 用这三个随机串和 M 异或得到 U：

$$M \oplus R \oplus S \oplus T = U$$

（3）Trent 将 U 给 Alice，S 给 Bob，T 给 Carol，U 给 Dave。

Alice、Bob 和 Carol、Dave 在一起可以重构此消息：

（4）Alice、Bob、Carol 和 Dave 一起计算：

$$R \oplus S \oplus T \oplus U = M$$

这是一个裁决协议，Trent 有绝对的权力，并且能够做他想做的任何事情。他可以把毫无意义的东西拿出来，并且申明是秘密的有效部分。在他们将秘密重构出来之前，没有人能够知道它。他可以分别交给 Alice、Bob、Carol 和 Dave 一部分，并且在以后告诉每一个人，只要 Alice、Carol 和 Dave 三人就可以重构出此秘密，然后解雇 Bob。由于这是由 Trent 分配的秘密，这对于他恢复信息是没有问题的。

然而，这种协议存在一个问题：如果任何一部分丢失了，并且 Trent 又不在，就等于将消息丢掉了。如果 Carol 有调味料制法的一部分，他跑去为对手工作，并带走了他的那一部分，那么其他人就很不幸了，她不可能重新产生这个秘方，但 Alice、Bob、Dave 在一起也不行。Carol 的那一部分对消息来说和其它部分的组合一样重要。Alice、Bob 和 Dave 知道的仅是消息的长度，没有其它更多的信息了。这是真的，因为 R、S、T、U 和 M 都有同样的长度；见到他们中的任何一个都知道它的长度。记住，M 不是通常单词意义的分割，它是用随机数异或的。

3.7 秘密共享

你正在为核导弹安装发射程序。你想确信一个疯子是不能够启动发射。你也想确信两个疯子也不能启动发射。在你允许发射前，五个官员中至少有三个是疯子。

这是一个容易解决的问题。做一个机械发射控制器，给五个官员每人一把钥匙，并且你在允许他们起爆时，要求至少三个官员的钥匙插入合适的槽中（如果你确实担心，可以使这些槽分隔远些，并要求官员们同时将钥匙插入。你不愿一个官员偷窃另两把钥匙，使他能够毁坏多伦多）。

我们甚至能把它变得更复杂。也许将军和两个上校被授权发射导弹，但如果将军正在打高尔夫球，那么启动发射需要五位上校。制造一个发射控制器，该发射器需要五把钥匙。给将军 3 把钥匙，给每位上校一把钥匙。将军和任何两位上校一起就能发射导弹。五个上校一起也能，但将军和一位上校就不能，四位上校也不行。

一个叫做**门限方案**的更复杂的共享方案，可在数学上做到这些甚至做得更多。起码，你可以取任何消息（秘密的秘方，发射代码，你的洗衣价目表），并把它分成 n 部分，每部分叫做它的“**影子**”或共享，这样它们中的任何 m 部分能够用来重构消息，更准确地说，这叫做（m,n）门限方案。

拿（3，4）门限方案来说，Trent 可以将他的秘密调味料秘方分给 Alice、Bob、Carol 和 Dave，这样把他们中的任意三个“影子”放在一起就能重构消息。如果 Carol 正在渡假，那么 Alice、Bob 和 Dave 可以做这样的事。如果 Bob 被汽车撞了，那么 Alice、Carol 和 Dave 能够做这件事。然而如果 Carol 正在渡假期间，Bob 被汽车撞了，Alice 和 Dave 就不可能重构消息了。

普通的门限方案远比上面所说的更通用。任何共享方案都能用它建造模型。你可以把消息分给你所在大楼中的人，以便重构它，你需要一楼的 7 个人和二楼的 5 个人，就能恢复此消息。如果有从三楼来的人，在这种情况下，你仅需要从三楼来的那个人和从一楼来的 3 个人及从二楼来的 2 个人，如果有从四楼来的人，在这种情况下，你仅需要从四楼来的那个人和从三楼来的 1 个人，或从四楼来的那个人和从一楼来的 2 个人及从二楼来的 1 个人，或者……好的，你明白这个概念了。

这种思想是由 Adi Shamir[1414] George .Blakley[182]两人分别创造的。并由 Gus Simmons 做了更广泛的研究，23.2 节讨论几种不同的算法。

有骗子的秘密共享

有许多方法可欺骗门限方案，，下面是其中的几种—

情景 1：上校 Alice、Bob 和 Card 在某个隔离区很深的地下掩体中。一天，他们从总统那里得到密码消息：“发射那些导弹，我们要根除这个国家的神经网络研究残余。”Alice、Bob 和 Carol 出示他们的“影子”，但 Carol 却输入一随机数。她实际上是和平主义者，不想发射导弹。由于 Carol 没有输入正确的“影子”，因此他们恢复的秘密是错误的，导弹还是停放在发射井中。甚至更糟糕的是，没有人知道为什么会这样。即使 Alice 和 Bob 一起工作，他们也不能证明 Carol 的“影子”是无效的。

情景 2：上校 Alice 和 Bob 与 Mallory 正坐在掩体中。Mallory 假装也是上校，其他人都不能识破。同样的消息从总统那里来了，并且每人都出示了他们的“影子”，“哈，哈！”Mallory 大叫起来，“我伪造了从总统那里来的消息，现在我知道你们两人的“影子”了。在其他人抓住他以前，他爬上楼梯逃跑了。

情景 3：上校 Alice、Bob 和 Carol 与 Mallory 一起坐在掩体中，Mallory 又是伪装的（记住，Mallory 没有有效的影子）。同样的消息从总统那里来了，并且每人都出示了他们的“影子”，Mallory 只有在看到他们三人的“影子”后才出示他的“影子”。由于重构这个秘密只需要三个影子，因此他能够很快地产生一有效的“影子”并出示。现在，他不仅知道了秘密，而且没有人知道他不是这个方案的一部分。

处理这些欺骗的一些协议在 23.2 节中讨论。

没有 Trent 的秘密共享

五个官员中有三个人插入他们的钥匙，才能打开银行的金库。这听起来像一个基本的（3，5）门限方案，但是有一个保险装置。没有人知道整个秘密，没有 Trent 来把秘密分成 5 部分。存在一种五个官员可以产生秘密的协议，通过这协议，每人得到一部分，使得官员们在重构它之前，无人知道这个秘密[443]。我不准备在这本书中讨论这些协议，详细资料见[756]。

不暴露共享的共享秘密

上述方案有一个问题。当每个人聚到一起重构他们的秘密时，他们暴露了他们的共享。其它不必出现这种情况。如果共享秘密是私钥（例如对数字签名），那么 n 个共享者中的每一个都可以完成文件的一部分签名。在第 n 部分签名后，文件已经用共享的私钥签名，并且共享者中没有人了解任何其它人的共享。秘密能够重用是关键，并且你不必用可信的处理器去处理它。Yvo Desmedt 和 Yair Frankel 对这个概念进行了进一步的探索。

可验证的秘密共享

Trent 给 Alice、Bob、Carol 和 Dave 每人一部分（秘密），或至少他说他这样做了。他们中的任何人想知道他们是否有有效部分，唯一的办法是尝试着去重构秘密。也许 Trent 发给 Bob 一个假的共享，或者由于通信错误，Bob 偶然接收到一个坏的共享。可验证的秘密共享允许他们中的每个人分别验证他们有一个有效的共享，而不用重构这个秘密[558，1235]。

带预防的秘密共享方案

一个秘密被分给 50 个人，只要任何 10 个人在一起，就可以重构这个秘密。这样做是容易的。但是，当增加约束条件，要 20 人在一起才能恢复秘密同时要防止其他重构秘密时，我们能实现这同一秘密共享方案吗？是否多少人共享都没有问题？已经证明，我们能够做到[153]。

数学当然十分复杂，但其基本思想是每个人得到两个共享：一个“是”和一个“否”的共享。当重构秘密时，每个人提交他们的一个共享。他们提交的实际共享依赖于他们是否希望重构秘密。如果有 m 或更多个“是”共享和少于 n 个“否”共享，那么秘密能够被重构，否则，不能重构。

当然，如果没有“否”共享的人（假设他们知道是谁），没有任何事情能防止足够数量的“是”共享的人钻牛角尖，去重构秘密。但是在每个人提交他们的共享进入中心计算机的情况下，这个方案可行。

带除名的秘密共享

你正在安装你的秘密共享系统，现在你想解雇你的一名共享者。你可以安装没有那个人的新方案系统，但很费时。有多种方法处理这个系统，一旦有一个参与者变成不可信时，允许立即启用新的共享方案 [1004]。

3.8 数据库的密码保护

任何组织的成员数据库都是有价值的。一方面，你想把数据库分配给所有成员，他们互相通信，交换想法，互相邀请吃黄瓜、三明治。另一方面，如果把成员数据库分配给每个人，副本必定会落入保险商人之手和其他恼人的垃圾邮件供应者之手。

密码学能够改善这个问题。可以加密数据库，使它易于提取单个人的地址，而难于提取所有成员的邮件名单。

[550，549]的方案是直截了当的。选用一个单向 hash 函数和对称加密算法。数据库的每个记录有二个字段。索引字段是成员的姓，用单向 hash 函数进行运算。数据字段是全名和成员的地址，用姓作为密钥对数据字段它加密。除非你知道这个人的姓，否则你不可能解密数据字段。

搜索一个指定的姓是容易的。首先，对姓进行 hash 运算，并在数据库的索引字段中搜寻散列值。如果匹配，那么这个人的姓就在数据库中。如果有几个匹配，那么就有几个人同姓，最后，对每个匹配的项，用姓作为密钥解密出全名和地址。

在[550]中，作者采用这种系统对 6000 个西班牙动词的字典进行保护。加密只引起很小的性能降低。[549]附加的复杂性就是处理搜寻多个索引，但思想还是相同的。这个系统的主要问题是当你不知道怎么拼写他们的名字时，就不可能搜寻所要找的人。你可以试各种拼法，直到你找到正确的拼法为止，但是当你搜寻“Schneier”时，扫描所有以 “Sch”开头的名字并不实际的。

这种保护并不完善。某个特别固执的保险商人通过试所有可能的姓，就可能重构成员的数据库。如果他有电话数据库，他就可以把它作为一个可能的姓氏表来建立数据库。在计算机上做此事可能要花费几星期时间，但却是可行的。在垃圾邮件社会中，做这种工作更难，而且“更难” 很快会变成“太贵”。

文献[185]提供了另一种途径，允许统计数字编辑在加密的数据中。

第四章　中级协议

4.1 时间戳服务

在很多情况中，人们需要证明某个文件在某个时期存在。版权或专利争端即是谁有产生争议的工作的最早的副本，谁就将赢得官司。对于纸上的文件，公证人可以对文件签名，律师可以保护副本。如果产生了争端，公证人或律师可以证明某封信产生于某个时间。

在数字世界中，事情要复杂得多。没有办法检查窜改签名的数字文件。他们可以无止境地复制和修改而无人发现。在计算机文件上改变日期标记是轻而易举的事，没有人在看到数字文件后说："是的，这个文件是在 1952 年 12 月 4 日以前创建的。"

Bellcore 的 Stuart Haber 和 W.Scott Stornetta 考虑了这个问题[682，683，92]。他们认为数字时间标志协议有下列三条性质：

- 数据本身必须有时间标记，而不用考虑它所用的物理媒介。
- 必定不存在改变文件的 1 个比特而文件却没有明显变化。
- 必定不可能用不同于当前日期和时间的日期和时间来标记文件。

仲裁解决方法

这个协议需要 Trent 和 Alice，Trent 提供可信的时间标记服务，Alice 希望对文件加上时间标记：

（1）Alice 将文件的副本传送给 Trent。

（2）Trent 将他收到文件的日期和时间记录下来，并妥善保存文件的副本。

现在，如果有人对 Alice 所声明的文件产生的时间有怀疑，Alice 只要打电话给 Trent，Trent 将提供文件的副本，并证明他在标记的日期和时间接收到文件。

这个协议是可行的，但有些明显的问题。第一，没有保密性，Alice 不得不将文件的副本交给 Trent。在信道上窃听的任何人都可以读它。她可以对文件加密，但文件仍要放入 Trent 的数据库中，谁知道这个数据库有多安全？

第二，数据库本身将是巨大的。并且发送大量的文件给 Trent 所要求的带宽也是非常大。

第三，存在潜在错误。传送错误或 Trent 的中央计算机中某些地方的电磁炸弹引爆将使 Alice 声明的时间标志完全无效。

第四，可能有些运行时间标记业务的人并不像 Trent 那样诚实。也许 Alice 正在使用 Bob 的时间标记和 Taco Stand 系统。没有任何事情能阻止 Alice 和 Bob 合谋，他们可以用他们想要的任何时间对文件作时间标记。

改进的仲裁解决方法

单向 hash 函数和数字签名能够轻而易举地解决大部分问题：

（1）Alice 产生文件的单向 hash 值。

（2）Alice 将 hash 值传送给 Trent。

（3）Trent 将接收到 hash 值的日期和时间附在 hash 值后，并对结果进行数字签名。

（4）Trent 将签名的散列和时间标记送回给 Alice。

这种方法解决了除最后一个问题的所有问题。Alice 再也不用担心展示她的文件内容，因为 hash 值就足够了。Trent 也不用存储文件的副本（或者甚至 hash 值），这样，大量的存储要求和安全问题被解决了（记住，单向 hash 函数不需要密钥），Alice 可以马上检查她在第（4）步中接收到的对时间标记的 hash 值的签名。这样，她将马上发现在传送过程中的任何错误。这里唯一存在的问题是 Alice 和 Trent 仍然可以合谋来产生他们想要的任何时间标记。

链接协议

解决这个问题的一种方法是将 Alice 的时间标记同以前由 Trent 产生的时间标记链接起来。这些时间标记很可能是为其他人产生，而不是为 Alice 产生的。由于 Trent 预先不知道他所接收的不同时间标记的顺序，Alice 的时间标记一定发生在前一个时间标记之后。并且由于后面来的请求是与 Alice 的时间标记链接，那么她必须出现在前面。Alice 的请求正好夹在两个时间之间。

如果 A 表示 Alice，Alice 想要做时间标记的 Hash 值是 H_n，并且前一个时间标记是 T_{n-1}，那么协议如下：

（1）Alice 将 H_n 和 A 发送给 Trent。

（2）Trent 将如下消息送回给 Alice。

$$T_n = S_k（n，A，H_n,，T_n；I_{n-1}，H_{n-1}，T_{n-1}，\ L_n）$$

这里，L_n 是由下面的 hash 链接信息组成：

$$L_n =H（I_{n-1}，H_{n-1}，T_{n-1}，L_{n-1}）$$

S_K 表示信息是用 Trent 的私钥签名的。Alice 的名字标识她是请求的发起者，参数 n 表示请求的序号：这是 Trent 发布的第 n 个时间标志。参数 T_n 是时间，另外的信息是标识、源 hash 值、时间和 Trent 对以前文件做的时间标记的 hash 值。

（3）在 Trent 对下一个文件做时间标记后，他将那个文件的发起者的标识符 I_{n+1} 发送给 Alice。

如果有人对 Alice 的时间标记提出疑问，她只同她的前后文件的发起者 I_{n-1} 和 I_{n+1} 接触就行了。如果对她前后文件也有疑问，他们可以同 I_{n-2} 和 I_{n+2} 接触等等，每个人都能够表明他们的文件是在先来的文件之后和后来的文件之前打上时间标记的。

这个协议使 Alice 和 Trent 很难合谋去产生一文件的时间标记，使它不同于实际的时间标记。Trent 不可能为 Alice 顺填文件的日期。因为那样的话， Trent 就要预先知道在它之前是哪个文件的请求。即使他伪造那个文件，但也得知道在那个文件前来的是什么文件的请求，等等。由于时间标记必须嵌入到马上发布的后一文件的时间标记中，并且那个文件也已经发布了，他不可能倒填文件的日期。破坏这个方案的唯一的办法是在 Alice 的文件前后创建一虚构的文件链，该链足够的长以至于可以穷举任何人对时间标记提出的疑问。

分布式协议

人死后，时间标记就会丢失。在时间标记和质询之间很多事情都可能发生，以使 Alice 不可能得到 I_{n-1} 的时间标记的副本，这个问题可以通过把前面 10 个人的时间标记嵌入 Alice

的时间标记中得到缓解，并且将后面 10 个人的标识都发给 Alice。这样 Alice 就会有更大的机会找到那些仍有他们的时间标记的人。

按相似的方法，下面的协议与 Trent 一起实现分布式协议。

（1）用 H_n 作为输入， Alice 用密码上安全的伪随机数发生器产生一串随机值：

$$V_1，V_2，V_3，……，V_k$$

（2）Alice 将这些值的每一个看作其他人的 I 身份标识。她将 H_n 发送给他们中的每个人。

（3）他们中的每个人将日期和时间附到 hash 值后，对结果签名，并将它送回给 Alice。

（4）Alice 收集并存储所有的签名作为时间标记。

第（1）步中密码上安全的伪随机数发生器防止了 Alice 故意选取不可靠的 I 作为证人。即使她在她的文件中作些改变以便构造一组不可靠的 I，她用这种方式逃脱的机会也是很小的，hash 函数使 I 随机化了。Alice 不可能强迫他们。

这个协议是可行的，因为 Alice 伪造时间标记的唯一办法是使所有的 K 个人都与她合作。由于她在第（1）步中随机地选择 K 个人，因此防备这种攻击的可能性是很高的。社会越腐败，K 值就应越大。

另外，应该有一些机制来对那些不能马上返回时间标记的人进行处理。K 的一些子集都应是有效时间标记所要求的。其细节由具体的实现来决定。

进一步的工作

时间标记协议的进一步改进已在[92]中提出。作者利用二叉树来增加时间标记的数目，这个时间标记的数目依赖于一个给定的时间标记，以进一步减少某些人产生虚拟时间标记链的可能性。他们也建议在公共地方公布每天的时间标记的 hash 值，例如发表在报纸上。这类似于在分布协议中发送 Hash 值给随机的人。事实上，从 1992 年来时间标记已经出现在每星期日的《纽约时报》上了。

这些时间标记协议是取得了专利权的[684，685，686]。原隶属于 Bellcore 公司的 Surety 技术公司拥有这些专利并将数字公证系统推向市场以支持这些协议。在他们的第一版中，客户发出“证明”请求给中央协调服务器。下述的 Merkle 技术使用 hash 函数构造树[1066]，服务器构造由 hash 值构成的树，树的叶子是在给定的时间秒期间所有接收的请求，并且服务器把从它的叶子到树根的路径上的 hash 值的列表发回给每位请求者。客户软件把它存储在本地，并能为已经证明的任何文件发布一个数字公证的“证书”。这些树的根的序列由在多个储存库地点用电子手段获得的“全程有效记录”组成（也在 CD-ROM 上发布）。客户软件也包括一个“有效”函数，允许用户测试文件是否已经被准确地用其当前形式证明（对适当的树根通过查询储存库，并把它与从文件和它的证书中重新计算的适当的 hash 值进行比较）。需要 Surety Technologies 有关信息可与下面的地址联系：

1 Main St。，Chatham，NJ，07928；（201）701-0600；Fax：（201）701-0601

4.2 阈下信道

假设 Alice 和 Bob 被捕入狱。他将去男牢房，而她则进女牢房。看守 Walter 愿意让 Alice 和 Bob 交换消息，但他不允许他们加密。Walter 认为他们可能会商讨一个逃跑计划，因此，他想能够阅读他们说的每个细节。

Walter 也希望欺骗 Alice 和 Bob，他想让他们中的一个将一份欺诈的消息当作来自另一个的真实消息。Alice 和 Bob 愿意冒这种欺骗的危险，否则他们根本无法联络，而他们必须商讨他们的计划。为了完成这件事，他们不得不欺骗看守，并找出一个秘密通讯的方法。他们不得不建立一个阈下信道，即完全在 Walter 视野内的他们之间的一个秘密通信信道，即使消息本身并不包含秘密信息。通过交换完全无害的签名的消息，他们可以来回传送秘密信息，并骗过 Walter，即使 Walter 正在监视所有的通信。

一个简易的阈下信道可以是句子中单词的数目。句子中奇数个单词对应“1”，而偶数个单词对应“0”。因此，当你读这种仿佛无关的段落时，我已将消息“101”送给了在现场的我方的人。这种技术的问题在于它仅仅是密码（见 1.2 节）；没有密钥，安全性依赖于算法的保密性。

Gustavus Simmons 发明了传统数字签名算法中阈下信道的概念[1458，1473]。由于阈下消息隐藏在看似正常数字签名的文本中，这是一种迷惑人的形式。Walter 看到来回传递的签名的无害消息，但他完全看不到通过阈下信道传递的信息。事实上，阈下信道签名算法与通常的签名算法不能区别至少对 Walter 是这样。Walter 不仅不能读阈下信道消息，而且他也不知道阈下信道消息已经出现：

一般地，协议看起来像下面这样：

（1）Alice 产生一个无害消息，最好随机；

（2）用与 Bob 共享的秘密密钥，Alice 对这个无害信息这样签名，她在签名中隐藏她的阈下信息（这是阈下信道协议的内容，见 23.3 节）；

（3）Alice 通过 Walter 发送签名消息给 Bob；

（4）Walter 读这份无害的消息并检查签名，没发现什么问题，他将这份签了名的消息传递给 Bob；

（5）Bob 检查这份无害消息的签名，确认消息来自于 Alice；

（6）Bob 忽略无害的消息，而用他与 Alice 共享的秘密密钥，提取阈下消息。

怎样欺骗呢？Walter 不相信任何人，别的人也不相信他。他是可以阻止通讯，但他没法构造虚假信息。由于他没法产生任何有效的签名，Bob 将在第五步中检测出他的意图。并且由于他不知道共享密钥，他没法阅读阈下信息。更重要的是，他不知道阈下信息在那里。用数字签名算法签名后的消息消息与在签名中嵌入到签名中的阈下消息看上去没有不同。

Alice 和 Bob 之间的欺骗问题就更多。在阈下信道的一些实现中，Bob 需要从阈下信道读的秘密信息与 Alice 需要签名的无害信息是相同的。如果这样，Bob 能够冒充 Alice。他能对消息签名而声称该消息来源于她，而对此 Alice 无能为力。如果她要给他发送阈下消息，她不得不相信他不会滥用她的私钥。

别的阈下信道实现中没有这个问题。由 Alice 和 Bob 共享的秘密密钥允许 Alice 给 Bob 发送阈下信息，但这个密钥与 Alice 的私钥不同， 并不允许 Bob 对消息签名。Alice 也就不

必相信 Bob 不会滥用她的私钥了。

阈下信道的应用

阈下信道的最显见的应用是在间谍网中。如果每人都收发签名消息，间谍在签名文件中发送阈下信息就不会被注意到。当然，敌方的间谍也可以做同样的事。

用一个阈下信道，Alice 可以在受到威胁时安全地对文件签名。她可以在签名文件时嵌入阈下消息，说“我被胁迫”。别的应用则更为微妙，公司可以签名文件，嵌入阈下信息，允许它们在文档整个文档有效期内被跟踪。政府可以“标记”数字货币。恶意的签名程序可能泄露其签名中的秘密信息。其可能性是无穷的。

杜绝阈下的签名

Alice 和 Bob 互相发送签名消息，协商合同的条款。他们使用数字签名协议。然而，这个合同谈判是用来掩护 Alice 和 Bob 间谍活动的。当他们使用数字签名算法时，他们不关心所签名的消息。他们利用签名中的阈下信道彼此传送秘密信息。然而，反间谍机构不知道合同谈判以及签名消息的应用只是表面现象。因此人们创立了杜绝阈下的签名方案。这些数字签名方案不能被变更让其包含阈下信道。细节见[480，481]。

4.3 不可抵赖的数字签名

一般的数字签名能够被准确复制。这个性质有时是有用的，比如公开宣传品的发布。在其它时间，它可能有问题。想象一下数字签名的私人或商业信件。如果到处散布那个文件的许多拷贝，而每个拷贝又能够被任何人验证，这样可能会导致窘迫或勒索。最好的解决方案是数字签名能够被证明是有效的，但没有签名者的同意，接收者不能把它给第三方看。

Alice 软件公司发布 DEW 软件（Do-Everything-Word）。为了确信软件中不带病毒，他们在每个拷贝中包括一个数字签名。然而，他们只想软件的合法买主能够验证数字签名，盗版者则不能。同时，如果 DEW 拷贝中发现有病毒，Alice 软件公司应该不可能否认一个有效的数字签名。

不可抵赖签名[343，327]适合于这类任务。类似于通常的数字签名，不可抵赖签名依赖于签名的文件和签名者的私钥。但不象通常的数字签名，不可抵赖签名没有得到签名者同意就不能被验证。虽然对这些签名，用像“不可改变的签名”一类的名称更好，但这个名称的由来是如果 Alice 被强迫承认或抵赖一个签名（很可能在法庭上），她不可能不实地否认她的真实签名。

数学描述是复杂的，但其基本思想是简单的：

（1）Alice 向 Bob 出示一个签名；

（2）Bob 产生一个随机数并送给 Alice；

（3）Alice 利用随机数和其私钥进行计算，将计算结果送给 Bob。Alice 只能计算该签名是否有效。

（4）Bob 确认这个结果。

也有另外的协议，以便 Alice 能够证明她没有对文件签名，又不能不实地否认签名。

Bob 不能转而让 Carol 确信 Alice 的签名是有效的，因为 Carol 不知道 Bob 的数字是随机数。Bob 很容易在文件上完成这个协议，而不用 Alice 的任何帮助，然后将结果出示给 Carol。Carol 只有在她与 Alice 本人完成这个协议后才能确信 Alice 的签名是有效的。现在或许没有什么意义，但是一旦你明白在 23.4 节的数学原理，就会显而易见了。

这个解决方法并不完美，Yvo Desmedt 和 Moti Yung 研究表明，在某些情况下，Bob 让 Carol 确信 Alice 的签名有效是可能的[489]。

例如，Bob 买了 DEW 的一个合法拷贝，他能在任何时候验证软件包的签名。然后，Bob 使 Carol 相信他是来自于 Alice 软件公司的销售商。他卖给她一个 DEW 的盗版。当 Carol 试图验证 Bob 的签名时，他同时要验证 Alice 的签名。当 Carol 发给 Bob 随机数时，Bob 然后把它送给 Alice。当 Alice 响应后，Bob 就将响应送给 Carol。于是 Carol 相信她是该软件的合法买主，尽管她并不是。这种攻击是象棋大师问题的一个例子，在 5.2 节中有详细的讨论。

即使如此，不可抵赖的签名仍有许多应用，在很多情况中，Alice 不想任何人能够验证她的签名。她不想她的个人通信被媒体核实、展示并从文中查对，或者甚至在事情已经改变后被验证。如果她对她卖出的信息签名，她不希望没有对信息付钱的那些人能够验证它的真实性。控制谁验证她的签名是 Alice 保护她的个人隐私的一种方法。

不可抵赖签名的一种变化是把签名者与消息之间的关系与签名者与签名之间的关系分开[910]。在这种签名方案中，任何人能够验证签名者实际产生的签名，但签名者的合作者还需要验证该消息的签名是有效的。

相关的概念是“受托不可抵赖签名”[1229]。设想 Alice 为 Toxins 公司工作，并使用不可抵赖的签名协议发送控告文件给报纸。Alice 能够对报社记者验证她的签名，但不向其它任何人验证签名。然而执行总裁 Bob 怀疑文件是 Alice 提供的，他要求 Alice 执行否认协议来澄清她的名字，Alice 拒绝了。Bob 认为 Alice 不得不拒绝的唯一理由是她有罪，于是便解雇她。

除了否认协议只能由 Trent 执行外，受托不可抵赖签名类似不可抵赖签名。Bob 不能要求 Alice 执行否认协议，只有 Trent 能够。如果 Trent 是法院系统，那么他将只执行协议去解决正式的争端。

4．4 指定的确认者签名

Alice 公司销售 DEW 软件的生意非常兴隆，事实上，Alice 验证不可抵赖签名的时间比编写新的功能部件的时间更多。

Alice 很希望有一种办法可以在公司中指定一个特殊的人负责对整个公司的签名验证。Alice 或任何其他程序员能够用不可抵赖协议对文件签名。但是所有的验证都由 Carol 处理。

结果表明，用指定的确认人签名[333，1213]是可行的。Alice 能够对文件签名，而 Bob 相信签名是有效的，但他不能使第三方相信。同时，Alice 能够指定 Carol 作为她签名后的确认人。Alice 甚至事先不需要得到 Carol 的同意，她只需要 Carol 的公开密钥。如果 Alice 不在家，已经离开公司，或者突然死亡了，Carol 仍然能够验证 Alice 的签名。

指定的确认人签名是标准的数字签名和不可抵赖签名的折中。肯定有一些场合 Alice 可能想要限制能验证她的签名的人。另一方面，如果 Alice 完全控制则破坏了签名的可实施性：Alice 可能在确认或否认方面拒绝合作，她可能声称用于确认或否认的密钥丢失了，或者她可能正好身份不明。指定的确认人签名让 Alice 既能保护不可抵赖签名同时又不让她滥用这种保护。Alice 可能更喜欢那种方式：指定的确认人签名能够帮助她防止错误的应用，如果她确实丢失了密钥，可以保护她，如果她是在渡假、在医院、甚至死了，也可以插手干预。

这种想法有各种可能的应用。Carol 能够把她自己作为公证人公开。她能够在一些地方的一些目录中发布她的公开密钥，人们能够指定她作为他们签名的确认人。她向大众收取少量的签名确认费用，使她可以生活得很好。

Carol 可能是版权事务所、政府机构、其它的很多事物。这个协议允许组织机构把签署文件的人同帮助验证签名的人分开。

4.5 代理签名

指定确认人签名允许签名者指定其他某个人来验证他的签名。例如，Alice 需要到一些地方进行商业旅行，这些地方不能很好地访问计算机网络（例如，到非洲丛林）。或者也许她在大手术后，无能为力。她希望接受一些重要的电子邮件，并指示她的秘书 Bob 作相应的回信。Alice 在不把她的私钥给 Bob 的情况下，该如何让 Bob 行使她的消息签名的权利呢？

代理签名是一种解决方案[1001]。Alice 可以给 Bob 代理，这种代理具有下面的特性：

- 可区别性。任何人都可区别代理签名和正常的签名。
- 不可伪造性。只有原始签名者和指定的代理签名者能够产生有效的代理签名。
- 代理签名者的不符合性。代理签名者必须创建一个能检测到是代理签名的有效代理签名。
- 可验证性。从代理签名中，验证者能够相信原始的签名者认同了这份签名消息。
- 可识别性。原始签名者能够从代理签名中识别代理签名者的身份。
- 不可抵赖性。代理签名者不能否认他创立的且被认可的代理签名。

在某些情况中，需要更强的可识别性形式，即既任何人都能从代理签名中确定代理签名者的身份。基于不同的数字签名方案的代理签名体制在[1001]中描述。

4.6 团体签名

David Chaum 在[330]中提出了下述问题：

一个公司有几台计算机，每台都连在局域网上。公司的每个部门有它自己的打印机（也连在局域网上），并且只有本部门的人员才被允许使用他们部门的打印机。因此，打印前，必须使打印机确信用户是在那个部门工作的。同时，公司想保密，不可以暴露用户的姓名。然而，如果有人在当天结束时发现打印机用得太频繁，主管者必须能够找出谁滥用了那台打印机，并给他一个帐单。

对这个问题的解决方法称为团体签名。它具有以下特性：

——只有该团体内的成员能对消息签名；

——签名的接收者能够证实消息是该团体的有效签名。

——签名的接收者不能决定是该团体内哪一个成员签的名；

——在出现争议时，签名能够被“打开”， 以揭示签名者的身份。

具有可信仲裁者的团体签名

本协议使用可信仲裁者：

（1）Trent 生成一大批公开密钥/私钥密钥对，并且给团体内每个成员一个不同的唯一私钥表。在任何表中密钥都是不同的（如果团体内有 n 个成员，每个成员得到 m 个密钥对，那么总共有 n*m 个密钥对）。

（2）Trent 以随机顺序公开该团体所用的公开密钥主表。Trent 保持一个哪些密钥属于谁的秘密记录。

（3）当团体内成员想对一个文件签名时，他从自己的密钥表中随机选取一个密钥。

（4）当有人想验证签名是否属于该团体时，只需查找对应公开钥主表并验证签名。

（5）当争议发生时，Trent 知道哪个公钥对应于哪个成员。

这个协议的问题在于需要可信的一方。Trent 知道每个人的私钥因而能够伪造签名。而且，m 必须足够长以避免试图分析出每个成员用的哪些密钥。

Chaum 在论文中列举了许多别的协议[330]，其中有些协议 Trent 不能够伪造签名，而另一些协议甚至不需要 Trent。另一个协议[348]不仅隐藏了签名者的身份，而且允许新成员加入团体内。在[1230]中还描述了另一个协议。

4.7 失败-终止 数字签名

让我们假想 Eve 是非常强劲的敌人。她有巨大的计算机网络和很多装满了 Cray 计算机的屋子（计算机能力比 Alice 大许多量级）。这些计算机昼夜工作试图破译出 Alice 的私钥，最终成功了。Eve 现在就能够冒充 Alice，随意地在文件上伪造她的签名。

由 Birgit Pfitzmann 和 Michael Waidner[1240]引入的失败-终止数字签名就能避免这种欺诈。如果 Eve 在穷举攻击后伪造 Alice 的签名，那么 Alice 能够证明它们都是伪造的。如果 Alice 对文件签名，然后否认签名，声称是伪造的，法院能够验证它不是伪造的。

失败-终止签名的基本原理是：对每个可能的公开密钥，许多可能的私钥和它一起工作。这些私钥中的每一个产生许多不同的可能的签名。然而，Alice 只有一个私钥，只能计算一个签名。Alice 并不知道别的任何私钥。

Eve 试图破解出 Alice 的秘密密钥。（在这种情况下，Eve 也可能是 Alice，试图为她自己计算第二个私钥。）她收集签名消息，并且利用她的 Cray 计算机阵列，试图恢复出 Alice 的私钥。即使 Eve 能够恢复出一个有效的私钥，但因为有许多可能的私钥，因而这个私钥可能是不同的一个。Eve 恢复出合适的私钥的概率非常小，可以忽略不计。

现在，当 Eve 利用她产生的私钥伪造签名时，它将不同于 Alice 本人对文件的签名。当 Alice 被传到法院，对同一消息和公开密钥她能够生成两个不同的签名（对应于她的私钥以

及 Eve 产生的私钥）以证明是伪造的。另一方面，如果 Alice 不能产生两个不同的签名，这时没有伪造，Alice 就要对她的签名负责。

这个签名方案避免了 Eve 通过巨大的计算能力来破译 Alice 的签名方案。它对下面这种更有可能发生的攻击却无能为力：当 Mallory 闯入 Alice 的住宅并偷窃她的私钥或者 Alice 签署了一个文件然后却丢失了她的私自钥时。为了防止前一种攻击，Alice 应该给她自己买条好的看门狗，这种事情已超出了密码学的范围。

其它的关于失败-终止签名的理论和应用能在[1239，1241，730，731]中找到。

4.8 用加密数据计算

Alice 想知道某个函数 f（X）对某些特殊的 x 值的解。不幸的是，她的计算机坏了，Bob 愿意为她计算 f（x），但 Alice 又不想让 Bob 知道她的 x。怎样做 Alice 才能在不让 Bob 知道 x 的情况下为她计算 f（x）呢？

这是加密数据计算的一般问题，亦称“对先知隐藏信息”问题。（Bob 是先知，他回答问题。）对某些函数来说有许多方法能够解决这个问题，在 23.6 节中讨论。

4.9 比特承诺

Alice，这位令人惊异的魔术天才，正表演关于人类意念的神秘技巧。她将在 Bob 选牌之前猜中 Bob 将选的牌！注意 Alice 在一张纸上写出她的预测。Alice 很神秘地将那张纸片装入信封中并封上。就在人们吃惊之时 Alice 将封好的信封随机地递给一观众。“取一张牌，Bob，任选一张”。他看了看牌而后将之出示给 Alice 和观众。是方块 7。现在 Alice 从观众那里取回信封，并撕开它。在 Bob 选牌之先写的预测，也是：方块 7！全场欢呼！

这个魔术的要点在于，Alice 在戏法的最后交换了信封。然而，密码协议能够提供防止这种花招的方法。这为什么有用？下面是一个更实际的故事：

股票经纪人 Alice 想说服投资商 Bob 她的选取赢利股票的方法很不错。

Bob 说：“给我选 5 支股票，如果都赢利，我将把生意给你。”

Alice 说：“如果我为你选了 5 只股票，你可以自己对他们投资，而不用给我付款。我为什么不向你出示我上月选的股票呢？”

Bob：“我怎样知道你在了解了上月股票的收益后没改变你上月选择的股票呢？如果你现在告诉我你选的股票，我就可以知道你不能改变他们。在我买你的方法以前我不在这些股票中投资。相信我。”

Alice：“我宁愿告诉你我上月选择的股票。我不会变，相信我。”

Alice 想对 Bob 承诺一个预测（即 1bit 或 bit 序列），但直到某个时间以后才揭示她的预测。而另一方面，Bob 想确信在 Alice 承诺了她的预测后，她没有改变她的想法。

使用对称密码算法的比特承诺

这个比特承诺协议使用对称密码：

（1）Bob 产生一个随机比特串 R，并把它发送给 Alice。

R

（2）Alice 生成一个由她想承诺的比特 b 组成的消息（b 实际上可能是几个比特），以及 Bob 的随机串。她用某个随机密钥 K 对它加密，并将结果送回给 Bob。

E_K（R，b）

这是这个协议的承诺部分，Bob 不能解密消息，因而不知道比特为何。

当到了 Alice 揭示她的比特的时候，协议继续：

（1）Alice 发送密钥给 Bob；

（2）Bob 解密消息以揭示比特。他检测他的随机串以证实比特的有效性。

如果消息不包含 Bob 的随机串，Alice 能够秘密地用一系列密钥解密她交给 Bob 的消息，直到找到一个给她的 bit，而不是她承诺的比特。由于比特只有两种可能的值，她只需试几次肯定可以找到一个。Bob 的随机串避免了这种攻击，她必须能找到一个新的消息，这个消息不仅使她的比特反转，而且使 Bob 的随机串准确地重新产生。如果加密算法好，她发现这种消息的机会是极小的。Alice 不能在她承诺后改变她的比特。

使用单向函数的比特承诺

本协议利用单向函数：

（1）Alice 产生两个随机比特串，R_1 和 R_2。

R_1，R_2

（2）Alice 产生消息，该消息由她的随机串和她希望承诺的比特（实际上可能是几比特）组成。

（R_1，R_2，b）。

（3）Alice 计算消息的单向函数值，将结果以及其中一个随机串发送给 Bob。H（R_1，R_2，b），R_1。

这个来自 Alice 的传送就是承诺证据。Alice 在第（3）步使用单向函数阻止 Bob 对函数求逆并确定这个比特。

当到了要 Alice 出示她的比特的时候，协议继续：

（4）Alice 将原消息发给 Bob。

（R_1，R_2，b）

（5）Bob 计算消息的单向函数值，并将该值及 R_1 与原先第（3）步收到的值及随机串比较。如匹配，则比特有效。

这个协议较前面一个的优点在于 Bob 不必发送任何消息。Alice 送给 Bob 一个对比特承诺的消息，以及另一揭示该 bit 的消息。

这里不需要 Bob 的随机串，因为 Alice 承诺的结果是对消息进行单向函数变换得到的。Alice 不可能欺骗，并找到另一个消息（R_1，R_2’，b’），满足

H（R_1，R_2’，b’）=（R_1，R_2，b）

通过发给 Bob R_1，Alice 对 b 的值作了承诺。如果 Alice 不保持 R_2 是秘密的，那么 Bob 能够计算 H（R_1，R_2，b’）和（R_1，R_2，b），并比较哪一个等于他从 Alice 那里接收的。

使用伪随机序列发生器的比特承诺

本协议是更容易的[1137]：

（1）Bob 产生随机比特串，并送给 Alice。

R_B

（2）Alice 为伪随机比特发生器生成一个随机种子。然后，对 Bob 随机比特串中的每一比特，她回送 Bob 下面两个中的一个：

（a）当 Bob 比特为 0，发生器的输出，或者

（b）如果 Bob 的比特为 1，发生器输出与她的比特的异或。

当到了 Alice 出示她的比特的时候，协议继续：

（3）Alice 将随机种子送给 Bob。

（4）Bob 完成第（2）步以确认 Alice 的行动是合理的。

如果 Bob 的随机比特串足够长，伪随机比特发生器不可预测，这时 Alice 就无有效的方法进行欺诈。

模糊点

Alice 送给 Bob 以便对比特承诺的这些串有时又叫模糊点。一个模糊点是一个比特序列，虽然在协议中没有说明它为什么必须这样，正如 Gilles Brassard 所说的，“只要是合理存在的就是有用的”[236]。模糊点有下面四个特性：

1.Alice 能够对模糊点承诺，通过承诺模糊点来承诺一个比特。

2.Alice 能够打开她所承诺的任何模糊点。当她打开模糊点时，她能让 Bob 相信在她对模糊点承诺时她所承诺的比特值。因此，她不能选择把任何模糊点作为 0 或 1 打开。

3.Bob 不知道 Alice 如何打开承诺了的但尚未打开的模糊点。即使 Alice 打开别的模糊点之后，也是如此。

4.模糊点所带的信息除 Alice 承诺的比特外，不再有任何信息。模糊点本身，连同 Alice 承诺和开启模糊点的过程，与 Alice 希望对 Bob 保密的别的东西不相关。

4.10 公平的硬币抛掷

是 Joe Kilian[831] 讲故事的时候了：

Alice 和 Bob 想抛掷一个公平的硬币，但又没有实际的物理硬币可抛。Alice 提出一个用思维来抛掷公平硬币的简单方法。“首先，你想一个随机比特，然后我再想一个随机比特，我们将这两个比特进行异或。”Alice 建议道。

“但如果我们中有人不随机抛掷硬币怎么办呢？”Bob 问道。

“这无关紧要，只要这些比特中的一个是真正随机的，它们之异或应该也是真正随机的。”Alice 这样回答。经过思考后，Bob 同意了。

没过多久，Alice 和 Bob 碰到一本关于人工智能的书，这本书被丢弃在路旁。 优秀公

民 Alice 说：“我们中有一个必须拣起这本书，并找到一个合适的垃圾箱。”Bob 同意并提议用抛币协议来决定谁必须将这本书扔掉。

“如果最后的比特是‘0’，那么你必须拣取那本书，如果是‘1’，那我必须那样做。”Alice 说。“你的比特是什么？”

Bob 答道：‘1’。

“为什么，我的也是 1”Alice 顽皮地说“我猜想今天不是你的幸运日。”

不用说，这个抛币协议有严重的缺陷，真正随机的比特 x 与任意独立分配的比特 y 异或仍得到真正随机的比特。Alice 的协议不能保证两个比特是独立分布的。事实上，不难验证不存在能让两个能力无限的团体公平抛币的思维协议。Alice 和 Bob 在收到来自密码学方面的一个无名的研究生的一封信后才走出了困境。信上的信息抽象得对任何人都不会有用，但随信用的信封却是随手可得的。

接下来，Alice 和 Bob 希望抛币，他们对原协议版本进行了修改。首先，Bob 确定一个比特，但这次他不立即宣布，只是将它写在纸上，并装入信封中。接下来，Alice 公布她选的比特。最后，Alice 和 Bob 从信封中取出 Bob 的比特并计算随机比特。只要至少一方诚实地执行协议，这个比特的确是真正随机的。Alice 和 Bob 有了这个可以工作的协议，密码学家梦想的社会关系实现了，他们从那以后过得很愉快。

那些信封很像比特承诺模糊点。当 Manuel Blum 通过调制解调器引入抛掷公平硬币问题时[194]，他利用比特承诺协议解决了此问题：

（1）Alice 利用在 4.9 节中所列的任一个比特承诺方案，对一个随机比特承诺。

（2）Bob 试图去猜测这个比特。

（3）Alice 出示这个比特给 Bob，如果 Bob 正确地猜出这个比特，他就赢得了这次抛币。

一般地，我们需要一个具有如下性质的协议：

——Alice 必须在 Bob 猜测之前抛币；

——在听到 Bob 的猜测后，Alice 必须不能再抛掷；

——Bob 在猜测之前必须不能知道硬币怎么落地的。

有几种方法可用来实现具有这些性质的协议。

采用单向函数的抛币协议

如果 Alice 和 Bob 对使用一个单向函数达成一致意见，协议非常简单：

（1）Alice 选择一个随机数 x，她计算 y=f（x），这里 f（x）是单向函数；

（2）Alice 将 y 送给 Bob；

（3）Bob 猜测 x 是偶数或奇数，并将猜测结果发给 Alice；

（4）如果 Bob 的猜测正确，抛币结果为正面;如果 Bob 的猜测错误，则抛币的结果为反面。Alice 公布此次抛币的结果，并将 x 发送给 Bob；

（5）Bob 确信 y=f（x）。

此协议的安全性取决于单向函数。如果 Alice 能找到 x 和 x’，满足 x 为偶而 x'为奇，且 y=f（x ）=f（x'），那么她每次都能欺骗 Bob。f（x）的没有意义的比特也必须与 x 不相关。否则，Bob 至少某些时候能够欺骗 Alice。例如，如果 x 是偶数，f（x）产生偶数的次数占

75%，Bob 就有优势。（有时没有什么意义的比特不是在这个应用中使用的最好的比特，因为它可能更易于计算）

采用公开密钥密码的抛币协议

这个协议既可与公开密钥密码又可与对称密码一起工作。其唯一要求是算法满足交换律，即：

$$D_{k_1}\left(E_{k_2}\left(E_{k_1}(M)\right)\right)=E_{k_2}(M)$$

一般地，对称算法中这个特性并不满足，但对某些公开密钥算法是正确的（例如，有相同模数的 RSA 算法）。协议如下：

（1）Alice 和 Bob 都产生一个公开/私钥密钥对。

（2）Alice 产生两个消息，其一指示正面，另一个指示反面。这些消息中包含有某个唯一的随机串，以便以后能够验证其在协议中的真实性。Alice 用她的公开密钥将两个消息加密，并以随机的顺序把他们发给 Bob，即

$E_A（M_1），E_A（M_2）$

（3）Bob 由于不能读懂其中任意一消息，他随机地选择一个。他用他的公开密钥加密并回送给 Alice，即

$E_B（E_A（M））$。

M 是 M_1 或 M_2。

（4）Alice 由于不能读懂送回给她的消息，就用她的私钥解密并回送给 Bob，即

$D_A（E_B（E_A（M））=E_B（M_1）$ 如果 $M = M_1$ 或

$E_B（M_2）$如果 $M = M_2$

（5）Bob 用他的私钥解密消息，得到抛币结果。他将解密后的消息送给 Alice

$D_B（E_B（M_1））= M_1$ 或 $D_B（E_B（M_2））= M_2$

（6）Alice 读抛币结果，并验证随机串的正确性。

（7）Alice 和 Bob 出示他们的密钥对以便双方能验证对方没有欺诈。

这个协议是自我实施的。任意一方都能即时检测对方的欺诈，不需要可信的第三方介入实际的协议和协议完成后的任何仲裁。让我们试图欺诈，看看协议是如何工作的。

如果 Alice 想欺骗，强制为正面，她有三种可能的方法影响结果。首先，她可以在第（2）步中加密两个“正面”的消息。在第（7）步 Alice 出示她的密钥时，Bob 就可以发现这种欺骗。第二种方法，Alice 在第（4）步时用一些其它的密钥解密消息，将产生一些乱七八糟的无用信息，Bob 可在第（5）步中发现。第三种方法，Alice 可在第（6）步中否认消息的有效性，当在第（7）步中 Alice 不能证明消息无效时，Bob 就可以发现。当然，Alice 可以在任何一步拒绝参与协议，那样，Alice 欺骗 Bob 的企图就显而易见了。

如果 Bob 想欺骗并强制为“反面”，他的选择性不大。他可以在第（3）步中不正确地加密一个消息，但 Alice 在第（6）步查看最终的消息时就可以发现它。他可以在第（5）步中进行不适当的操作，但这也会导致乱七八糟的无用信息，Alice 可在第（6）步中发现。他

可以声称由于 Alice 那方面的欺诈使他不能适当地完成第（5）步的操作，但这种形式的欺诈能在第（7）步中发现。最后，他可能在第（5）步中给 Alice 一个“反面”的信息，而不管他解密获得的信息是什么，但 Alice 能在第（6）步中立即检查消息的真实性。

掷币入井协议

注意到在所有这些协议中，Alice 和 Bob 不能同时知道掷币的结果。每个协议有一个点，在这个点上其中一方（如开始的两个协议中的 Alice，最后一个协议中的 Bob）知道掷币结果，但不能改变它。然而，这一方能推迟向另一方泄露结果。这被称作掷币入井协议。设想一口井，Alice 在井的旁边，而 Bob 远离这口井。Bob 将币扔进井里去，币停留在井中，现在 Alice 能够看到井中的结果，但她不能到井底去改变它。Bob 不能看到结果，直到 Alice 让他走到足够近时，才能看到结果。

采用抛币的密钥生成

这个协议的实际应用是会话密钥生成。掷币协议能让 Alice 和 Bob 产生随机会话密钥，以便双方都不能影响密钥生成的结果。假定 Alice 和 Bob 加密他们的交换，这个密钥生成方法在存在窃听时也是安全的。

4.11 智力扑克

这是一个类似于公平硬币抛掷协议的协议，它允许 Alice 和 Bob 通过电子邮件打扑克。Alice 在这里不是地生成和加密两个消息，一个 “正面”和一个“反面”，她要生成 52 个消息 M_1，M_2，……，M_{52}，每个代表一副牌中的一张牌。Bob 随机选取 5 张牌，用他的公开密钥加密，然后回送给 Alice。Alice 解密消息并回送给 Bob，Bob 解密它们以确定他的一手牌。然后，当 Bob 接收到 Alice 发送的消息，他随机地选择另外 5 个消息，并发给 Alice，Alice 解密它们，并且它们变成她的一手牌。在游戏期间，可通过重复这些过程来为任意一方发其它的牌。在游戏结束时，Alice 和 Bob 双方出示他们的牌和密钥对使得任意一方确信另一方没有作弊。

三方智力扑克

人较多时玩牌会更有趣。基本的智力扑克协议可以很容易地扩展到三个或更多个玩牌者。在这种情况下，密码算法也必须是可交换的。

（1）Alice、Bob 和 Carol 都产生一个公钥/私钥密钥对。

（2）Alice 产生 52 个消息，每个代表一副牌中的一张牌。这些消息应包含一些唯一的随机串，以便她能在以后验证它们在协议中的真实性。Alice 用她的公钥加密所有这些消息，并将它们发送给 Bob。

$$E_A（M_n）。$$

（3）Bob，由于不能阅读任何消息，他随机地选择 5 张牌。他用他的公钥加密，并把它们回送给 Alice。

E_B（E_A（M_n））。

（4）Bob 将余下的 47 张牌送给 Carol。

E_A（M_n）。

（5）Carol，由于不能阅读任何消息，也随机选 5 个消息。她用她的公钥加密，并把它们送给 Alice。

E_C（E_A（M_n））。

（6）Alice 也不能阅读回送给她的消息，她用她的私钥对它们解密，然后送给 Bob 或 Carol（依据来自谁而定）。

$$D_A（E_B（E_A（M_n））） = E_B（M_n）$$

$$D_A（E_C（E_A（M_n））） = E_C（M_n）$$

（7）Bob 和 Carol 用他们的密钥解密并获得他们的牌。

$$D_B（E_B（M_n）） = M_n$$

$$D_C（E_C（M_n）） = M_n$$

（8）Carol 从余下的 42 张牌中随机取 5 张，把它们发送给 Alice。

E_A（M_n）。

（9）Alice 用她的私钥解密消息获得她的牌。

$$D_A（E_A（M_n））=M_n$$

（10）在游戏结束时，Alice，Bob 和 Carol 都出示他们的牌以及他们的密钥，以便每人都确信没有人作弊。

其它的牌可以用同样的方式处理。如果 Bob 或 Carol 想要牌，任何一个人能够取被加密的牌，并和 Alice 一起履行该协议。如果 Alice 想要一张牌，当前得到牌的任何人都随机地发给她一张牌。

在理想情况下第（10）步是不必要的。协议结束后，不应该要求所有选手都出示他们的牌；只有那些没有出完牌的人被要求。由于第（10）步是只设计来抓住骗子的部分，因而也可能有改进。

在扑克中，人们只对赢家是否欺骗感兴趣。只要他们仍然失败，其它每个人也能进行他们想要的欺骗。（事实上，这确实是不对的。当失败时，可以收集其它牌手的玩牌风格的数据）。让我们看看不同选手赢牌的情况。

如果 Alice 赢了，她出示她的牌和她的密钥。Bob 能够用 Alice 的私钥确认 Alice 是合法地进行了第（2）步——52 个消息的每个都对应一张不同的牌。Carol 通过用 Alice 的公钥加密牌，并验证是与她在第（8）步中送给 Alice 的加密消息相同，从而确认 Alice 没有对出的牌撒谎。

如果 Bob 或 Carol 赢了，赢牌者将出示他们的牌以及密钥。Alice 可以通过检查她的随机串来确信这些牌是合法的。她也能确信通过用赢家的公钥对牌加密的牌是发的牌，并验证是与她在第（3）步或第（5）步中收到的加密消息相同。

当防止恶意的牌手间串通时，这个协议便不安全了。Alice 和别的牌手可以有效地联合对付第三方，可以在不引起怀疑的情况下骗取其所有东西。因此，每次检查牌手出的牌中的随机串以及所有的密钥是重要的。如果你与两个从未出示他们牌的人坐在虚拟桌子上，且其

中一个为发牌者（上述协议中的 Alice）时，你就应该停止玩了。

对扑克协议的攻击

密码学家已经证明，如果使用 RSA 算法，那么这些扑克协议会泄漏少量的信息 [453，573]。具体地，如果牌的二进制表示是二次方程的残数（见 11.3 节），那么牌的加密亦为二次方程的残数。这个特性可被用来标记某些牌——比如，所有的“A”。虽然不能泄露许多牌，但在诸如扑克游戏中，在最后即便是一个微小的比特信息也会有用。

Shafi Goldwasser 和 Silvio Micali[624]设计了一个两人玩的智力扑克游戏协议，它解决了这个问题，但由于其复杂性，使其太理论化而不能实用。在[389]中设计了消除信息泄漏问题的通用 n 方扑克协议。

别的对扑克协议的研究可在[573，1634，389]中找到。一个允许牌手不出示他们的牌的复杂协议可在[390]中找到。Don Coppersmith 讨论了在利用 RSA 算法的智力扑克游戏中两种作弊的方法。

匿名密钥分配

在人们利用这个协议通过调制解调器玩扑克是不太可能的时候，Clarles Pfleeger 讨论了这样一种情况，使得这类协议迟早有用。

考虑密钥分配问题。如果我们假定人们不能生成他们自己的密钥（它们必须为某种形式，或必须被某组织签名，或类似的要求）。我们必须设置密钥分配中心（KDC），用来生成和分配密钥。问题是我们必须找出一些密钥分配方法使得（包括服务器）没有人知道谁得到了什么密钥。

下面的协议解决了这个问题：

（1）Alice 生成一个公钥/私钥密钥对。对这个协议，她保持这两个密钥秘密。

（2）KDC 产生连续的密钥流。

（3）KDC 用它自己的公钥，逐个地将这些密钥加密。

（4）KDC 逐个地将这些加密后的密钥传送到网上。

（5）Alice 随机选择一个密钥。

（6）Alice 用她的公钥加密所选的密钥。

（7）Alice 等一段时间（要足够长使得服务器不知道她选择了哪个密钥），将这个双重加密的密钥送回 KDC。

（8）KDC 用它的私钥解密双重加密的密钥，得到一个用 Alice 的公开密钥加密的密钥。

（9）服务器将此加密密钥送给 Alice。

（10）Alice 用她的私钥解密这个密钥。

Eve 在这个协议过程中也不知道 Alice 选择了什么密钥。她在第（4）步看到了连续密钥流通过。当 Alice 在第（7）步将密钥送回给服务器时，是用她的公开密钥加密的，而公开密钥在协议期间也是秘密的。Eve 没法将它与密钥流关联起来。当服务器在第（9）步将密钥送回给 Alice 时，也是用 Alice 的公开密钥加了密的。仅当 Alice 在第（10）步解密密钥时，才知道密钥。

如果你用 RSA，这个协议以每个 1 比特的速度泄漏信息。它又是二次方程的残数。如果你准备用这种方式分配密钥，必须确保泄露是无关紧要的。来自 KDC 的密钥流也必须足够长，以阻止穷举攻击。当然，如果 Alice 不信任 KDC，那么她就不应该从 KDC 得到密钥。恶意的 KDC 可以预先记录它所生成的所有密钥。然后，它能搜索所有的密钥，决定哪一个是 Alice 的。

这个协议也假定 Alice 行为正当。利用 RSA 算法，她能够做其它事情来得到更多的信息。在我们的这个方案中，这不成其为问题，但在其它环境可能存在问题。

4.12 单向累加器

Alice 是 Cabal 公司的一个成员，有时候，她必须在光线暗淡的旅馆与其他成员会晤。问题是旅馆的光线非常暗，以至于她难于知道桌子对面的人是否也是他们的成员。

Cabal 公司可以选择几种解决方案。每个成员可以携带一个成员名单，这有两个问题。一是现在每人必须携带一个大的数据库，二是他们必须很好地保护成员名单。另一种选择，一个值得信任的秘书能够发布数字签名的身份卡。这样做增加了让外来者验证成员的好处。（例如，在本地食品店打折），但是它需要可信任的秘书，在 Cabal 公司没有人能够被信任到那种程度。

新的解决方案是使用叫做“单向累加器”的东西[116]。除了可交换外，它类似单向 hash 函数。也就是说，用任何顺序对成员数据库进行 hash，都得到相同的值，这是可能的。而且，把成员加入到 hash 中，得到新的 hash，它也是与顺序无关的。

那么，这儿是 Alice 做的事情。她计算除她自己外的每个成员名字的累加和。然后，她把那个值与她的名字存在一起。Bob 和其他每个成员都做和 Alice 一样的事。现在，当 Alice 和 Bob 在光线暗淡的旅馆会面时，他们简单的互相交换累加值和名字，Alice 确信 Bob 的名字加上他的累加值等于 Alice 的名字加上她的累加值。Bob 做同样的事情。现在他们两人知道另一个是公司成员。同时，没有人能够知道任何其他人的身份。

甚至更好，非会员能知道每个人的累加值。现在 Alice 能够对非会员验证他的会员资格（也许，在他们的本地反间谍商店为会员打折），非会员不可能计算出全部会员资格的名单。

只要到处发送新会员的名字就可把新会员加入到累加值中。不幸的是，删除会员的唯一方法是给每个会员发送新名单并让他们重新计算他们的累加值。但是如果有人辞职，Cabal 公司需要那样做；死亡的会员可以保留在名单上（很奇怪，这绝不会有问题。）

在没有集中签名者的情况下，无论什么时候你想要与数字签名有同样的效果时这是一个聪明的想法，并已得到应用。

4.13 秘密的全或无泄露

假设 Alice 是前苏联的前代理商，现在失业了，Alice 为了挣钱，便出卖机密，任何愿意付钱的人都可以买秘密。Alice 甚至还有一个目录，所有的秘密都编号列出，并加上一个非常撩人的标题：“Jimmy Hoffa 在那里？”谁在秘密控制着三方委员会？“为什么鲍里斯·叶

利钦总是看上去像吞了一只活青蛙？”等等。

Alice 不愿为一个秘密的价格而泄露两个秘密或者泄露秘密的任何一部分信息。Bob 是一个潜在的买主，他不想为随意的秘密付钱，他也不想告诉 Alice 他想要哪个秘密。这并不关 Alice 的事，此外，那么 Alice 可能在她的目录中加上“Bob 对什么感兴趣“这一条。

在这种情况下不能使用扑克协议，因为在协议的末尾 Alice 和 Bob 必须互相摊牌。Bob 也能进行欺骗而得到不止一个秘密。

这个解决方案就叫做秘密的全或无泄露（ANDOS）。因为，一旦 Bob 得到了不管是 Alice 的秘密中哪一个的任何信息，他就失去了获知任何其他秘密的任何东西的机会。

在密码学文献中有几个 ANDOS 协议，其中一些将在 23.9 节中讨论。

4.14 密钥托管

下面这段话摘自 Silvio Micali 的专题介绍[1084]：

当前，法院授权许可的搭线窃听是防止犯罪并将罪犯绳之以法的有效方法。更重要的是，照我们的观点，通过阻止对正常网络通信的非法使用也防止了犯罪的进一步扩散。因此，法律上比较关心的是，公开密码学的广泛应用可能对犯罪和恐怖组织有很大帮助。实际上，很多议案提议：一个适当的政府机关，在法律允许的情况下，应当可以获得任何通过公共网络进行通讯的明文。目前，这个要求可能意味着强迫市民（1）要么使用弱的密码系统，即有关当局（当然也可以是任何其他的人）经过一定的努力可以破开的密码体制；或（2）要么事先把他们的秘密密钥交给当局。如果这种替代方法会从法律上提醒许多有关的市民，让他们觉得国家安全和法律强制应在隐私之上的话，这并不令人惊奇。

密钥托管是美国政府的 Clipper 计划和它的托管加密标准的核心。这里面临的挑战是开发一个密码系统，要保护个人隐私但同时又要允许法院授权的搭线窃听。

托管加密标准通过防窜改的硬件来实现安全性。每个加密芯片有一个唯一的 ID 号和秘密密钥，密钥被分为两部分，并与 ID 号一起由两个不同的托管机构存储。芯片每次加密数据文件，它首先用唯一的秘密密钥加密会话密钥，然后通过通信通道发送加密的会话密钥和它的 ID 号。当一些法律执行机构想用这些芯片中的一个解密加密的信息流时，它监听 ID 号，从托管机关收集适当的密钥，把它们异或起来，解密会话密钥，然后使用会话密钥解密信息流。面对欺诈者，为了使这个方案可行，它可能更复杂；细节见 24.16 节。同样的事情能够用软件实现，也可用公开密钥密码实现[77，1579，1580，1581]。

Micali 称他的思想为公平密码系统[1084，1085]。（据传美国政府在它们的托管加密标准中为了使用他的专利花了 1 百万美元[1086，1087]；然后，Banker’s Trust 购买了 Micali 的专利。）在这些密码系统中，私钥被分成许多部分，发给不同的机构。类似秘密共享方案，这些机构可集中到一起并重新构造私钥。但是，这些密钥碎片具有一种附加的性质：无需重新构造私钥，便能分别验证这些密钥碎片是否正确。

Alice 可以产生她自己的私钥并给几个托管人每人一部分密钥。这些托管人中没有人能恢复出 Alice 的私钥。然而，所有这些托管人都能验证他们的那一部分是私钥的有效部分，Alice 不可能送给一个托管人随机比特串，并希望他带着逃跑了。如果法院授权搭线窃听，

有关法律执行机构可以遵照法庭的命令让 n 个托管人交出他们的那一部分密钥。用所有这 n 部分，执行机构重新构造出私钥，并能够对 Alice 的通讯线路进行搭线窃听。另一方面，Mallory 为了能重新构造出 Alice 的密钥并侵犯她的隐私，将不得不破坏所有 n 个托管人。

协议执行的情况如下：

（1）Alice 产生出她的私钥/公钥密钥对，她把私钥分成几个公开和秘密部分。

（2）Alice 送给每个托管人一个公开的部分及对应的秘密部分。这些消息必须加密。她也把公开密钥送给 KDC（密钥分配中心）。

（3）每个托管人独立地完成计算以确认所得到的公开部分和秘密部分都是正确的。每个托管人将秘密部分存放在安全的地方并把公开部分发送给 KDC。

（4）KDC 对公开部分和公开密钥执行另一种计算。假设每一件事都是正确的，KDC 在公开密钥上签名，然后把它送回给 Alice 或把它邮寄给某处的数据库。

如果法庭要求进行搭线窃听，那么每个托管人就把他或她的那部分交给 KDC，KDC 能重新构造出私钥。在交出密钥前，无论是 KDC 还是任何一个托管人都不能重新构造出私钥，所有托管人一起才能重新构造出这个密钥。

用这种方式能把任何公开密钥密码算法都做成是公正的。在 23.10 节中讨论了一些特殊算法。Micali 的文章[1084，1085]讨论了把门限方案与这个协议结合起来的办法，使得只需要托管人的一个子集（例如，五个中的三个）便能重新构造出私钥。他又讲述了怎样将不在意传输（见 5.5 节）与这个协议结合起来，使得托管人不知道是谁的私钥正在被重新构造。

公平密码系统不是完美的，罪犯能够利用这个系统，他能够使用阈下信道（见 4.2 节）把另一个秘密密钥嵌入到他的那部分中。采用这种方法，使用阈下密钥，不用担心法院授权的搭线窃听，他就能够安全地与其他人通信。另一个叫做防故障密钥托管解决了这个问题[946，833]。23.10 节描述了算法和协议。

密钥托管的政治

除了政府的密钥托管计划外，几个商业密钥托管正在付诸实施。这导致了明显的问题：对用户来说，密钥托管的好处是什么？

实际上没有任何好处，用户不能从密钥托管得到任何东西。如果他愿意，他可以备份他的密钥（见 8.8 节）。密钥托管保证：即使使用了加密，警察也能够窃听他的谈话或阅读他的数据文件。它保证：即使使用加密，NSA 不经批准也能够窃听他的国际电话。也许，他将被允许在现在反对密钥托管的国家使用密码，这似乎好象是唯一的好处。

密钥托管有相当大的缺陷。用户不得不相信托管机构的安全性程序，以及参与人的诚实。他不得不相信托管机构没有改变他们的策略，政府没有改变他的法律，那些得到密钥的执法机构和托管机构会合法地和负责地做事。设想一个大恐怖分子袭击纽约时，对警察来说，还有什么样的限制不能抛到一边呢？

难于想象托管加密方案工作会象他们的发起人设想的那样没有一些法律的压力。很明显的下一步是禁止使用非托管加密，这可能是使商业系统付费的唯一办法，并且它肯定是使技术上富有经验的罪犯和恐怖分子使用它的唯一方法。不清楚要使非托管密码成为非法将会遇到什么阻力，或它怎么影响作为研究学科的密码学。就我而言，没有软件非托管加密设备，我能研究面向软件的加密算法吗？我还需要特别的许可吗？

还有法律上的问题，如果有加密数据被破开，托管密钥怎么影响用户的责任？如果美国政府试图保护托管结构，是不是有隐含的假设，在用户或托管机构都会危及到秘密的安全时，泄密的一定是用户？

对于政府或商业性的密钥托管服务而言，它的整个托管密钥数据库被偷盗了会怎么样？如果美国政府试图对它保持一段时间的沉默又会怎么样呢？很清楚，这会对使用密钥托管的用户愿望产生影响，如果不是自愿的，这样的一些丑闻又将增加政治压力，迫使政府要么让其成为自愿，要么对该产生增加复杂的新规定。

更为危险的是现政府的政治对手、对某些情报或警察机构坦率直言的批评家已经被监视多年的丑闻会公诸于世。这可能引起公众强烈地反对托管加密的情绪。

如果签名密钥和加密密钥一样被托管，存在更多的问题。当局使用签名密钥执行操作反对可疑罪犯能否被接受？基于托管密钥签名的真实性在法庭上会被接受吗？如果当局签一些不宜的合同，以帮助国家扶持的工业，或只是为了偷窃金钱，而使用他们的签名密钥，用户会有什么样的追索权呢？

密码的全球化导致了另外一些问题，密钥托管的政策在其它国家将会一致吗？跨国公司为了保持与各种地方法律一致，他们必须在每个国家保持单独的托管密钥吗？如果没有某种一致性，密钥托管方案的好处之一（强加密的国际化使用）必将崩溃。

如果有些国家根本不接受托管机构的安全性会怎么样呢？用户在那里怎么做生意呢？他们的数字化合同能得到当地法院的支持吗？或者他们的签名密钥托管在美国的事实会允许他们在瑞士声称别的人也可能签署他的电子合同吗？在这些国家做生意的人是否有特殊的弃权呢？

工业间谍又会怎么样呢？没有理由相信那些目前正在为其重要的或政府性质的公司从事间谍活动的国家会放弃在密钥托管加密系统上做手脚。的确，由于事实上没有哪个国家会允许其他国家监视自己的情报工作，所以，托管加密的广泛使用必将可能增加搭线窃听的盛行。

即使具有良好公民权记录的国家，其使用密钥托管只是为了合法追踪罪犯和恐怖分子，但它肯定也用于别的地方以跟踪异已分子、有敲诈勒索倾向的政敌等等。数字通信在监视公民的行动、意见、购买和集会等一整套工作上提供的机会比模拟世界可能提供的机会大得多。

人们不清楚 20 年以后这种情况对商用密钥托管将有怎样的影响，向土耳其或中国出售现成的密钥托管系统，就类似于 70 年代向南非出售电棍和 80 年代为伊拉克建立化工厂。更糟糕的是，由于这种对通信的窃听十分易行且不可能被跟踪，因而可能诱使许多政府对其大多数公民的通信进行跟踪，甚至连以前不打算这样做的政府也会如此。因而不能保证自由民主社会就能抵御这种诱惑。

第五章 高级协议

5.1 零知识证明

下面是另一个故事：

Alice：“我知道联邦储备系统计算机的口令，汉堡包秘密调味汁的成分以及 Knuth 第四

卷的内容。”

Bob：“不，你不知道。”

Alice：“我知道。”

Bob：“你不知道!”

Alice：“我确实知道!”

Bob：“请你证实这一点!”

Alice：“好吧，我告诉你!”（她悄悄地说出了口令）。

Bob：“太有趣了!。现在我也知道了。我要告诉《华盛顿邮报》”

Alice：“啊呀!”

不幸的是，Alice 要证明一些事情给 Bob 看的通常方法是 Alice 告诉 Bob。但这样一来 Bob 也知道了这些事情。现在，Bob 就可以告诉他想要告诉的其它人，而且 Alice 对此毫无办法。（在文献中，协议常常使用不同的人物。Peggy 通常扮成证明者，而 Victor 则扮成验证者，他们的名字将代替 Alice 和 Bob 出现在下面的例子中。）

Peggy 可使用单向函数进行零知识证明[626]。这个协议向 Victor 证明 Peggy 确实拥有一部分信息，但却没有给予 Victor 确定这个信息是什么的办法。

这些证明采取了交互式协议的形式。Victor 问 Peggy 一系列问题，如果 Peggy 知道那个信息，她就能正确地回答所有问题，如果她不知道，她仍有正确回答的机会--在如下例子中有 50%的机会。大约在十个问题之后，将使 Victor 确信 Peggy 知道那个信息。然而，所有的问题或回答都没有给 Victor 提供关于 Peggy 所知信息的任何信息——只有她知道这个信息。

基本的零知识协议

Jean-Jacques Quisquater 和 Louis Guillou[1281]用一个关于洞穴的故事来解释零知识，如图 5.1 所示，洞穴里面有一个秘密，知道咒语的那些人能打开 C 和 D 之间的密门。对其他任何人来说，两条通道都是死胡同。

Peggy 知道这个洞穴的秘密。她想对 Victor 证明这一点，但她不想泄露咒语。下面是她如何使 Victor 相信的过程：

（1）Victor 站在 A 点；

（2）Peggy 一直走进洞穴，到达 C 点或者 D 点；

（3）在 Peggy 消失在洞穴中之后，Victor 走到 B 点；

（4）Victor 向 Peggy 喊叫，要她：

（a）从左通道出来，或者

（b）从右通道出来；

（5）Peggy 答应了，如果有必要她就用咒语打开密门；

（6）Peggy 和 Victor 重复步骤（1）至（5）n 次。

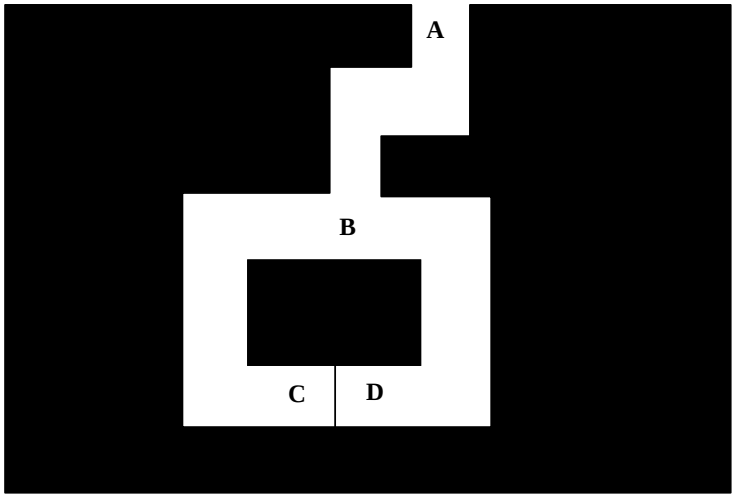

图 5.1　零知识洞穴

假设Victor有一个摄像机并记录下他所看到的一切。他记录下Peggy消失在洞中的情景，记录下他喊叫 Peggy 从他选择的地方出来的时间，记录下 Peggy 走出来。他记录下所有 n 次试验。如果他把这些记录给 Carol 看，她会相信 Peggy 知道打开密门的咒语吗？肯定不会。在不知道咒语的情况下，如果 Peggy 和 Victor 事先商定好 Victor 喊叫什么，那将如何呢？Peggy 会确信她走进 Victor 叫她出来的那一条路。然后她就可以在不知道咒语的情况下在 Victor 每次要她出来的地方出来。或许他们不那么做，Peggy 会走进其中一条通道，Victor 会发出一个随机的要求。如果 Victor 猜对了，好极了；如果他猜错了，他们会从录像带中删除这个试验。总之，Victor 能获得一个记录，它准确显示与实际证明 Peggy 知道咒语相同的事件顺序。

这说明了两件事情。其一，Victor 不可能使第三方相信这个证明的有效性。其二，它证明了这个协议是零知识的。在 Peggy 不知道咒语的情况下，Victor 显然不能从记录中获悉任何信息。但是，因为无法区分一个真实的记录和一个伪造的记录，所以 Victor 不能从实际证明中了解任何信息——它必定是零知识。

协议使用的技术叫做分割选择，因为它类似于将任何东西等分的经典协议：

（1）Alice 将东西切成两半；

（2）Bob 给自己选择一半；

（3）Alice 拿走剩下的一半。

Alice 最关心的是在步骤（1）中的等分，因为 Bob 可以在步骤（2）中选择他想要的那一半。Michael Rabin 是第一个在密码学中使用分割选择技术的人[1282]。交互式协议和零知识的概念是后来才正式提出的[626，627]。

分割选择协议起作用是因为 Peggy 没有办法重复猜出 Victor 要她从哪一边出来。如果 Peggy 不知道这个秘密，那么她只能从进去的路出来。在协议的每一轮（有时叫一次鉴别）

中她有 50%的机会猜中 Victor 会叫她从哪一边出来，所以她有 50%的机会欺骗他。在两轮中她欺骗 Victor 的机会是 25%。而所有 n 次她欺骗 Victor 机会是 2^n 分之一。经过 16 轮后，Peggy 只有 65536 分之一的机会欺骗 Victor。Victor 可以安全地假定，如果所有 16 次 Peggy 的证明都是有效的，那么她一定知道开启 C 点和 D 点间的密门的咒语。（洞穴的比拟并不完美。Peggy 可能简单地从一边走进去，并从另一边出来；这里并不需要任何分割选择协议，但是，数学上的零知识需要它。）

假设 Peggy 知道一部分信息而且这个信息是一个难题的解法，基本的零知识协议由下面几轮组成。

（1）Peggy 用她 的信息和一个随机数将这个难题转变成另一难题，新的难题和原来的难题同构。然后她用她的信息和这个随机数解这个新的难题。

（2）Peggy 利用比特约定方案提交这个新的难题的解法。

（3）Peggy 向 Victor 透露这个新难题。Victor 不能用这个新难题得到关于原难题或其解法的任何信息。

（4）Victor 要求 Peggy 或者：

（a）向他证明新旧难题是同构的（即两个相关问题的两种不同解法）。

（b）公开她在步骤（2）提交的解法并证明是新难题的解法。

（5）Peggy 同意。

（6）Peggy 和 Victor 重复步骤（1）至（5）n 次。

还记得洞穴协议中的摄像机吗？在此你可以做同样的事。Victor 可以做一个在他和 Peggy 之间交换的副本。他不能用这个副本让 Carol 信服，因为他总能串通 Peggy 制造出一个伪造 Peggy 知识的模拟器。这个论点可以用来论证这样的证明是零知识的。

这类证明的数学背景是很复杂的。这个问题和这个随机变换一定要仔细挑选，使得甚至在协议的多次迭代之后，Bob 仍不能得到关于原问题解法的任何信息。不是所有难题都能用作零知识证明，但很多可以。

图同构

举个例子来解释这个概念可能要费很多笔墨。这个概念来自图论[619，622]。连结不同点的线构成的网络称为图。如果两张图除点的名字不同其它都一样，它们叫“同构”。对于一个非常大的图，找出两个图是否同构需要计算机工作几百年的时间；这是在 11.1 节中讨论的那些 NP—完全问题之一。

假设 Peggy 知道图 G_1 和 G_2 之间同构，下面的协议将使 Victor 相信 Peggy 的知识：

（1）Peggy 随机置换 G_1 产生另一个图 H，并且 H 和 G_1 同构。因为 Peggy 知道 G_1 和 H 同构，她也就知道 H 和 G_2 同构。对其他人来说，发现 G_1 和 H 或 H 和 G_2 之间同构与发现 G_1 和 G_2 之间同构一样难。

（2）Peggy 把 H 送给 Victor。

（3）Victor 要求 Peggy 或者：

（a）证明 G_1 和 H 同构，或者

（b）证明 G_2 和 H 同构。

（4）Peggy 同意。她或者：

（a）证明 G_1 和 H 同构，但不证明 G_2 和 H 同构，或者

（b）证明 G_2 和 H 同构，但不证明 G_1 和 H 同构。

（5）Peggy 和 Victor 重复步骤（1）至（4）n 次。

如果 Peggy 不知道 G_1 和 G_2 之间的同构性，她就不能创造出和这两个图都同构的图 H。她只能创造一个图或者与 G_1 同构或者与 G_2 同构。同前面的那个例子一样，她只有 50%的机会猜中 Victor 在第（3）步中会要求她执行哪一个证明。

这个协议没有给 Victor 任何有用的信息以帮助他了解 G_1 和 G_2 之间的同构性。因为 Peggy 在协议的每一轮都产生一个新图 H，故不管他们经过多少轮协议 Victor 也得不到任何信息，他不能从 Peggy 的答案中了解 G_1 和 G_2 的同构性。

在每一轮中，Victor 都得到 H 的一个新的随机置换，以及 H 和 G_1 或 G_2 之间的同构性。Victor 也可以自己来产生这个协议。因为他能做一个此协议的模拟器，它能被证明是零知识的。

汉密尔顿圈

另一不同的例子是由 Manuel Blum 最先提出的[196]。Peggy 知道一条沿图线走向的环形连续路径，通过每个点仅一次，这个环形连续路径被称为“汉密尔顿圈”。找到一个汉密尔顿圈是另一难题。Peggy 拥有这部分信息——她可能通过利用某个汉密尔顿圈来构造图而得到该信息——这正是她想要 Victor 相信她知道的信息。

Peggy 知道一个图 G 的汉密尔顿圈。Victor 知道图 G，但是不知道它的汉密尔顿圈。在不暴露汉密尔顿圈的情况下，Peggy 要向 Victor 证明她知道这个汉密尔顿圈。下面是她的做法：

（1）Peggy 随机地置换图 G。她移动这些点并改变他们的标号，生成一个新图 H。因 G 和 H 在拓扑上同构（即相同的图），如果她知道 G 的汉密尔顿圈，那么她能很容易地找到 H 的汉密尔顿图。如果她不是自己创造 H，则确定两个图之间的同构性将是另一难题；它也需花费计算机几百年的时间。她加密 H 得到 H´（这必定是一种对 H 的每一条线的概率加密，即，对 H 的每一条线加密 0 或加密 1）。

（2）Peggy 给 Victor 一个 H´的副本。

（3）Victor 要求 Peggy 或者

（a）向他证明 H´是 G 的同构副本的加密，或者

（b）向他出示 H 的汉密尔顿圈。

（4）Peggy 同意，她或者

（a）通过揭示置换和解密一切证明 H´是 G 的同构副本的加密，但不出示 G 或 H 的汉密尔顿圈，或者

（b）仅通过解密构成汉密尔顿圈的那些线出示 H 的汉密尔顿圈，但不证明 G 和 H 在拓扑上同构。

（5）Peggy 和 Victor 重复步骤（1）至（4）n 次。

如果 Peggy 诚实，她就能给 Victor 提供步骤（4）中两个证明中的任何一个。但是，如果她不知道 G 的汉密尔顿圈，她就不能创造一个加了密的图 H'，这个图 H'能满足两个要求。她最多能做到的是使其所创造的图或者与 G 同构，或者具有相同数目的点线及一个有效的

汉密尔顿圈。虽然她有 50%的机会猜中 Victor 在执行第（3）步中将要他完成哪一个证明，但 Victor 可将协议重复足够多次来使他自己确信 Peggy 知道 G 的汉密尔顿圈。

并行零知识证明

基本的零知识协议包括 Peggy 和 Victor 之间的 n 次交换。可以把它们全部并行完成：

（1）Peggy 使用她的信息和 n 个随机数把这个难题变成 n 个不同的同构难题，然后用她的信息和随机数解决这 n 个新的难题。

（2）Peggy 提交这 n 个新难题的解法。

（3）Peggy 向 Victor 透露这 n 个新难题。Victor 无法利用这些新难题得到关于原问题或其解法的任何信息。

（4）对这 n 个新难题中的每一个，Victor 要求 Peggy 或者：

（a）向他证明新旧难题是同构的，或者

（b）公开她在步骤（2）中提交的解法，并证明它是这个新难题的解。

（5）Peggy 对这 n 个新难题中的每一个都表示同意。

很不幸，事情并非如此简单。该协议没有同前协议相同的零知识性质。在第（4）步，Victor 可以把第（1）步所提交的所有值的单向 HASH 函数作为询问，这样就使副本不可冒充。它仍然是零知识的，但属于不同种类。实际应用中它似乎是安全的，但是没有人知道怎样证明它。我们确实知道，在某些环境下，针对某些问题的某些协议可以并行运行，并同时保留它们的零知识性质[247，106，546，616]。

非交互式零知识证明

不能使 Caro 相信是因为这个协议是交互式的，并且她没有介入交互中。为了让 Carol 和其他感兴趣的人相信，我们需要一个非交互式的协议。

人们已经发明了非交互式零知识证明的协议[477，198，478，197]。这些协议不需要任何交互作用，Peggy 可以公布他们，从而向任何花时间对此进行检验的人证明协议是有效的。

这个基本协议类似于并行零知识证明，不过只是用单向 hash 函数代替了 Victor：

（1）Peggy 使用她的信息和 n 个随机数把这个难题变换成 n 个不同的同构问题，然后用她的信息和随机数解决这 n 个新的难题。

（2）Peggy 提交这 n 个新的难题的解法。

（3）Peggy 把所有这些提交的解法作为一个单向 hash 函数的输入。（这些行为终归不过是一些比特串），然后她保存这个单向 hash 函数输出的头 n 个比特。

（4）Peggy 取出在步骤（3）中产生的 n 个比特。对每个难题，她依次针对第 i 个新难题取出这 n 个比特中的第 i 个比特并且：

（a）如果它是 0，她则证明新旧问题是同构的，或者

（b）如果它是 1，她则公布她在第（2）步中提交的解法，并证明它是这个新问题的解法。

（5）Peggy 将步骤（2）中的所有约定及步骤（4）中的解法都公之于众。

（6）Victor 或 Carol 或其他感兴趣的人，可以验证步骤（1）至（5）是否被正确执行。

这很令人惊异：Peggy 可以公布一些不含有关她的秘密的信息、却能让任何人相信这个秘密的存在。如果把这个问题作为初始消息和要签名的消息的单向 hash，则这个协议也可

用于数字签名方案。

这个协议起作用的原因在于单向 hash 函数扮演了一个无偏随机比特发生器的角色。如果 Peggy 要进行欺骗，她必须能预测这个单向 hash 函数的输出。（记住，如果她不知道这个难题的解法，她可以完成步骤（4）的（a）或（b），但不能两者一起。）如果由于什么原因她知道了这个单向 hash 函数会叫她做什么，那么她可以进行欺骗。然而，Peggy 没有办法强迫这个单向 hash 函数产生哪些比特或猜中它将产生哪些比特。这个单向 hash 函数在协议中实际上是 Victor 的代替物——在步骤（4）中随机地选择两个证明中的一个。

在一个非交互式协议中，必定有更多的问/答序列迭代。不是 Victor 而是 Peggy 在用随机数挑选这些难题，她可以挑选不同的问题，因此有不同的提交矢量，直到这个 hash 函数产生她希望的东西为止。在一个交互式协议中，10 次迭代——Peggy 能进行欺骗的概率为 2^{10} 分之一（1024 分之一）——是很好的了。但是，对非交互式零知识证明是不够的；记住，Mallory 总能完成步骤（4）的（a）或(b)，他能设法猜测会要他完成哪一步，处理完步骤（1）直到步骤(3)，并弄清他是否猜对。如果他没有猜对，可以再试——反反复复。在计算机上进行 1024 次猜测不是难事。要防止这种穷举攻击，非交互式协议需要 64 次迭代，甚至 128 次迭代才是有效的。

这就是使用单向 HASH 函数的全部要点：Peggy 不能预测 HASH 函数的输出，是因为她不能预测其输入。只有在她解决了新的难题以后，才能知道作为输入的提交。

一般性

Blum 证明了任何数学定理都能被转化为一个图，使得这个定理的证明等价于证明图的汉密尔顿圈。假设有了单向函数并因此有了好的加密算法，则任何 NP 命题包括一个零知识证明，这种一般情况已在[620]中得到证明。任何数学证明都能被转化成一个零知识证明。采用这项技术，研究人员能向世人证明他们知道一个特殊定理的解法但又不会泄露那个证明是什么。Bium 可以公布他的结果，同时又不泄露它们。

也存在一些最小泄露证明[590]。在最小泄露证明中，具有以下性质：

1．Peggy 不能欺骗 Victor。如果 Peggy 不知道证明，她使 Victor 相信她知道这个证明的概率是非常小的。

2．Victor 不能欺骗 Peggy。除了 Peggy 知道证明这个事实，他得不到关于证明的轻微线索。尤其在他自己没有完完全全地证明它的情况下，Victor 不可能向其它人论证这个证明。

零知识证明有一个附加条件：

3．除了 Peggy 知道证明这个事实，Victor 不能从 Peggy 处得到任何东西，因为没有 Peggy 他不能自己得到有用的信息。

最小泄露证明与零知识证明之间存在相当大的数学区别。这个区别超越了本书的范围，但欢迎愿意深入研究的读者参阅参考书籍。概念介绍请看[626，619，622]。关于概念的更详尽的描述，基于不同数学假设，已在[240，319，239]阐述。

这里也有不同类型的零知识证明：

——**完美的**。有一个使副本与正本具有相同分布的模拟器（例如汉密尔顿圈和图的同构）。

——**统计的**。除了一定数量的例外，有一个使副本与正本具有相同分布的模拟器。

——**计算的**。有一个使副本与正本不能区别的模拟器。

——**无用的**。可能不存在模拟器，但是我们可以证明 Victor 不能从证明中得到任何更多的信息（例如并行证明）。

在这些年中，关于最少泄露和零知识证明，无论是理论还是应用上，人们已做了广泛的工作。Mike Burmester 和 Yvo Desmedt 发明了广播交互式证明，其间的一个证明者能将零知识交互式证明广播给一大群验证者[280]。密码学家们证明，能用一个交互式证明证明的每一件事，也能用一个零知识交互式证明来证明[753，137]。

关于该题目的好综述文章是[548]。更多的数学上的细节、变化、协议和应用，请参看[590，619，240，319，620，113，241，1528，660，238，591，617，510，592，214，104，216，832，97，939，622，482，615，618，215，476，71]，关于这个主题的论文比比皆是。

5.2 身份的零知识证明

在现实世界中，我们用物理信物作为身份证明：护照、驾驶执照、信用卡等等。这些信物包含了把它与一个人连系起来的东西：通常是照片或签名，但可能最方便的是一个指纹，一个视网膜扫描图或牙齿的 X 光片。用数字方式来做这件事难道不好吗？

使用零知识证明来作身份证明最先是由 Uriel Feige, Amos Fiat 和 Adi Shamir 提出的[566，567]。Alice 的私钥成为她（的）“身份”的函数。通过使用零知识证明，她能够证明她知道她的私钥，并由此证明她的身份。这类算法在 23.11 节介绍。

这个想法是相当有用的，它使一个人不用任何实际信物便能证明他的身份。但是，它不是完美无缺的。它存在一些弊端。

国际象棋特级大师问题

Alice 是一个甚至连国际象棋的规则也不知道的人，这里要介绍她怎样击败一个特级大师（有时叫做“国际象棋特级大师”问题。）。她在一场比赛中挑战加里 • 卡斯帕罗夫（Gary Kasparov）和安纳托利 • 卡尔波夫（Anatoly Karpov），选择同一时间和地点，但在不同的房间。她执白棋对卡斯帕罗夫而执黑棋对卡尔波夫，两个特级大师都不知道对方。

卡尔波夫执白先行，走了第一步，Alice 记住这一步，并走进卡斯帕罗夫的房间。她执白对卡尔波夫走了同样一步。卡斯帕罗夫走了一步黑棋。Alice 记下这一步，走进卡尔波夫的房间，走了同样一步。这样持续下去直到她赢了一盘比赛并输掉了另一盘，或两盘比赛都以平局告终。

实际上，卡斯帕罗夫是在同卡尔波夫对局，而 Alice 只是简单地扮作了中间人，模仿每一个特级大师在另一个的棋盘上行棋。然而，如果卡尔波夫和卡斯帕罗夫都不知道对方在场，他们都会对 Alice 的棋艺留下相当深刻的印象。

这种欺骗可以用于攻击身份的零知识证明[485，120]。在 Alice 向 Mallory 证明她的身份时，Mallory 同时能向 Bob 证明他是 Alice。

黑手党骗局

当讨论 Adi Shamir 的零知识识别协议时，他说[79]：“我可以去一个黑手党拥有的商店连续一百万次，而他们仍然不能冒充我。”

下面是黑手党如何能够做到的过程。Alice 正在 Bob 的餐馆——一家黑手党拥有的餐馆

吃饭，Carol 正在一家高档珠宝店 Dave 商场买东西，Bob 和 Carol 都是黑手党成员，并且他们正通过一条秘密的无线电线路通信。Alice 和 Dave 都不知道这个骗局。

当 Alice 吃完饭，她准备付帐并对 Bob 证明她的身份时，Bob 给 Carol 发信号通知她准备开始这场骗局。Carol 买了一些贵重的钻石，并准备对 Dave 证明她的身份。现在，当 Alice 对 Bob 证明她的身份时，Bob 用无线电告知 Carol，Carol 则同 Dave 执行相同协议。当 Dave 问协议中的一个问题时，Carol 用无线电把问题回告 Bob，然后 Bob 再问 Alice 这个问题。当 Alice 回答后，Bob 又用无线电将正确答案告诉 Carol。实际上，Alice 只是在对 Dave 证明她的身份，而 Bob 和 Carol 只是简单地在协议中间来回传递消息。当协议完成时，Alice 已对 Dave 证明了她的身份，并买了一些贵重的钻石（Carol 随之消失）。

恐怖分子骗局

如果 Alice 愿意与 Carol 合作，她们也能欺骗 Dave。在这个协议中，Carol 是一个臭名昭著的恐怖分子。Alice 帮助 Carol 进入这个国家。Dave 是移民局的官员。Alice 和 Carol 通过一条秘密的无线电路联系。

当 Dave 询问 Carol 零知识协议中的一部分问题时，Carol 用无线电将它们发给 Alice，Alice 自己回答这些问题。Carol 向 Dave 复述答案。实际上，是 Alice 在向 Dave 证明她的身份，Carol 只是作为一条通信路径。当协议完成时，Dave 认为 Carol 是 Alice 并让她进入这个国家。三天后，Carol 同一辆装满炸药的微型车在某政府大楼出现。

建议的解决方法

黑手党和恐怖分子的骗局有可能成功，因为同谋者可以通过一条秘密的无线电线路通信。阻止这一切发生的一个办法是要求所有的识别在法拉第罩内发生，这样可以防止所有的电磁辐射。在恐怖分子的例子中，这将使移民局官员 Dave 确信 Carol 没有从 Alice 那里收到她的答案。在黑手党的例子中，Bob 可以简单地在他的餐馆内建一个有缺陷的法拉第罩，但珠宝商 Dave 得有一个正常工作的法拉第罩，Bob 和 Carol 仍不能通信。为了解决国际象棋特级大师问题，应强迫 Alice 坐在她的位置上，直到对弈结束。

Thomas Beth 和 Yvo Desmedt 提出了另一种解决办法，这种办法使用了很精确的时钟[92]。如果协议中每一步都必须在一个给定的时间发生，同谋者就没有时间通信。在国际象棋特级大师问题中，如果每一局的每步棋都必须在时钟敲响一分钟时走，那么 Alice 就没时间从一个房间跑到另一个房间。在黑手党故事中 Bob 和 Carol 也没时间互相间传递问题和答案。

多重身份骗局

在[485，120]中还讨论了其他一些零知识身份证明的滥用问题。在一些实现中，当个人注册一个公开密钥时不作检验。因此，Alice 可有几个私钥，因而有几个身份。如果她想搞税款骗局，这可能大有帮助。Alice 也可以犯了罪然后消失。首先，她创造并公布几个身份，其中一个她没有使用，接着，她使用那个身份一次并进行犯罪，故对她进行身份验证的人就是证人。然后，她立即停止使用那个身份，证人知道犯罪人的身份，但如果 Alice 不再使用那个身份——她也就难以被发现。

为了防止这种欺骗，必须有某种机制来保证每个人只有一个身份。在[120]中，作者提出了防止掉包婴儿的“古怪”想法，这些婴儿都不能克隆，并都包含一个独一无二的编号作为他们遗传密码的一部分。他们还建议让每个婴儿在出生时都得到一个身份。（实际上，由

于婴儿会被别人占有，所以父母须在孩子出生时就做这项工作，这可能很容易被滥用；父母可能在孩子出生时为他提供多重身份。归根结底，个体的唯一性仍基于信任。

出租护照

Alice 想到扎伊尔去旅游，但该政府不给她签证。Carol 提出把她的身份租给 Alice。（Bob 首先提议，但有一些明显的问题。）Carol 把她的私钥卖给 Alice，Alice 伪装成 Carol 去扎伊尔。

Carol 不但因为她的身份得到报酬，而且她还有一个完美的辩解。当 Alice 在扎伊尔期间 Carol 犯了罪。“Carol”已经在扎伊尔证明了她的身份；她怎么能回家作案呢？

当然，Alice 也可以随意作案。她或者在离开前或者返回后在 Carol 家附近作案。首先，她证明自己是 Carol（她有 Carol 的私钥，故她能轻易做到），然后作案潜逃，警察将会来找 Carol，Carol 宣称她把身份租给了 Alice，但谁会相信这个荒谬的故事呢？

问题在于 Alice 并没有在真正地证明她的身份，她只是在证明她知道一部分秘密信息。正是那个属于信息和人之间的联系被滥用了。防止掉包婴儿的解决办法可防止这类骗局，如同在一个警察国家，那里所有的市民必须经常证明他们的身份（每天晚上，每个街道拐角处，等等）。

成员资格证明问题

Alice 想向 Bob 证明她是某超级秘密组织的成员，但她不想暴露她的身份。这个问题类似于但又不同于身份证明问题，在[887，906，907，1201，1445]中也有研究。有些解决方法和分组签名问题有联系。

5.3 盲签名

数字签名协议的一个基本特征是文件的签署者知道他们在签署什么。这是个好的构想，除非当我们不想让他们知道时。

有时候我们想要别人签署一个他们从未看过其内容的文件。也有办法让签名者能大体知道他们要签什么，只不过不准确而已。

完全盲签名

Bob 是一个公证员，Alice 要他签一个文件，但又不想让他知道他在签什么。Bob 不关心文件中说些什么，他只是证明他在某一时刻公证过这个文件。他愿意这样进行：

（1）Alice 取出文件并将它乘以一个随机值，这个随机值称为盲因子；

（2）Alice 送这份隐蔽好的文件给 Bob；

（3）Bob 在这个隐蔽好的文件上签名；

（4）Alice 将其除以隐蔽因子，留下 Bob 签过的原始文件。

只有当签名函数和乘法函数是可交换时，这个协议才能有效。如果不是，有别的方法可代替乘法来修改文件。相关的算法出现在 23.12 节中。现在，假设运算是乘法，并且所有数学上的要求都满足。

Bob 能进行欺骗吗？他能收集到他所签的文件的任何信息吗？如果盲因子是真正随机的并使隐蔽文件真正随机的，那么他不能。在步骤（2）中，Bob 签的隐蔽好的文件一点也不象 Alice 开始用的文件。在步骤（3）中，带有 Bob 签名的隐蔽好的文件也一点不象步骤

（4）末的已签了名的文件。即使 Bob 可染指这个文件，并且文件上带有他的签名，在完成这个协议之后，他也不能证明（向他自己或任何其他人）他在那个特殊的协议中签了这个文件。他知道他的签名是有效的。他也可以象其他人一样验证他的签名。但是，他没有办法把已签了名的文件和他在协议中收到的任何信息相关联。如果他用这个协议签了一百万份文件，他照样没有办法知道在哪种情况下他签了哪一份文件。

完全盲签名的性质是：

——Bob 在文件上的签名是有效的。签名就是 Bob 签署这份文件的证据。如果把文件给 Bob 看，Bob 确信他签署过这份文件。在 2.6 节中也讨论过的数字签名具有的所有其他性质。

——Bob 不能把签署文件的行为与签署了的文件相关联。即使他记下了他所作的每一个盲签名，他也不能确定他在什么时候签署了该文件。

Eve 在中间观看了这个协议，他得到的信息甚至比 Bob 还少。

盲签名

用完全盲签名协议，Alice 能让 Bob 签任何东西：“Bob 欠 Alice 一百万美元”，“Bob 欠 Alice 的头生子”，“Bob 欠 Alice 一袋软糖。”可能的事远远不止于此。这个协议在许多场合都无用。

然而，有一个办法可以让 Bob 知道他在签什么，同时仍保持盲签名的有用性质。这个协议的核心是分割选择技术，考虑一个例子，每天很多人进入这个国家，而移民局要确信他们没有走私可卡因。官员们可以搜查每一个人，但他们换用了一种概率解决办法。他们检查入境人中的十分之一。十个人中有一个人的行李被检查，其余的九个畅通无阻。长期的走私犯会在大多数时间里逍遥法外，但他们有 10%的机会被抓住。并且如果法院制度有效，则抓住一次的处罚将远远超出其它九次所得到的。

如果移民局想增大抓住走私犯的可能性，他们将不得不搜查更多的人。如果他们要减少这种可能性，他们只须搜查少量的人。通过操纵概率，他们可以控制协议抓住走私犯的成功程度。

盲签名协议以类似的方式发挥作用。Bob 将得到一大堆不同的隐蔽好的文件，他打开即检查除一个文件以外的所有文件，然后对最后一个文件签名。

把隐蔽文件想象为装在信封里，隐蔽文件的过程就是把文件装进信封，去除隐蔽因子就是打开信封。当文件在信封里时，没人能读它。文件则是用一张复写纸签署在信封里：当签名人签署信封时，他的签名通过复写纸也签在了文件上。

这个剧情涉及到一组反间谍人员。他们的身份是秘密的，甚至反间谍机构也不知道他们是谁。这个机构的头子想给每个特工一份签名的文件，文件上写有：“这个签名文件的持有人（这里插入特工的化名）享有完全的外交豁免权。”所有的特工有他们自己的化名名单，故这个机构不能仅仅是分发签名文件。特工们不想把他们的化名送给所属机构，敌方或许已经破坏了这个机构的计算机。另一方面，机构也不想盲目地签特工送来的文件。聪明的特工可能会代之一条消息，象：“特工（名字）已经退休并获得一年一百万美元的养老金。签名：总统先生”。在这种情况下，盲签名可能是有用的。

假设所有特工都有十个可能的化名，这些化名都是他们自己选的，别人不知道。同时假

设特工们并不关心他们将在哪个化名下得到外交豁免权。再假设这个机构的计算机是情报局大型情报计算机 ALICE，我们的特定代理部门是波哥大行动局：BOB。

（1）BOB 准备了 n 份文件，每一个使用不同的化名，并给予那个特工外交豁免权。

（2）BOB 用不同的盲因子隐蔽每个文件。

（3）BOB 把这 n 份隐蔽好的文件给 ALICE。

（4）ALICE 随机选择 n-1 份文件并向 BOB 索要每份文件的盲因子。

（5）BOB 向 ALICE 发送适当的盲因子。

（6）ALICE 打开（即去掉盲因子）n-1 份文件，并确信它们是正确的——而不是退休授权。

（7）ALICE 在第十个文件上签名并把它送给 BOB。

（8）BOB 去掉盲因子并读出他的新化名：“The Crimson Streek”。签署的文件在那个名字下给予他外交豁免权。

这个协议能防止 BOB 欺骗。他要欺骗，他必须准确地预测 ALICE 不会检查哪一份文件。他这样做的机会是 n 分之一，不是很好。ALICE 也知道这一点并且有把握签一份她不可能检查的文件。用这份文件，这个协议就和先前的盲签名协议一样，并保持了它所有的匿名性质。

有一种方法可以使 BOB 的欺骗机会更小，在步骤（4）中，ALICE 随机选择 n/2 份文件提出质疑，并在步骤（5）中发送给她合适的盲因子。在步骤（7）中，ALICE 使所有非质疑文件一块儿相乘并签署这分大文件。在步骤（8）中，BOB 去掉所有的盲因子，ALICE 的签名只有在它是 n/2 相同文件乘积的有效签名时才是可以接受的。要欺骗 BOB 就得能够准确地猜测 ALICE 将质疑哪一个子集；其机会要比猜测 ALICE 不会质疑哪一份文件的机会小得多。

BOB 有另一种方法进行欺骗。他可产生两份不同的文件，一份 ALICE 愿意签署，一份 ALICE 不愿签署。然后他可以找两个不同的盲因子，把每份文件变成相同的隐蔽文件。这样，如果 ALICE 要检查文件，BOB 就给她把文件变成良性文件的盲因子；如果 ALICE 不要求看文件，并签署文件，则他可以使用盲因子把文件变为恶意文件。虽然这在理论上是可行的，但涉及到特定算法，使 BOB 能找到这样一对盲因子的机会变得微乎其微。实际上，可以使它同 BOB 能自己在一份任意消息上产生签名的机会一样小。这个问题在 23.12 节中进一步讨论。

专利

CHAUM 已取得了几种盲签名的专利（见表 5.1）。

TABLE 5.1

Chaum’s 盲签名专利

美国专利	时期	名称
4,759,063	7/19/88	盲签名系统[323]
4,759,064	7/19/88	盲的非参与签名系统[324]
4,914,698	3/3/90	一次显示盲签名系统[326]
4,949,380	8/14/90	返回值盲签名系统[328]
4,991,210	2/5/91	不可预测的盲签名系统[331]

5.4 基于身份的公钥密码

Alice 想发一秘密消息给 Bob。她不想从密钥服务器中获得他的公开密钥；她不想在他的公钥证书上验证某个第三方的签名；她甚至不愿在她自己的计算机上存贮 Bob 的公开密钥。她只想给 Bob 发送一份秘密消息。

基于身份的密码体制，有时叫作非交互式密钥共享（NIKS）体制，可以解决发送秘密消息问题 [1422]。Bob 的公开密钥是基于他的名字和网络地址的（或者电话号码，或者实际街区地址，或者其它什么东西）。对一般的公钥密码体制，Alice 需要一个使 Bob 的身份同他的公钥相关的已签过名的证书。对基于身份的密码体制 Bob 的公钥就是他的身份。这是一个真正绝妙的主意，就象邮政系统一样方便：如果 Alice 知道 Bob 的地址，她就可以给 Bob 发送保密邮件。它使密码变得尽可能透明。

这个体制是建立在 Trent 依据其身份给用户发布私钥的基础上。如果 Alice 的私钥泄露，她就必须在某些方面改变他的身份，以求得到另一个私钥。一个更严重的问题是系统的设计方法应使不诚实用户串通也无法伪造密钥。

已对有关这些种类的方案的数学问题做了大量研究工作——大部分在日本——它证明使保密变得异常复杂。许多建议的解决方案中涉及 Trent 为每个用户选择一个随机数——我认为这点可以解决系统的真正要害。在 19 章和 20 章讨论的一些算法可以是基于身份的。有关细节、算法和密码分析，见[191，1422，891，1022，1515，1202，1196，908，692，674，1131，1023，1516，1536，1544，63，1210，314，313，1545，1539，1543，933，1517，748，1228]。一种不依赖任何随机数的算法见[1035]。在[1546，1547，1507]中讨论的系统对选择公开密钥攻击是不安全的；建议的系统如 NIKS-TAS[1542，1540，1541，993，375，1538]也是如此。老实说，迄今为止所提系统没有一个是既实用又安全的。

5.5 不经意传输

密码员 Bob 正在拼命地想将一个 500 比特的数 n 进行因子分解。他知道它是 5 个 100 比特的数的乘积，但不知道任何更多的东西。（这是一个问题。如果他不能恢复这个密钥，他就得加班工作，势必错过他和 Alice 每周一次的智力扑克游戏。）

你知道什么？现在 Alice 来了：

“我碰巧知道那个数的一个因子，”Alice 说，“并且我要一百美元才把它卖给你。那是一比特一美元。”为了表明她的诚意，她使用一个比特提交方案并分别提交每一比特。

Bob 很感兴趣，但他只有 50 美元。Alice 又不愿降价，只愿意以一半的价格卖给 Bob 一半的比特。“这将会节省你相当多的工作，”她说。

“但是我怎么知道你的数确实是 n 的一个因子呢？如果你给我看那个数并让我验证它是一个因子，那么我将同意你的条件”， Bob 说。

他们陷入了僵局。Alice 不能在不透露 n 的情况下让 Bob 相信她的数是 n 的一个因子，而 Bob 也不愿买一个可能毫无用处的数的 50 比特。

这个借自 Joe Kilian[831]的故事，介绍了不经意传输的概念。Alice 传送一组消息给 Bob，Bob 收到了那些消息的某个子集，但 Alice 不知道他收到了那些消息。然而这并没有彻底解

决上面的问题。在 Bob 收到那些比特的任意一半后，Alice 就还得用一个零知识证明来使他相信她发送的那些比特是 n 的部分因子。

在下面的协议中，Alice 将发送给 Bob 两份消息中的一份。Bob 将收到其中一条消息，并且 Alice 不知道是哪一份。

（1）Alice 产生两个公开密钥/私钥密钥对，或总共四个密钥。她把两个公开密钥发送给 Bob。

（2）Bob 选择一个对称算法（例如 DES）密钥。他选择 Alice 的一个公开密钥并用它加密他的 DES 密钥。他把这个加了密的密钥发送给 Alice，且不告诉她他用的是她的哪一个公开密钥加密的 DES 密钥。

（3）Alice 解密 Bob 的密钥两次，每次用一个她的私钥来解密 Bob 的密钥。在一种情况下，她使用了正确的密钥并成功地解密 Bob 的 DES 密钥。在另一种情况下，她使用了错误的密钥，只是产生了一堆毫无意义，而看上去又象一个随机 DES 密钥的比特。由于她不知道正确明文，故她不知道哪个是正确的。

（4）Alice 加密她的两份消息，每一份用一个不同的在上一步中产生的 DES 密钥（一个真的和一个毫无意义的），并把两份消息都发送给 Bob。

（5）Bob 收到一份用正确 DES 密钥加密的消息及一份用无意义 DES 密钥加密的消息。当 Bob 用他的 DES 密钥解密每一份消息时，他能读其中之一，另一份在他看起来是毫无意义的。

Bob 现在有了 Alice 两份消息中的一份，而 Alice 不知道他能读懂哪一份。很遗憾，如果协议到此为止，Alice 有可能进行欺骗。另一个步骤必不可少。

（6）在协议完成，并且知道了两种可能传输的结果后，Alice 必须把她的私钥给 Bob，以便他能验证她没有进行欺骗。毕竟，她可以用第（4）步中的两个密钥加密同一消息。

当然，这时 Bob 可以弄清楚第二份消息。

因为 Alice 无法知道两个 DES 密钥中的哪一个是真的，故这个协议能防止 Alice 的攻击。她加密两份消息，但 Bob 只能恢复出其中之一——一直到第（6）步。它同样能防止 Bob 的攻击，因为在第（6）步之前，他没有办法得到 Alice 的私钥来确定加密另一份消息的 DES 密钥。这可能看起来仍不过象一个较复杂的通过 Modem 掷硬币的方法，但当把它用于较复杂的协议时，它具有广泛的意义。

当然，没有办法阻止 Alice 发送给 Bob 两份完全无用的消息：“Nyah nyah”和“You sucker”。这个协议确保 Alice 发送给 Bob 两份消息中的一份；它不保证 Bob 想收到其中的任何一份。

在文献中还有其他的不经意传输协议。其中有些是非交互式的，即 Alice 可以公布她的两份消息，并且 Bob 只能收到其中一份。他能自己做此事，不必与 Alice 通信[105]。

没有人真正注意实际中能否进行不经意传输，但这个概念却是其它协议的重要构成部分。尽管有许多种不经意传输——我有两个秘密你得到一个；我有 n 个秘密你得到一个；我有一个秘密你可能得到其中的 1/2；等等——它们都是等价的[245，391，395]。

5.6 不经意签名

说实话，我不认为它们好用，但是有两种类型[346]：

1．Alice 有 n 份不同的消息。Bob 可以选择其中之一给 Alice 签名，Alice 没有办法知道她签的哪一份消息。

2．Alice 有一份消息。Bob 可以选择 n 个密钥中的一个给 Alice 签署消息用，Alice 无法知道她用的哪一个密钥。

这是一个巧妙的想法；我相信它在某些地方有用。

5.7 同时签约

带有仲裁者的签约

Alice 和 Bob 想订立一个合约。他们已经同意了其中的措词，但每个人都想等对方签名后再签名。如果是面对面的，这很容易：两人一起签。如果距离远的，他们可以用一个仲裁者。

（1）Alice 签署合约的一份副本并发送给 Trent。

（2）Bob 签署合约的一份副本并发送给 Trent。

（3）Trent 发送一份消息给 Alice 和 Bob，指明彼此都已签约。

（4）Alice 签署合约的两份副本并发送给 Bob。

（5）Bob 签署合约的这两份副本，自己留下一份，并把另一份发送给 Alice。

（6）Alice 和 Bob 都通知 Trent 他们每个人都有了一份有他们两人合签的合约副本。

（7）Trent 撕毁在每一份上只有一个签名的两份合约副本。

这个协议奏效是因为 Trent 防止了双方中的某一方进行欺骗。如果在步骤（5）中 Bob 拒绝签约，Alice 可以向 Trent 要求一份已经由 Bob 签署的合约副本。如果在步骤（4）中 Alice 拒绝签名，Bob 也可以这么做。当在步骤（3）中 Trent 指明他收到了两份合约，Alice 和 Bob 知道彼此已受到和约的约束。如果 Trent 在步骤（1）和（2）中没有收到这两份合约，他便撕掉已收到的那份，则两方都不受合约约束。

无仲裁者的同时签约（面对面）

如果 Alice 和 Bob 正面对面坐着，那么他们可以这样来签约[1244]：

（1）Alice 签上她名字的第一个字母，并把合约递给 Bob。

（2）Bob 签上他名字的第一个字母，并把合约递给 Alice。

（3）Alice 签上她名字的第二个字母，并把合约递给 Bob。

（4）Bob 签上他名字的第二个字母，并把合约递给 Alice。

（5）这样继续下去，直到 Alice 和 Bob 都签上他们的全名。

如果你忽视掉这个协议的一个明显问题（Alice 的名字比 Bob 长），这个协议照样有效。在只签了一个字母之后，Alice 知道法官不会让她受合约条款约束。但签这个字母是有诚意的举动，并且 Bob 回之以同样有诚意的举动。

在每一方都签了几个字母之后，或许可以让法官相信双方已签了合约，虽然如此，细节却是模糊的。当然在只签了第一个字母后他们确实不受约束，正如在签了全名之后他们理所

当然受合约约束一样。在协议中哪一点上他们算是正式签约呢？在签了他们名字的一半之后？三分之二之后？四分之三之后？

因为 Alice 或 Bob 都不能她或他受约束的准确点，他们每一位至少有些担心她或他在整个协议上都受合约约束。Bob 在任一点上都无法说："你签了四个字母而我只签了三个，你受约束，但我不受。"Bob 也没有理由不继续这个协议。而且，他们继续得越久，法官裁决他们受合约约束的概率越大。另外，也不存在不继续执行这个协议的理由。毕竟他们都想签约，他们只是不想先于另一方签约。

无仲裁者的同时签约（非面对面）

这个协议使用了同一类型的不确定性[138]。Alice 和 Bob 轮流采用小步骤签署，直到双方都签约为止。

在这个协议中，Alice 和 Bob 交换一系列下面这种形式的签名消息："我同意我以概率 P 接受这个合约约束。"

消息的接方可以把它提交给法官，法官用概率 p 考虑被签署的合约。

（1）Alice 和 Bob 就签约应当完成的日期达成一致意见。

（2）Alice 和 Bob 确定一个双方都愿意用的概率差。例如，Alice 可以决定她不愿以超过 Bob 概率 2%以上的概率受合约约束。叫 Alice 的概率差为 a，叫 Bob 的概率差为 b。

（3）Alice 发送给 Bob 一份 p=a 的已签消息。

（4）Bob 送给 Alice 一份 p=a+b 已签署的消息。

（5）令 p ´ 为 Alice 在前一步中从 Bob 那里收到消息的概率。Alice 发送给 Bob 一份 p= p ´ +a 或 1 中较小的已签署消息。

（6）令 p ´ 为 Bob 在前一步中从 Alice 那里收到消息的概率。Bob 发送给 Alice 一份 p= p ´ +b 或 1 中较小的已签署消息。

（7）Alice 和 Bob 继续交替执行步骤（5）和步骤（6），直到双方都收到 p=1 的消息，或者已通过在第（1）步中达成一致的日期。

随着协议的进行，Alice 和 Bob 都以越来越大的概率同意接受合约约束。例如，Alice 还定义她的的 a 为 2%，Bob 可以定义他的 b 为 1%（如果他们选择较大的增量则更好；我们会在这里停留片刻）。Alice 的第一份消息可能声明她以 2%的概率受约束，Bob 可能回答他以 3%的概率接受约束。Alice 的下一份消息可能声明她以 5%的概率受约束，等等，直到双方都以 100%的概率受约束。

如果由于完成日期 Alice 和 Bob 二者完成这个协议，则万事大吉。否则，任何一方都可把合约拿给法官，并同时递上另一方的最后签的消息，法官在看合约之前在 0 或 1 之间随机选择一个。如果这个值小于另一方签名的概率，则双方都受合约约束。如果这个值大于那个概率，则双方都不受约束（法官接着保存这个值，以防需判定涉及同一合约的其它事件）。这就是以概率 p 受合约约束的意思。

这是一个基本的协议，但还可以有更复杂的协议。法官可在一方缺席的情况下作出判决，法官的判决可约束双方或哪一方都不受约束；不存在一方受约束而另一方不受约束的情况。而且只要一方愿意有比另一方稍微高一点（不管多小）的概率受约束，这个协议将终止。

无需仲裁者的同时签约（使用密码技术）

这种密码协议使用了同样小步进方法[529]。在协议描述中使用了 DES，但也可用任一种对称算法。

（1）Alice 和 Bob 二者随机选择 2n 个 DES 密钥，分成一对对的。这些密钥对没有什么特别之处，它们只是因协议要求而那样分组。

（2）Alice 和 Bob 都产生 n 对消息，例如 L_n 和 R_n：“这是我的第 i 个签名的左半部分”和：“这是我的第 i 个签名的右半部分。”标识符 i 从 1 取到 n。每份消息可能也包含合约的数字签名以及时戳。如果另一方能产生一个单签名对的两半 L_i 和 R_i，那么就认为合约已被签署。

（3）Alice 和 Bob 二者用每个 DES 密钥对加密他们的消息对，左半消息用密钥对中的左密钥，右半消息用密钥对中的右密钥。

（4）Alice 和 Bob 相互发送给对方 2n 份加密消息，弄清哪份消息是哪对消息的哪一半。

（5）Alice 和 Bob 利用每一对的不经意传输协议相互送给对方，即， Alice 送给 Bob 或用于独立地加密 n 对消息中每一对左半消息的密钥；或用于加密右半消息的密钥。Bob 也这样做。他们可以交替地发送这些“半消息”或者先送 100 对，接着再送其余的——这都没有关系。现在 Alice 和 Bob 都有每一对密钥中的一个密钥，但都不知道对方有哪一半。

（6）Alice 和 Bob 用收到的密钥解密他们能解的那一半消息。他们确信解密消息是有效的。

（7）Alice 和 Bob 都把所有 2n 个 DES 密钥的第一个比特发送给对方。

（8）Alice 和 Bob 对所有 2n 个 DES 密钥的第二个比特、第三个比特。重复步骤（7），如此继续下去，直到所有 DES 密钥的所有比特都被传送出去。

（9）Alice 和 Bob 解密剩余一半消息对，合约被签署。

（10）Alice 和 Bob 交换在第（5）步的不经意传输中使用的私钥，并各方验证对方没有欺骗。

为什么 Alice 和 Bob 必须通过所有步骤呢？让我们假设 Alice 想要欺骗，看看会发生什么。在第（4）步和第（5）步中，Alice 可以通过送给 Bob 一批毫无意义的比特字符串来破坏这个协议。Bob 能在第（6）步中发现这一点，即在他试图解密他收到的那一半时，Bob 就可以安全地停止执行协议，此后 Alice 便不能解密 Bob 的任何消息对。

如果 Alice 非常聪明，她可能只破坏协议的一半。她可以正确地送出每对的一半，但送一个毫无意义的字符串作为另一半。Bob 只有 50%的机会收到正确的一半，故 Alice 在一半的时间里可以进行欺骗。但是，这只有在只有一对密钥的情况下起作用。如果有两对密钥，这类欺骗可在 25%的时间里成功。这就是 n 必须很大的原因。Alice 必须正确地猜出 n 次不经意传输协议的结果；她有 2^n 分之一的机会成功。如果 n=10，Alice 有 1024 分之一的机会欺骗 Bob。

Alice 也可以在第（8）步中给 Bob 发送随机比特。也许 Bob 直到收到了全部密钥并试图解密余下的一半消息时才知道 Alice 送给他的是随机比特。但是，Bob 这边也有机会发现。他已经收到了密钥的一半，并且 Alice 不知道是哪一半。如果 n 足够大，Alice 如果确实送给他一个无意义的比特到他已收到的密钥中，则他能立即发现 Alice 在试图欺骗他。

也许 Alice 将继续执行第（8）步直到她有足够多的密钥比特使用穷举攻击，然后再停

止传送比特。DES 有一个 56 比特长的密钥。如果她收到 56 比特中的 40 个比特，她只须试验 2^{16}（65，536）个密钥便能读出这份消息——这个任务对计算机来说当然是轻而易举的。但是 Bob 有同样多数量的她的密钥比特（或最坏是少一个比特），故他也可以读出消息。Alice 除了继续这个协议外别无选择。

基本点是 Alice 必须公正地进行这个协议，因为要欺骗 Bob 的机会太小。在协议结束时，双方都有 n 个签名消息时，其中之一就足以作为一个有效的签名。

有一个 Alice 可以进行欺骗的办法；她可以在第（5）步中发给 Bob 相同消息。Bob 直到协议结束都不能察觉这点，但是他可以使用协议副本让法官相信 Alice 的欺骗行为。

这类协议有两个弱点[138]。首先，如果一方比另一方有强大得多的计算能力，就会产生一个问题。例如，如果 Alice 使用穷举攻击的速度比 Bob 快，那么她能在第（8）步中较早地停止发送比特，并自己推算出 Bob 的密钥。Bob 不能在一个合理的时间内同样做到这一步，将会很不幸。

其次，如果一方提前终止协议，也会产生一个问题。如果 Alice 突然终止协议，双方都面对同样的计算量，但 Bob 没有任何实际合法的追索权，例如，如果合约要求他在一周内做一些事，而在 Alice 真正承诺前的某一时刻终止协议，使得 Bob 将不得不花费一年的计算量，那么就有了一个问题。这里的实际困难是没有一个整个过程能嘎然，而且双方都受约束或者双方都不受约束的近期截止期限。

这些问题也适用于 5.8 节和 5.9 节中的协议。

5.8 数字证明邮件

用作签约的同时不经意传输协议也可以用于计算机证明邮件[529]，但要做一些修改。假设 Alice 要把一条消息送给 Bob，但如果没有签名的收条，她就不让他读出。确实，在实际生活中是邮政工作者处理这一过程，但相同的事可以使用密码术来做。Whitfield Diffie 最先在文献[490]中讨论了这个问题。

乍一看，同时签约协议能做此事。Alice 简单地用一个 DES 密钥加密她的消息。她的一半协议可以是象这样一些东西：“这是这个 DES 密钥的左半：32F5。”Bob 的那一半可以是这样：“这是我的收条的左半。”其他一切保持不变。

要弄明白为什么这个协议不能工作，请记住这个协议依赖这样一个事实，即在第（5）步中的不经意传输保证双方是诚实的。他们两人都知道他们发送给另一方一个有效的半密钥，但都不知道是哪一半。他们在第（8）步中没有进行欺骗是因为做了坏事而不被发觉的机会太小。如果 Alice 要发送给 Bob 的不是一份消息，而是一个 DES 密钥的一半，在第（6）步中 Bob 没有办法检查这个 DES 密钥的有效性。Alice 仍然能检查 Bob 的收条的有效性，所以 Bob 仍然不得不是诚实的。Alice 可以随便发送给 Bob 一些无用的 DES 密钥，直到 Alice 收到一个有效的收条时才知道其中的不同。Bob 实在是命运多舛。

克服这个问题需要对协议进行一些调整：

（1）Alice 用一个随机的 DES 密钥加密她的消息，并把它发送给 Bob。

（2）Alice 产生 n 对 DES 密钥。每对密钥的第一个密钥是随机产生的；每对密钥的第

二个密钥是第一个密钥和消息加密密钥的异或。

（3）Alice 用她的 2n 个密钥的每一个加密一份假消息。

（4）Alice 把所有加密消息都发送给 Bob，保证他知道哪些消息是哪一对的哪一半。

（5）Bob 产生 n 对随机 DES 密钥。

（6）Bob 产生一对指明一个有效收条的消息。比较好的消息可以是“这是我收条的左半”和“这是我收条的右半”，再附加上某种类型的随机比特串。他做了 n 个收条对，每个都编上号。如同先前的协议一样，如果 Alice 能产生一个收条的两半（编号相同）和她的所有加密密钥，这个收条被认为是有效的。

（7）Bob 用 DES 密钥对加密他的每一对消息，第 i 份消息用第 i 个密钥，左半消息用密钥对中的左密钥，右半消息用密钥对中的右密钥。

（8）Bob 把他的消息对发送给 Alice，保证 Alice 知道哪些消息是哪一对的哪一半。

（9）Alice 和 Bob 利用不经意传输协议发送给对方每个密钥对。那就是说，对 n 对中的每一对而言，Alice 或者送给 Bob 用来加密左半消息的密钥，或者送给 Bob 用来加密右半消息的密钥。Bob 也同样这么做。他们可以或者交替传送这些一半，或者一方发送 n 个，然后另一方再发送 n 个这都没有关系。现在 Alice 和 Bob 都有了每个密钥对中的一个密钥，但是都不知道对方有哪些一半。

（10）Alice 和 Bob 都解密他们能解的那些一半，并保证解密消息是有效的。

（11）Alice 和 Bob 送给对方所有 2n 个 DES 密钥中的第一个比特（如果他们担心 Eve 可能会读到这个邮件消息，那么他们应当对相互的传输加密）。

（12）Alice 和 Bob 对所有 2n 个 DES 密钥中的第二比特、第三比特都重复第（11）步，如此继续下去，直到 所有 DES 密钥的所有比特都传送完。

（13）Alice 和 Bob 解密消息对中的余下一半。Alice 有了一张来自 Bob 的有效收条，而 Bob 能异或任一密钥对以得到原始消息加密密钥。

（14）Alice 和 Bob 交换在不经意传输协议期间使用的私钥，同时每一方验证另一方没有进行欺骗。

Bob 的第（5）至第（8）步和 Alice 和 Bob 的第（9）至第（12）步都和签约协议相同。意想不到的手法是 Alice 的所有假消息。它们给予 Bob 一些办法来检查第（10）步中 Alice 的不经意传输的有效性，这可以迫使 Alice 在第（11）至第（13）步期间保持诚实。并且如同同时签约协议一样，完成协议要求 Alice 的一个消息对的左右两半。

5.9 秘密的同时交换

Alice 知道秘密 A；Bob 知道秘密 B。如果 Bob 告诉 Alice B，Alice 愿意告诉 Bob A。如果 Alice 告诉他 A，Bob 愿意告诉 Alice B。这个协议可以在一个校园里被遵守——但很明显它不起作用：

（1）Alice：“如果你先告诉我，我就告诉你。”

（2）Bob：“如果你先告诉我，我就告诉你。”

（3）Alice：“不，你先讲。”

（4）Bob：“噢，好吧”（Bob 悄悄说了）。

（5）Alice：“哈！我不告诉你。”

（6）Bob：“那不公平。”

密码技术可以使它变得公平。前面的两个协议是这个比较通用协议的实现，这个协议允许 Alice 和 Bob 可以同时交换秘密[529]。与其重复整个协议，倒不如简单介绍对数字证明邮件协议的修改情况。

Alice 使用 A 作消息完成第（1）至第（4）步。Bob 用 B 作他的消息完成类似的步骤。Alice 和 Bob 在第（9）步中执行不经意传输，在第（10）步中解密他们能解密的那些一半消息，并在第（11）和第（12）步中处理完那些迭代。如果他们要防范 Eve，他们应当加密他们的消息。最后，Alice 和 Bob 解密消息对余下的一半，并异或任一密钥对来得到原始消息加密密钥。

这个协议使 Alice 和 Bob 可以同时交换秘密，但没有谈到所交换秘密的质量。Alice 可以允诺给 Bob Minotaur 迷宫的解法，但实际上送给他一张波士顿地铁系统交通图。Bob 将得到 Alice 送给他的任何秘密，无论这个秘密是什么。其他的协议见[1286，195，991，1524，705，753，259，358，415]。

第六章 深奥的协议

6.1 保密选举

除非有一个协议既能防止欺骗又能保护个人隐私，否则计算机化的投票永远不会在一般选举中使用。理想的协议至少要有这样六项要求：

（1）只有经授权的投票者才能投票。

（2）每个人投票不得超过一次。

（3）任何人都不能确定别人投谁的票。

（4）没有人能复制其它人的选票。（这一点证明是最困难的要求。）

（5）没有人能修改其他人的选票而不被发现。

（6）每个投票者都可以保证他的选票在最后的表中被计算在内。

此外，有些投票方案可能有如下要求：

（7）每个人都知道谁投了票及谁没有投。

在讨论具有这些特性的复杂投票协议前，我们先看几个比较简单的协议。

简单投票协议#1

（1）每个投票者利用中央制表机构（CTF）的公钥加密他们的选票。

（2）每个投票者把他们的选票送给 CTF。

（3）CTF 将选票解密，制表，公布结果。

这个协议问题成堆。CTF 不知道选票从何而来，故它甚至不知道选票是否来自合格的投票者。他们也不知道这些合格的投票者是否投了一次以上的票。在正的一方，没有人能改变其他人的选票，但是当你可以相当容易地将你的选择结果投无数次时，也就没有人试图去修改其他人的选票。

简单投票协议#2

（1）每个投票者用他的私钥在选票上签名。

（2）每个投票者用 CTF 的公开密钥加密他们的签了名的选票。

（3）每个投票者把他的选票送给 CTF。

（4）CTF 解密这些选票，检查签名，将选票制表并公布结果。

这个协议满足了性质 1 和性质 2：只有被授权的投票者才能投票，并且任何人都不能投一次以上的票。CTF 在第（3）步中记录收到的选票。每张选票都用投票者的私钥签名，故 CTF 知道谁投了票？谁没有投票？以及每个投票者投了多少次？如果出现没有由合格投票者签名的选票，或者出现另一张由一个已投过票的投票者签名的选票，那么机构可以不计这张选票。没人能改变其他任何人的选票，即使他们在第（2）步截获了它。这个协议的问题在于签名附在选票上，故 CTF 知道谁投了谁的票。用 CTF 的公开密钥加密选票阻止了任何人在协议进行中窃收，并了解谁投了谁的票，但是你得完全信任 CTF。它类似于有一个选举监督员在背后盯着你把票投入票箱。

这两个例子说明要满足安全投票协议的前三个要求是多么困难，更别说其它的要求了。

使用盲签名投票

我们需要以某种办法切断投票者与选票的关系，同时仍能保持鉴别。盲签名协议正好可以做到这一点。

（1）每个投票者产生 10 个消息集，其中对每一种可能结果都有一张有效选票（例如，如果选票是一个“Yes”或“No question”，则每个集合中包含两张选票，一张“Yes”且另一张“No”）。每条消息也包含一个随机产生的识别号，这个数要大到足以避免和别的投票者重复。

（2）每个投票者分别隐蔽所有的消息（见 5.3 节）并把它们同盲因子一道送给 CTF。

（3）CTF 检查它的数据库以保证投票者先前不曾以他们的签名提交过隐蔽好的选票。它打开 9 个集合以检查它们是否正确形成，然后它分别签名这个集合中的每一条消息。接着把它们送还给投票者，并把投票者的名字存在它的数据库中。

（4）投票者除去这些消息的隐蔽，留下由 CTF 签名的一组选票（这些选票签了名，但未加密，故投票者能轻易地知道哪 张选票是“Yes”及哪张是“No”）。

（5）投票者选择其中一张选票，（哈，很民主）并用 CTF 的公开密钥对它加密。

（6）投票者投出他们的选票。

（7）CTF 将选票解密，检查签名，检查它的数据库是否有重复的识别号，保存这个序号并将选票制表。公布选举结果以及每个序号和其相关的选票。

一个恶意的投票者，我们不妨称之为 Mallory，他不可能欺骗这个系统。盲签名协议确保他的选票是独一无二的。如果他试图在同一次选举中投两次票，则 CTF 将会在第（7）步中发现重复的系列号并把第二张选票扔掉。如果他试图在第（2）步中得到多张签了名的选票，则 CTF 将在第（3）步中发现这一点。因为 Mallory 不知道这个机构的私钥，故他也不能产生他自己的选票。同样他也不能截取和改变其他人的选票。

第（3）步的分割——选择协议是为了保证选票的唯一性。没有这一步，Mallory 可以制造出大量的相同的选票，除了识别号不同，并且使这些选票全都有效。

一个恶意的 CTF 不可能了解个人如何投票。因为盲签名协议防止了这个机构在人们投票前看到选票上的序码，则 CTF 无法把它签名的隐蔽好的选票与最终投出的选票联系起来。公布系列号清单和它们的相关选票使得投票者能肯定他们的选票被正确地统计制表。

这里仍然有问题。如果第（6）步不是匿名的，CTF 能记录下谁投了哪 张选票，那么它就能知道谁投谁的票。但是，如果它收到的选票在一个锁着的选票箱里，并且随后把它们制表，则它就不能记录。两样，还有，虽然 CTF 不能把选票同个人联系起来，但它能产生大量签名的有效选票，供他自己进行欺骗。而且如果 Alice 发现 CTF 修改了她的选票，她没有办法证明。[1195，1370]中有一个类似的协议试图弥补这些问题。

带两个中央机构的投票

一种解决办法是将 CTF 一分为二。没有哪一方自己有能力进行欺骗。

下面这个协议使用一个中央合法机构（CLA）来证明投票者，以及一个单独的 CTF 来计票[1373]。

（1）每个投票者发送一条消息给 CLA 要求得到一个有效数字。

（2）CLA 送还给投票者一个随机的有效数字。CLA 保持一张有效数字的列表，CLA

也保留一张有效数字接受者的名单，以防有人试图再次投票。

（3）CLA 把有效数字的列表送给 CTF。

（4）每个投票者选择一个随机识别号。他们用该识别号、从 CLA 收到的有效数字和他们的选票一起产生一条消息，把这消息送给 CTF。

（5）CTF 对照它在第（3）步中从 CLA 收到的列表来检验有效数字。如果数字存在，CTF 就把它划掉（防止任何人投票两次）。CTF 把识别号加到投了某位候选者的人员名单上，并在记数中加一。

（6）在收到了所有的选票后，CTF 公布结果、识别号以及这些识别号所有者投了谁的票。

就象前面的协议一样，每个投票者能够看到识别号的列表，并在其中找到他自己的识别号，这就证明他的选票被计了数。当然，协议中各方之间传递的所有消息应当加密并签名，以防止一些人假冒另一些人或截取传送。

因为每个投票者都要寻找他们的识别字符串，故 CTF 不能修改选票。如果投票者找不到他的识别号，或者发现他的识别号在不是他们所投票的记录中，他会立即知道这中间有舞弊行为。因为 CTF 受 CLA 监督。所以它不能把假选票塞进投票箱。CLA 知道有多少个投票者正被证明及他们的鉴别数字，并会检测到任何窜改。

Mallory 不是一个合格的投票者，他可以试图通过猜测有效数字来进行欺骗。但通过使可能的有效数字比实际有效数字大得多的方法可使这种威胁降到最低限度。例如，一百万个投票者的一百位数字。当然，有效数字必须是随机产生的。

尽管这样，CLA 在一些方面仍是一个可信任的机构。它能验证出不合格的投票者。它能对合格投票者多次验证。通过让 CLA 公布被验证的投票者（但不是他们的鉴别数字）的清单可使这种风险最小化。如果这个清单上投票者的数目小于已造表的选票的数目，那么肯定其中有诈。如果被验证的投票者比已造表的选票多，可能意谓着一些被验证的人未投票。很多人注册投票，但却没有将选票投进票箱。

这个协议也易受 CLA 和 CTF 的合谋攻击。如果它们两个串通一气，那它们可以将数据库联系起来并知道谁投了谁的票。

带一个中央机构的投票

使用一个更复杂的协议能克服 CLA 和 CTF 合谋的危险[1373]。这个协议和前面一个基本相同，但作了两处修改：

- CLA 和 CTF 是一个组织，并且
- ANDOS（见 4.13 节）用来在第（2）步中匿名分配有效数字。

因为匿名的密钥分配协议防止了 CTF 知道哪个投票者得到了哪个有效数字，故 CTF 没有办法把收到的选票和有效数字联系起来。虽然仍必须相信 CTF 不会把有效数字给不合格的投票者。即使如此你也可以使用盲签名来解决这个问题。

改进的带单个中央机构的投票

这个协议也使用 ANDOS[1175]。它满足一个好的投票协议的所有六个要求。它不满足第 7 个要求；但有另外两个性质，我们将其列在这一节的开始：

（7）投票者在一个给定的时间内改变主意。（即，收回他们的选票并重新投票。）

（8）如果投票者发现他们的选票被误计，他们能够鉴别并纠正这个问题，同时不会危害到他投票的秘密。

下面是这个协议：

（1）CTF 公布所有合法投票者的名单。

（2）在一个指定的截止日期内，每一投票的人都把他的投票意图告诉 CTF。

（3）CTF 公布参加选举的投票者。

（4）每个投票者都使用 ANDOS 协议，收到一个鉴别数字 I。

（5）每个投票者产生一个公钥/私钥密钥对：k 和 d。如果 v 是选票，他们产生出下列消息并将它送给 CTF：

$$I, E_k（I，v）$$

该消息必须匿名发送。

（6）CTF 通过公布 E_k（I，v）确认收到选票。

（7）每个投票者送给 CTF：

$$I，d$$

（8）CTF 用 d 解密选票。在选举结束时，它公布选举结果和对每张不同选票公布所有包含那张选票的所有 E_k（I，v）值的列表。

（9）如果投票者发现他们的选票没有被正确计数，他们通过给 CTF 发送：

$$I, E_k（I，v）, d$$

来表示抗议。

（10）如果投票者想把他的选票从 v 改到 v ´（这在一些选举中是可能的），他们送给 CTF：

$$I, E_k（I，v）v ´$$

有一种不同的投票协议用盲签名代替 ANDOS，但其本质是相同的[585]。步骤（1）至步骤（3）是实际投票的开端。它们的目的是弄清楚并公布实际投票者的总数。虽然一些人可能不参加，但它减少了 CTF 增加假冒选票的可能性。

在第（4）步中，两个投票者得到相同的鉴别数字是可能的。通过让可能的鉴别数字远远超过实际的投票者的数目能使这种可能性最小化。如果两个投票者提交了带相同鉴别标记的选票，CTF 就产生一个新的鉴别数字 I ´，选择两张选票中的一张并公布：

$$I ´ E_k（I，v）$$

这张选票的所有者认出它来，并通过重复第（5）步，交上第二张带新的鉴别数字的选票。

第（6）步让所有投票者能够查知 CTF 确实收到了他们的选择。并且如果他们的投票被误计了，他们可以在第（9）步中证明这一情况，假设在第（6）步中投票者的选票是对的，则在第（9）步中他们发送的消息构成了他们选票被误计的证据。

这个协议的一个问题是一个腐败的 CTF 可给在第（2）步中响应的人分发选票，但不能给实际不投票的人分发选票。另一个问题是 ANDOS 协议的复杂性。作者建议把一大群投票人分成几个小群，如同一个个选区。

另一个更为严重的问题是 CTF 可能漏计选票。这个问题无法解决：Alice 宣称 CTF 故

意漏计她的选票，但是 CTF 宣称投票人没有投票。

不带中央制表机构的投票

这个协议完全省却了使用 CTF，投票者互相监督。它是由 Michael Merrit 设计的[452，1076，453]。它是如此难操作以至只有在少数几个人中才能实际地实现，然而我们可以从中学到一些东西。

Alice、Bob、Carol 和 Dave 正在对一个特殊问题进行是或否（0 或 1）的投票。假设每个投票者都有一个公开密钥和一个私钥。也假设每个人都知道其他人的公开密钥。

（1）每个投票者选择一张选票并做以下事情：

（a）在他们的选标上附一个随机字符串。

（b）用 Dave 的公开密钥加密步骤（a）的结果。

（c）用 Carol 的公开密钥加密步骤（b）的结果。

（d）用 Bob 的公开密钥加密步骤（c）的结果。

（e）用 Alice 的公开密钥加密步骤（d）的结果。

（f）在步骤（e）的结果中附上一个新的随机字符串，并用 Dave 的公开密钥对它加密。他们记下这个随机字符串的值。

（g）在步骤（f）的结果中附上一个新的随机字符串，并用 Carol 的公开密钥对它加密。他们记下这个随机字符串的值。

（h）在步骤（g）的结果中附上一个新的随机字符串，并用 Bob 的公开密钥对它加密。他记下这个随机字符串的值。

（i）在步骤（h）的结果中附上一个新的随机字符串，并用 Alice 的公开密钥对它加密。他们记下这个随机字符串的值。

如果 E 是加密函数，R 是一个随机字符串，且 V 是选票，则选票看起来像：

$$E_A(R_5, E_B(R_4, E_C(R_3, E_D(R_2, E_A(E_B(E_C(E_D(V, R_1)))))))))$$

所有的投票者记下计算中每一点的中间结果。在协议中后面将会用这些结果来确定他们的选票被计了数。

（2）每个投票者把他的选票送给 Alice。

（3）Alice 用她的私钥对所有的选票解密，接着将那一级中所有随机字符串删去。

（4）Alice 置乱所有选票的秩序并把结果送给 Bob。每张选票现在看起来像这个样子：

$$E_B(R_4, E_C(R_3, E_D(R_2, E_A(E_B(E_B(E_C(E_D(V, R_1)))))))))$$

（5）Bob 用他的私钥对所有的选票解密，查看他的选票是否在选票集中，删去那一级中所有随机字符串，置乱所有的选票然后把结果送给 Carol。每张选票现在看起来像这个样子：

$$E_C(R_3, E_D(R_2, E_A(E_B(E_C(E_D(V, R_1)))))))$$

（6）Carol 用她的私钥对所有的选票解密，查看她的选票是否在选票集中，删去那一级所有的随机字符串，置乱所有的选票，然后把结果送给 Darve。

每张选票现在看起来像这个样子：

$$E_D(R_2, E_A(E_B(E_C(E_D(V, R_1))))))$$

（7）Dave 用他的私钥对所有的选票解密，查看他的选票是否在选票集中，删去那一级

中所有随机字符串，置乱所有的选票，并把结果送给 Alice。

每张选票现在看起来像这个样子：

$$E_A（E_B（E_C（E_D（V，R_1））））$$

（8）Alice 用她的私钥对所有选票解密，查看她的选票是否在选票集中，签名所有选票，并把结果送给 Bob、Carol 和 Dave。每张选票现在看起来像这个样子：

$$S_A（E_B（E_C（E_D（V，R_1））））$$

（9）Bob 验证并删去 Alice 的签名。他用他的私钥对所有的选票解密，查看他的选票是否在选票集中，对所有的选票签名，然后把结果送给 Alice、Bob 和 Dave。每张选票现在看起来是这个样子：

$$S_B（E_C（E_D（V，R_1））$$

（10）Carol 验证并删去 Bob 的签名。她用她的私钥对所有选票解密，，查看她的选票是否在选票集中，对所有的选票签名，然后把结果送给 Alice、Bob 和 Dave。每张选票现在看起来是这个样子：

$$S_C（E_D（V，R_1））$$

（11）Dave 验证并删去 Carol 的签名。他用他的私钥对所有选票解密，，查看他的选票是否在选票集中，对所有的选票签名，然后把结果送给 Alice、Bob 和 Carol。每张选票现在看起来是这个样子：

$$S_D（V，R_1）$$

（12）所有人验证并删去 Dave 的签名。通过检验以确信他们的选票在选票集中（通过在选票中寻找他们的随机字符串）。

（13）每个人都从自己的选票中删去随机字符串并记录每张选票。

这个协议不仅起作用，而且还能自我判决的。如果有人试图进行欺骗，Alice、Bob、Carol 和 Dave 将立即知道。这里不需要 CTF 和 CLA。为了弄清楚这是怎样起作用的，让我们来试演行骗。

如果人有想把假票塞进票箱，Alice 在第（3）步当她收到比人数多的选票时就会发现这一企图。如果 Alice 试图把假票塞进票箱，Bob 将在第（4）步中发现。

一种更狡猾的欺骗方法是用一张选票替换另一张。因为选票是用各种不同的公开密钥加密的，任何人都能按其需要创造很多有效的选票。这里解密协议有两轮：第一轮包括第（3）至第（7）步，第二轮包括第（8）至第（11）步。替换选票会在不同轮次被分别发现。

如果有人在第二轮中用一张选票替换另一张，他的行为会立即被发现。在每一步上选票被签名并送给所有投票者。如果一个（或更多）的投票者注意到他的选票不再在选票集中，他就立即中止协议。因为选票在每一步都签了名，并且因为每个人都能反向进行协议的第二轮，故很容易发现谁替换了选票。

在协议的第一轮用一张选票替换另一张显得更为高明。Alice 不能在第（3）步中这样做，因为 Bob、Carol 或 Dave 会在第（5）、（6）、（7）步中发现。Bob 可以第（5）步中这样做。如果他替换了 Carol 或 Dave 的选票（记住，他不知道哪张选票对应哪个投票者），Carol 或 Dace 将在第（6）或第（7）步中发现，他们不知道谁窜改了他们的选票（虽然这一定是某个已经处理过选票的人），但他们知道他们的选票被窜改了。如果 Bob 幸运地挑选了 Alice

的选票来替换，她要到第二轮才会发现。接着，Alice 在第（8）步中会发现她的选票遗失了。但她仍然不知道谁窜改了她的选票。在第一轮中，选票在从一步到另一步时被搅乱并且未被签名；任何人都不可能反向跟踪协议以确定谁窜改了选票。

另一种形式的骗术是试图弄清楚谁投了谁的票。因为置乱是在第一轮，故任何人都不可能反向跟踪协议，并把投票者与选票联系起来。在第二轮中删去随机字符串对保护匿名性来说关系重大。如果它们未被删除，通过用置乱者的公开密钥对出现的选票重新加密便能将选票的置乱还原。由于协议的固有性质，选票的机密性是有保障的。

更有甚者，因为有初始随机字符串 R_1，即使一样的选票在协议的每一步都被加密成不同的选票。直到第（11）步人们才能知道选票的结果。

这个协议的问题是什么呢？首先，这个协议计算量特别大。前面所述的例子仅有四个投票者，就已经很复杂了。这个协议在实际的选举无法凑效，因为有成千上万的投票者。其次，Dave 先于其他人知道选举结果。虽然他还不能影响选举结果，但这给了他一些别人没有的权力。另一方面，带有中央化的投票方案也是合乎实际情况的。

第三个问题是 Alice 能拷贝其他人的选票，即使事先她并不知道它是什么。为了弄清这是一个问题的原因，设想一个 Alice，Bob，和 Eve 的三人选举。Eve 并不关心选举结果，但是她想知道 Alice 是怎样投票的。因此她拷贝 Alice 的选票，保证选举的结果等于 Alice 的投票。

其他投票方案

人们已经提出许多复杂的安全选举协议。它们来自两个不同风格的基本协议。有一些混合协议，象“没有中央制表机构的协议”，这里每人的选票都被混合以便没有人能把选票与投票者联系起来。

也有被分开的协议，单独的选票在不同的制表机构被分散开，单独的一个机构不能欺骗投票者[360，359，118，115]。这个协议仅在政府的（或管理投票的机构的）“不同”部门不串通起来对付投票者的情况下才能保护投票者的隐私。（将一个中央机关分成不同部门，仅在它们都聚在一起时才可信任的想法来自文献[316]）。

文献[1371]中有一个被分开的协议。它的基本思想是每一个投票者把他的投票分成几分。例如，如果选票是“yes”或“no”，可以用 1 代表“yes”而 0 代表“no”；然后投票者产生几个数字，它们的和为 1 或 0。每一分送给制表机构，一个部分一份，并且每一分都被加密邮寄。每个中心标记它收到的那些部分（有一个验证标记是否正确的协议），最终的投票结果是所有的标记之和。还有一个协议保证每个投票者的部分数值和为 1 或 0。

David Chaum 提出的另一个协议[322]，它确保跟踪任何企图破坏选举的投票人。但是，那样做必须重复选举过程，并在不得干扰投票者的情况下；这种方法对于大规模选举是不实用的。

另一个更复杂的投票协议解决了一些这方面问题[770，771]。甚至有一个投票协议使用了多密钥密码[219]。还有一个投票协议，宣称对大规模选举是实用的，见文献[585]。文献[347]允许投票者弃权。

投票协议有效、但他们使买卖选票变得更加容易。这种动机变得相当强烈，因为买方相信出售的选票是合法的。一些协议被设计成不要收条的，这使得投票者以某种方式向其他人

证明他的投票变得不可能[117，1170，1372]。

6.2 保密的多方计算

保密的多方计算是一种协议，在这个协议中，一群人可在一起用一种特殊的方法计算许多变量的任何函数。这一群中的每个人都知道这个函数的值，但除了函数输出的明显东西外，没有人知道关于任何其他成员输入的任何事情。下面是几个例子：

协议#1

一群人怎样才能计算出他们的平均薪水而又不让任何人知道其他人的薪水呢？

（1）Alice 在她的薪水上加一个秘密的随机数，并把结果用 Bob 的公开密钥加密，然后把它送给 Bob。

（2）Bob 用他的私钥对 Alice 的结果解密。他把他的薪水加到他从 Alice 那里收到的结果上，用 Carol 的公开密钥对结果加密，并把它送给 Carol。

（3）Carol 用她的私钥对 Bob 的结果解密。她把她的薪水和她从 Bob 那收到的结果相加，再用 Dave 的公开密钥对结果加密，并把它送给 Dave。

（4）Dave 用他的私钥对 Carol 的结果解密。他们他的薪水和他从 Carol 那收到的结果相加，再用 Alice 的公开密钥对结果加密，并把它送给 Alice。

（5）Alice 用她的私钥对 Dave 的结果解密。她减去第一步中的随机数以恢复每个人薪水之总和。

（6）Alice 把这个结果除以人数（在这里是 4），并宣布结果。

这个协议假定每个人都是诚实的。如果参与者谎报了他们的薪水，则这个平均值将是错误的。一个更严重的问题是 Alice 可以对其他人谎报结果。在第五步她可以从结果中减去她喜欢的数，并且没有人能知道。可以通过使用任何 4.9 节中的比特提交方案要求 Alice 提交她的随机数来阻止 Alice 这样做，但当 Alice 在协议结束时泄露了她的随机数，Bob 就可以知道他的薪水。

协议#2

Alice 和 Bob 一起坐在一家餐馆中，正在争论谁的年纪大。然而他们都不想告诉对方他们的年龄。他们可以都把他们的年龄悄悄地告诉一个可信赖的中立方（例如侍者），那个人在脑中比较两个数并对 Alice 和 Bob 宣布这个结果。

关于上面这个协议存在两个问题。一个是，普通侍者不能处理比确定两个数之中哪一个大更复杂问题的计算能力。另一个是，如果 Alice 和 Bob 真的关心他们的信息的秘密，他们不得不将这个侍者扔进一个汤碗淹死，以免他告诉酒楼服务员。

公开密钥密码术提供了一个远非如此残暴的解决办法。有一个协议是 Alice 知道一个值 a，且 Bob 知道一个值 b，他们能一起确定 a 是否小于 b，而 Alice 得不到 b 的任何信息，Bob 也得不到 a 的任何信息。并且，Alice 和 Bob 能相信计算的有效性。所有的密码学算法是这个协议的重要部分，详细情况参见 23.14 节。

当然，这个协议不能防止主动欺骗者。没有办法能防止 Alice（或 Bob）谎报他们的年

龄。如果 Bob 是一个隐蔽执行这个协议的计算机程序，那么 Alice 通过反复执行这个协议可以知道他的年龄（一个计算机程序的年龄是从程序被写出的时间长度？还是从它开始运行的时间长度？）。Alice 可以在执行这个协议时，指定她的年龄为 60。在得知她的年纪大时，她可以将她的年龄指定为 30，再次执行这个协议。在得知 Bob 要大一些之后，她可以称她的年龄为 45 再次执行这个协议，依此继续下去，直到 Alice 发现 Bob 的年龄达到她所希望的精确度。

假设参与者不主动欺骗，很容易把这个协议推广到多个参与者。任何数量的人通过一系列诚实应用这个协议可以发现他们的年龄的顺序；并且没有一个参与者能够得知另一个参与者的年龄。

协议#3

Alice 喜欢用玩具熊做一些古怪的事。Bob 则沉迷于大理石桌子。他们都对他们的癖好特别难为情，但却想找一个有共同生活风格的伴侣。

在保密的多方计算约会服务（Secure Multiparty Computation Dating Service）中，我们为这样的人设计了一个协议。我们已将一个令人惊奇的嗜好名单编号，从“土豚”到“阻特层装”。Alice 和 Bob 彼此分开，通过一个调制解调器相连，他们便能参与一个保密的多方协议。他们可以一起确定他们是否有同样的癖好。如果有的话，他们可以期望建立一种终生的幸福关系。如果没有，他们可以彼此分开，而且他们的特殊个人信息仍保持机密。没有人，甚至是保密多方计算约会服务也不知道。

下面是该协议如何工作的：

（1）使用一个单向函数，Alice 将她的癖好做散列得到一个 7 个数字的字符串。

（2）Alice 用这 7 个数字作为一个电话号码，拨号，给 Bob 留下一条消息。如果没有人回答或电话号码无效，Alice 给这个电话号码申请一个单向函数直到她找到一个与她有相同癖好的人。

（3）Alice 告述 Bob 她为她的癖好申请一个单向函数所需的时间。

（4）Bob 花了与 Alice 相同的时间散列他的癖好。他也用这 7 个数字作为电话号码，询问其他人是否有留给他消息。

注意 Bob 有选择明文攻击。他可以散列一般的癖好并拨所得的电话号码，查找给他的消息。只有在不可能得到足够多的明文消息的情况下这个协议才能真正执行。

也有一个数学的协议，类似于协议#2。Alice 知道 a, Bob 知道 b，并且他们在一起可以确定是否 a=b，但 Bob 不知道关于 a 的任何事且 Alice 不知道关于 b 的任何事。详细情况请参见 23.14 节。

协议#4

对保密的多方计算存在另一问题，见文献[1373]：这里有一个七方委员会，他们定期开会对某一些问题秘密表决。（不错，他们秘密地统治着世界——不要把我告诉你的事告诉任何人。）所有委员会成员可以投票表决 Yes 或 No。另外，有两方有权利投“超级选票”：S—Yes 和 S—no。他们不一定要投“超级选票”，如果他们愿意，他们可以投一般选票。如果没有人投“超级选票”，则选票的多数决定这个问题。在一张或两张结果相同的“超级选票”情况下，所有普通选票被忽略。在两张结果相反的“超级选票”情况下，普通选票的多数决定这一问题。我们需要一个能安全执行这类投票的协议。

下面两个例子将阐述投票过程。假设有五个普通投票者：N_1 到 N_5，和两个超级投票者：S_1 和 S_2，下面是关于问题#1 的选票：

S_1	S_2	N_1	N_2	N_3	N_4	N_5
S - yes	no	no	no	no	yes	yes

在这个例子中唯一起作用的选票是 S_1 的票并且结果是 yes。下面是关于问题#2 的选票：

S_1	S_2	N_1	N_2	N_3	N_4	N_5
S - yes	S -no	no	no	no	yes	yes

这里两张超级选票抵消，普通选择的多数 no 选票决定这一问题。

如果隐藏“超级选票”抑或普通选票是决定性选票的信息无需隐藏，这就是一个安全投票协议的简单应用。如果隐藏这一点很重要，就需要一个更复杂的保密多方计算协议。

这种投票可能会发生在现实生活中。它可以是一个公司组织结构的一部分，在这里某些人比其他人有更大的权力，或者它是联合国的做法的一部分，其中某些国家比其他国家有更大的权力。

无条件多方安全协议

这只是一般定理的一种简单情况：任何 n 输入的函数可以被 n 个人用这种办法计算，使得所有人都知道函数的值，但任何少于 n/2 个人的一群人都得不到除了他们自己的输入以及输出信息值之外的任何附加信息。细节见[136，334，1288，621]。

保密电路计算

Alice 的输入为 a，Bob 的输入为 b。他们希望一起计算一些普通函数 f（a，b），这样使得 Alice 不知道 Bob 的输入情况 Bob 也不知道关于 Alice 的输入情况。保密多方计算的一般性问题也称为保密电路计算。这里，Alice 和 Bob 可以创造一个任意的布尔电路，这个电路接受来自 Alice 和来自 Bob 的输入并产生一个输出。保密电路计算是一个完成下面三件事的协议：

（1）Alice 可以键入她的输入且 Bob 不能知道它。

（2）Bob 可以键入他的输入且 Alice 不能知道它。

（3）Alice 和 Bob 都能计算出这个输出，双方都确信输出是正确的且没有一方能窜改它。

保密电路计算的详细情况参见文献[831]。

6.3 匿名报文广播

你无法同一群密码员一起出去进餐而不引起争吵。在文献[321]中，David Chaum 提出了密码员进餐问题：

> 三位密码员正坐在他们最喜欢的三星级餐馆准备进餐。侍者通知他们这是 Maitre d’hotel 安排的且需匿名支付帐单。其中一个密码员可能正在付帐，或者可能已由 NSA 付过了。这三位密码员都尊重彼此匿名付帐的权利，但他们要知道是不是 NSA 在付帐。

这三个密码员分别叫 Alice、Bob 和 Carol，他们怎样才能确定他们之中的一个正在付帐同时又要保护付帐者的匿名呢？

Chaum 接着解决了这个问题：

每个密码员在他的菜单后，在他和他右边的密码员之间抛掷一枚无偏硬币，以致只有他们两个能看到结果。然后每个密码员都大声说他能看到两枚梗币——他抛的一个和他左手邻居抛的那个——落下来是同一面还是不同的一面。如果有一个密码员付帐，他就说所看到的相反的结果。在桌子上说不同的人数为奇数表明有一个密码员在付帐；不同为偶数表明 NSA 在付帐（假设晚餐只付一次帐）。还有，如果一个密码员在付帐，另两个人都不能从所说的话中得知关于那个密码员付帐的任何事。

为了明白这是如何起作用的。不妨想象 Alice 试图弄清其他哪个密码员为晚餐付了帐（假设既不是她也不是NSA付的）。如果她看见两个不同的硬币，那么另两个密码员 Bob 和 Carol，或者都说“相同”，或者都说“不同”（记住，密码员说“不同”的次数为奇数，表明他们中有一个付了帐）。如果都说“不同”，那么付帐者是最靠近与未看见的硬币不同的那枚硬币的密码员。但是，如果 Alice 看见两枚硬币是相同，那么或者 Bob 说“相同”而 Carol 说“不同”，或者 Bob 说“不同”而 Carol 说“相同”。如果未看见的硬币和她看到的两枚硬币是相同的。那个说“不同”的密码员是付帐者。如果隐藏的硬币和她看到的两枚硬币是不同的，那么说“相同”的密码员是付帐者。在所有这些情况中，Alice 都需要知道 Bob 和 Carol 抛掷硬币的结果以决定是他们中的哪一位付的款。

这个协议可以推广到任意数量的密码员：他们全都坐成一圈并在他们中抛掷硬币。甚至两个密码员也能执行这个协议；当然他们知道谁付的帐，但是观看这个协议的人只知道是一个密码员付的帐还是 NSA 付的帐；他们不会知道是哪个密码员付的帐。

这个协议的应用远远超出了围坐在一家餐桌的范围。这是一个无条件的发方和收方不可追踪性的例子。在网络上的一群用户可以用这个协议发送匿名报文。

（1）用户把他们自己排进一个逻辑圆圈。

（2）在一定的时间间隔内，相邻的每对用户对在他们之间抛掷硬币，使用一些公正的硬币抛掷协议防止窃听者。

（3）在每次抛掷之后每个用户说“相同”或“不同”。

如果 Alice 希望播发一条报文，那她可以简单地对应着报文的二进制表示的 1 开始颠倒那些轮中的陈述。例如，如果消息是“1001”，他颠倒她的陈述，说出真情，然后再颠倒她陈述。假设她抛掷的结果是“不同”“相同”“相同”“相同”，她将说“相同”“相同”“相同”“不同”。

如果 Alice 发现协议的所有结果都和他们要发送的报文不匹配，那她知道其他人同时也正在试图发送一条报文。然后她停止发送报文，并再次试图发送前等待一个随机数目的轮次。虽然人们必须基于这个网络上报文通信的数量算出准确参数，但这个想法应该是清楚的。

为了让这些事更令人感兴趣，可以用其他用户的公开密钥加密。然后，当每个人都收到这个报文（这个协议的真正实现应该加上一种标准的报文开始和报文结束字符串）时，只有想要接收的人能解密并读出这份报文。其他人都不知道谁发送的它，。其他人也不知道谁能读它。虽然信息流量分析可以跟踪并搜集编辑人们通信的模式，即使这些报文本身加了密，但对此也无能为力。

代替在相邻方之间抛掷硬币的一种方法是他们保留一个随机比特的共同文件。他们也许可以把它们放在一个 CD-ROM 上，或者这一对中一方可以产生一堆随机比特并把它们送给

另一方（当然是加密的）。另一个办法是，他们可以在他们之间商定一个保密的伪随机数产生器，并且他们每一个都能为协议产生相同的伪随机比特字符串。

这个协议的一个问题是虽然一个恶意的参与者不能读出报文，但他能通过在第（3）步中说谎来破坏系统。修改先前的协议可以检测到破坏[1578，1242]；这个问题被称为“在迪斯科舞厅里吃饭的密码员”。

6.4 数字现金

现金是一个问题。它难于搬运，传播病菌并且别人能从你那里把钱偷走。支票和信用卡大大减少了社会上实际现金的流通量，但根本不可能完全取消现金。这永远不会发生，毒品贩子和政治家永远不会赞成它。支票和信用卡具有审计线索，你不可能隐瞒你把钱给了谁。

另一方面，支票和信用卡使别人可以侵犯你的隐私，其程序是以前想象不到的。你决不会同意警察跟随你一生，但是警察可以查看你的金融交易。他们能看你在哪里买汽油，在哪里买食物，你和谁通电话——所有这一切都逃不过他们的计算机终端。人们需要有一种方法来保护他们的匿名权、藉以保护他们的隐私。

幸运的是，有一个复杂协议容许消息可以确证，但不可跟踪。说客 Alice 能把**数字现金**转移给参议员 Bob，并使得新闻记者 Eve 不知道 Alice 的身份，Bob 然后可把这笔电子货币存入他的帐户，即使是银行也不知道 Alice 是谁。但是如果 Alice 试图以用来贿赂 Bob 的同一笔数字现金来买可卡因，那么她会被银行检测出来。如果 Bob 试图把同一笔数字现金存入两个不同帐户，她也会被发现——但 Alice 仍保持匿名。为了把它与带审计追踪的数字现金如信用卡相区别，有时把它叫做**匿名数字现金**。这类东西有着很大的社会需求。随着网上商业交易的发展，商业上日益需要更多的称为网络隐私和网络匿名的东西。（人们有很好的理由不愿意通过互联网传送他们的信用卡号。）另一方面，银行和政府似乎不愿放弃目前银行系统提供的审计追踪控制。尽管如此，他们不得不放弃。一些可信赖的机构愿意将数字现金转换为真正的现金，所有的银行也都将提供数字现金服务。

数字现金协议非常复杂。我们将构建一个，一步一步来。较正式的细节，参见文献[318，339，325，335，340]。要认识到这只是一个数字现金协议；还有其他的。

协议#1

前面几个协议是密码协议的具体应用。这第一个协议是一个有关匿名汇票的简单化的物理协议：

（1）Alice 准备了 100 张 1000 美元的匿名汇票。

（2）Alice 把每张汇票和一张复写纸放进 100 个不同信封内，她把这些全部交给银行。

（3）银行开启 99 个信封并确认每个都是一张 1000 美元的汇票。

（4）银行在余下的一个未开启的信封上签名，签名通过复写纸印到汇票上。银行把这个未开启的信封交还 Alice，并从她的帐户上扣除 1000 美元。

（5）Alice 打开信封并在一个商人处花掉了这张汇票。

（6）商人检查银行的签名以确信这张汇票是合法的。

（7）商人拿着这张汇票到银行。

（8）银行验证它的签名并把 1000 美元划入这个商人的帐户。

这个协议能起作用。银行从未看到它签的那张汇票，故当这个商人把它带到银行时，银行不知道它是 Alice 的，虽然如此，因为这个签名的缘故银行还是相信它有效，那是因为这个签名的缘故。银行相信未开启的汇单是 1000 美元的（既不是 100 000 美元也不是 100 000 000 美元），那是因为采用分割选择协议（见 5.1 节）的缘故。它验证了其他 99 个信封，故 Alice 仅有 1%的机会欺骗银行。当然，银行对于欺诈将进行足够狠的惩罚以致它与机会相比是不值的。如果银行只是拒绝在最后一张支票上签名（如果 Alice 被发现在欺骗）而不惩罚 Alice 的话，她将继续尝试直到她碰上大运。坐牢是一种较好的威慑。

协议#2

前一个协议防止了 Alice 在一张汇票上写入比她宣称的更多的钱，但它没有防止 Alice 将这张汇票照相复制并两次花掉它。这叫做“双重花费问题”，为了解决这个问题，我们需要一个复杂的协议：

（1）Alice 准备 100 张每张 1000 美元的匿名汇票。在每一张汇票上包含了一个不同且随机的唯一的字符串，字符串长到足以使另一个人也用它的机会变得微乎其微。

（2）Alice 把每张汇票都和一张复写纸一起装入 100 个不同的信封，并把它们全交给银行。

（3）银行开启 99 个信封并确认每张票都是一个 1000 美元，而且所有随机唯一字符串都是不同的。

（4）银行在余下的一个未开启的信封上签名，签名通过复写纸印到汇票上。银行把这个未开启的信封交回 Alice，并从她的帐户上扣除 1000 美元。

（5）Alice 打开信封并在一个商人处花掉这张汇票。

（6）这个商人检查银行的签名以确信汇票是合法的。

（7）这个商人拿着这张汇票来到银行。

（8）银行验证它的签名，并检查它的数据库以确信有相同的唯一字符串的汇票先前没有被存过。如果没有，银行把 1000 美元划到这个商人的帐上。银行在一个数据库中记录这个随机字符串。

（9）如果它先前被存过，银行不接受这张汇票。

现在，如果 A lice 试图使用这张汇票的影印件，或者如果这个商人试图用这张汇票的影印件存款，银行都会知道。

协议#3

前一个协议保护了银行不受欺骗者的欺骗，但它没有识别出这些欺骗者。银行不知道是买这张汇票的人（银行不知道是 Alice）试图欺骗这个商人或者是这个商人试图欺骗银行。这个协议纠正如下：

（1）Alice 准备了 100 张每张 1000 美元的匿名汇票。在每一张汇票上包含了一个不同的唯一的随机字符串，字符串长到足以，使另一个人也用它的机会变得微乎其微。

（2）Alice 把每张汇票都和一张复写纸一起装入 100 个不同的信封，并把它们全交给银行。

（3）银行开启其中 99 个信封并确认每张都是一个 1000 美元的汇票，而且所有随机字符串都是不同的。

（4）银行在余下的一个未开启的信封上签名，签名通过复写纸印到汇票上。银行把这个未开启的信封交回 Alice，并从她的帐户上扣除 1000 美元。

（5）Alice 打开信封并在一个商人处花掉了这张汇票。

（6）这个商人检查银行的签名以确信汇票是合法的。

（7）商人要求 Alice 在汇票上写一个随机识别字符串。

（8）Alice 同意。

（9）这个商人拿着这张汇票来到银行。

（10）银行验证签名并检查它的数据库以确信具有相同唯一字符串的汇票先前没有被存过。如果没有，银行把 1000 美元划归商人的帐上。银行在一个数据库中记下这个唯一字符串和识别字符串。

（11）如果这个唯一字符串在数据库中，银行拒收这张汇票。接着，它将汇票上的识别字符串同存在数据库中的识别字符串比较。如果相同，银行知道这个商人影印了这张汇票。如果不同，银行知道买这张汇票的人影印了它。

这个协议假设一旦 Alice 在汇票上写上这个识别字符串，那个商人就不能变它。汇票可能有一系列小方格，这个商人会要求 Alice 用 X 或 O 填充这些小方格。汇票可能是用如果要抹去就撕掉的纸做成。

由于商人和银行之间的交互作用发生在 Alice 花钱之后，故这个商人可能和一张空头汇票牵连在一起。这个协议的具体实现可以要求 Alice 在商人——银行交互期间在柜台前等着，很像是今天的信用卡交易操作的方式。

Alice 可能会试图陷害这个商人。她可以第二次花一张汇票的拷贝，在第（7）步中给一个同样的识别字符串。除非这个商人保持一个已收到的汇票的数据库，否则他将遭到欺骗。下一个协议消除这个问题。

协议#4

如果证明是买汇票的人试图欺骗这个商人，银行会希望知道那个人是谁。为了做到这一点，要求我们从实际模拟中出来进入密码界。

秘密分割技术可以用来在数字汇票中隐藏 Alice 的名字。

（1）Alice 对给定数量的美元准备 n 张匿名汇票。

每张汇票都包含了一个不同的随机唯一字符串 X，X 有足够长，足以使得有两个字符串相同的机会微乎其微。

在每一张汇票上也有 n 对鉴别比特字符串 I_1，I_2，…，I_n。（在每张支配上那都是 n 个不同的对。）这些对中的每一个都是按如下产生的：Alice 创造一个给出她的名字、地址以及任何其他银行希望见到的鉴别信息的字符串。接着，她用秘密分割协议（见 3.6 节）将它分成两部分。然后，她使用一种比特提交协议提交每一部分。

例如，I_{37} 由两部分组成：I_{37_L} 和 I_{37_R}。每一部分是一个可以要求 Alice 打开的比特提交分组，其正确打开与否也可以立即验证。任何对如：I_{37_L} 和 I_{37_R}，但不是 I_{37_L} 和 I_{38_R} 都会揭

示 Alice 的身份。

每张汇票看起来像这个样子：

总数

唯一字符串：X

鉴别字符串：I_1=（I_{1L}，I_{1R}）

I_2=（I_{2L}，I_{2R}）

…

I_n=（I_{nL}，I_{nR}）

（2）Alice 用盲签名协议隐蔽所有 n 张汇票。她把它们全部给银行。

（3）银行要求 Alice 恢复出随机的 n-1 张汇票并确认它们都是合格的。银行检查总数、唯一字符串并要求 Alice 出示所有鉴别字符串。

（4）如果银行对 Alice 没有任何进行欺骗的企图感到满意，它就在余下的一张隐蔽汇票上签名。银行把这张隐蔽汇票交回 Alice，并从她的帐户上扣除这笔钱。

（5）Alice 恢复这张汇票，并在一个商人那里花掉它。

（6）商人验证银行的签名以确信这张汇票是合法的。

（7）商人要求 Alice 随机按汇票上每个鉴别字符串的左半或右半。实际上，商人给 Alice 一个随机的 n 比特选择字符串，b_1，b_2，…， b_n。Alice 根据 b_I 是 0 还是 1 公开 I_I 的左半或右半。

（8）Alice 同意。

（9）商人拿着这张汇票来到银行。

（10）银行验证这个签名并检查它的数据以确信有相同唯一字符串的汇票先前没有被存过。如果没有，银行把这笔钱划到商人的帐上。银行在它的数据库中记下这个唯一字符串和所有识别信息。

（11）如果这个唯一字符串在数据库中，银行就拒收汇票。接着，它把汇票上的识别字符串同它数据库中存的相比较。如果相同，银行知道是商人复制了汇票。如果不同，银行知道是买汇票的人影印了它。由于接收这张汇票的第二个商人交给 Alice 一个和第一个商人不同的选择字符串，银行找出一个比特位，在这个比特位上，一个商人让 Alice 公开了左半，而另一个商人让 Alice 公开了右半。银行异或这两半以揭露 Alice 的身份。

这是一个相当迷人的协议，故让我们从不同角度来看看它。

Alice 能进行欺骗吗？她的数字汇票不过是一个比特字符串所以可以拷贝它。第一次花它不会有问题，她只需完成协议，则一切进展顺利。商人在第（7）步中给她一个随机的 n 比特选择字符串，并且 Alice 在第（8）步中将公开每个 I_I 的左半或右半。在第（10）步中，银行将记录所有这些数据，连同汇票的唯一字符串。

当她试图第二次使用同一张数字汇票时，商人（同一个商人或另一商人）将在第（7）步中给她一个不同的随机选择字符串。Alice 必须在第（8）步中同意；不这样做势必立即提醒商人有些事值得怀疑。现在，当这个商人在第（10）步中将汇票带到银行时，银行会立即发现带相同唯一字符串的汇票已经存过。银行接着比较鉴别字符串中所有公开的部分。两个随机选择字符串相同的机会是 2^n 分之一，在下一个冰期前是不可能发生的。现在，银行找

出这样一对，其中一半第一次被公开，另一半第二次被公开。它把这两半一起异或，马上得到 Alice 的名字，于是银行知道谁试图两次花这一张汇票。

应当指出，这个协议不能让 Alice 不进行欺骗，但它能几乎肯定地检测她的欺骗。如果 Alice 进行欺骗，她不可能不暴露身份。她不可能改变唯一字符串或任何的识别字符串，因为此时银行的签名不再有效。这个商人将在第（6）步中马上意识到这点。

Alice 可能试图偷一张空头汇票骗过银行，这张汇票上的识别字符串不会泄露她的名字，或最好是一张其识别字符串泄露其他人名字的汇票。她在第（3）步中进行这种欺诈骗过银行的机会是 n 分之一。这些并非是不可能的机会，但如果你作出的惩罚足够严厉的话，Alice 不敢以身试法。或者，你可以增加 Alice 在第（1）步中制作的多余汇票的数目。

这个商人能进行欺骗吗？他的机会甚至更小。他不能将这张汇票存两次；银行将会发现选择字符串被重复使用。他不能以捏造陷害 Alice，只有 Alice 才能打开任意的识别字符串。

甚至 Alice 和商人合谋也不能欺骗银行。一旦银行在带唯一字符串的汇票上签名，银行就确信只能使用这张汇票一次。

银行又怎样呢？它能不能知道它从商人那儿收到的汇票是它为 Alice 签的那张呢？在第（2）至第（5）中的盲签名协议保护了 Alice。银行无法作出判断，即使它保留了每次交易的完整记录。说得更重些，银行和商人在一起也无法知道 Alice 是谁。Alice 可以走进商店并且完全匿名地购买东西。

Eve 可以进行欺骗。如果她能窃听 Alice 和商人之间的通讯，并能在商人到达银行之前先到达银行，她就能第一个把这笔数字现金存入她的帐户。银行将会接受，甚至更糟的是，当商人试图去存入数字现金时他会被认为是一个欺骗者。如果 Eve 偷到数字现金并在 Alice 之前花掉它，那么 Alice 会被认为是一个欺骗者。没有办法防止这种情况；它是现金匿名的直接后果。当他们要使用纸币时，Alice 和商人都必须保护好他们的每一比特信息。

这个协议是介于被仲裁协议和一个自我执行协议之间的协议。Alice 和商人都相信银行能兑现汇票，但 Alice 不必信任知道她购物的银行。

数字现金和极为高明的犯罪

数字现金也有它不利的一面。有时人们并不需要那么多的隐私。看看 Alice 进行的高明犯罪[1575]：

（1）Alice 绑架了一个婴儿。

（2）Alice 准备了 10，000 张每张 1000 美元的匿名汇票（或多到她想要的那么多）。

（3）Alice 用盲签名协议隐蔽所有 10，000 张汇票，她把它们送给当局并威胁除非按下列指示去做，否则要杀死婴儿：

（a）让银行签所有 10，000 张汇票。

（b）在报纸上公布结果。

（4）当局同意。

（5）Alice 买了一张报纸，恢复那些汇票，并开始花它们。当局没有办法靠追踪这些汇票来抓到她。

（6）Alice 放了这个婴儿。

注意这种情况比任何涉及实际特征的情况都更糟糕—例如现金。因为没有物理接触，警

察很难有机会抓住绑架者。

通常话虽如此，但数字现金对犯罪分子来说也算不上理想。问题是匿名只有一种方式奏效：消费者是匿名的而商人不是。而且，商人不能隐藏他收到钱的事实。数字现金使政府容易知道你挣了多少钱，但不可能知道你把钱花在什么上。

实用化的数字现金

一家荷兰公司 DigiCash 拥有数字现金的大部分专利并已经在实际产品中实现数字现金协议。任何感兴趣的人可按以下地址联系：

DigiCash BV

Kruislaan 419

1098 VA Amsterdam

NETHERLANDS

其他数字现金协议

有一些其它的数字现金协议；见文献[707，1554，734，1633，973]。它们中的一些涉及了相当复杂的数学问题。通常，各种数字现金协议可以分为不同的种类。在线式系统需要商人在每次销售时和银行联系，很象今天的信用卡协议。如果有问题，银行不会接受数字现金，Alice 不能行骗。

脱线式系统，如协议#4，直到商人与顾客交易之后它都不需要商人和银行之间的通信。这类系统可以发现 Alice 行骗但不能防止 Alice 行骗。协议#4 中可以通过验证 Alice 的身份知道她是否试图欺骗来发现她的欺骗。Alice 知道这种情况会出现，因此她不会欺骗。

另一种方法是制造一个特定的智能卡（见 24.13 节），它包含一个叫做“观察者”[332，341，387]的防窜改芯片。“观察者”芯片保存了一个所有关于智能卡上花掉的数字现金的信息的袖珍数据库。如果 Alice 企图复制数字现金并再次花掉它，这个被嵌入在内的“观察者”芯片就会发现并禁止交易。因为“观察者”芯片是防窜改的，Alice 不能抹掉袖珍数据库，除非永久性的损坏智能卡。数字现金以它自己的方式在经济领域中流通；当它最终被存入银行时，银行可以检查数字现金并发现是否有人进行欺骗以及谁在进行欺骗。

数字现金协议也可以被分在另一类。**电子货币**有固定的价值；使用这个系统的人们需要若干不同面额的硬币。电子支票可以用在任何数量直到最大值上，然后作为退款返回没有用完的部分。

两个优秀而且完整的不同的脱线式电子货币协议见文献[225，226，227]和[563，564，565]。有一个系统叫做 Netcash，有弱匿名性质，也在文献[1048，1049]中提出。另一新系统见文献[289]。

在文献[1211]中，Tatsuaki Okamoto 和 Kazuo Ohta 列出了一个理想数字现金系统的六个性质：

（1）独立性。数字现金的安全性不依赖于任何物理位置。现金能通过计算机网络传送。

（2）安全性。数字现金不能被拷贝和重用。

（3）隐私性（不可追踪性）。用户的隐私受到保护，没有人能追踪发现用户和他们的所购物之间的关系。

（4）脱线付款。当一个用户用电子现金为所购物付款时，用户和商人之间的协议是脱线执行的。那就是说，商店不必与一台主机相连以处理用户的付款。

（5）可转移性。数字现金可被转移给其他用户。

（6）可分性。给定数量的数字现金能被分成较小数额的几份数字现金。（当然，每份最

后加起来总数还是那么多。）

上面讨论的协议满足性质（1）、（2）、（3）和（4），但不满足性质（5）和（6）。这里有一些满足除性质（4）以外的所有性质的在线式数字现金系统[318，413，1243]。在文献[339]中提出了第一个满足性质（1）、（2）、（3）和（4）的脱线式数字现金系统，它与这里讨论的那个类似。Okamoto 和 Ohta 提出了一个满足性质（1）至性质（5）的系统[1209]；他们也提出了一个满足性质（1）至性质（6）的系统，但购买一件物品要求的数据大约为 200 兆字节。其它的脱线式可分货币系统在文献[522]中讨论。

由同一个作者在文献[1211]中提出了一个满足性质（1）至（6）的数字现金系统，这个系统无庞大的数据要求。一次付款的总数据传输量大约是 2 万字节，并且协议能在几秒钟内完成。作者认为这是第一个理想的不可追踪的电子现金系统。

匿名信用卡

文献[988]中的这个协议用在若干不同的银行以保护顾客的身份。每一位顾客在两个不同银行拥有一个帐户，第一个银行知道此人的身份并愿意发给他信用卡，第二个银行仅仅知道他的假名（类似于瑞士银行的帐号）。

顾客可以通过证明帐户是他的从第二个银行取出资金。但是，银行不知道这个人也不愿发给他信用卡。第一个银行知道这个顾客并转帐给第二个银行——用不着知道假名。顾客然后就可以匿名地使用这笔资金。在月末，第二个银行给第一个银行一分帐单，这是银行真正应该支付的。第一个银行把顾客应该支付的帐单送给顾客。当顾客付帐后，第一个银行把附加资金转帐到第二个银行。所有的交易都是通过一个中间媒介处理的，就象电子联邦储备所作的那样：在银行之间结算帐目，登记消息，和产生审计追踪。

顾客、商人和各个银行的交易在文献[988]中描述。除非每个人都串通起来陷害顾客，顾客的匿名是可以保证的。然而，这不是数字现金；银行很容易进行欺骗。这个协议使顾客在不泄露隐私的情况下保护其信用卡的利益。

第二篇　密码技术

第七章　密钥长度

7.1 对称密钥长度

对称密码体制的安全性是算法强度和密钥长度的函数：前者更加重要而后者则更容易描述。

假设算法具有足够的强度，实际上这点极难做到，不过本例却很容易。这里我所指的足够是：除了用穷举攻击的方式试探所有的密钥外没有更好的方法破译该密码系统。

为了发动对密码系统的攻击，密码分析者需要少量的密文和对应的明文，穷举攻击是一种已知明文的攻击。对分组密码来说，密码分析者需要密文分组和对应的明文分组：通常是 64 比特。获得明文和密文比你想象的要容易，密码分析者可通过一些手段获取明文消息的副本而后去截取相应的密文。他们可能知道有关密文格式的一些信息：如，它是一个 WordPerfect 文件，它有一个标准的电子邮件消息头，它是一个 UNIX 目录文件，一幅 TIFF 图像，或用户数据库中的一个标准记录。而所有这些格式都有一些予定义字节。密码分析者不需要太多的明文来发起这种攻击。

很容易计算一次穷举攻击的复杂程度：如果密钥长度为 8 比特，那么有 2^8=256 种可能的密钥，因而找出正确的密钥将需要 256 次尝试，在 128 次尝试后找到正确密钥的概率是百分之五十。假如密钥长度为 56 比特，会有 2^{56} 种可能密钥。设想有一台每秒能检验一百万个密钥的超级计算机，也需要 2285 年时间才能找出正确的密钥。如果密钥长度为 64 比特，则将需要 585,000 年才能在 2^{64} 种可能的密钥中找出正确的密钥；如果密钥长 128 比特，则需要 10^{25} 年的时间。宇宙也只有 10^{10} 年的历史，相对而言 10^{25} 年太长了。对于一个长为 2048 比特的密钥，用每秒尝试百万个密钥的百万个计算机并行工作要 10^{597} 年才能完成。到那时宇宙或许早已爆炸或膨胀得无影无踪了。

在你急着去发明一个 8K 字节密钥的密码体制之前，请记住其强度问题的另一面：加密算法必须非常安全，以致于除穷举攻击外没有其它更好的方法来破译它。这并不像看上去那样容易。密码学是一门奇妙的艺术，看上去完美的密码系统往往是非常脆弱的。很强的密码系统，哪怕是一点点的改变就会使它变得非常脆弱。对业余密码设计人员的忠告是要对任意新的算法进行健康的、执着的怀疑，最好去信任那些专业密码人员分析了多年而未能攻破的算法，以及去怀疑那些设计者宣称其安全性是如何好的算法。

回忆一下 1.1 节里的一个重要点：密码体制的安全性应依赖于密钥，而不是依赖于算法的细节。假设密码分析者已经获得了你的算法的所有细节；假设他们能够得到发起唯密文攻击的足够多的密文；假设他们能得到所需要的尽可能多的数据发起明文攻击；甚至假设他们能进行选择明文攻击。在这些情况下，如果你的密码体制仍然是安全的，那么它就达到所需要的安全性要求了。

忠告归忠告，在密码学领域业余密码人员还是有许多事是可以做的。其实，这里讨论的安全性在许多情况下并非必须，大多数敌手并不具备这方面的知识，也不具有一个强大国家所

拥有的计算资源，甚至他们连破译密码的兴趣都没有。要是你正密谋推翻一个强大的政府，就得依靠本书后面讲的那些可靠而正确的算法。剩下的，就是玩得开心了。

穷举攻击所需时间和代价估计

要记住，穷举攻击是一种典型的已知明文攻击,它需要少量的密文及相应的明文。如果你假设穷举攻击是对算法最有效的攻击的话——一个大的假设——密钥必须足够长以致使攻击不可行，那么密钥要多长呢？

有两个参数决定穷举攻击的速度：需测试的密钥量及每个测试的速度。大多数对称密码算法接受一个固定比特长度模式作为密钥，DES 有 56 比特的密钥；共 2^{56} 个可能密钥。本书讨论的算法一些有 64 比特的密钥，也就是说有 2^{64} 个可能密钥，另外一些有 128 比特密钥。

每一个可能密钥的测试速度也是一个因素，但重要性次之。为了分析的目的，我们将假定每种不同算法的测试时间是相同的。实际上测试一种算法的速度可能比另一种算法的速度快两、三倍，甚至十倍。但由于我们正在寻找的密钥长度比破译密码的难度大百万倍之多，这个密钥长度是可行的，所以测试速度小小的差异是可以忽略的。

在保密通信中关于穷举攻击效率的多数讨论都集中在 DES 算法上。1977 年 Whitfield Diffie 和 Martin Hellman[497]假设了一种专用破译 DES 的机器。这台机器由一百万个芯片组成，每秒能测试一百万个密钥。这样它可在 20 个小时内测试 2^{56} 个密钥，如果制造出来用于破译 64 比特密钥的算法，它可在 214 天内尝试所有的密钥。

穷举攻击必须有并行处理器的存在。每个处理器测试密钥空间中的一个子集，它们之间不需要什么通信，要说通信那就是报道成功的消息，并且它们没有共享的内存。设计这样一台具有一百万个并行处理器的机器，并让它们彼此独立地工作是很容易的。

最近，Micheal Wiener 决定设计一台穷举攻击机器[1597，1598]。（这台机器是为攻击 DES 设计的，但对任何算法都适用。）他设计了专门的芯片、主板、支架，并且估算了其价格。他发现只要给定一百万美元就能制造出一台这样的机器使其在平均 3.5 小时（最多不超过 7 小时）里能破译密钥长为 56 比特的 DES 算法。他还发现机器的价格和破译速度之比是呈线性的，表 7.1 列出了各种密钥长度的对应数据。记住 Moore 定律：大约每经 18 个月计算机的计算能力就翻一番。这意味着每 5 年价格就会下降到原来的百分之十，所以在 1995 年所需要的一百万美元到了 2000 年就只用花十万美元。流水线计算机能够做得更好[724]。

表 7.1　1995 年硬件穷举攻击的平均时间估计

Cost	**Length Of Key In Bits**					
	40	**56**	**64**	**80**	**112**	**128**
$100K	2 seconds	35 hours	1 year	70,000years	10^{14}years	10^{19}years
$1M	.2 seconds	3.5 hours	37years	7000 years	10^{13}years	10^{18}years
$10M	.02 seconds	21 minutes	4 days	700 years	10^{12}years	10^{17} years
$100M	2 milliseconds	2 minutes	9 hours	70 years	10^{11} years	10^{16} years
$1G	.2 milliseconds	13 seconds	1 hour	7 years	10^{10} years	10^{15} years
$10G	.02 milliseconds	1 seconds	5.4minutes	245 days	10^{9} years	10^{14} years

$100G	2 microseconds	.1 seconds	32seconds	24 days	10^8 years	10^{13} years
$1T	.2 microseconds	.01 seconds	3seconds	2.4 days	10^7 years	10^{12} years
$10T	.o2microseconfs	1milliseconds	.3seconds	6 hours	10^6 years	10^{11} years

对一个 56 比特密钥，穷举攻击所需金额对很多大公司和一些犯罪组织来说还是可以承受的。对于 64 比特密钥，则只有一些发达国家的军事预算才能承受。而破译 80 比特密钥现在仍然不行，但是如果按目前的形势继续发展下去的话，这种情况将会在 30 年内发生改变。

当然，估计未来 35 年计算机的计算能力是很可笑的，一些科幻小说里出现的技术突破可能会觉得上述预测很可笑。相反，目前一些未知的物理限制又使人们产生不切实际的乐观。在密码学中悲观一点是很明智的。用 80-比特密钥的加密算法是一种目光非常短浅的行为，还是坚持用至少 112-比特的密钥吧。

如果攻击者想要不择手段地破译一个密钥，他们必须要做的全部事情就是花费钱。所以，试图估计一下密钥的最小“价值”似乎是明智的：在试图破译一个有经济价值的密钥之前，要确信它到底有多大价值呢？举个极端的例子，如果一个加密了的消息值 1.39 美元，一台价值 1000 万美元的破译机来寻找密钥在经济上就毫无意义了。另一方面，如果明文消息值 1 亿美元，那么造一台破译机破译此信息是值得的，何况有些信息的价值会随时间迅速减少。

软件破译机

没有功能特殊的硬件设备和大规模并行计算机，穷举攻击很难有效地工作。对密码系统的软件攻击比硬件攻击大约慢一千倍。

基于软件穷举攻击的真正威胁：并非是它的确定性，而是它的“自由”性。装配一台微型计算机来测试可能的密钥，无论其是否在测试都不涉及到成本问题，如果找到正确的密钥——那太好了，如果没找到，也不会失去什么。建立整个这样的微机网络同样不涉及成本问题。最近一次对 DES 的试验，在一天内用了 40 个工作站的空闲时间对 2^{34} 个密钥进行了测试[603]。照这样的速度，将需要四百万天来测试所有密钥，但是如果足够多的人试图像这样破译的话，总有某个人会在某处交上好运。文献[603]中这样写到：

软件威胁的关键是十足的坏运气。设想一个具有 512 个工作站的大学计算机网络全部联网，对某些校园来说这是一个中等规模的网络，它们甚至可以遍及世界，通过电子邮件来协调各自行动。假设每个工作站能以每秒加密 15,000 次的速率运行…考虑测试和更换密钥的时间，便得到…每台每秒测试 8,192 次。用这样的速度测试完[一个 56]比特密钥空间将需要 545 年（假定网络每天工作 24 小时）。但是请注意，假设学生破译者进行同样的计算，他们在一天内破译出密钥的机会是 1/200,000，在一周内破译出的机会增加到 1/66,000，他们的设备越先进，或者使用的机器越多，成功的机会自然就越大。这种概率对靠赛马谋生来说机会太小了，但在某些情况下它还算好的，如，同政府的博彩中奖相比它要好得多，“一百万分之一？”“一千年能发生一次吗？”，要简单地说清这些事已不再可能了。这种不断发展的冒险能接受吗？

用一个 64 比特密钥来替换 56 比特密钥的密码算法来说，其破译难度要大 256 倍。而对于 40 比特的密钥，情况就远比这简单得多。一个由 400 台，每台每秒能加密 32,000 次的计

算机组成的网络，能够在短短一天内完成对 40 比特密钥的破译（1992 年， 40 比特密钥的 RC2 和 RC4 算法已经被成功破译了——见 13.8 节）。

对 128 比特密钥来说，进行穷举攻击是十分荒唐的。工业专家估计：到 1996 年止，世界上将有二亿台计算机在运行，这种估计包括从巨大的 Cray 大型机到笔记本电脑在内的计算机。如果所有的这些计算机同时一起进行穷举攻击，并且每台以每秒一百万次的加密速度进行，破译这个 128 比特的密钥也得需要一百万倍宇宙年龄的时间才能成功。

神经网络

神经网络对密码分析来说并不是非常有用，这主要是其解空间的模式导致的，因为神经网络最擅于处理那些连续解的问题，这些解一些比另一些更好。这就允许神经网络去学习并在学习中得出越来越好的解。而破译密码算法并没有提供太多学习“机会”的方法：你要么找到密钥要么没有。（至少对稍微好的算法是这样。）神经网络适用于那些结构化环境，在那里一些东西必须要学习，而在信息熵很高，随机性非常强的密码学领域就不适用了。

病毒

让数以百万台计算机放在一起进行穷举攻击的所遇到的最大难题是说服这些计算机的拥有者同意他们的机器参与，你会有礼貌地请求，但那是浪费时间，因为他们会说“不”；你也可以试图侵占他们的机器，但那更是浪费时间，并且你还可能被捕；你还可以利用计算机病毒来散布攻击程序，以更加有效地覆盖到尽可能多的计算机上。

这是在文献[1593]中首次提到的特别有趣的思想。攻击者写下并散布计算机病毒，这些病毒并不重新格式化硬盘驱动器和删除文件，它只是在受染计算机的空闲时间处理穷举攻击的密码分析问题。各种各样的研究已表明微型计算机 70%至 90%的时间是空闲的，于是病毒可以无任何阻碍地找到时间来完成它的任务。如果它是良性的，在它发作的时候甚至可能逃过人们的注意。

最后，某个机器也会偶然碰到正确的密钥，此时会有两条路可以选择：其一，攻击病毒将引发另一个不同的病毒。它除了复制有关正确密钥的信息和删除攻击病毒留下的其它的信息外不做任何事情。然后，新的病毒通过计算机网络传播，直到它回到书写原始病毒的人的计算机上为止。

其二，卑鄙的方法将是病毒在屏幕上显示以下信息：

> 本机中有严重的病毒存在！
>
> 请拨 1-800-123-4567 并且读入下面的 64 比特数字给操作人员：
>
> ×××× ×××× ×××× ××××
>
> 对于第一个报告该病毒的人给予 100 美元的奖金。

这种攻击的有效性有多大呢？假设典型的被感染了病毒的计算机可以每秒测试一千个密钥，远远低于计算机的最大潜力，因为我们不得不假设它会干其他事情，我们也假设典型的病毒感染了一千万台机器，这样这个病毒可以在 83 天里破译一个 56 比特密钥，在 58 年内破译一个 64 比特的密钥。你或许会收买抗病毒软件的制造者，但那是你们的问题，任何计算机速度或者病毒感染速度的增长必然会使这种攻击更加有效。

中国式抽彩法

中国式抽彩法是一种折衷的，但对于大规模并行密码分析机来说是一个可行的建议[1278]。设想一种用于穷举攻击的、每秒有百万次测试速度的破译芯片被安装在每台收音机与电视机中售出，每块芯片在通过广播收到一对明文/密文后便自动利用其中的程序检测不同的密钥。于是，每当想要破译密钥时，就广播这个数据，这时所有的收音机和电视机便开始嘎嚓嘎嚓地开动起来。最后，正确的密钥将会在某个地方被某人找到，这个人也因此得到抽彩所中的奖金，这样便确保了结果会有迅速而正确的报告，且也有助于带有破译芯片的收音机和电视机的销售。

如果中国每个成年男人、妇女和小孩都拥有一台这样的收音机或电视机，那么 56 比特算法的正确密钥在 61 秒内可找到。如果仅仅十分之一的中国人拥有收音机或电视机——这更接近于现实——正确的密钥将在 10 分钟内找到。64 比特算法的密钥将在 4.3 小时内找到。——如果百分之一的人拥有收音机或电视机的话要 43 小时。

为了使得攻击能付诸实际，要求进行一些修改。首先，使每块芯片测试随机的密钥，而不是唯一的一组密钥将会更加容易，这样将使攻击减慢大约 39%左右——从目前我们正处理的数字来看这并不算多。如果每一个人都在测试中，最后，有人就在其“发现密钥”的灯灭了时要打电话通报这个结果，然后读出出现在他们屏幕上的一串数字。

表 7.2 显示了中国式抽彩对不同国家和不同长度密钥的有效性。中国显然是最有利于实施这种攻击的地方，但他们不得不使每一个人都拥有电视机或收音机。美国人数较少，但有多得多的设备。怀俄明州能够在不到一天之内破译一个 56 比特的密钥。

表 7.2　使用中国式抽彩的穷举攻击估计

国家	人口	电视和收音机数	破译时间 56-bit	破译时间 64-bit
China	1,190,431,000	257,000,000	280secs.	20hrs
U.S.	260,714,000	739,000,000	97secs.	6.9hrs
Iraq	19,890,000	4,730,000	4.2hrs	44days
Israel	5,051,000	3,640,000	5.5hrs	58days
Wyoming	470,588	1,330,000	15hrs	160days
Winnemucca,NV	6,100	17,300	48days	34yrs

(所有数据摘自《1995 World Almanac and Book of Facts》)

生物工程技术

如果生物芯片是可能的，那么不用它作为分布式的穷举攻击密码分析工具将是愚蠢的，考虑一种假想的动物；很不幸地称它为“DESosaur”[1278]，它由能够检验可能密钥的生物细胞组成，这些生物细胞可以通过光学通道接收明文/密文对（细胞是透明的你可以看见）。而密钥解则通过生物体内的特殊细胞经由循环系统送到 DESosaur 的发音器官。

典型的恐龙古生物具有 10^{14} 个细胞（包括细菌），如果它们每秒能完成一百万次加密（应该承认，这是一个大的假设），破译一个 56 比特的密钥需要一万分之七秒，破译一个 64 比特的密钥少于十分之二秒的时间。但破译一个 128 比特的密钥仍需 10^{11} 年。

另一种生物方法是利用遗传密码分析海藻，它能够完成对密码算法的穷举攻击[1278]。这些生物体使制造多处理器的分布式机器成为可能，因为他们能够覆盖很大的区域。每个明/密文对可通过卫星广播，如果一个生物体找到了结果，它附近的细胞将改变颜色并把结果传回卫星。

假设典型的海藻细胞大小为 10 立方微米（这或许是一个大的估计），那么 10^{15} 个可占据一立方米，把它们放入海洋，覆盖深一米的 200 平方英里（518 平方公里）的海水（你可以考虑怎样做到这点——我只是出主意的人），这样将有 10^{23}（超过一千亿加仑）个海藻飘浮在海洋中（超过了一千亿加仑的石油）。如果它们中的每一个每秒能够尝试一百万个密钥，那么它们得花超过 100 年的时间才能破译 128 比特的密钥（让海藻的处于繁荣期是一个问题），不管海藻处理速度、海藻直径、或者甚至能够在海洋中的扩散区域的大小，其中任何一个取得突破，都将有效地减少这些数字。

请不要问我有关纳米工艺的问题。

热力学的局限性

热力学第二定律的结论是：信息的表达需要一定的能量。通过改变系统的状态，记录单独的 1 比特所需要的能量不少于 kT，其中 T 表示系统的绝对温度，k 是 Boltzman 常数。（依靠我，物理课程几乎就结束了。）

假定 $k=1.38*10^{-16}$erg/° Kelvin，宇宙的环境温度为 3.2K，那么在此温度下运行的计算机每设置或清除 1 比特将消耗 $4.4*10^{-16}$erg 的能量。在比宇宙辐射温度低的环境中运行一台计算机会需要附加的能量来运行热泵。

目前，太阳每年辐射出的能量约为 $1.21*10^{41}$erg。这一能量足以使理想的计算机上的 $2.7*10^{56}$ 个单独比特发生改变，也足使一个 187 比特计算器中的所有状态值发生改变。如果绕太阳建造一个 Dyson 球，并且让它一点也不少地吸收 32 年的能量，那么我们就能使一台计算机的能量增加到 2^{192}。当然，它不会让能量剩余，以便计算器完成任何有用的计算。

但是，这仅仅是一颗星体，所有星体中很渺小的一颗。一颗典型的超新星释放的能量可达 10^{51}erg。（约是能量以微中子形式释放的一百倍，还得现在就开始。）如果所有的这些能量被用以运算，那么一个 219 比特的计算器会循环其所有的状态值。

批注 [admin1]:

这些数字与设备的技术性能无关，它只是热力学允许的最大值。所以，我们可以断言：对 256-比特的密钥进行穷举攻击是绝对行不通的，除非在超空间里用超物质制造计算机。

7.2 公钥密钥长度

2.3 节中已经讨论了单向函数。例如，两个大素数进行相乘就是一个单向函数，得到相乘的结果很容易，但是由这个结果分解得到两个素数却非常困难（见 11.3 节）。公钥密码体

制就是利用这种思想做成单向陷门函数。实际上，那不过是一个谎言；因子分解被推测是一个难题（见 11.4 节）。众所周知，它似乎是这样。如果它的确是，也没有人能够证明难题就真的很难。大多数人都假定因子分解是困难的，但它从来没有被数学证明过。

还有必要再啰嗦一点。不难想象 50 年后的情形，人门围坐在一起，兴致勃勃地谈论着过去的美好时光：那时候，人们习惯于认为因子分解是难的；密码体制正基于这种难度；而公司实际上依靠这种素材大赚其钱。也可以很容易地设想到，未来在数论方面的发展使因子分解更加容易或者复杂度定理的发展使因子分解变得毫无意义。没有理由相信这会发生——并且大多数懂得很多，而有主见的人也会告诉你这是不可能的——但是也没有什么理由相信它不会发生。

无论怎么样，今天主要的公钥加密算法都是基于分解一个大数的难度，这个大数一般是两个大素数的乘积。（其它一些算法基于称为离散对数问题的东西，这里我们作相同的讨论。）这些算法也会受到穷举攻击的威胁，只不过方式不同。破译它们的出发点并不是穷举所有的密钥进行测试而是试图分解那个大数（或者在一个非常大的有限域内取离散对数——一个类似的问题）。如果所取的数太小，那么就无安全可言；如果所取的数足够大，那会非常安全：集中世界上所有的计算机力量从现在开始工作直到太阳变成一颗新星为止都不能奈何它——当然是基于目前对数学的理解。11.3 节更注重数学细节，讨论了大数因子分解的有关问题，这里我们仅讨论分解不同长度的数需要花费多长时间。

对大数进行因子分解是困难的。不幸地是，对算法设计者来说它正变得越来越容易。更加糟糕的是，它比数学家们所希望的还快。1976 年，Richard Guy 写道：“本世纪内如果有人不采用特殊的方式成功地对 10^{80} 大小的数进行因子分解的话那我将非常地惊呀！”[680]。1977 年，Ron Rivest 说过分解一个 125 个十进制位的数据需要 $40*10^{15}$ 年[599]。可是，一个 129 位（十进制）的数据在 1994 年被成功分解。如果有什么教训的话，那就是作出预言将是很愚蠢的。

表 7.3 列出了过去十年有关因子分解的记录。其间，最快的分解算法是二次筛选法（参见 11.3 节）。

这些数据是触目惊心的。今天已经很不容易看到 512 比特的数据在操作系统中使用了，因为对这些数据进行因子分解，这势必危及到系统的安全，是非常可能的：Internet 上的蠕虫可在一个周末完成。

表 7.3 使用二次筛选算法进行因子分解

Year	**# of decimal digits factored**	**How many times harder to factor a 512-bit number**
1983	71	>20 million
1985	80	>2 million
1988	90	250,000
1989	100	30,000
1993	120	500
1994	129	100

计算机的计算能力是以 mips-years 来衡量的，即每秒一百万条指令的计算机运行一年，大约 $3*10^{13}$ 条指令。根据约定，一台 1-mips 的计算机就等同于一台 DEC VAX 11/780。因此，一 mips-year 就相当于一台 VAX 11/780 运行一年，或等同的机器。（一台 Pentium 100 计算机大约是 50mips，而一台 1800-node Intel Paragon 机器则是 50000。）

1983 年对一个有 71 个十进制位的数字进行因子分解需要 0.1mips-years；1994 年对一个 129 位的则需要 5000mips-years。计算能力这一令人激动的增长很大程度上归功于分布式计算的引入：利用网络上工作站的空闲时间。这一趋势是由 Bob Silverman 发起而通过 Arjen Lenstra 和 Mark Manasse 大力发展的。1983 年的分解在一台单独的 Cray X-MP 上使用了 9.5 个小时；1994 年的分解却使用了 5000 个 mips-years 并耗费了全球范围 1600 台计算机将近 8 个月的空闲时间。可见，现代的因子分解从这种分布式应用中大受其益。

情况变得更糟了。一种新的因子分解算法取代了二次筛选法：一般数域筛选法。1989 年数学家们会告诉你一般数域筛选法永远不会行得通；到了 1992 年他们又说一般数域筛选是可行的，但是它仅仅当被分解的数大于 130—150 位（十进制）时才比二次筛选法快。可是今天，众所周知，对小于 116 位的数的一般数域筛选法也快得多[472,635]。同时分解一个 512 位的数一般数域筛选法要比二次筛选法快 10 倍。在一台 1800-node Intel Paragon 机器运行该算法不需要一年的时间。表 7.4 给出了应用该算法对一系列不同大小的数进行因子分解所需的 mips-years 数[1190]。

然而，一般数域分解法的速度仍在加快。数学家们则努力紧跟着这些新技巧、新优化、新技术。现在还没有任何原因认为这种趋势不会继续。与其相关的一个算法：特殊数域筛选法，现在已经能够分解一些特殊形式的数——一般并不用于密码分析——其速度比一般数域筛选法分解同样大小的数还要快。假定一般数域筛选法优化后也能做得这样快是不无道理的，或许 NSA 已经知道怎样做。表 7.5 给出了用特殊数域筛选法进行因子分解的 mips-years 数[1190]。

1991 年在 EISS（欧洲系统安全研究所）的一个实验室里，参与者们一致认为一个 1024-比特的模数可以足够保密到 2002 年[150]。然而，他们又警告说：“尽管实验室的参与者在各自的领域中都有很高的资格，但是对这些声明（关于安全的持续性）要提高警惕。”这是个好建议。

在选择公钥密钥长度时，明智的密码人员应是极端的保守主义者。为了断定你所需要的密钥有多长你必须考虑你想要的安全性和密钥的生命周期，以及了解当前因子分解的发展水平。今天使用 1024-比特长的数仅获得 80 年代初期 512-比特的安全性。如果你希望你的密钥能够保持 20 年的安全，那么 1024-比特似乎太短了点。

表 7.4　利用一般数域筛选法因子分解

# of bits	**Mips-years required to factor**
512	30,000
768	$2*10^{8}$
1024	$3*10^{11}$
1280	$1*10^{14}$

1536	$3*10^{16}$
2048	$4*10^{20}$

表 7.5　利用特殊数域筛选法因子分解

# of bits	Mips-years required to factor
512	<200
768	100,000
1024	$3*10^{7}$
1280	$3*10^{9}$
1536	$2*10^{11}$
2048	$4*10^{14}$

即使你的特殊安全性并不值得花力气对模数因子分解，但仍然处于危险之中。设想一下用 RSA 加密的自动银行系统吧。Mallory 站在法庭上大声说："难道你没读报，1994 年 RSA-129 就已经被破译，任何组织只要花上几百万美元等上几个月就能将 512-比特的数分解吗？我的银行就是使用 512-比特的数进行保密的，顺便说一下，我并没有不断地撤消更换。"即使他在撒慌，法官也会责令银行证实的。

为什么不用 10000-比特的密钥呢？可以用，但你必须为密钥变长所需计算时间付出代价。通常是，你既想密钥足够长又想计算所需的时间足够短。

在本节的开始时我就说过作出预测是愚蠢的。但是现在我也预测一些。表 7.6 给出了公钥密钥多长才安全的一些忠告。其中，每年列出了三个密钥长度：分别针对个人，大的公司和大的政府。

下面是摘自文献[66]的一些假设：

我们相信能够获得十万台没有超人的，缺乏职业道德能力的机器。也就是说，我们绝对不会释放任何 Internet 蠕虫或病毒为我们寻求资源。很多组织都有几千台计算机在网上。充分利用这些设备的确需要很有技术性的外交能力，但不是不可能的。假定平均计算能力是 5mips，那么一年的时间就能从事一项需要 50 万 mips-years 的工程。

因子分解 129-digit 的数这样一个工程估计需要整个 Internet 计算能力的 0.03%[1190]，这不会令它感到困难。假定一个非常引人注意的工程需要花一年时间使用全球 2%的计算力量并非不可理解。

表 7.6　公钥密钥长度的推荐值(比特)

Year	Vs. Individual	Vs. Corporation	Vs. Government
1995	768	1280	1536
2000	1024	1280	1536
2005	1280	1536	2048
2010	1280	1536	2048
2015	1536	2048	2048

假设一个专注的密码分析家能得到 10,000 mips-years 的计算能力，一个大公司能得到

10^7 mips-years，并且一个政府能得到 10^9 mips-years。同时假定计算机的计算能力每 5 年增长 10 倍。最后假定数学领域因子分解的进步能够让我们以特殊数域筛选法的速度分解一个一般的数。（然而这是不可能的，但话又说回来，技术突破随时可能发生。）表 7.6 推荐了不同年份不同的安全密钥长度。

一定要记住需考虑密钥的价值。公开密钥经常用来加密那些时间长，价值大的东西：银行顾客用于数字化取款系统的密钥；政府用于检验其护照的密钥；以及公证人的公开数字签名密钥。或许并不值得花上几个月的计算机时间对一个私钥进行攻击，如果你能用一个破译了的密钥印制自己的钞票的话那它就非常吸引人了。一个 1024-比特的密钥足以作为那些用一个星期，或一个月，甚至几年才验证的东西的签名。但是你并不想拿着一份数字签名的文件从现在开始在法庭上站上 20 年，并且不断让对手演示如何用相同的签名伪造文件。

对更远的未来进行预测会更加愚蠢。谁能知道 2020 年计算机，网络，数学等方面发展成什麽样？然而，如果你看得更远点，你就会发现每个年代我们分解数的长度是上个十年的一倍。这引出了表 7.7。

另一方面，分解技术早在 2045 年之前或许就达到它的极限值。从现在起的 20 年里我们可以分解任何的数。尽管如此，我认为未必。

不是每个人都同意我的推荐的。NSA 指令 512-比特到 1024-比特的密钥作为数字签名标准（参见 20.1 节）——远远低于我所推荐的长期安全的数值。Pretty Good Privacy（参见 24.12 节）使用了最大长度的 RSA 密钥，2047 比特。Arjen Lenstra，世界上最成功的因子分解专家，在过去十年里拒绝作出预测[949]。表 7.8 给出了 Ron Rivest 对密钥长度的建议值，它们最早是在 1990 年作出的，但我却认为也太乐观了点[1323]。当他的分析在纸面上看来非常合理时，最新的历史却展示了令人惊奇的事正规则的发生着。这对你在选择密钥后面对未来的“惊奇”重新保持沉默是非常有意义的。

25000 美元的预算，使用二次筛选算法，每年的技术进步为 20%假定是较低的估计，25000,000 美元的预算，使用一般数域筛选法，每年 33%的技术进步为一般的估计，而较高的估计假定是预算为 25 亿美元，使用一般二次筛选法运行在特殊数域筛选法的速度下，每年有 45%的技术进步。

一直有这样一种可能性存在，那就是因子分解方面的进步同样令我吃惊，但归因于我得计算。但是为什么要相信我呢？我也只是通过预测证明了我的愚蠢。

表 7.7 对未来因子分解的预测

Year	**Key Length (in bits)**
1995	1024
2005	2048
2015	4096
2025	8192
2035	16384
2045	32768

表 7.8　Rivest 乐观的密钥长度推荐值

Year	Low	Average	High
1990	398	515	1289
1995	405	542	1399
2000	422	572	1512
2005	439	602	1628
2010	455	631	1754
2015	472	661	1884
2020	489	677	2017

DNA 计算法

现在的情况正变得不可思议。1994 年，Lenard M. Adleman 在一个生物化学实验室里竟然演示了 NP 完全问题的解决方法（NP-Complete Problem）（参见 11.2 节），他用 DNA 分子链描述对该问题解的推测[17]。Adleman 所解决的问题是引向哈密尔顿问题（Directed Hamiltonian Path Problem）的一个实例：汉密尔顿问题是指，给定一张有关城市的地图，其中这些城市是由单向马路连接，要求在地图上找到一条从城市 A 到城市 Z 而恰恰仅一次通过所有城市的路径。在 Adleman 的演示中，每一个城市由一系列不同的随机的 20 基点的 DNA 链表示，用传统的微生物技术，Adleman 综合处理了 50 皮摩尔（相当于 $3*10^{13}$ 个分子）的 DNA 链。每一条路也是由一个 20 基点的 DNA 链表示，但这些 DNA 链并不是随机选取的，Adleman 聪明地选择它们以使表示每条路（如 Road PK）的 DNA 链的开始端连到该路的起始城市（如 P）DNA 链，而其尾部则连到终止城市（如 K）。

Adleman 使用了 50 皮摩尔的 DNA 链表示每一条路，再用一条表示所有城市的 DNA 链将它们混合在一起，然后加入一种捆绑酵素使所有的 DNA 分子的尾部相连。酵素利用 DNA 路和 DNA 城市之间非常技巧的关系把各个表示路的 DNA 链连成了一个正当的模式。也就是，路 PK 的“出”端连到了始于城市 K 的所有路的“入”端，而不是其他路的“出”端或始于别的城市的那些路的“入”端。这些混合物经过一段时间的反应（时间是精心计算的），酵素会生成大量的 DNA 链，这些 DNA 链就表示地图上合法的但是随机的多路路径。

在所有的这些路径中，Adliman 可以观察到非常细微的痕迹——甚至单个分子——表示该问题解的 DNA 链。利用微生物学的一般技术，他丢弃那些表示路径过长或过短的 DNA 链 。（在所希望得到的路径中，路的数目应该等于城市的个数减一。）接着，他依次丢弃那些不经过城市 A，B，….Z 的 DNA 链。如果最后某个 DNA 链“幸存”，那就检测它以找到它所表示的路的序列：这就是有向汉密尔顿路径问题。

根据定义，任何 NP 完全问题（NP-Complete）的一个实例都可以在多项式时间里变换成其他 NP 完全问题的一个实例，当然也可以转换成有向汉密尔顿问题的一个实例。从 20 世纪 70 年代开始，密码学家才试着将 NP 完全问题用于公钥密码体制中。

由于 Adleman 解决的这个实例非常的“朴素”（地图上仅有七个城市，用眼睛观察也不过几分钟就能解决。），这项技术还处于初期，并且在发展上没有太大的障碍，所以，有关基于 NP 完全问题的密码协议的安全性讨论到目前为止才开始。“假定破译者有一百万个处理

器，每一个每秒能测试一百万次”，或许很快就被改成“假定破译者有一千个发酵桶，每一个有 20，000 升的容量。”

表 7.9 能阻止穷举攻击的对称密码和公钥密码的密钥长度

对称密码的密钥长度	公钥密码的密钥长度
56bits	384bits
64bits	512bits
80bits	768bits
112bits	1792bits
128bits	2304bits

量子计算法

现在，已经更不可思议了。量子计算法的基本原理就是爱因斯坦的玻粒二象性：光子可以同时存在于许多状态。一个典型的例子就是，当光射到银白色的镜面时，它会有波一样的特性，既可以反射也可以传播，就象波浪撞击一堵带有缺口的防波堤，有的会翻回去，有的却可以穿过去。然而对光子进行测量时它又表现出粒子的特性，有一个唯一的被测量的状态。

在文献[1143]中，Peter Shor 阐述了一个基于量子力学的因子分解机器设计模型。不象一般的计算机在某一特定时刻可以认为有一个单一、固定的状态，量子计算机有一个内部波动函数，这个函数是所有可能基状态的联合重叠。计算机在单步运算中通过改变整套的状态值来改变波动函数。在这个意义上，量子计算机是基于经典的有限状态机改进而成的：它利用量子特性允许在多项式时间里进行因子分解。理论上可以用来破译基于大数分解或离散对数问题的密码体制。

舆论一致认为，量子计算机与基本量子力学定理是可和谐共存的。然而，在可预见的未来制造出一台量子因子分解机基本上是不可能的。其最大的障碍是非连贯性问题，因为它容易导致叠加后的波形丢失某些特性，从而使计算失败。不连贯性会使运行在 1Kelvin 下的计算机仅 1 纳秒后就死机。另外，制造一台量子因子分解设备需要超大量的逻辑门，这使得制造不太可能。Shor 的设计需要一部完整的模取幂计算机。由于没有内部时钟，数以千万甚至上亿的独立门被用于分解密码上非常大的数，如果 n 个量子门有很小的错误概率 p，则每成功运行一次所需实验次数就是（1-/（1-p））^{n}。量子门的数目可假定按被分解的数的长度（比特）呈多项式增加，那么，实验次数将随该长度呈超级指数增长——这比用试除法进行分解还要糟糕！

所以，虽然量子因子分解法在学术上非常令人兴奋，但它在不远的将来被用于实践却是不可能的。别说我没有提醒你！

7.3 对称密钥和公钥密钥长度的比较

一个系统往往是在其最弱处被攻击。如果你同时用对称密钥体制算法和公开密钥算法设计一个系统，那么你应该好好选择每一种算法的密钥长度，使它们被不同的方式攻击时有着

同样的难度。128-比特的对称密钥算法和 386-比特的公钥算法同时使用将毫无意义，就如同使用 56-比特的对称密钥算法和 1024-比特的公钥算法一样。

表 7.9 列出了一系列攻击难度相同的对称密钥长度和公钥密钥长度。

该表说明如果你认为对称密码算法密钥必须 112 比特才安全的话，那你的公开密钥算法的模数就得是 1792 比特。尽管如此，一般而言你应该选择比你的对称密钥算法更安全的公钥密钥长度，因为公钥密钥算法通常持续时间长，且用来保护更多的信息。

7.4 对单向 Hash 函数的生日攻击

对单向 Hash 函数有两种穷举攻击的方法。 第一种是最明显的：给定消息的 Hash 函数 H（M），破译者逐个生成其他文件 M ‘，以使 H（M）= H（M’）。第二种攻击方法更巧妙：攻击者寻找两个随机的消息：M，M ‘，并使 H（M）=H（M’）。这就是所谓的冲突攻击法（Collision Attack）。这比第一种方法来得容易。

生日悖论是一个标准的统计问题。房子里面应有多少人才有可能使至少一人与你生日相同？答案是 253。既然这样，那么应该有多少人才能使他们中至少两个人的生日相同呢？答案出人意料地低：23 人。对于仅有 23 个人的屋里，在屋里仍有 253 个不同对的人。

寻找特定生日的某人类似于第一种方法；而寻找两个随机的具有相同生日的两个人则是第二种攻击。第二种方法通常被称为生日攻击（Birthday Attack）。

假设一个单向 Hash 函数是安全的，并且攻击它最好的方法是穷举攻击。假定其输出为 m 比特，那么寻找一个消息，使其单向函数值与给定函数值相同则需要计算 2^m 次；而寻找两个消息具有相同的 hash 值仅需要试验 $2^{m/2}$ 个随机的消息。每秒能 hash 运算一百万次的计算机得花 600,000 年才能找到第二个消息与给定的 64-比特 hash 值相匹配。同样的机器可以在大约一个小时里找到一对有相同的 hash 值的消息。

这就意味着如果你对生日攻击非常担心，那么你所选择的 hash 函数值其长度应该是你本以为可以的两倍。例如，如果你想让他们成功破译你的系统的可能低于 $1/2^{80}$，那么应该使用 160-比特的单向 Hash 函数。

7.5 密钥应该多长

答案并不固定，它要视情况而定。为了断定你需要多高的安全性，你应该问自己一些问题：你的数据价值有多大？你的数据要多长的安全期？攻击者的资源情况怎样？

一个顾客清单或许值 1000 美元；一起令人痛苦的离婚案件的财政数据或许值 10000 美元；一个大公司的广告和市场数据应该值 1 百万美元；而一个数据取款系统的主密钥价值可能会超过亿元。

在商品贸易的世界里，保密仅需要数分钟，在报纸行业，今天的秘密将是明天的头条标题。产品开发信息或许需要保密一到两年，根据法律美国人口普查数据要保密 100 年。

参加你小妹妹令人惊讶的生日晚会的客人名单只能引起那些爱管闲事的亲戚们的兴趣；公司的贸易秘密是那些竞争公司所感兴趣的；对敌军来说军事秘密是值得感兴趣的。

你可以据此阐述你的安全需求。例如，

密钥长度必须足够长，以使破译者花费一亿美元在一年中破译系统的可能性也不超过 $1/2^{32}$，甚至假设技术在此期间每年有 30%的增长速度。

表 7.10 部分摘自文献[150]，给出了对各种信息的安全需要的估计。

表 7.10　不同信息的安全需要

信息类型	时间	最小密钥长度
战场军事信息	数分钟/小时	56-64 bits
产品发布、合并、利率	几天/几周	64 bits
贸易秘密	几十年	112 bits
氢弹秘密	>40 年	128 bits
间谍的身份	>50 年	128 bits
个人隐私	>50 年	128 bits
外交秘密	>65 年	至少 128 bits
美国普查数字	100 年	至少 128 bits

将来计算机能力是难以估计的，但这里有一个合情合理的经验方法：计算机设备的性价比每 18 个月翻一番或以每 5 年十倍的速度增长。这样，在 50 年内最快的计算机将比今天快 10^{10} 倍，且这些数字仅对于普通用途的计算机而言；谁能知道某种特制的密码破译机在下个 50 年内如何发展呢？

假定一种加密算法能用 30 年，你就能对它是多么安全有一个概念。现在设计的一种算法或许直到 2000 年才会普遍使用，并将在 2025 年仍然运用它来为那些需保密至 2075 年或更晚的信息加密。

7.6 总结

胡扯了整个一章。有关对未来 10 年计算能力的预测是十分滑稽的，更不用说 50 年了。那些计算仅可以理解为一个指导，仅此而已。如果说过去就是一个向导的话，已经表明，未来与我们预测的的情况会有很大不同。

保守一点吧。如果你的密钥比你认为必须有的长度还要长的话，那么没有太多令人惊讶的技术能够伤害你。

第八章 密钥管理

Alice 与 Bob 有一个保密通信系统。他们玩智力扑克游戏，签定了合同，甚至相互交换数字现金。他们的通信协议是安全的，算法也是一流的。不幸地是，他们的密钥来自 Eve 的“R-Us 密钥”，该店的口号是“你可以相信我们：安全性是我们前任婆婆的旅行社在 Kwik-E-Mart 遇到的某个人的别名”。

Eve 不需要去破译这些算法，也不依靠协议的微小缺陷，她尽可以使用他们的密钥阅读所有 Alice 与 Bob 的通信而不动一根破译的指头。

在现实世界里，密钥管理是密码学领域最困难的部分。设计安全的密钥算法和协议是不容易的，但你可以依靠大量的学术研究。相对来说，对密钥进行保密更加困难。

密码分析者经常通过密钥管理来破译对称密码体制和公钥体制，假如 Eve 能从粗心的密钥管理程序中很容易找到密钥，她何必为破译而操心烦恼呢？如果花一千美元能贿赂一个书记员，她何必花一千万去制造一台破译机器呢？利用一百万美元收买外交大使馆里一个职位不错的通信记录员是笔划得来的买卖。美国海军的加密密钥很多年前就已经被内贼们卖到了苏联。CIA（中央情报局）的军事情报人员，包括他们的妻子，可以为低于 200,000 美元所引诱，这可比建造大型的攻击机器和雇佣聪明的密码分析专家便宜得多。Eve 还可以偷到密钥，也可以逮到或绑架知道密钥的人。当然，Eve 利用色相对知晓密钥者进行勾引，也能得到密钥。（保卫莫斯科美国大使馆的海军陆战兵们对此并不具备免疫力。）在人身上找到漏洞比在密码体制中找到漏洞更容易。

Alice 与 Bob 必须像保护他们的数据那样保护他们的密钥。如果一个密钥不经常更改，那么分析者可获得大量的数据。不幸地是，许多商业产品只简单地标注“使用 DES”而将其他事情抛在脑后。这样的结果往往并不令人振奋。

举个例子，大多数软件店出售的对 Macintosh（2.1 版）的磁盘锁程序声称具有 DES 加密算法的安全性。他们的文件的确是用 DES 加密的，并且对 DES 算法的实现也正确。然而磁盘锁将 DES 密钥放在加密后的文件中。如果你知道到哪里去寻找密钥，并想阅读用磁盘锁的 DES 加密后的文件，从加密的文件中恢复密钥，然后用它解密。这与是否使用 DES 加密该程序没有任何关系——这种应用是完全不安全的 。

有关密钥管理的进一步资料可见文献[457，98，1273，1225，775，357]。下面几节深入讨论一些细节及其解决方案。

8.1 密钥生成

算法的安全性依赖于密钥，如果你用一个弱的密钥生成方法，那么你的整个体制是弱的。因为能破译你的密钥生成算法，所以 Eve 就不需要试图去破译你的加密算法了。

减少的密钥空间

DES 有 56 比特的密钥，正常情况下任何一个 56 比特的数据串都能成为密钥，所以共有 2^{56}（10^{16}）种可能的密钥。Norton Discreet for MS-DOS（8.0 版或更低的版本）仅允许 ASCII 码的密钥，并强制每一字节的最高位为零。该程序将小写字母转换成大写（使得每个字节的

第 5 位是第 6 位的逆），并忽略每个字节的最低位。这样就导致该程序只能产生 2^{40} 个可能的密钥。这些糟糕的密钥生成程序使 DES 的攻击难度比正常情况低了一万倍。

表 8.1 给出了在不同输入限制下可能的密钥数。表 8.2 给出了在每秒一百万次测试的情况下，寻找所有这些密钥消耗的时间，记住，穷举搜索 8 字节密钥与搜索 4、5、6、7、8 字节密钥在所费时间上有很小的区别。

表 8.1　不同密钥空间的可能密钥数

	4-BYTES	5-BYTES	6-BYTES	7-BYTES	8-BYTES
Lowercase letters(26)	460,000	1.2×10^{7}	3.1×10^{8}	8.0×10^{9}	2.1×10^{11}
Lowercase letters and digits(36)	1,700,000	6.0×10^{7}	2.2×10^{9}	7.8×10^{10}	2.8×10^{12}
Alphanumeric charactets(62)	1.5×10^{7}	9.2×10^{8}	5.7×10^{10}	3.5×10^{12}	2.2×10^{14}
Printable characters(95)	8.1×10^{7}	7.7×10^{9}	7.4×10^{11}	7.0×10^{13}	6.6×10^{15}
ASCII charactrers(128)	2.7×10^{8}	3.4×10^{10}	4.4×10^{12}	5.6×10^{14}	7.2×10^{16}
8-bit ASCII characters(256)	4.3×10^{9}	1.1×10^{12}	2.8×10^{14}	7.2×10^{16}	1.8×10^{19}

表 8.2　不同密钥空间穷举搜索时间（假设每秒测试一百万次）

	4-BTES	5-BYTES	6-BYTES	7-BYTES	8-BYTES
Lowercase letters(26)	0.5sec.	12secs.	5mins.	2.2hrs.	2.4days
Lowercase letters and digits(36)	1.7sec.	1min.	36mins.	22hrs.	33days
Alphanumeric charactets(62)	15.0sec.	15 mins.	16hrs.	41days.	6.9yrs.
Printable characters(95)	1.4mins.	2.1hrs.	8.5days.	2.2yrs.	210yrs.
ASCII charactrers(128)	4.5mins.	9.5hrs.	51days	18yrs.	2300yrs.
8-bit ASCII characters(256)	1.2hrs	13days	8.9yrs.	2300yrs	580,000yrs.

特定的穷举攻击硬件和并行工具将在这里工作，每秒测试一百万次（或是一台机器或是多台机器并行运转），那么将能破译小写字母和小写字母与数字的 8 字节的密钥，7 字节的字母数字符号密钥，6 字节长的印刷字符和 ASCII 字符密钥，5 字节长的 8 比特—ASCII—字符密钥。

记住，计算机的计算能力每 18 个月翻倍。 如果你期望你的密钥能够抵抗穷举攻击 10 年，那么你最好相应的做出计划。

弱密钥选择

人们选择自己的密钥时，通常选择一个弱密钥。他们常常喜欢选择“Barney”而不是“*9（hH/A。”。并不能总归罪于不良的安全实践，而是“Barney”的确比“*9（hH/A。”更容易记忆。最安全的密码体制也帮不了那些习惯用他们配偶的名字作为密钥或者把密钥写下来揣在兜里的人。一个聪明的穷举攻击并不按照数字顺序去试所有可能的密钥，它们首先尝试

最可能的密钥。

这就是所谓的“字典攻击”，因为攻击者使用一本公用的密钥字典。Daniel V.Klein 用这个系统能够破译一般计算机上 40%的口令[847，848]。试图登录时，他并不是一个口令接一个口令的试验，他把加密的口令文件副本下来然后进行离线攻击。下面是他所试验的：

（1） 用户的姓名、简写字母、帐户姓名和其他有关的个人信息都是可能的口令，基于所有这些信息可以尝试到 130 个口令。对于一个名叫“Daniel V.Klein”， 帐户名为“Klone”的用户 ，用来尝试口令的一些词是“klone，klone0，klonel， klone123，dvk ，dvkdvk，dklein，Dklein， leinad，nielk，dvklein，danielk，DvkkvD，DANIEL-KLEIN，(klone)，KleinD 等等。

（2） 使用从各种数据库中得到的单词。这些单词是男人和女人的姓名名单（总共约达 16，000）；地点（包括像“spain”、“spanish”和“spaniard”这样的排列也全被考虑在内）；名人的姓名；卡通漫画和卡通人物；电影和科幻小说故事的标题、有关人物和地点；神话中的生物名字（从《Bulfinch 神话故事》和神话动物字典中产生出的）；体育活动（包括球队名、一些浑名，和职业队名称）；数字（比如“2001”和写出的“twelve”）；一串字母和数字（“a”“aa”“aaa”“aaaa”，等等）；中文音节（选自汉语拼音字母或在英文键盘上输入中文的国际标准系统）；《圣经》的权威英译本；生物术语；公用的粗话（如“fuckyou”、“ibmsux”和“deadhead”）；键盘模式（如“qwerty”、“asdf”和“zxcvbn”）；缩写（如“roygbiv”——彩虹的七种颜色和“ooottafagvah”——帮助记忆头部十二条神经的东西）；机器名称（可从 letc/hosts 中获得）；莎士比亚作品中的人物、戏剧和地点；常用的犹太语；小行星名称和 Klein 以前出版的技术论文中搜集到的单词。综上，每个使用者可以考虑超过 66，000 个独立的单词（舍弃字典内外复制的那些）。

（3）第（2）步得到的单词的不同置换形式。这包括使第一个字母大写或作为控制符，使整个单词大写，颠倒单词的顺序（不管前面有无大写），将字母“O”换成数字“0”（使得单词“scholar”变作“sch0lar”），将字母“l”换成数字“1”（使单词“scholar”变成“scho1ar”），以及进行同样操作将字母“Z”换成数字“2”，“S”换成“5”。另一种测试是将单词变为复数形式（不管它是否为名词），非常聪明的将“dress”变为“dresses”、“house”变为“houses”，并且“daisy”变为“daisies”， Klein 并不考虑复数规则，“datum”可以变为“datums”（不是“data”）“sphynx”变为“sphynxs”（而不是“sphynges”），同样地，将后缀“-ed”，“-er”和“-ing”加到单词上，如“phase”变为“phased”，“phaser”和“phasing”。这些附加的测试使得每一位使用者可能的口令清单增加了的 1,000,000 个单词。

（4）从第（2）步得到的单词的不同的大写置换形式，不考虑第（3）步。这包括所有单字母的单个大写置换（如“michael”可换为“mIchael”，“miChael”，“michAel”等等）双字母大写置换（“MIchael”，“MiChael”，“MicHael” ……“mIChael”，“mIcHael”等等）。三字母置换，等等。对于每一个使用者，单字母置换增加了大约 400,000 个单词，双字母置换增加 1,500,000 个单词，三字母置换增加至少 3,000,000 个单词。必须要有足够的时间来完成测试，测试完成 4，5，6 个字母的置换没有充足的计算机“马力”是不可能的。

（5） 对外国用户要尝试外语单词，对有中文名称的用户要使用中文口令来进行特别的测试。汉语拼音字母组成单音节、双音节或三音节的单词，但由于不能测试确定它们是否是

实际存在的，所以要启动穷举搜索。(在汉语拼音中共有 298 个音节，158,404 个双音节词，稍多于 16,000,000 个三音节词。) 一种类似的攻击方式，就是穷举构造出来的可以发音但并不存在的单词，可以很容易的被用于英语中。

（6） 尝试词组。自然测试所耗费的数字量是令人惊愕的。为了简化测试，只有在/usr/dict / words 中存在，且仅有 3，4 个字母长的才被测试。即使这样，词组数目也有千万。

当字典攻击被用作破译密钥文件而不是单个密钥时就显得更加有力。单个用户可以很机灵地选择到好密钥，如果一千个人各自选择自己的密钥作为计算机系统的口令，那么至少有一个人将选择攻击者字典中的词作为密钥。

随机密钥

好密钥是指那些由自动处理设备生成的随机的比特串。如果密钥为 64 比特长，每一个可能的 64 比特密钥必须具有相等可能性。这些密钥比特串要么从可靠的随机源中产生（见 17.14 节），要么从安全的伪随机比特发生器中产生（见 16 和 17 章）。如果自动处理办不到，就抛硬币或掷骰子吧！

这是非常重要的，但不要太拘泥于声源产生的随机噪音和放射性衰减产生的噪音谁更随机。这些随机噪声源都不是很完美的，但已经是足够好了。用一个好的随机数发生器产生密钥是很重要的，然而更加重要的是要有好的加密算法和密钥管理程序。如果你对密钥的随机性产生怀疑的话，请用后面讲的密钥碾碎技术。

许多加密算法有弱的密钥：特定的密钥往往比其它密钥的安全性差。建议对这些弱密钥进行测试，并且发现一个就用一个新的代替。DES 在 2^{56} 个密钥中仅有 16 个弱密钥，因此生成这些密钥的机会小。已经证明，密码分析者对使用弱密钥一无所知，因而也就不能从这个偶然的使用中获得什么，不用弱密钥加密反而给密码分析者提供了信息。然而，对几个弱密钥进行检测是如此容易以至让我们轻率地不去做它 。

对公钥密码体制来说，生成密钥更加困难，因为密钥必须满足某些数学特征（必须是素数的，是二次剩余的，等等）。11.5 节讨论了产生大随机素数的技术。从密钥管理的观点看，发生器的随机种子也必须是随机的，这一点是应该记住的。

产生一个随机密钥并不总是可能的。有时你需要记住密钥（看你用多长时间能记住这串字符：25E856F2E8BAC820）。如果你必须用一个容易记忆的密钥，那就使它苦涩难懂。比较理想的情况是该密钥既容易记忆，又难以被猜中。下面是一些建议：

——词组用标点符号分开，例如“turtle*moose”或者“zorch! splat”。

——由较长的短语的首字母组成字母串。例如由“Mein Luftkissenfahrzeug ist voller Aale”产生密钥“MlivA！”。

通行短语

一个比较好的办法是利用一个完整的短语代替一个单词，然后将该短语转换成密钥。这些短语被称为通行短语。一项称为密钥碾碎的技术可以把容易记忆的短语转换为随机密钥，使用一个单向 Hash 函数可将一个任意长度的文字串转换为一个伪随机比特串。

例如，易于记忆的文本串：

My name is Ozymandias，king of kings。Look on my works，ye　mighty，and despair。

可以被碾碎成一个 64 比特密钥：

E6C1 4398 5AE9 0A9B

当然，向一个显示关闭的计算机输入完整的短语是很困难的，所以，解决该问题的好建议是值的推荐的。

如果这个短语足够长，所得到的密钥将是随机的。“足够长”的确切涵义还有待解释。信息论讲到，在标准的英语中平均每个字符含有 1.3 比特的信息（参见 11.1 节）。对于一个 64 比特的密钥来说，一个大约有 49 个字符或者 10 个一般的英语单词的通行短语应当是足够的。通常地讲，每 4 个字节的密钥就需要 5 个单词。这是保守的假设，因为字母的大小写、空格键、及标点符号并没有考虑之内。

这种技术甚至可为公开密钥体制产生私钥：文本串被“碾碎”成一个随机种子，该种子被输入到一个确定性系统后就能产生公钥/私钥对。

如果你正打算选择一个通行短语的话，就选择独特而容易记忆的，最好别在文学著作里面选取——如，“Ozymandias”就是很差的。莎士比亚的全本著作和《星球大战》里的对话都在线提供，并且很容易被用来进行字典攻击。要选择难懂，但有个性的词，包括一些标点和大写字母，如果能够，还可以包括数字和非字母符号。糟糕的或者不正确的英语，甚至一门外语都会使通行短语对字典攻击缺乏敏感性。对使用通行短语有一个建议就是“鬼扯”：容易记住但不可能被写出的胡言乱语。

不管上面怎么叙述，难懂决不是真正随机的代替品。最好的密钥还是随机密钥，尽管很难记住。

X9.17 密钥生成

ANSI X9.17 标准规定了一种密钥生成方法（见图 8.1）[55]。它并不生成容易记忆的密钥，它更适合在一个系统中产生会话密钥或伪随机数。用来生成密钥的加密算法是三重--DES，它就像其它算法一样容易。

设 E_k（X）表示用密钥 K 对 X 进行三重 DES 加密。K 是为密钥产生器保留的一个特殊密钥。V_0 是一个秘密的 64 比特种子，T 是一个时间邮戳。欲产生随机密钥 R_i，计算：

$$R_i = E_k(E_k(T_i) \oplus V_i)$$

欲产生 V_{i+1}，计算：

$$V_{i+1} = E_k(E_k(T_i) \oplus R_i)$$

要把 R_i 转换为 DES 密钥，简单地调整每一个字节第 8 位奇偶性就可以。如果你需要一个 64 比特密钥，按上面计算就可得到。如果你需要一个 128 比特的密钥，产生一对密钥后再把它们串接起来便可。

DoD 密钥生成

美国国防部建议用 DES 在 OFB 方式（参见 9.8 节）下生成随机密钥[1144]。由系统中断向量、系统状态寄存器和系统计数器生成 DES 密钥；由系统时钟、系统 ID 号、日期、时间生成初始化向量；外部生成的 64-比特量可作为明文输入（V_0）：如，系统管理员键入的 8

个字符。这样，其输出就可作为你的密钥。

8.2 非线性密钥空间

假设你是一个军事密码组织，为你的手下制造了一批加密设备。你想使用一个安全的算法，但又怕这些设备落入敌手。最后你想做的就是你的敌人能够用这些设备保护*他们*的秘密。

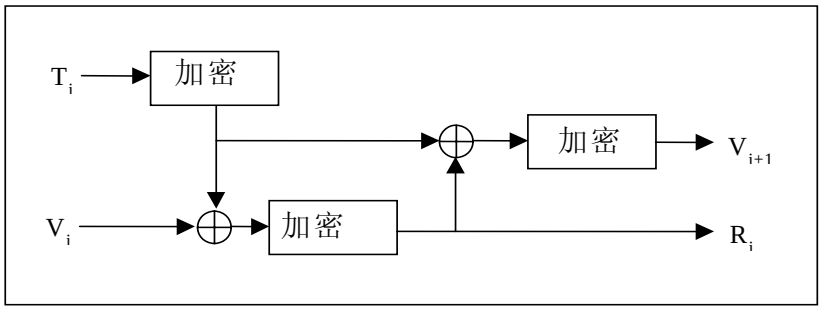

图 8.1 ANSI X9.17 密钥生成

如果你能将你的算法加入到一个防窜改模块中，下面就是你能做的。你可以要求有特殊保密形式的密钥，而其它的密钥都会引起模块用非常弱的算法加解密。你这样做可以使那些不知道这个特殊形式的人偶然碰到正确密钥的机会几乎为零。

这就是所谓的非线性密钥空间，因为所有密钥的强壮程度并不相等。（与之相对应的是线性或平坦密钥空间。）一个很简单的方法可以做到非线性密钥空间，就是按照两部分来生成密钥：密钥本身和用该密钥加密的某个固定字符串。模块用这个密钥对字符串进行解密；如果它收到那个固定的字符串它就很正常地解密，否则就用另一个非常弱的算法进行。如果该算法有一个 128-比特的密钥和一个 64-比特的字符块，也就是总密钥长度为 192 比特，那么共有有效密钥 2^{128} 个，但是随机选择一个好密钥的机会却成了 $1/2^{64}$。

你甚至更巧妙，可以设计一种算法使某些密钥比别的密钥更好。一个没有弱密钥，至少是没有很明显弱密钥的算法仍然可以有非线性密钥空间。

使用非线性密钥空间仅当在算法是安全的，并且敌人不能对其进行反控制，或者密钥强度的差异是足够细微，以至于敌人不能计算出来时才可行。美国国家安全局对 Overtake 模块中的安全算法进行非线性密钥空间的使用（参见 25.1 节）。他们会不会对 Skipjack 算法（见 13.12 节）做同样的事情呢？没人知道。

8.3 发送密钥

Alice 和 Bob 采用对称加密算法进行保密通信：他们需要同一密钥。Alice 使用随机密钥发生器生成一个密钥，然后必须安全地送给 Bob。如果她能在某个地方碰见 Bob（一个僻静

的小巷，一个无窗的小屋或是木星的某个卫星上），她能将密钥副本交给他，否则就会出问题。公钥加密算法用最小的预先安排可以很好地解决了这个问题，但是这一技术并不总是有效（参见 3.1 节）。一些系统使用被公认安全的备用信道，Alice 可以通过一个可靠的通信员把密钥传送给 Bob，也可以用合格的邮政或通宵传递业务来传送。或者，她可能同 Bob 一起建立另一个希望无人窃听的通信信道。

Alice 可通过他们加密的通信信道把对称密钥送给 Bob。但这是愚蠢的，因为如果信道能够保证加密，那么在同一个信道上明文发送加密密钥就能够保证在该信道上的任何偷听者都能破解全部通信。

X9.17 标准描述了两种密钥：密钥加密密钥和数据密钥。密钥加密密钥加密其它需要分发的密钥；而数据密钥只对信息流进行加密。除少数例外，密钥加密密钥必须进行手工分发（尽管也可在一个防窜改设备里安全进行，如，智能卡）。数据密钥的分发更加频繁，更多细节参见文献[75]，密钥分发中用到了这两种密钥的概念。

对密钥分发问题的另一个解决方法是将密钥分成许多不同的部分（见 3.6 节）然后用不同的信道发送出去：有的通过电话，有的通过邮寄，有的还可以通过通宵专递或信鸽等等（见图 8.2）。即使截获者能收集到密钥，但缺少某一部分，他仍然不知道密钥是什么，所以该方法可以用于除个别特殊情况外的任何场合。3.6 节讨论了拆分密钥的有关方案。Alice 甚至可以用秘密共享方案（见 3.7 节），允许 Bob 在传输过程丢失部分密钥时能重构完整密钥。

Alice 可以面对面的或者用刚讨论过的密钥拆分技术将密钥加密密钥送给 Bob，一旦他们两人同时拥有了该密钥，Alice 就可以先用它对每天的数据密钥进行加密，然后在同一信道上把它传送给 Bob。由于用密钥加密密钥对大量的数据加密速度非常慢，它不须经常改动。然而，密钥加密密钥遭到损害就会使那些用它本身加密的密钥加密的所有信息受到损害，所以必须对它进行安全的存储。

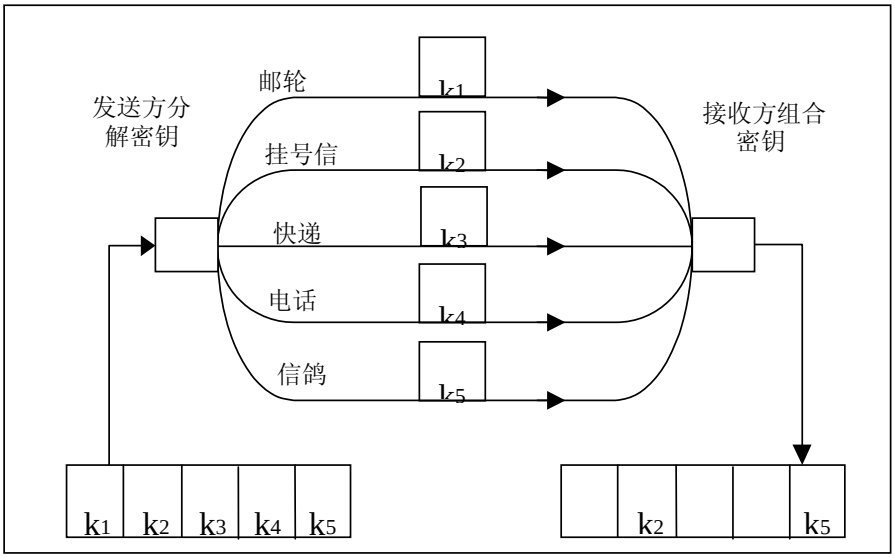

图 8.2 通过并行信道的密钥分发

大型网络的密钥分发

在小型网络中，每对用户可以很好地使用密钥加密密钥。如果网络变大，将很快变得令人讨厌。因为每对用户必须交换密钥，n 个人的网络总的交换次数为 n（n-1）/2。

6 人网络需交换 15 次密钥。1000 人网络则需近 500,000 次，在这种情况下，建造一个中央密钥服务器(或服务器组) 会使操作更加有效。

另外，在 3 .1 节中讲到的对称密码或公钥密码协议制都提供了安全的密钥分发。

8.4 验证密钥

当 Bob 收到密钥时，他如何知道这是 Alice 传送的而不是其他人伪装 Alice 传送的呢？如果是 Alice 亲自递给他的，那自然简单；如果 Alice 通过可靠的信使传送密钥，Bob 必须相信信使；如果密钥由密钥加密密钥加密， Bob 必须相信只有 Alice 才拥有那个加密密钥；如果 Alice 运用数字签名协议来给密钥签名，那么当 Bob 验证签名时就必须相信公开密钥数据库；如果某个密钥分配中心（KDC）在 Alice 的公钥上签名，Bob 必须相信 KDC 的公开密钥副本不曾被窜改过。

结果，控制了 Bob 周围整个网络的人都能够使 Bob 认为控制者想干什么。Mallory 可以传送一个加密和签名的消息而将它伪装成来自 Alice，当 Bob 试图访问公钥数据库以验证 Alice 的签名时，Mallory 可以用他自己的公钥来代替。他可以发明自己的假 KDC，并把真正的 KDC 公钥换成她自己产生的公钥。Bob 不会察觉。

利用该缺陷的一些人声称公钥密码体制是无用的。既然 Alice 与 Bob 保证他们密钥不被窜改的唯一方式是面对面交换，公钥体制对提高安全性一点用处也没有。

这种观点是幼稚的。理论上这种说法是错误的，但实际情况却复杂得多。采用数字签名和可信赖 KDC 的公钥体制，使得一个密钥代替另一个密钥变得非常困难，Bob 从来都不能绝对肯定 Mallory 没有控制他的整个网络，但他相信那样做需要比 Mallory 能够访问到的大多数资源还要多。

Bob 可以通过电话核实 Alice 的密钥，那样他可以听到她的声音。声音辨别是一个真正的好的鉴别方案。如果这是一个公钥，他可以当着大家的面通过电话安全的背诵出来。如果是一个秘密密钥，他就用一个单向 Hash 函数来核实密钥。PGP（参见 24.12 节）和 AT&T TSD（参见 24.18 节）就是用这种方法对密钥进行验证的。

有时，核实一个公开密钥到底属于谁并不重要，核实它是否属于去年的同一个人或许是有必要的。如果某人送了一个签名提款的信息到银行，银行并不关心到底谁来提款，它仅关心是否与第一次来存款的人属同一个人。

密钥传输中的错误检测

有时密钥在传输中会发生错误。密钥错误就意味着大量的密文无法解密，所以这是个问题。所有的密钥必须附着一些检错和纠错位来传输。这样，密钥在传输中的错误很容易地被检查出来，并且如果需要，密钥可被重传。

最广泛采用的一种方法是用密钥加密一个常量，然后把密文的前 2-4 字节与密钥一起发送。在接收端，做同样的工作，如果接收端加密后的常数能与发端常数匹配，则传输无错。

检查不出错误的机会是 2^{-16} 到 2^{-32} 之间。

密钥在解密过程中的错误检测

有时收方想检查他拥有的某个密钥是否是正确的对称解密密钥。如果明文与 ASCII 码类似，他就试着去解密，并阅读验证它，如果得到的明文是随机的，那么密钥就有问题。

最简单的方法是附加一个验证分组：加密之前给明文加一个已知的报头。在接收端，Bob 解密报头，并验证它的正确性。这是可行的，但是它却给 Eve 提供了已知的明文来帮助分析密码系统。它也使得对 DES 这样的短密钥密文和对所有可出口的密码攻击变得容易。一旦对每个密钥的校验和进行了预计算，就可以用它来确定之后截取到的任何信息的密钥。这就是校验和的特性，它不包含随机的数据，至少在每一个校验和中没有随机的数据。当利用通行短语生成密钥时，它在概念上类似于使用 “salt”。

下面是更好的方法[821]：

（1）产生一个初始化向量 IV（不用于消息）。

（2）用该初始化向量 IV 生成一个大的比特块：譬如，512 比特。

（3）进行单向 Hash 运算

（4）使用 Hash 运算结果的相同固定位置上的比特位（如 32 比特）作为密钥校验和。

这种方法不可避免地给 Eve 提供一些信息，但非常少。如果她试图利用最后 Hash 值的低 32 比特采取穷举攻击的话，她必须进行多重加密并对每一个候选密钥作 Hash 运算；对密钥本身进行穷举攻击将更迅速。

再者，她也得不到更多的已知明文信息以供攻击，如果她能设法选择我们的随机值，那她永远也得不到我们的明文，因为它已经在她看到之前被 Hash 运算了。

8.5 使用密钥

软件加密是可怕的。一台微机在一种程序控制下的时代已过去了，现在有 Macintosh 系统 7、Windows NT 和 UNIX，谁也说不准什么时候操作系统将会中止加密的运行，将一些东西写在磁盘上，和处理另外一些急需的工作。当操作系统最后回头加密那些正被加密的东西时一切好象还是那么好。没有谁意识到操作系统已把加密应用程序写到磁盘上，并同时将密钥也写了下来。这些密钥未被加密，在计算机重新覆盖那个存贮区之前它一直留在磁盘上，或许几分钟，或许几个月，甚至永远。当攻击者采用好的工具彻底搜索该硬盘驱动器时，密钥可能还放在那里。在一个可抢先的、多任务环境中，你可以设置你的加密操作足够高的优先权以防止被中断。尽管这样可以减轻危险度，但仍有风险。

硬件实现更安全。如果受到损害，许多加密设备被设计成能够擦出密钥，例如 IBM PS/2 加密卡有一个包含 DES 芯片、电池和内存的环氧单元。当然，你得相信硬件制造者正确地实现了这些特征。

在一些通信应用中，如电话加密机，可以使用会话密钥（Session Keys）。会话密钥是指用于一次通信会话——一次单独的电话通话，通话完毕就抛弃的密钥。这种密钥使用一次后

不再储存，并且如果你使用一些密钥交换协议将密钥从一端传到另一端，在使用之前也不会被储存。这样就减小密钥被损害的可能性。

控制密钥使用

在一些应用中控制怎样使用会话密钥是有意义的，有的用户需要它或许仅仅是为了加密，有的或许是为了解密，而会话密钥应该被授权仅用于某一特定机器或时间。运用这些限制的一个方案是在密钥后面附加一个控制向量（Control Vector，CV）——用它来标定密钥的使用和限制（参见 24.1 节）[1025，1026]。对 CV 取单向 Hash 运算，然后与主密钥异或；把得到的结果作为密钥对会话密钥进行加密，再把合成的加密了的会话密钥跟 CV 存在一起。恢复会话密钥时，对 CV 取 Hash 运算再与主密钥异或，最后用结果进行解密。

该方案的好处是 CV 可以任意长，并且总是以明的方式与加过密的密钥一起储存。本方案假定硬件防窜改和用户不能直接得到密钥。本系统进一步的讨论请参见 24.1 节和 24.8 节。

8.6 更新密钥

设想你每天都想改变加密的数据链路的密钥。有时，每天进行新的密钥分发的确是一件痛苦的事。更容易的解决办法是从旧的密钥中产生新的密钥，有时称为密钥更新。

更新密钥使用的是单向函数。如果 Alice 和 Bob 共享同一密钥，并用同一个单向函数进行操作，他们就会得到相同的结果。那么就可以从结果中得到他们所需要的数据来产生新的密钥。

密钥更新是可行的，但记住新密钥只是与旧密钥一样安全。如果 Eve 能够得到旧密钥，她自己可以完成密钥更新功能。然而，如果 Eve 得不到旧密钥，并试图对加密的数据流进行唯密文攻击的话，那对 Alice 和 Bob 来说这是一个很好的保护他们数据的方法。

8.7 存储密钥

最不复杂的的密钥存储问题是单用户的密钥存储， Alice 加密文件以备以后用。因为只涉及她一个人，且只有她一人对密钥负责。一些系统采用简单方法：密钥存放于 Alice 的脑子中，而决不放在系统中，Alice 只需记住密钥，并在需要对文件加密或解密时输入。

该系统的一个例子是 IPS[881]。用户可直接输入 64-比特密钥，或输入一个更长的字符串，系统自动通过密钥碾碎技术从这个字符串生成 64-比特密钥。

其它解决方案有：将密钥储存在磁条卡中，嵌入 ROM 芯片的塑料密钥（称为 ROM 密钥）或智能卡 [556，557，455]。用户先将物理标记插入加密箱上或连在计算机终端上的特殊读入装置中，然后把密钥输入到系统中。当用户使用这个密钥时，他并不知道它，也不能泄露它。他只用这种方法使用它，且为了显示控制向量时使用它。

ROM 密钥是一个很聪明的主意。人们已经对物理钥匙很熟悉了，知道它们意味着什么和怎样保护它们。将一个加密密钥做成同样的物理形式就会使储存和保护它更加地直观。

把密钥平分成两部分，一半存入终端一半存入 ROM 密钥使得这项技术更加安全。美国政府的 STU-III 保密电话就是用的这种方法。丢失了 ROM 密钥并不能使加密密钥受到损害——换掉它一切就正常如初。丢失终端密钥情况也如此。这样，两者之一被损害都不能损害整个密钥——敌人必须两部分都有才行。

可采用类似于密钥加密密钥的方法对难以记忆的密钥进行加密保存。例如，一个 RSA 私钥可用 DES 密钥加密后存在磁盘上，要恢复密钥时，用户只需把 DES 密钥输入到解密程序中即可。

如果密钥是确定性地产生的（使用密码上安全的伪随机序列发生器），每次需要时从一个容易记住的口令产生出密钥会更加简单。

理想的情况是密钥永远也不会以未加密的形式暴露在加密设施以外。这始终是不可能的，但是可以作为一个非常有价值的奋斗目标。

8.8 备份密钥

Alice 是保密有限公司的首席财政官员——“我们不能告诉你我们的秘密。”像任何好的公司官员一样，她遵守公司的保密规则，把她的所有数据都加密。不幸地是，她没注意到公司旁边的过街警告线而被一辆卡车撞倒了。公司的董事长，Bob 该怎么办呢？

除非 Alice 留下了她的密钥的副本，否则 Bob 麻烦就大了。加密的意义就是使文件在没有密钥时不能恢复。除非 Alice 是弱智，并用糟糕透顶的加密软件，否则她的文件便永远地丢失了。

Bob 有几种方法可避免这种事情发生。最简单的方法，有时称密钥托管方案（见 4.14 节）：他要求所有雇员将自己的密钥写下来交给公司的安全官，由安全官将文件锁在某个地方的保险柜里（或用主密钥对它们进行加密）。现在，当 Alice 在州际公路上被撞倒后，Bob 可向他的安全官索取她的密钥。Bob 保证自己也可以打开保险箱，否则，如果安全官被另一辆卡车撞倒了，Bob 只得再次倒霉。

与密钥管理相关的问题是 Bob 必须相信他的安全官不会滥用任何人的密钥。更重要地是，所有雇员都必须相信安全官不会滥用他们的密钥。一个更好的方法是采用一种秘密共享协议（见 3.7 节）。

当 Alice 产生密钥时，她将密钥分成若干片，然后，她把每片——当然加密——发给不同的公司官员，单独的任何一片都不是密钥，但是某人可以搜集所有的密钥片，并重新把密钥恢复出来。于是，Alice 对任何恶意者做了防备，Bob 也对 Alice 被撞引起的数据丢失做了预防。或者，她可以用每一位官员不同的公钥把不同的片段加密，然后存入自己的硬盘之中。这样在需要进行密钥管理之前，没有人卷入到密钥管理中。

另一个备份方案是[188]用智能卡（参见 24.13 节）作为临时密钥托管。Alice 把加密她硬盘的密钥存入智能卡，当她不在时就把它交给 Bob。Bob 可以利用该卡进入 Alice 的硬盘，但是由于密钥被存在卡中，所以 Bob 不知道密钥是什么。并且，系统具有双向审计功能：Bob 可以验证智能卡能否进入 Alice 的硬盘；当 Alice 回来后可以检查 Bob 是否用过该密钥，并用了多少次。

这样的方案对数据传输来说是毫无意义的。对于保密电话，密钥在通话的时间有效，以后就无效了。象刚描述的那样，对数据储存来说密钥托管方案可能是个好主意。虽然我的记忆力比大多数人都好，但大约每隔五年我就丢失一把钥匙。如果两亿人都用密码，那么按这个概率每年约有 4 千万个密钥丢失。我把房门钥匙复制下来放到邻居那里以防止把它弄丢。如果房门钥匙就象密钥一样让我给丢了，那我永远也不能进入房子，并挽回财产了。就象我把数据换个地方备份一样，对数据加密密钥进行备份是非常有意义的。

8.9 泄露密钥

本书中所有的协议、技术、算法仅当在密钥（公钥体制中的私钥）保密的情况下安全，如果 Alice 的密钥丢失、被盗、出现在报上、或以其它方式泄露，则她的所有的保密性都失去了。

如果对称密码体制泄露了密钥，Alice 必须更换密钥，并希望实际损失最小。如果是一个私钥，问题就大了，她的公钥或许就在所有网络的服务器上。Eve 如果得到了 Alice 的私钥，她就可以在网络上冒充 Alice：读加密邮件、对信件签名、签合同等等。Eve 能很有效地变成 Alice。

私钥泄露的消息通过网络迅速蔓延是最致命的。任何公钥数据库必须立即声明一个特定私钥被泄露，以免怀疑有人用该泄露的密钥加密消息。

希望 Alice 能知道她的密钥是何时泄密的。如果 KDC 正在管理密钥，Alice 应该通知它密钥已经泄露。如果没有 KDC，她就要通知所有可能接收到她消息的人。有人应该公布在丢失密钥之后再收到她的任何消息都是值得怀疑的，以及其他人也不应该再用与丢失密钥相对应的公钥给 Alice 发送消息。实际应用中应该采用各种时间邮戳，这样用户就能识别哪些消息合法，哪些是值得怀疑的。

如果 Alice 不知道她的密钥泄露的确切时间，事情就难办多了。Alice 要求撕毁合同，因为偷密钥者冒名代替她签了名，如果系统允许这样，那么任何人都可以以密钥已泄密为由在签名前撕毁合同。这对于裁决者来说的确是个难题。

这是非常严重的问题，并且它给 Alice 带来一个危险信号：将所有身份约束到一个单一密钥上。对 Alice 来说不同的应用用不同的密钥会更好——就象她钱柜上不同的锁有不同的钥匙一样。该问题的其它解决办法包括生物统计学、限制密钥的作用、时间延迟、会签。

这些程序和建议很难最佳，但这是我们能做的最好解决办法。所有这些目的就是为了保护密钥，其中最重要的是保护私钥。

8.10 密钥有效期

没有哪个加密密钥能无限期使用，它应当和护照、许可证一样能够自动失效。以下有几个原因：

——密钥使用时间越长，它泄露的机会就越大。人们会写下密钥，也会丢失，偶然事件也会发生的。如果你使用一年，那泄露的可能性比你使用一天要大得多。

——如果密钥已泄露，那么密钥使用越久，损失就越大。如果密钥仅用于加密一个文件服务器上的单个预算文件，它的丢失仅意味着该文件的丢失。如果密钥用来加密文件服务器上所有预算信息，那损失就大得多。

——密钥使用越久，人们花费精力破译它的诱惑力就越大——甚至采用穷举攻击法。破译了两个军事单位使用一天的共享密钥，就会使某人能阅读当天两个单位之间的通信信息。破译所有军事机构使用一年的共享密钥，就会使同样的人获取和伪造通行全球一年的信息。在我们的意识里，冷战后的世界里，哪个密钥会受到攻击呢？

——对用同一密钥加密的多个密文进行密码分析一般比较容易。

对任何密码应用，必须有一个策略能够检测密钥的有效期。不同密钥应有不同有效期，基于连接的系统，如电话就是把通话时间作为密钥有效期，当再次通话时就启用新的密钥。

专用通信信道就不这么明显了。密钥应当有相对较短的有效期，这主要依赖数据的价值和给定时间里加密数据的数量。每秒千兆位的通信链路所用的密钥自然应该比只有 9600 波特的 Modem 所用的密钥更换得频繁。假定存在一种有效方法传送新密钥，那么会话密钥至少每天就得更换。

密钥加密密钥无需频繁更换，因为它们只是偶尔地（一天很难用到一次）用作密钥交换。这只给密钥破译者提供很少的密文分析，且相应的明文也没有特殊的形式。然而，如果密钥加密密钥泄露，那么其潜在损失将是巨大的：所有的通信密钥都经其加密。在某些应用中，密钥加密密钥仅一月或一年更换一次。你必须在保存密钥的潜在危险和分发新密钥的潜在危险之间权衡一下。

用来加密保存数据文件的加密密钥不能经常地变换。在人们重新使用文件前，文件可以加密贮藏在磁盘上数月或数年，每天将它们解密，再用新的密钥进行加密，这无论如何都不能加强其安全性，这只是给破译者带来了更多的方便。一种解决方法是每个文件用唯一的密钥加密，然后再用密钥加密密钥把所有密钥加密，密钥加密密钥要么被记忆下来，要么保存在一个安全地点，或某个地方的保险柜中。当然，丢失该密钥意味着丢失所有的文件加密密钥。

公开密钥密码应用中的私钥的有效期是根据应用的不同而变化的。用作数字签名和身份识别的私钥必须持续数年（甚至终身），用作抛掷硬币协议的私钥在协议完成之后就应该立即销毁。即使期望密钥的安全性持续终身，两年更换一次密钥也是要考虑的。许多网络中的私钥仅使用两年，此后用户必须采用新的私钥。旧密钥仍需保密，以防用户需要验证从前的签名。但是新密钥将用作新文件签名，以减少密码分析者所能攻击的签名文件数目。

8.11 销毁密钥

如果密钥必须定期替换，旧钥就必须销毁。旧密钥是有价值的，即使不再使用，有了它们，攻击者就能读到由它加密的一些旧消息[65]。

密钥必须安全地销毁（参见 10.9 节）。如果密钥是写在纸上的那么必须切碎或烧掉。小心地使用高质量切碎机，市面上有许多低质的切碎机。本书中的算法对花费上百万美元及上

百万年的穷举攻击是安全的，如果攻击者在你的垃圾中获取到一包切碎的文件碎片，然后支付某贫困县城的 100 个失业工人每小时 10 美分让他们用一年的时间将这些碎片拼凑起来，这样重新找到密钥将只需花 26,000 美元。

如果密钥在 EEPROM 硬件中，密钥应进行多次重写。如果在 EPROM 或 PROM 硬件中，芯片应被打碎成小碎片四散开来。如果密钥保存在计算机磁盘里，应多次重写覆盖磁盘存贮的实际位置(参见 10.9 节)或将磁盘切碎。

一个潜在的问题是，在计算机中密钥可以很容易地进行副本和存储在多个地方。自己进行内存管理的任何计算机，不断地接收和刷新内存的程序，这个问题更加严重，没有办法保证计算机中密钥被安全的销毁，特别是在计算机操作系统控制销毁过程的情况下。谨慎的做法是：写下一个特殊的删除程序，让它查看所有磁盘，寻找在未用存贮区上的密钥副本，并将它们删除。还要记住删除所有临时文件或交换文件的内容。

8.12 公开密钥的密钥管理

公开密钥密码使得密钥较易管理，但它有自己的问题。无论网络上有多少人，每个人只有一个公开密钥。如果 Alice 想传送一段信息给 Bob，她必须知道 Bob 的公开密钥，这有以下几种方式：

——她可以从 Bob 处获得。

——她可以从中央数据库获得。

——她可以从她自己的私人数据库获得。

2.5 节讨论了基于 Mallory 用自己的密钥代替 Bob 密钥引起的针对公钥算法的多种可能的攻击。该攻击是 Alice 想给 Bob 发送信息，她进入公开密钥数据库获得了 Bob 的公开密钥。但是 Mallory 偷偷摸摸地用他自己的密钥代替了 Bob 的（如果 Alice 直接向 Bob 询问，Mallory 必须截取 Bob 的通信，并用他的密钥取代 Bob 的）。Alice 使用 Mallory 的密钥加密她的消息，并传给 Bob，Mallory 窃听到消息，破译并阅读该消息。他再重新用 Bob 的密钥加密，并传给 Bob，Alice 与 Bob 都被蒙在鼓里。

公钥证书

公钥证书是某人的公开密钥，由一个值得信赖的人签发。证书可用来防范用一个密钥替换另一个密钥的攻击[879]。Bob 的证书在公钥数据库中包含比他的公钥更多的数据，它含有关于 Bob 姓名、地址等等信息，并由 Alice 相信的某个人签名：Trent（通常作为证书机关或 CA）。通过对 Bob 的密钥及其有关信息签名，Trent 证实有关 Bob 的信息是正确的，且公钥属于 Bob 而非其他人的。证书在许多公钥协议中，如 PEM[825]（参见 24.10 节）和 X.509[304]（参见 24.9 节）扮演了重要的角色。

这类系统存在一个复杂的非密码学的问题：证书到底意味着什么？或者换个角度，谁值得信任，他给谁发证书？任何人都可以给其他人签发证书，但总得需要一些办法过滤掉那些值得怀疑的证书：例如，由别的公司的 CA 给公司雇员签发的证书。一般情况下，一个证书链是这样传送信任的：一个唯一的可信任的实体认证多个可信任的代理机构，这些机构再认

证一些公司 CA，最后这些公司的 CA 再认证他们的雇员。

下面是一些值得思考的问题：

——通过某人的证书能够对他的身份信任到什么程度 ？

——某人与给他公钥证书的 CA 之间是什么关系？怎样从证书中断定这种关系？

——谁可以被信任作为证书链的最高层：唯一的可信任实体？

——证书链可以有多长？

理想的情况是，Bob 在 CA 签发证书之前遵循某种鉴别程序。另外，各种时间邮戳或证书有效期对防止密钥泄露是很重要的[461]。

使用时间邮戳是远远不够的。密钥或者因为泄露或者由于管理的原因在没有到期之前就已经无效。 所以，CA 保存一个合法的证书清单是很重要的，这样用户就可以定期的查看它。密钥撤消仍然是很难解决的问题。

使用一个公钥/私钥密钥对仍然是不够的。当然，任何好的公钥密码的实现需要把加密密钥和数字签名密钥分开。分离密钥要考虑到不同的安全级别、有效期、备份过程等等。有人或许用储存在智能卡中的 2048-比特的密钥给消息签名，并能保密 20 年，然而他或许用储存在计算机中的只能保密 6 个月的 768-比特的密钥给它加密。

同样，单独一对加密和签名密钥还是不够的。象身份证一样，私钥证明了一种关系，而人不止有一种关系：Alice 分别可以以私人名义、Monolith 公司的副总裁 、和她的组织的主席名义给某个文件签名。其中有些密钥比其他的密钥更有价值，所以它们更应该更好地保护。Alice 必须将工作密钥的备份储存在公司的安全官那里；但她并不想公司拥有她用来对抵押契据签名的密钥。就象她口袋里有多把物理钥匙一样，Alice 也将拥有多个密码密钥。

分布式密钥管理

在有些情况，进行集中密钥管理是不可能的，或许没有 Alice 和 Bob 都相信的 CA，或许他们只相信他们的朋友，或许他们谁都不相信。

分布式密钥管理，如用于 PGP（参见 24.12），通过“介绍人”解决了此问题。介绍人是系统中对他们朋友的公钥签名的其他用户。例如，当 Bob 产生出他的公钥时，把副本给他的朋友 Carol 和 Dave，他们认识 Bob，两人分别在 Bob 的密钥上签名并给 Bob 一个签名副本。现在，当 Bob 把他的密钥送给 Alice 这个新来者时，他就将两个介绍人的签名一起给 Alice。如果 Alice 也认识并相信 Carol，她有理由相信 Bob 的密钥是合法的。如果 Alice 不太认识和信任 Carol 和 Dave，她也有理由认为 Bob 的密钥有效。如果她既不认识 Carol 也不认识 Dave 便没有理由相信 Bob 的密钥。

随着时间的推移，Bob 将收集更多的介绍人。如果 Alice 和 Bob 在同一个社交圈子里，Alice 很可能会认识 Bob 的介绍人。为了防止 Mallory 替换 Bob 的密钥，介绍人必须在签名前确信密钥是属于 Bob 的。也许介绍人需要面对面地收到密钥或通过电话证实它。

此机制的好处是不需要人人都得相信的 CA。缺点就是当 Alice 接收到 Bob 的密钥时，并不能保证她认识介绍人中的哪一个。因此就不能保证她相信密钥的合法性。

第九章　算法类型和模式

对称密码算法有两种基本类型：分组密码和序列密码。分组密码算法是在明文分组和密文分组上进行运算——通常分组长为 64 比特，但有时更长。序列密码算法作用在明文和密文数据流的 1 比特或 1 字节上，有时甚至是一个 32 比特的字。利用分组算法，相同的明文用相同的密钥加密永远得到相同的密文。用序列算法，每次对相同的明文比特或字节加密都会得到不同的密文比特或字节。

密码模式通常是基本密码、一些反馈、和一些简单运算的组合。运算是简单的，因为安全性依赖于基本密码，而不依赖模式。强调一点，密码模式不会损害算法的安全性。

还有其他安全的考虑事项：明文的模式应当隐藏；输入密文应当是随机的；通过向密文中引入错误来对明文进行控制应当是困难的；以及用同一个密钥加密多个信息应当是可能的。这些问题将在以后章节里详细讨论。

效率是另一个值得考虑的事情。运算模式将不会明显地降低基本密码的效率。在一些情况下，密文和明文大小相同是非常重要的。

第三个考虑事情是容错。一些应用需要并行加密或解密，而其它一些则需要能够尽可能多的进行预处理。无论怎样，在丢失或增加比特的密文流中，解密过程能够从比特错误中恢复是很重要的。正如我们将看到的，不同的模式将有不同的特征子集。

9.1　电子密码本模式

电子密码本（ECB）模式是使用分组密码算法的最明显方式：一个明文分组加密成一个密文分组。因为相同的明文分组永远被加密成相同的密文分组，所以在理论上制作一个包含明文和其相对应的密文的密码本是可能的。然而，如果分组的大小是 64 比特，那么密码本就有 2^{64} 项——对预计算和储存来说太大了。记住，每一个密钥有一个不同的密码本。

这是最容易的运行模式。每个明文分组可被独立地进行加密。你不必按次序进行，你可以先加密中间 10 分组，然后是尾部分组，最后加密最开始的分组。这对加密随机存取的文件，如数据库，是非常重要的。如果一个数据库用 ECB 模式进行加密的，那么任意一个记录都可以独立于其他记录被添加、删除、或者解密——假定记录是由离散数量的加密分组组成。如果你有多重加密处理器，当然处理是并行的，那么它们就可以独立地对不同的分组进行加解密而不用相互干涉。

ECB 模式所带来的问题是：如果密码分析者有很多消息的明密文，那他就可在不知道密钥的情况下编辑密码本。在许多实际情形中，消息格式趋于重复，不同的消息可能会有一些比特序列是相同的。计算机产生的消息，如电子邮件，可能有固定的结构。这些消息在很大程度上是冗余的或者有一个很长的 0 和空格组成的字符串。

如果密码分析者知道了明文“5e081bc5”被加密成密文“7ea593a4”，那么无论它在什么时候出现在另一段消息中，他就能立即将其解密。如果加密的消息具有一些冗余信息，那么这些信息趋向于在不同消息的同一位置出现，密码分析者可获得很多信息。然后他就可以

对明文发动统计学攻击，而不去考虑密文分组的长度。

消息的开头和结尾是致命之处，因为那儿规定了消息头和消息尾，其中包含了关于发送者、接收者、日期等的信息。这个问题有时叫做格式化报头和格式化结尾。

该模式好的一面就是用同一个密钥加密多个消息时不会有危险。实际上，每一个分组可被看作是用同一个密钥加密的单独消息。密文中数据出了错，解密时，会使得相对应的整个明文分组解密错误，但它不会影响其它明文。然而，如果密文中偶尔丢失或添加一些数据位，那么整个密文序列将不能正确的解密，除非有某中帧结构能够重新排列分组的边界。

填充

大多数消息并不是刚好分成 64-比特（或者任意分组长）的加密分组，它们通常在尾部有一个短分组。ECB 要求是 64-比特分组。处理该问题的一个方法是填充。

用一些规则的模式——0、1 或者 0、1 交替一——把最后的分组填充成一个完整的分组。如果你想在解密后将填充位去掉，在最后一分组的最后一字节中加上填充字节的数目。例如，假定分组的大小是 64 比特，且最后一个分组含有 3 字节（24 比特）。也就是说，需要填充 5 字节以使最后一分组达到 64 比特，这时就要添加 4 个字节的 0 然后再用 5 填充最后一个字节。解密后删除最后分组的后面 5 个字节就是了。因为该方法能正确工作，所以每一个消息都必须填充。即使明文以分组的边界结束，也必须添加一个整分组。然而，你可以用一个文件结束字符表示明文的最后一个字节，然后在该字符后面进行填充。

图 9.1 是一个可供选择的方案，称为密文挪用[402]。P_{n-1} 是最后一个完整的明文分组，P_n 是最后一个短的明文分组。C_{n-1} 是最后一个完整的密文分组，C_n 是最后一个短的密文分组。C' 仅作为一个中间结果，并不是传输密文的一部分。

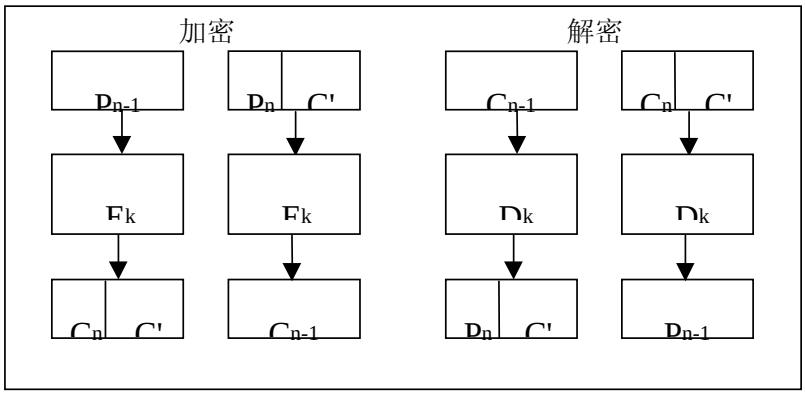

图 9.1　ECB 模式中的密文挪用

9.2　分组重放

ECB 模式最严重的问题是敌人可以在不知道密钥情况下能够修改被加密过的消息，用这种办法可以欺骗指定的接收者。最早在文献[291]中讨论过这个问题。

为了说明这个问题，考察在不同银行的帐号之间银行间的资金往来的资金转帐系统，为

使计算机系统方便有效，银行制定了一个标准的消息格式用来转帐，格式如下：

银行 1：发方	1.5 个分组（数据分组）
银行 2：收方	1.5 个分组
存款者姓名	6 个分组
存款者帐户	2 个分组
存款金额	1 个分组

上述的一个“分组”对应着一个 8-字节的加密分组。这些消息用 ECB 模式下的某个分组算法进行加密。

Mallory 正在窃听银行间（如 Alice 所在的银行与 Bob 所在的银行）的通信线路，他可以利用这些信息致富。首先，他用自己的计算机记录下所有从 Alice 银行到 Bob 银行的加密消息，接着他从 Alice 的银行传送 100 美元到他在 Bob 的银行的帐户上。然后，他又重复该过程。利用计算机记录下的信息，他可以找到一对完全相同的消息，这些消息就是授权将 100 美元转到他的帐户上。如果发现有多于一对的消息相同（实际生活中更有可能），他就作另一个款项转移，并将结果记录下来，最终他能分离出授权他的款项转移的消息。

现在他可以按照他的意愿在通讯链路中插入消息，每次他给 Bob 银行发送一则消息，就有 100 美元进入他的帐户，当两个银行核对他们的往来帐目（总有那么一天），他们将注意到幽灵般的转款授权现象。但如果 Mallory 聪明的话，他早就将款取出，并逃到没有引渡法律的中南美洲国家去了，并且他可在许多不同的银行用同样的手法每次搞到远多于 100 美元的款项。

乍看一眼，就可以发现银行可以通过在消息中附加时间邮戳来防止这种情况的发生。

日期/时间邮戳	1 分组
银行 1：发方	1.5 分组
银行 2：收方	1.5 分组
存款者姓名	6 分组
存款者帐户	2 分组
存款金额	1 分组

用这种系统可以很容易识别两段完全相同的消息。尽管如此，使用一种叫分组重放的技术，Mallory 仍然可以发财。如图 9.2 所示，Mallory 可以挑选 8 个与他的名字和帐号相对应的密文分组：分组 5 到分组 12。这时一阵恶魔般的笑声传来，Mallory 已作好了准备。

他截取从银行 1 到银行 2 的随机消息，并用他的名字和帐号替代分组 5 和分组 12 间相应的比特消息，然后将其发往银行 2。他不必知道原先的存户；甚至不必知道那个帐号（虽然他可以比较他窜改后存入他帐户金额的消息，来确定对应于同样金额的加密分组），他只需简单地将姓名和帐号换上他自己的，然后查看他的进帐就行了。

银行将花费一天以上的时间才能发现，当他们每天核对帐目时，一切都会正常，直到可能某一天合法客户注意到钱最终没存进他的帐户，或什么时候某人意外地发现 Mallory 的帐户出奇地活跃，银行才会发现问题。Mallory 并不蠢，到那时他将取消他的帐户，改名换姓在阿根延买了一幢别墅。

银行可通过频繁的改变密钥，尽可能地降低风险。但这只是意味着 Mallory 的行动要更

加迅速。然而，增加一个 MAC 字段可解决此问题。即使这样，这仍然是 ECB 模式的根本性问题。Mallory 仍可以按自己的意愿或删除或重复或改换密文分组。该问题的解决方法是采用称为分组链接的技术。

分组号

1	2	3	4	5	6	7	8	9	10	11	12	13
时间标记	发送银行	接收银行		储户名						储户帐号		存款-额

字段

图 9.2　一个记录实例的加密分组

9.3　密码分组链接模式

链接将一种反馈机制加进分组密码中：前一个分组的加密结果被反馈到当前分组的加密中，换句话说，每一分组被用来修改下一分组的加密。每个密文分组不仅依赖于产生它的明文分组，而且依赖于所有前面的明文分组。

在密码分组链接（CBC）模式中，明文被加密之前要与前面的密文进行异或运算。图 9.3（a）展示了分组链接是如何工作的，第一个分组明文被加密后，其结果也被存在反馈寄存器中，在下一明文分组加密之前，它将与反馈寄存器进行异或作为下一次加密的输入，其结果又被存进反馈寄存器，再与下一分组明文进行异或，如此这般直到消息结束。每一分组的加密都依赖于所有前面的分组。

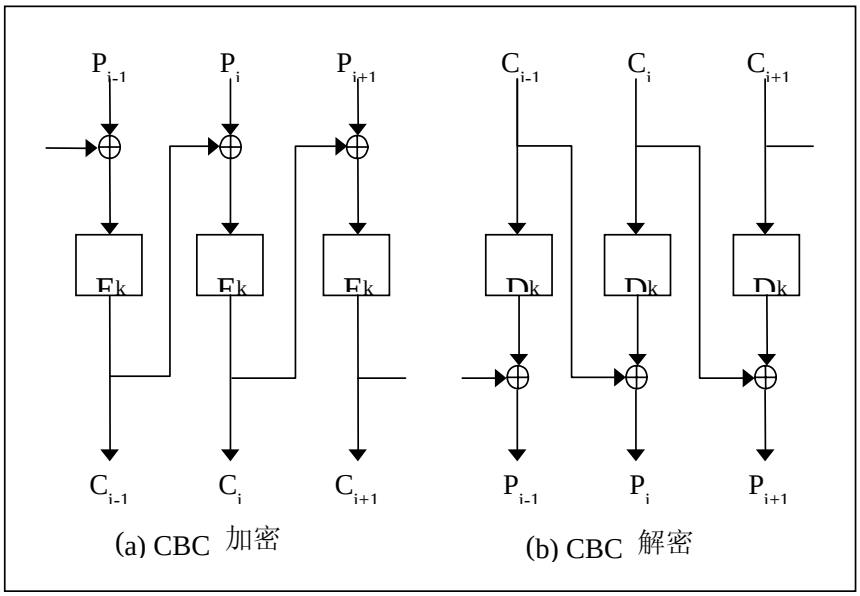

图 9.3　密码分组链接模式

解密一样简单易行（见图 9.3b）。第一个分组密文被正常的解密，并将该密文存入反馈寄存器，在下一分组被解密后，将它与寄存器中的结果进行异或。接着下一个分组的密文被

存入反馈寄存器，如此下去直到整个消息结束。

用数学语言表示为：

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

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

初始向量

CBC 模式仅在前面的明文分组不同时才能将完全相同的明文分组加密成不同的密文分组，因此两个相同的消息仍将加密成相同的密文。更糟糕的是，任意两则消息在它们的第一个不同之处出现前，将被加密成同样的结果。

一些消息有相同的开头：如，一封信的信头，“发件人”行，或其它东西。虽然使用分组重放是不可能的，但这些相同的开头的确给密码分析者提供了一些有用的线索。

防止这种情况发生的办法是用加密随机数据作为第一个分组，这个随机数据分组被称之为“初始化向量（IV），“初始化变量”，或“初始连接值”。IV 没有任何意义，它只是使每个消息唯一化。当接收者进行解密时，只是用它来填充反馈寄存器，然后将忽略它。时间邮戳是一个好的 IV，当然也可以用一些随机比特串作为 IV。

使用 IV 后，完全相同的消息可以被加密成不同的密文消息。这样，偷听者企图再用分组重放进行攻击是完全不可能的，并且制造密码本将更加困难。尽管要求用同一个密钥加密的消息所使用的 IV 是唯一的，但这也不是绝对的。

IV 不需要保密，它可以明文形式与密文一起传送。如果觉得这样错了，那就看一下如下讨论：假设我们有一个消息的各个分组，B_1、B_2、….B_i，B_1用 IV 加密，B_2使用 B_1的密文作为 IV 进行加密，B_3用 B_2的密文作为 IV 进行加密，如此类推。所以，如果有 n 个分组，即使第一个 IV 是保密的，那仍然有 N—1 个“IV”暴露在外。因此没有理由对 IV 进行保密；它只是一个虚拟密文分组——你可以将它看作链接开始的 B_0 分组。

填充

就象 ECB 模式一样进行填充，但在许多应用中需要使密文与明文有同样的长度。或许明文被加密后就放在内存原来的位置，这样，你必须对最后那一个短分组进行不同的加密处理。假定最后一分组有 j 比特，在对最后一个完整分组加密之后，再将其加密，然后选择密文的最左边 j 比特让其跟不完整分组（短分组）进行异或运算。如图 9.4 所示。

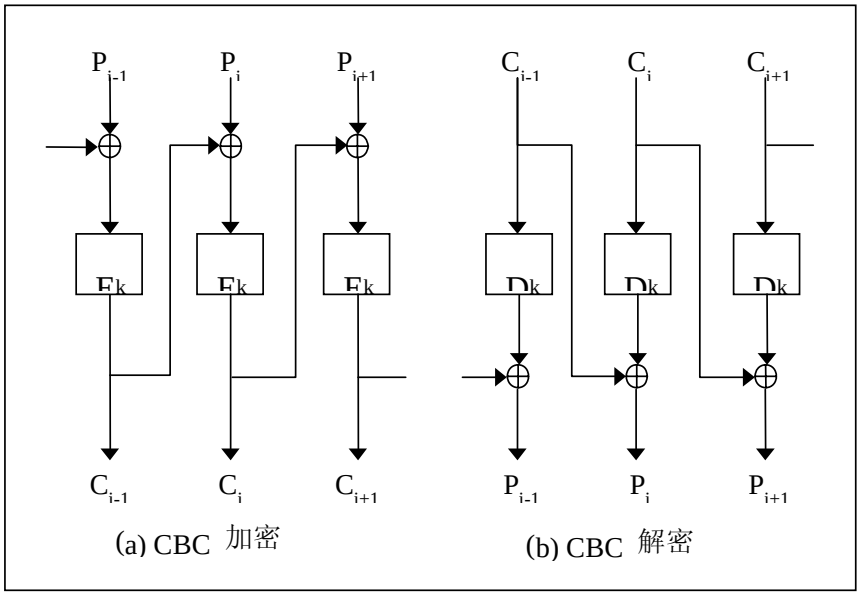

图 9.4　CBC 模式最后短分组的加密方法

这种方法的不足之处是当 Mallory 不能恢复最后明文分组时，他可以通过修改密文的一些个别比特系统地改变它们。如果最后 n 比特密文含有重要信息，这将是一个弱点，如果最后几位只含一些简单的不重要的东西，就无关紧要。

密文挪用是一个很好的方法（参见图 9.5）[402]。P_{n-1} 是最后一个完整的明文分组，P_n 是最后的短明文分组。C_{n-1} 是最后一个完整的密文分组，C_n 是最后短的密文分组。C' 是一个中间结果并非传送密文的一部分。该方法的好处是明文消息的所有位都通过了加密算法。

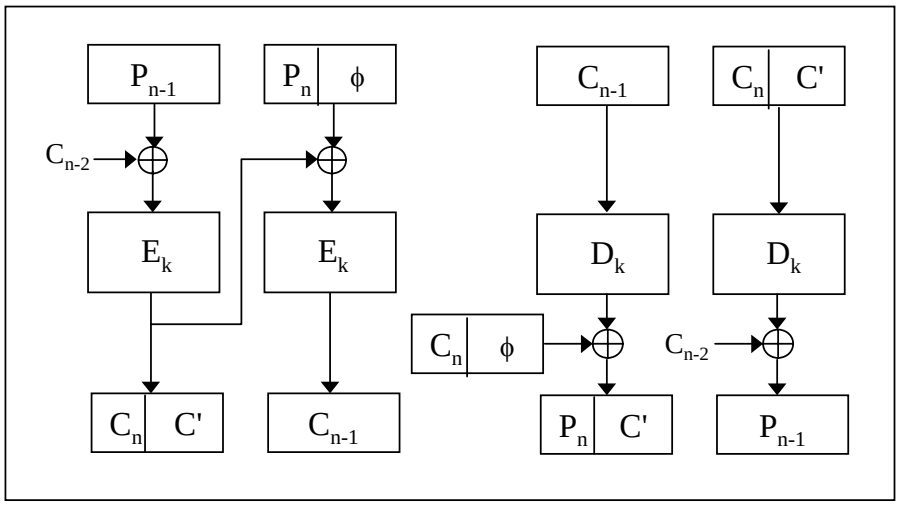

图 9.5　CBC 模式下的密文挪用

错误扩散

CBC 具有在加密端是密文反馈和解密端是密文前馈的性质，这意味着要对错误进行处理。明文分组中单独一位发生错误将影响密文分组以及其后的所有密文分组。这没什么大不了，因为解密将反转这种影响，恢复的明文也还是那个错误。

密文错误更加常见。信道噪音或存储介质损坏很容易引起这些错误。在 CBC 模式中，密文中一个单独比特的错误将影响一个分组以及恢复明文的 1 比特错误。含有 1 比特错误的分组是完全不能恢复，随后的分组在同样的位置有 1 比特的错误。

密文的小错误能够转变成明文很大的错误，这种现象叫做错误扩散。它是最烦人的事。错误分组的第二分组之后的分组不受错误影响，所以 CBC 模式是“自己恢复的”。虽然两个分组受到一个错误的影响，但系统可以恢复并且对所有后面的分组都不受影响。CBC 是用于自同步方式的分组密码算法的一个实例，但仅在分组级。

尽管 CBC 很快能将比特错误恢复，但它却不能恢复同步错误。如果从密文流中增加或丢失 1 比特，那么所有后续分组要移动 1 比特，并且解密将全部是错误的。任何使用 CBC 的加密系统都必须确保分组结构的完整，要么用“帧”，要么在有多个分组大小的内存分组中存储。

安全问题

一些潜在的安全问题是由 CBC 的结构引起的。首先，因为密码分组都是用同样的方式影响后面的分组，所以 Mallory 可以在加密消息的后面加上一些分组而不被发觉。当然，它或许被解密成一堆杂乱的数据，但在很多情况下这是不需要的。

如果你正在使用 CBC，你应当组织好明文使你能知道消息在何处结束，并且能检测出额外附加的分组。

其次，Mallory 可以通过改变一个密文分组，控制其余解密的明文分组。例如，如果 Mallory 切换一个密文位，那么就使得整个密文分组不能被正确解密，但紧接的分组在相应的同一位置出现 1 比特错误。在很多情况下这是所需要的。整个明文消息应当包括某些控制冗余或鉴别。

最后，尽管通过链接把明文的模式被隐藏起来了，但很长的消息仍然有其模式。由生日悖论可以预知 $2^{m/2}$ 个分组后就有完全相同的分组，其中 m 为分组的大小。对一个 64-比特的分组，也就是约 34G 字节。在消息必须足够长时，才有这样的问题。

9.4 序列密码算法

序列密码算法将明文逐位转换成密文 。该算法最简单的应用如图 9.6 所示。密钥流发生器（也称为滚动密钥发生器）输出一系列比特流：K_1，K_2，K_3，……K_i 。密钥流（也称为滚动密钥）跟明文比特流，P_1，P_2，P_3，……P_i ，进行异或运算产生密文比特流。

$$C_i = P_i \oplus K_i$$

在解密端，密文流与完全相同的密钥流异或运算恢复出明文流。

$$P_i = C_i \oplus K_i$$

由于

$$P_i \oplus K_i \oplus K_i = P_i$$

所以该方式是正确的。

系统的安全性完全依靠密钥流发生器的内部机制。如果它的输出是无穷无尽的“0”序列，那么密文就是明文，这样整个系统一文不值。如果它吐出的是一个重复性的 16 比特模式，那么该算法仅是一个可忽略安全性的异或运算（参见 1.4 节）。如果是一系列无尽的随机流（是真正的随机，不是伪随机——见 2.8 节）那就有一次一密乱码本和非常完美的安全。

实际的序列密码算法其安全性依赖于简单的异或运算和一次一密乱码本。密钥流发生器生成的看似随机的密钥流实际上是确定的，在解密的时候能很好的将其再现。密钥流发生器输出的密钥越接近随机，对密码分析者来说就越困难。

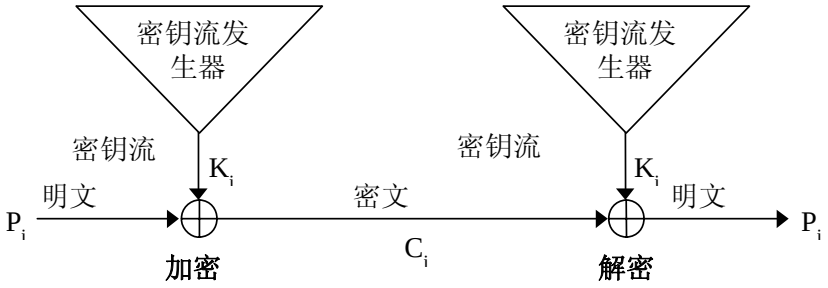

图 9.6　序列密码

然而，如果密钥流发生器每次都生成同样的密钥流的话，对攻击来说，破译该算法就容易了。举个实例看一下为什么。

如果 Eve 得到一份密文和相应的明文，她就可以将两者异或恢复出密钥流。或者，如果她有两个用同一个密钥流加密的密文，她就可以让两者异或得到两个明文互相异或而成的消息。这是很容易破译的，接着她就可以用明文跟密文异或得出密钥流。

现在，无论她再拦截到什么密文消息，她都可以用她所拥有的密钥流进行解密。另外，她还可以解密，并阅读以前截获到的消息。一旦 Eve 得到一明文/密文对，她就可以读懂任何东西了。

这就是为什么所有序列密码也有密钥的原因。密钥流发生器的输出是密钥的函数。这样，Eve 有一个明文/密文对，但她只能读到用特定密钥加密的消息。更换密钥，攻击者就不得不重新分析。序列密码算法对加密那些永不结束的通信数据流是特别有用的：如，两台计算机之间的 T-1 连接。

密钥流发生器有三个基本组成部分（见图 9.7）。内部状态描述了密钥流发生器的当前状态。两台密钥流发生器如果有相同的密钥和内部状态，那么就会产生相同的密钥流。输出函数处理内部状态，并产生密钥流；下个状态函数处理内部状态，并生成新的内部状态。

9.5　自同步序列密码

自同步序列密码就是密钥流的每一位是前面固定数量密文位的函数[1378]。军方称为密

文自动密钥（CTAK）。其基本思想是在 1946 年成形的[667]。

图 9.8 描述了其工作原理。其中，内部状态是前面 n 比特密文的函数。该算法的密码复杂性在于输出函数，它收到内部状态后生成密钥序列位。

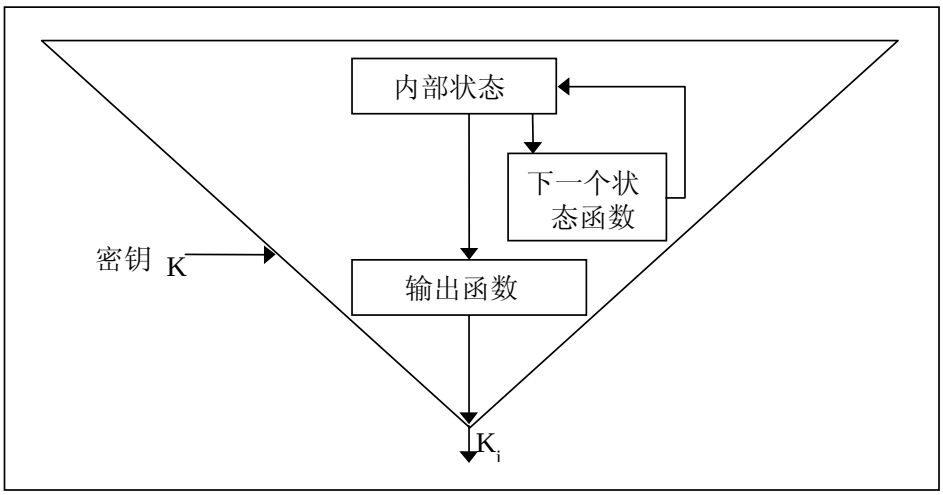

图 9.7　　密钥流发生器内部机理

因为内部状态完全依赖前面 n 个密文位，所以解密密钥流发生器在收到 n 个密文位后自动跟加密密钥流发生器同步。

在该模式的智能化应用中，每个消息都以随机的 n 位报头开始。这个报头被加密、传输、解密，在 n 位密文之前整个解密是不正确的，直到之后两个密钥流发生器同步。

自同步密码的缺点是错误扩散。对传输中每一个密文位被窜改，解密密钥流发生器就有 n 位密钥流位不能正确生成。因此，一位密文错误就会导致 n 位相应的明文错误，直到内部状态里面不再有该错误位。

安全问题

自同步序列密码算法同样对回放攻击很敏感。Mallory 先记录下一些密文位。接着，一段时间后他就用这些记录代替当前数据流。在一些初始位过后，当收端重新同步时，一些旧的密文仍正常的解密。接收端没有办法知道它是不是当前数据，但旧的数据则可以被回放。如果不用时间邮戳，Mallory 就可以回放相同的信息（当然假定密钥还没被更换）使银行相信并将大笔的钱一遍又一遍的存入自己帐户。在频繁再同步情况下，该方案的还有其它可利用的弱点[408]。

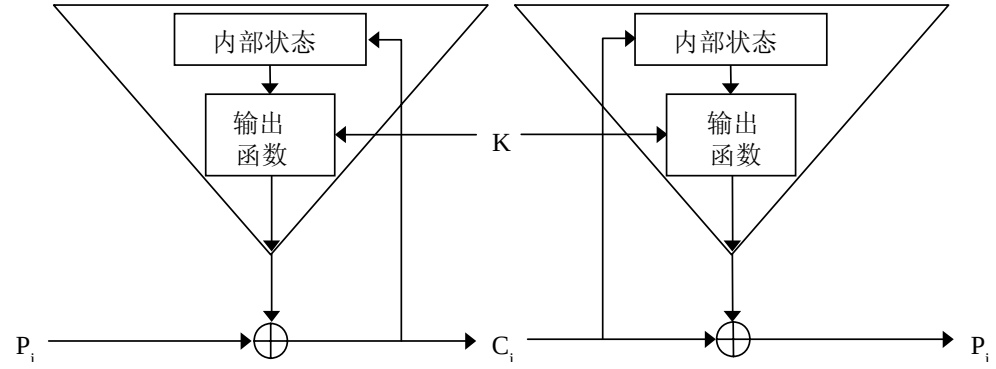

图 9.8　自同步密钥流发生器

9.6　密码反馈模式

分组密码算法也可以用于自同步序列密码，就是所谓的密码反馈模式（CFB）。在 CBC 模式下，整个数据分组在接收完之后才能进行加密。对许多网络应用来说，这是一个问题。例如，在一个安全的网络环境中，当从某个终端输入时，它必须把每一个字符马上传给主机。当数据在一字节大小的分组里进行处理时，CBC 模式就不能做到了。

在 CFB 模式下，数据可以在比分组小得多的单元里进行加密。下面这个例子就是一次加密一个 ASCII 字符（称为 8-比特 CFB），这里数字“8”没任何特殊性，你可以用 1-比特 CFB 一次加密一位数据。尽管用完整的分组加密算法对单独一位进行加密好象也能工作，但用序列密码算法更好。（并不提倡利用减少分组的大小来加快速度[1269]）你也可以使用 64-比特 CFB 或者任意 n-比特 CFB（其中 n 小于或等于分组大小）。

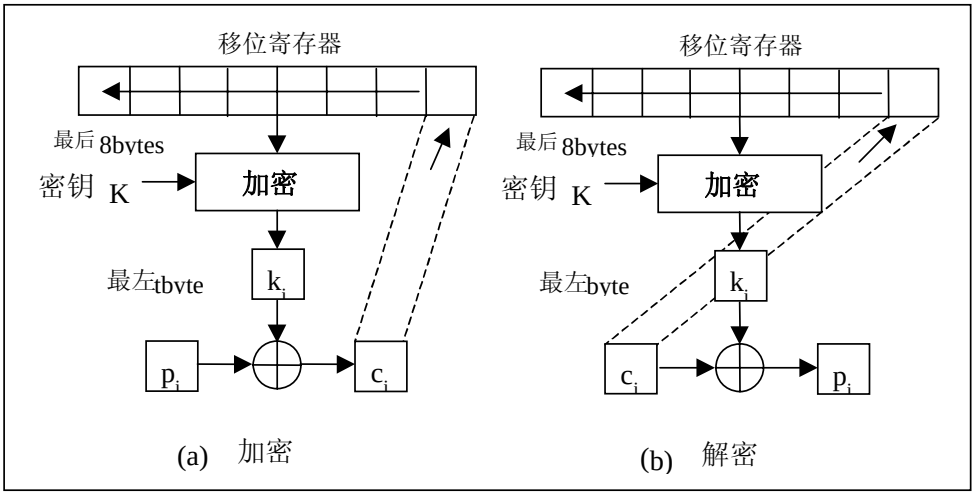

图 9.9　8 比特密码反馈方式

图 9.9 说明了 64-比特分组算法下的 8-比特 CFB 模式的工作原理。CFB 模式下的分组算法对输入分组大小的队列进行操作。开始，该队列就象在 CBC 模式下一样用一个 IV 填充。

整个队列被加密取其最左面 8 位与明文第一个 8 比特字符进行异或得到密文的第一个 8 比特字符。这个字符现在就可以传输了，同时该字符被移动到队列的最右边字节位置，然后其它字节向左移动 8 位。最左面 8 比特丢弃。其它明文字符如法炮制。解密是一个逆过程。在加密解密两端，分组算法用于其加密模式中。

如果算法的分组是 n 比特，那么 n 比特 CFB 就象（见图 9.10）：

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

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

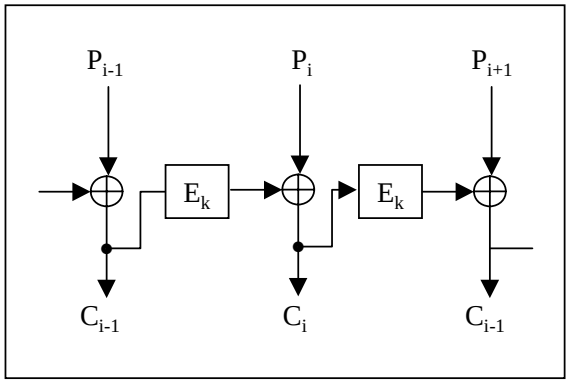

图 9.10 n-比特分组算法下的 n-比特 CFB 模式

象 CBC 模式，CFB 模式将明文字符连接起来以使密文依赖所有以前的明文。

初始化向量

为了初始化 CFB 过程，分组算法的输入必须用 IV 初始化。就象在 CBC 模式使用 IV 一样，它并不需要保密。

尽管那样，IV 必须是唯一的。（这是与 CBC 模式不同的地方，CBC 模式中 IV 应该唯一但不是必须。）如果在 CFB 模式下 IV 不是唯一的，密码分析者就可以恢复出相应的明文。对不同的消息，IV 必须更换，它可以是一系列号，每一个消息后增大，以保证在密钥有效期内不会重复。对用于存储的信息加密，它可以是用来查找数据指针的函数。

错误扩散

CFB 模式中，明文的一个错误就会影响所有后面的密文以及在解密过程中的逆。密文出现错误就更有趣了：首先，密文里单独一位的错误会引起明文的一个单独错误。除此之外，错误进入移位寄存器，导致密文变成无用的信息，直到该错误从移出寄存器的另一端移出。在 8-比特 CFB 模式中，密文中一比特的错误会使加密明文产生 9 字节的错误。之后，系统恢复正常，后面的密文也被重新正确解密。通常情况下，在 n-比特 CFB 模式中单独的密文错误会影响当前和紧跟的 m/n-1 分组的解密，其中 m 是分组大小。

该类错误扩散的一个严重问题是如果 Mallory 熟悉某个正在传输的明文，他就可以窜

改某个分组里的某些位，使它们解密成自己想要的信息。“下一分组”会被解密成“垃圾”，但破坏已经发生了。他还可以更改消息的最后一些位而不被发现。

CFB 模式对同步错误来说同样是可以自我恢复的。错误位进入移动寄存器就可以使 8 字节的数据毁坏，直到它从另一端移出寄存器为止。CFB 是分组密码算法用于自同步序列密码算法的一个实例（分组级）。

9.7 同步序列密码

在同步序列密码中密钥流是独立于消息流而产生的。军方称之为密钥自动密钥（KAK）。加密端密钥流发生器一位接一位地“吐出”密钥，在解密端另一个发生器产生出完全相同的密钥。两个密钥发生器同步以后，这种一致就开始了。如果其中一个发生器跳过一个周期或者一个密文位在传输过程中丢失了，那么错误后面的每一个密文字符都不能正确解密。

如果错误不幸发生了，发方和收方就必须在继续进行之前使两个密钥发生器重新同步。他们必须这样做，以保证密钥流的任意部分不会重复，重新设置发生器回到前一个状态，这个简单的法子是不行的。

好的一面，同步密码并不扩散传输错误。如果有一位在传输中改变了，比丢失一位可能性大的多，那么只有该位不能正确解密。所有进程和结果都不会受影响。

由于在加解密两端密钥流发生器必须产生同样的输出，所以它必须是确定的。因为它是用限状态机器实现的（如计算机），密钥序列终会重复。这些密钥流发生器被称为周期性的。除一次一密乱码本外，所有密钥流发生器都是周期性的。

发生器的周期必须非常长，要比密钥更换之前发生器所能输出的位的长度还要长得多。如果其周期比明文还要短，那么明文的不同部分将用同样的加密——这是一个严重的弱点。如果密码分析者熟悉这样的一批明文，他就可以恢复出密钥流，然后恢复出更多的明文。即使分析者仅有密文，他也可以用同一密钥流加密的不同部分密文相异或得到明文跟明文的异或。这只是一个有非常长密钥的单一异或运算罢了。

周期需要多长取决于应用。用于加密连续 T-1 连接通信的密钥发生器每天加密 2^{37} 比特。那么它的周期应该比这个数大几个数量级，尽管密钥每天都要更换。如果周期足够长，你仅仅需要每周甚至每月才更换密钥。

同步序列密码同样可防止密文中的插入和删除，因为它们会使系统失去同步而立即被发现。然而，却不能避免单个位被窜改。就象 CFB 模式下的分组密码算法，Mallory 更换数据流中的某个比特，如果他熟悉明文，他就可以使那些比特被解密成他想要的。后面的比特仍被正确的解密，所以在很多应用中 Mallory 仍可进行某些毁坏。

插入攻击

同步序列密码对插入攻击非常敏感[93]。Mallory 作了一些密文记录，但他并不知道明文或用来加密明文的密钥流。

源明文： p_1 p_2 p_3 p_4……

源密钥流： k_1 k_2 k_3 k_4……

源密文： c_1 c_2 c_3 c_4……

Mallory 在明文 p_1 后面插入一个单独的已知位 p'，他能够使修改后的明文被相同的密钥流加密。他记录下新的密文：

新明文： p_1 p' p_2 p_3 p_4……

源密钥流： k_1 k_2 k_3 k_4 k_5……

更新的密文： c_1 c_2' c_3' c_4' c_5' ……

假定他知道 p' 的值，他可以根据原始密文和新的密文确定整个明文：

由 $k_2 = c_2' \oplus p'$ 得到 $p_2 = c_2 \oplus k_2$

由 $k_3 = c_3' \oplus p_2$ 得到 $p_3 = c_3 \oplus k_3$

由 $k_4 = c_4' \oplus p_3$ 得到 $p_4 = c_4 \oplus k_4$

Mallory 并不需要知道插入位的确切位置，他只用比较一下原始密文和更新后的密文从哪个地方不同就是了。为了防止这种攻击，永远不要使用同一个密钥流加密两个不同的消息。

9.8 输出反馈模式

“输出反馈（OFB）”模式是运行分组密码作为同步序列密码算法的一种方法。它与密码反馈模式相似，而 OFB 是将前一个 n-比特输出分组送入队列最右边位置（见图 9.11）。解密是一个逆过程。称它为 n-比特 OFB，在加解密两边，分组算法都以加密模式使用。这种方法有时也叫“内部反馈”，因为反馈机制独立于明文和密文而存在的[291]。

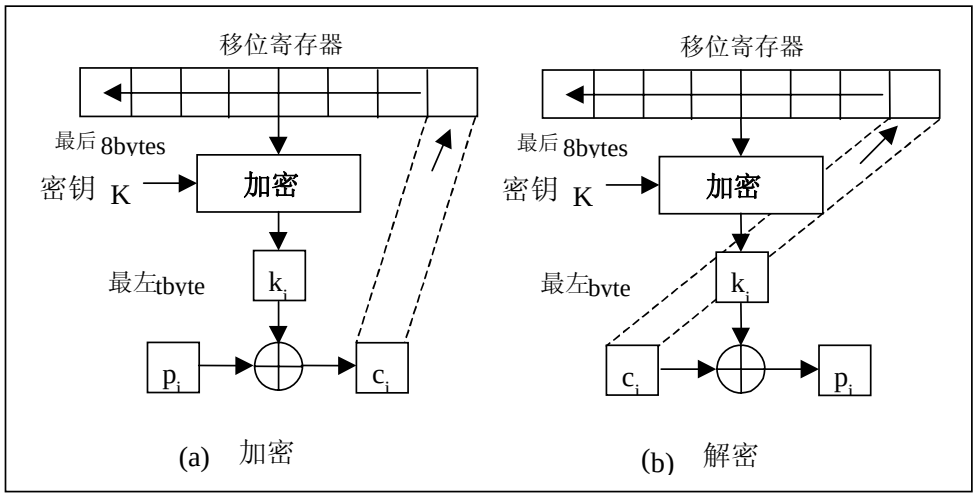

图 9.118－比特输出反馈方式

如果 n 是该算法分组的大小，那么 n-比特 OFB 看上去象（参见图 9.12）：

$C_i = P_i \oplus S_i$ ； $S_i = E_k(S_{i-1})$

$P_i = C_i \oplus S_i$ ； $S_i = E_k(S_{i-1})$

S_i 是状态，它独立于明文和密文。

OFB 模式有一个很好的特性就是大部分工作可以离线进行，甚至在明文存在之前。当消息最终到达时，它可以与算法的输出相异或产生密文。

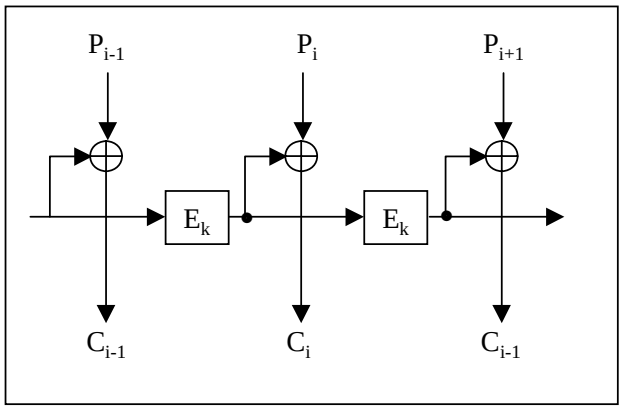

图 9.12　8-比特 OFB

初始化向量

OFB 移位寄存器也必须装入 IV 初始化矢量， IV 应当唯一但不须保密。

错误扩散

OFB 模式没有错误扩散。密文中单个比特的错误只引起恢复明文的单个错误，这点对一些数字化模拟传输非常有用，象数字化声音或视频，这些场合可以容忍单比特错误，但不能容忍扩散错误。

另一方面，失步是致命的。如果加密端和解密端移位寄存器不同，那么恢复的明文将是一些无用的杂乱数据，任何使用 OFB 的系统必须有检测失步的机制，和用新的（或同一个）IV 填充双方移位寄存器重新获得同步的机制。

关于 OFB 的安全性

OFB 模式的安全分析[588，430，431，789]表明，OFB 模式仅当反馈量大小跟分组大小相同时才有用。例如，在 64-比特 OFB 模式中你只能用 64-比特分组算法。即使美国政府授权在 DES[1143]中使用其他大小的反馈，但应尽量避免。

OFB 模式将密钥流与明文异或。密钥流最终会重复。对同一个密钥使密钥流不重复是很重要的，否则，就毫无安全可言。当反馈大小与分组大小相同时，分组密码算法起到 m-比特数值置换（m 是分组长度）的作用，并且平均周期长度为 2^m-1，对 64-比特的分组长度，这是一个很大的数。当反馈大小 n 小于分组大小时，平均周期长度将降到约 $2^{m/2}$。对 64-比特分组算法，就是 2^{32}——不够长。

OFB 模式下的序列密码

用 OFB 模式也能产生序列密码。在这种情况下，密钥影响下一状态函数（见图 9.13）。输出函数并不依赖密钥；它经常简单地使用内部状态的某个单独位或多位的异或值。密码复杂性在于下一状态函数，该函数依赖于密钥。这种方法被称为内部反馈[291]，因为对密钥产生算法来说，反馈机制存在于内部。

在该模式的一个变体中，密钥只决定密钥流产生器的初始状态。在密钥设置好产生器的

内部状态后，产生器就不再受到干扰。

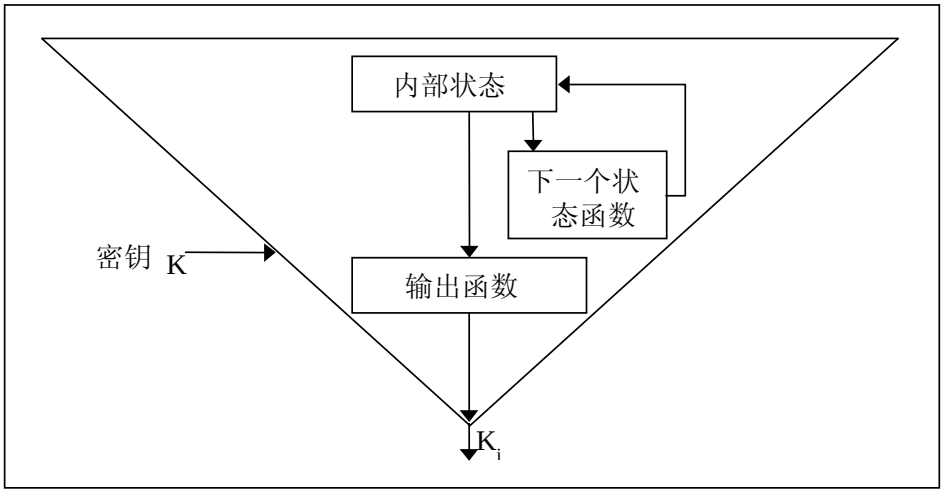

图9.7　　输出反馈方式中的密钥流发生

9.9 计数器模式

计数模式下的分组密码算法使用序列号作为算法的输入[824，498，715]。不是用加密算法的输出填充寄存器，而是将一个计数器输入到寄存器中。每一个分组完成加密后，计数器都要增加某个常数，典型值是 1。该模式的同步和错误扩散特性同 OFB 模式完全一样。计数模式解决了 OFB 模式小于分组长度的 n-比特输出问题。

没有什么是专供计数器用的，它不必根据可能的输入计数。你可以使用 16、17 章讲到的如何随机序列发生器作为分组算法的输入，而不管其密码上是否安全。

计数模式中的序列密码

计数模式中的序列密码算法有简单的下一状态函数和复杂的依赖于密钥的输出函数。这种技术，如图 9.14 所示，在文献[498，715]中提出的。下一状态函数可以是跟计数器一样简单的东西，只要在前一状态上加 1 就行。

使用计数模式序列密码算法，不用先生成前面所有的密钥位，就可直接生成第 i 个密钥比特 k_i 。简单的手工设置计数器到第 i 个内部状态，然后产生该比特。这在保密随机访问数据文件时是非常有用。你不用解密整个文件就可以直接解密某个特殊数据分组。

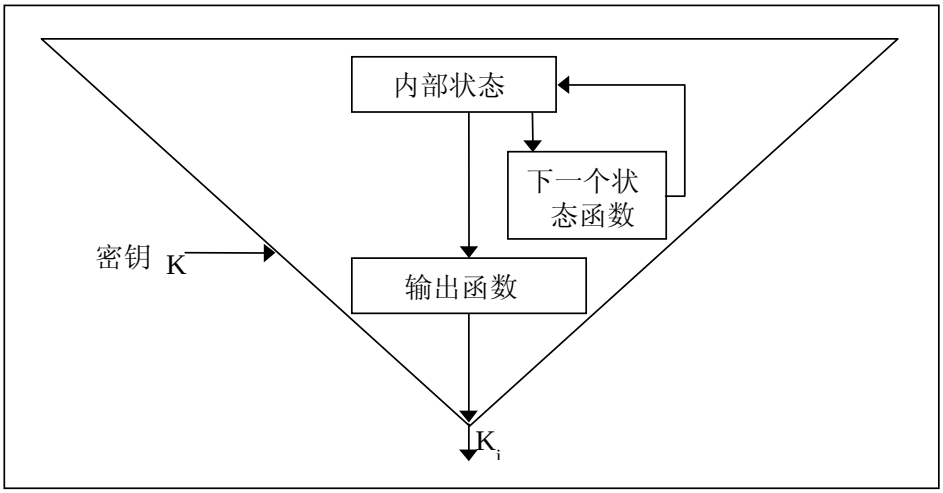

图 9.14　计数模式下的密钥流发生器

9.10　其他分组密码模式

分组链接模式

为了在分组链接（BC）模式中使用分组算法，可以简单地将分组密码算法的输入跟所有前面密文分组的异或值相异或。就象 CBC 算法一样，过程要从一个初始向量 IV 开始。

数学式表达如下：

$C_i = E_K(P_i \oplus F_i)$；　$F_{i+1} = F_i \oplus C_i$

$P_i = F_i \oplus D_K(C_i)$；　$F_{i+1} = F_i \oplus C_i$

象 CBC，BC 模式的反馈过程具有扩散明文错误的性质，BC 的这个问题是由于密文分组的解密依赖于所有前面的密文分组而引起的，密文中单一的错误都将导致所有后续密文分组在解密中出错。

扩散密码分组链接方式

“扩散密码分组链接（PCBC）”[1080]模式与 CBC 模式相似，只是它在加密前，前面的明文分组、密文分组（或解密后）都与当前明文分组相异或（见图 9.15）。

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

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

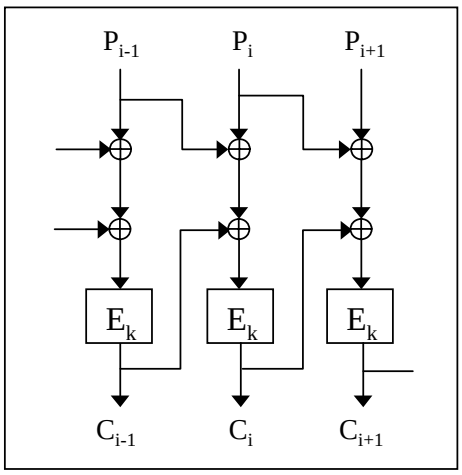

图 9.15　扩散密码分组链接方式

PCBC 被用于 Kerberos 版本 4（见 24.5 节）进行加密，并在一次传递中完成加密和完整性检查。在 PCBC 模式中，密文分组的一个错误将引起所有后续分组在解密时产生错误，这意味着检验消息尾的一个标准分组将能确保整个消息的完整性。

不幸的是这个模式有一个问题[875]，交换两个密文分组，将使两个对应的明文分组不能正确解密，但根据明文和密文异或的性质，错误将抵消。所以，如果完整性检查只检查最后几个解密的明文分组，它可能欺骗接收者接收部分错误的消息。尽管现在还没有人对这个弱点进行“开发使用”，但 Kerberos 版本 5 还是在发现上述缺点后以 CBC 模式取代了它。

带校验和的密码分组链接

带校验和的密码分组链接（CBCC）是 CBC 的一个变体[1618]。该模式保存所有明文分组的异或，并在加密前与最后的明文分组异或。CBCC 保证任何对密文的改动都将引起最后分组解密输出的改动。如果最后分组包含某种完整校验或常数，那么用很小的额外操作就能检验解密明文的完整性。

带非线性函数的输出反馈

带非线性函数的输出反馈（OFBNLF）[777]是 OFB 和 ECB 的一个变体，它的密钥随每一个分组而改变：

$$C_i = E_{Ki}(P_i) \quad ; \quad K_i = E_K(K_{i-1})$$

$$P_i = D_{Ki}(C_i) \quad ; \quad K_i = E_K(K_{i-1})$$

密文的一个比特错误扩散到一个明文分组。然而，如果一位丢失或增加，那就有无限的错误扩散。使用一个有复杂的密钥编排算法的分组算法（如 DES），该模式是很慢的。据我所知还没有对该模式的密码分析。

更多的模式

可能还有其它的模式，虽然它们并没有被广泛应用。“明文分组链接（PBC）”模式与 CBC 相似，只是前一明文分组与当前明文分组进行异或，而不是与密文分组异或，明文反

馈（PFB）与 CFB 相似，只是明文而非密文用于反馈。为了抵制已知明文攻击，这两中模式准许采用选择明文攻击。还有“明文差分密文分组链接（CBCPD）”模式，我肯定它会变得更加离奇。

如果密码分析者有一台穷举密钥搜索机器，如果能猜出一个明文分组的话，那么就能恢复出密钥。在使用加密算法之前利用一些陌生的模式对加密可以起到奇妙的作用：如将文本与一确定秘密串进行异或，或者对文本进行置换。几乎任何非标准的东西都将有助于挫败这类密码分析。

9.11 选择密码模式

如果你所关心的主要是简单和速度的话，ECB 是最简单和最快的分组密码的模式，当然也是最弱的。除了容易受到重放攻击外，ECB 模式中的算法也是最易分析的。建议不要使用 ECB 作为信息加密。

加密随机数据，如别的密钥，ECB 是一个很好的模式。由于数据短而随机，对这种应用 ECB 几乎没什么缺点。

对一般的明文，请使用 CBC，CFB 或 OFB 模式。所选择的模式依赖于你的特殊需要。表 9.1 列出了各种模式的安全性和效率。

表 9.1 分组密码模式一览表

ECB: **安全性：** —不能隐藏明文模式 —分组密码的输入并不是随机的，它与明文一样 + 一个密钥可以加密一个或多个消息 —明文很容易窜改，分组可被删除，再现或互换 **效率：** + 速度跟分组密码一样 —由于填充，密文比明文长 —不可能进行预处理 + 处理过程并行进行 **容错性：** ——一个密文错误会影响整个明文分组 —同步错误不可恢复	CBC： **安全性：** + 通过跟前一个密文分组相异或明文模式被隐藏 + 与前一个密文分组异或后分组密码的输入是随机的 + 用同一个密钥可以加密多个消息 +/— 篡改明文稍有点难度；分组可以被从消息头和尾处删除，第一块分组的数据可被更换，并且复制允许控制的改变 **效率：** + 速度同分组密码一样 —密文比明文长，不计算 IV —不能进行预处理 +/— 加密不是并行的，解密是并行的且有随机存取特性 **容错性：** ——一个密文错误会影响正个明文分组以及下一个分组的相应位

	—同步错误不可恢复
CFB： **安全性：** + 可以隐藏明文模式 + 密文块的输入是随机的 + 假设用不同的 IV ，同一个密钥可加密多个消息 +/-对明文的篡改稍难；分组可以被从消息头和尾处删除，第一块分组的数据可被更换，并且复制允许控制的改变 **效率：** + 速度同分组密码相同 - 密文与明文同大小，不计算 IV +/- 加密不是并行的，解密是并行的且有随机存取特性 - 在分组出现之前作些预处理是可能的，前面的密文分组可以被加密 **容错性：** - 一个密文错误会影响明文的相应位及下一整个分组 + 同步错误是可恢复的，1-比特 CFB 能够恢复单独位的添加或丢失	OFB/Counter： **安全性：** + 明文模式被隐藏 + 密文分组的输入是随机的 + 用不同的 IV，同一个密钥可以加密多个消息 - 明文很容易被控制篡改，任何对密文的改变都会直接影响明文 **效率：** + 速度同分组密码一样 - 密文跟明文有同样大小，不计算 IV + 消息出现前作些预处理是可能的 -/+ OFB 处理过程不是并行的，计数器处理是并行的 **容错性：** + 一个密文错误仅影响明文的响应位 - 同步错误不可恢复

CBC 最好用来加密文件。安全性增加是很有意义的；并且当存储数据中有某些错误位时，同步错误几乎从不发生。如果你的应用是基于软件的，CBC 总是最好的选择。

CFB——特别是 8-比特 CFB——通常是加密字符流所选择的模式，此时每个字符需要分别对待处理，比如在终端和主机链路中。OFB 通常用在不能容忍错误扩散的高速同步系统中。如果需要预处理那么 OFB 也是可以选择的模式。

OFB 是在容易出错的环境所选择的模式，因为它没有错误扩散。除了所谓“神奇”模式，ECB、CBC、OFB 和 CFB 四种模式之一几乎能够满足任何应用需要，这些模式既不过分复杂也不会减少系统的安全性。尽管复杂的模式或许能增加安全性，大多数情况下它仅仅是增加复杂性。没有什么神奇模式具有好的错误扩散特性或错误恢复能力。

9.12 交错

大多数模式中，对 1 比特（或分组）的加密依赖于前面的比特（或分组）的加密。这就使得并行处理成为可能。例如，假定有一个工作在 CBC 模式下的加密硬件盒，假设它由 4 块加密芯片组成，其中仅有一块能在任意时间工作，其余的芯片需要得到前一芯片的结果后

才开始工作。

解决方案就是交错多重加密序列（不是 15.1 节和 15.2 节讲到的多重加密）。代替使用单一的 CBC 链，它使用 4 个链。第一、第五及每隔四块的分组使用同一个 IV 的 CBC 模式下加密；第二、第六及每隔四块之后再用另一个 IV 在 CBC 模式下加密，依次类推。总的 IV 要比没有交错时长的多。

可把它想成是用相同的密钥和四个不同的 IV 对四个不同的消息进行加密。所有这些消息也都是相互交错的。

该技巧可用于提高硬件的整体加密速度。如果你有三个加密芯片，每一个加密速度是 33 兆比特每秒，那么你使用交错就可以对一个 100 兆比特每秒的单一信道进行加密。

图 9.16 示出 CFB 模式下三个并行交错的序列加密。这种思想也可以用任意数目的并行序列在 CBC 和 OFB 模式下工作。只是要记住每个序列需要自己的 IV，不能共享。

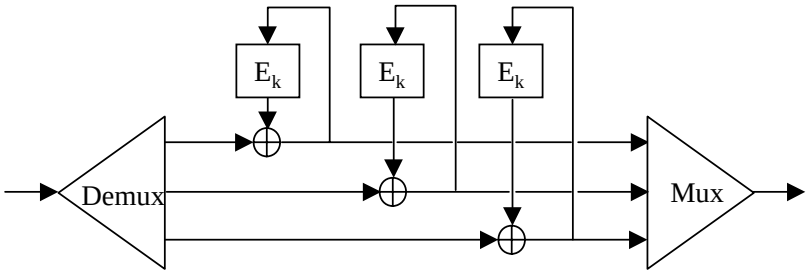

图 9.16　交错的三个 CFB 模式加密

9.13 分组密码算法与序列密码算法

尽管分组和序列密码算法非常不同，但分组密码也可作为序列密码使用，反之亦然。我所见到的对两者最好的区别定义是在文献[1362]中：

> 分组密码算法是对一个大的明文数据块（分组）进行固定变换的操作；
>
> 序列密码算法是对单个明文比特的随时间变换的操作。

在现实中，分组密码似乎更加通用（如，它们能够用四种模式中的任意一种），而序列密码好象更容易用于数学分析。有大量序列密码分析和设计方面的理论著作——由于某些原因，大多数在欧洲完成。自从电子技术发明以来，它们被各国军方所使用。不过，情况好象在改变，近年来，有大量的论文是关于分组密码设计的。或许不久以后，分组密码设计方面的理论就能象现在序列密码设计的理论一样丰富。

然而，两者之间的区别主要体现在实现上。每次只能对一个数据比特进行加解密的序列密码算法并不适用于软件实现。分组密码算法就可以很容易的用软件实现，因为它可以避免耗时的位操作，并且它易于处理计算机界定大小的数据分组。当然另一方面，序列密码更适合用硬件实现，因为使用硅材料可以非常有效的实现它。

这些是值得考虑的事情。对数字通信信道上的硬件加密设备来说，每经过一位就加密一位是非常有意义的。这是这些设备的长处。反之，用软件加密设备加密每一个分离的单个位将没有意义。也有一些特殊的场合：在一个计算机系统中逐位、逐字节加密是必须的——例

如，对键盘和 CPU 之间的连接进行加密——但是通常地讲加密分组至少是数据总线的宽度。

第十章　使用算法

安全性——数据安全性、通信安全性、信息安全性，诸如此类——就是一条链子。整个系统的安全性仅是最脆弱的连接的安全性。每一样东西都必须安全：加密算法、协议、密钥管理，以及更多。如果算法很好，但是随机数发生器非常糟，那么聪明的分析者就能通过该发生器攻击系统。如果你修复了一个漏洞，但忘了安全的删除包含密钥的那块内存，那么分析者通过程序可以攻破你的系统。如果你的什么东西都是正确的，但偶尔将一封含有你安全文件副本的 E-mail 发给了“华尔街日报”，你可能得到同样的结果。

太不公平了。作为一个安全系统的设计者，你必须想到每一个可能的攻击方法及其对策，而一个密码分析者只需找到你的安全漏洞及怎样利用它。

密码学仅是安全性的一部分，甚至经常是很小的一部分。它仅在数学上使一个系统安全，这与实际使系统安全是两码事。密码学有它的“size queens”：人们在花费很多的时间讨论密钥应该有多长时，他们忘记了其它的事情。如果秘密警察想知道你计算机里有什么，那么对他们来说潜入你的房中安置一个摄象机，让它记录你的计算机屏幕总要比分析你的硬驱要容易得多。

此外，对计算机密码学，传统的看法是“间谍与反间谍”技术，并且这种看法越来越不恰当。世界上超过 99%的密码学并没有用来保护军事机密，它们被用于诸如：银行卡、付费电视，道路收费、办公大楼及计算机访问令牌、抽彩设备、预付款电子计量器等[43，44]。在这些应用中，密码的作用就是使卑鄙的犯罪更困难，对那些高额聘请有才能的大量密码分析者和满屋子计算机的攻击者并不适用。

这些应用大部分使用了性能差的密码算法，但是成功的攻击它们与密码分析没有多大关系。他们与受欺骗的雇员、聪明的敲诈行为、愚蠢的实现、频繁地说漏嘴、随便的举止等有关。（我强烈建议上面所述的某些人读一下 Ross Anderson 的论文“密码系统为什么会失效”。），甚至 NSA 也承认，在它关注领域的大多数安全失败是由工作运作错误引起的，而不是算法或协议上的失败[1119]。在这些场合，密码算法再好也没有什么用处，成功的攻击完全可以绕过它。

10.1 选择算法

当开始估计并选择算法时，人们有下面几种选择：

——他们可以选一公开算法，基于相信一个公开算法已经受到许多分析者的攻击，如果还没有人破开它，说明它很好。

——他们可以相信制造者，基于相信一个很有名的制造商不会用他们的名誉冒险去出售具有缺陷算法的设备。

——他们可以相信私人顾问，基于相信一个公正的很有名望的顾问对市场上算法的估计很有见解。

——他们可以相信政府，基于相信政府是最值得信赖的，它不会欺骗其公民。

——他们可以写自己的算法，基于相信他们的算法不次于别人的，并且他们除了自己，不信任任何人。

当然这些选择都有些问题，但第一种选择似乎是最明智的。相信一个制造商、顾问、政府就等于是自找麻烦。那些自称是安全顾问的多数人（甚至来自很有名气的公司），通常连一点加密都不懂。大多数的安全产品制造商也不会更好，NSA 有一大批世界著名的密码学家为他工作，但他们从来未告诉你所有他们知道的，他们所追逐的利益并不与他们公民的利益相一致。即使你是个天才，在得不到同层人（协议上）审查的情况下，使用你自己编写的算法也将是愚蠢的。

本书中的所有算法都是公开的；他们已公开发行并被这方面专家分析过。我列出了所有公布过的结果，其中有正面也有反面的。我并没有接近世界上任何军事安全组织所做的分析（它们也许比学术组织做的好—他们已做了很长时间并有很高的薪水），所以这些算法也许很容易被破译，即使这样，它们也很可能比那些在某个公司的地下室秘密设计和实现的算法更安全。

所有这些理由的漏洞是我们并不知道各种军事密码分析组织的能力。

NSA 可以破译什么样的算法？对我们中的大多数来说，确实无法知道。如果你同一块用 DES 加密的计算机硬盘一起被逮捕的话，在审讯你时，FBI 是不可能引用硬盘的解密明文的；他们能够攻破一个算法这个事实经常比破译出的任何信息更值得保密。二战期间，盟国被禁止使用解密的德国过激的信息，除非能够巧妙的从别的地方获得这些信息。NSA 承认，能够攻破某个给定的算法的唯一方法是加密一个非常有价值的东西，然后公布该加密的消息。或者，可能更好，制造一个真正的滑稽笑话，然后通过加密的 E-mail 把它传给一个不可靠的人。NSA 的雇员同样是人，我对他们能为一个好笑话保住秘密持怀疑态度。

一个好的、可行的假设是 NSA 可以读到它选择的任何信息，而不是它不能读任何它选择的信息。NSA 受资源限制不得不在各种目标中进行挑选。另一个好的假设是他们宁愿破译一些关键部位而不破译整个密码 ；这种偏爱非常强烈，以致于当他们想为他们曾读过该消息保密时，就不得不求助于攻击整个密码。

无论怎么样，我们能够做的最好选择是选公开算法，它们已经历过了公开检验和分析。

算法的出口

美国出口算法必须经美国政府批准（实际上是 NSA——见 25.1 节），广泛认为这些批准输出的算法 NSA 都能破译它。虽然没有人承认它，但谣传 NSA 对那些希望输出他们的密码成果的公司曾私下建议道：

——偶尔泄露一比特密钥，将其藏在密文中。

——在 30 比特内隐埋有效密钥。比如，一个算法可能接收 100 比特的密钥，但大部分可能是等价的。

——使用固定的 IV，或者在每则加密消息的开头加密一个确定报头，以允许选择明文攻击。

——产生一些随机字节，并用密钥将其加密，接着将这些随机字节的明文和密文放在被加密消息的开头，这也允许已知明文攻击。

NSA 得到一个源码副本，但加密算法细节对外仍保密，当然没有人大肆宣扬这些细致的弱点，但请意识到你买的是得到出口许可的美国加密产品。

10.2 公钥密码与对称密码

公开密码与对称密码哪一个好呢？这个问题没有任何意义，但是自从公开密码体制产生以来就一直争论不休。这个争论假定两种密码算法可以基于同一个基础点进行比较。事实上并非如此。

Needham 和 Schroeder[1159]指出使用公开密钥算法的消息的数量和长度比对称算法大得多，他们的结论是对称算法比公开密钥算法更有效，尽管这是正确的，但这种分析忽视了公开密钥方案的安全性意义。

Whitfield Diffie 在文献[492，494]中写道：

按照把公开密钥作为一种新的密码体制而不是新的密钥管理形式的观点，从安全性与性能两方面考虑，我站在批评者的一边。反对者立即指出 RSA 体制运行速度是 DES 的千分之一且要求 10 倍长的密钥。尽管从一开始公开密钥体制就被限制用于传统（对称）密码的密钥交换，明显这也是不必要的。在这种情况下建立一种混合密码体制[879]的建议被作为一个新的发现得到了很大的响应。

公开密钥密码与对称密码是不同的两种东西，它们解决不同的问题。对称密码算法适合加密数据，它速度极快并且对选择密文攻击不敏感。公开密钥密码可以做对称密码所不能做的事情，它最擅长密钥分配和第一部分讨论的大量协议。

在第一部分还讲了其他一些术语：单向 Hash 函数，消息鉴别码，等等。表 10.1 列出了各种不同算法及其性质[804]。

表 10.1 各算法级别

算法	机密性	验证	完整性	密钥管理
对称加密算法	YES	N O	NO	YES
公钥加密算法	YES	NO	NO	YES
数字签名算法	NO	YES	YES	NO
密钥共识算法	YES	可选	NO	YES
单向 Hash 函数	NO	NO	YES	NO
消息验证密码	NO	YES	YES	NO

10.3 通信信道加密

这是一个典型的 Alice—Bob 问题：Alice 想传送一个安全的消息给 Bob，她怎样去做？她将消息加密。

理论上，加密可以在 OSI（开放系统互连）通信模型的任何层进行。（更多信息参见 OSI 安全结构标准）事实上，加密一般在最底层（第一或第二层）或较高层。如果在最底层就称

为链-链加密，通过特定数据连接的任何数据都要被加密。如果发生在较高的层就称为端-端加密，数据被选择性加密，并且只在最后的接收端进行解密。两种方法各有优缺点。

链—链加密

最容易加密的地方是在物理层（见图 10.1），这叫做“ 链—链”加密。通常物理层接口是标准的，并且在此处最容易连接硬件加密设备。这些设备对通过它们的所有数据进行加密，包括数据、路由信息、协议信息等，它们可以被用于任何类型的数据通信链路上。另一方面，发送端与接收端之间的任何智能交换或存储节点都必须在处理这些数据流之前对其进行解密。

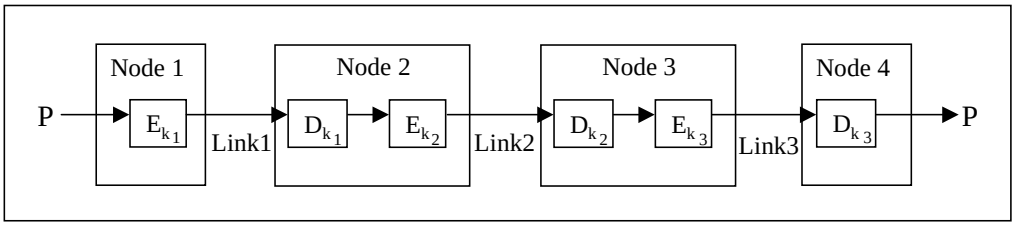

图 10.1　链路加密

这种类型的加密是非常有效的，因为任何东西都被加密，密码分析者得不到任何关于消息结构的信息，他们不知道谁正跟谁通话，发送消息多长，哪天进行的通信等等，这叫做“流量保密”。敌人不仅不能存取消息，而且也不知道消息的出处与去处。

系统安全性不依赖于任何传输管理技术。密钥管理也简单，仅仅是两端需要共同的密钥，他们可以独立于网络其他部分而更换密钥。

设想一个用 1 比特 CFB 加密的同步通信线路。在初始化后，线路可以无限的运行，比特和同步错误被自动恢复。无论什么时候有消息从一端发往另一端线路都被加密，没有消息时就加解密随机数据。Eve 不知道什么时候发送消息什么时候不发，也不知其开始与结束。她所见的只是无穷无尽的看上去随机的比特流。

如果通信线路是异步的， 1 比特 CFB 模式同样可以使用。不同的是敌方可以知道信息的传输率，如果这个信息必须隐蔽，那么必须做些防备，以使消息在空余时间悄悄传过。

物理层加密的最大问题是：网络中每个物理链路都必须加密，如果有一处没加密就会危及整个网络的安全，如果网络很大，这类加密的开销会变得很大，以致于限制了它的实施。

另外，必须保护网络中每个节点，因为它处理未加密的数据。如果网络中每个用户都相互信任，并且每个节点都很安全，这或许可以接受。但这是不可能的，即使在一个公司里面，信息可能必须在某个部门里保密，如果网络偶然将信息发错了路线，任何人都可以看到它。表 10.2 是链-链加密的优缺点。

端—端加密

另一种处理是将加密设备放在网络层和传输层之间，加密设备必须根据低三层的协议理解数据，并且只加密传输层的数据单元。这些加密的数据单元与未加密的路由信息重新结合，然后送到下一层进行传输。

表 10.2 链—链加密：优缺点

优点：	易操作，因为它对用户是透明的。即，在通过链路传送之前所有数据都被加密。每一次连接仅需要一组密钥。因为任何路由信息都被加密，所以能够提供安全的通信流。加密是在线的。
缺点：	在介质间节点数据被暴露

这种处理避免了在物理层中出现的加解密问题。通过“端—端”加密，数据一直保持加密状态，直到到达目的地才被解密（见图 10.2）。端—端加密的主要问题是路由信息未被加密；一个好的密码分析者可以据此知道谁和谁通信，何时以及有多长，而并不需要知道通信内容。其次，密钥管理也很困难，因为每个用户必须确保他们与其他人有共同的密钥。

制造端—端加密设备是困难的。每一个特殊的通信系统有其自身的协议，有时在这些层的接口没有很好的定义也使任务更加困难。

如果加密发生在通信系统的高层，象应用层或表示层，那么它可以独立于网络所用的通信结构。它仍是端—端加密，但加密实现并不扰乱线路编码、两个调制解调器间的同步、物理接口等等。在早期的电子机械密码体系中，完全是脱线加解密，这里只是向前走了一步。

在这些层加密会与相应层的软件相互作用。这些软件对不同的计算机结构是不同的，因而必须针对不同的计算机系统，通过软件自身或特殊的硬件进行加密。后一种情况下，计算机在发送到低层进行传输之前数据必须用特殊的硬件加密。这种处理需要智能终端而不适合非智能终端，另外可能还有各种不同计算之间的兼容问题。

端—端加密的最大缺点是它允许流量分析。流量分析主要分析：数据从哪里来到哪里去，传送多长时间，什么时候发送，发送频繁程度，是否与其它事件如会议等有关联等等。大量有用的信息埋藏在传送的数据中，密码分析者可得到很大帮助。表 10.3 列出端对端加密的正反两面。

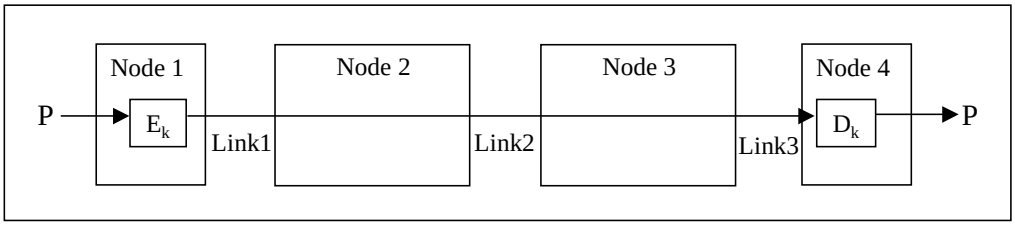

图 10.2　端对端加密

表 10.3 端—端加密：优点与缺点

优点：	更高的保密级别
缺点：	需要更复杂的密钥管理系统。流量分析是可能的，因为路由信息未被加密。离线加密。

两者的结合

表 10.4，主要摘自文献[1244]，对上面两种加密进行了比较。将端—端与链—链加密相结合，尽管很昂贵，却是一种有效的网络安全方法，加密每个物理链路使得对路由信息分析成为不可能，而端—端加密减少网络节点中未加密数据处理带来的威胁。对两种方案的密钥管理可以完全分开：网络管理人员可以只关心物理层，而每个用户只负责相应的端—端加密。

表 10.4 链-链加密与端-端加密的比较

	链-链加密	端-端加密
主机内部 安全性	发送主机内部数据暴露 交换节点数据暴露	发送主机内部消息被加密 在交换节点数据被加密
使用规则	发送主机使用 对用户不可见 主机保持加密 对所有用户便利 可以硬件完成 或者所有或者没有消息被加密	发送过程使用 用户实现加密 用户必须挑选算法 用户选择加密 软件更易完成 对每一条消息用户可选择加密或者否
有关实现	每一主机对需要一个密钥 每一台主机需要加密硬件或软件 提供节点验证	每一用户对需要一个密钥 在每一个节点需要加密硬件或软件 提供用户验证

10.4 加密数据存储

用于存储的加密数据、以后可以在 Alice-Bob 模式里检查。Alice 仍可向 Bob 发送消息，只是此处的“Bob”是某时间后的 Alice。然而，该问题又有本质的不同。

在通信信道里，传输中的数据并没有价值。如果 Bob 不接收某个特定消息，Alice 将一直发送下去。这对于用存储的数据加密来说是不正确的。如果 Alice 不能对她的消息进行解密，她将不能回过头去重新加密。或许她会永远地失去这些数据。这就意味着用于数据存储的加密实现应该有某些机制，以防止由于密文中的错误引起的不可恢复。

加密密钥有着与加密消息同样的价值，只是它小一点罢了。事实上，加密就是将大秘密变成小秘密。由于较小，所以它更容易丢失。密钥管理程序应当假定同一个密钥可以被一遍又一遍地使用，并且数据可以在磁盘上保存数年，直到被销毁。

此外，密钥应该保存很长一段时间。用于通信的密钥，理想化的仅存在通信维持时间。而用于数据存储的密钥则需要数年，因此必须安全地保存数年。

文献[357]列出针对计算机数据加密存储的其它一些问题：

——数据可以以明文形式存在，或者在别的磁盘上、或者计算机、或者就写在纸上。对密码分析者来说进行已知明文攻击有很大的机会。

——在数据库应用中，数据片比大多算法的数据分组小。这会使得密文比明文要大。

——I/O 设备的速度要求快速的加解密，这可能需要硬件加密。在一些应用中，可能需

要特殊的高速算法。

——需要安全，长期的密钥储存。

——密钥管理更复杂，因为不同的人需要存取不同的文件，或同一文件的不同部分等。

如果加密文件不是以记录和字段构造（象文本文件），恢复将很容易：整个文件在使用之前进行解密。如果加密文件是数据库文件，解决会有很多问题。为存取单个记录而解密整个数据库文件是无效率可言的，但独立地加密各个记录又容易受到分组重放之类的攻击。

另外，你必须确认加密后未加密的文件已删除（见 10.9 节），进一步的细节、研究请参见文献[425，569]。

非关联化密钥

当对一个大的硬盘驱动器进行加密时，有两种方法。你可以用一个单独的密钥对所有数据进行加密。但这给分析者提供了大量用于分析的密文，并使多个用户只查看驱动器的一部分成为不可能。或者，你可以用不同的密钥对各个文件进行加密，这迫使用户去记住每个文件的密钥。

解决办法是使用独立的密钥对每一个文件进行加密，然后用一个各个用户都知道的密钥加密这些密钥。每个用户只须记住一个密钥，不同的用户可以有一个用他们的密钥加密的文件加密密钥子集。并用主密钥对每一个文件加密密钥加密。这会更加安全，因为文件加密密钥是随机的，并对字典攻击不敏感。

驱动器级与文件级加密

有两种方法加密硬盘驱动器：文件级和驱动器级。文件级加密是指每一个文件被单独加密。为了使用被加密的文件，你必须先解密，再使用，再重新加密。

驱动器级加密保持在用户的一个逻辑驱动器上，对所有的数据加密。如果做得好，可以提供很好的安全性，它比选择好的通信字更安全，不需要为用户担心。然而，因为必须要处理一些诸如驱动器的安装、文件新扇区的划分、文件老扇区的反复应用、逻辑磁盘上的数据的随机存取和更新请求等，所以这个驱动器级加密肯定比单一的文件加密程序复杂得多。

标准情况是，启动前驱动器会提示用户输入一个口令，它被用来生成解密主密钥，然后用主密钥解密真正的不同数据的解密密钥。

提供加密驱动器的随机存取

大多数系统都期望能够随机存取单个磁盘扇区。这给任何链接模式下使用许多序列密码和分组密码增加了不少难度。有很多可能的解决办法。

可以使用扇区的地址为每一个被加/解密的扇区生成一个唯一的 IV。缺点就是每一个扇区将一直用同一个 IV 进行加密。确信这不是一个安全问题。

为了得到主密钥，可生成一个与某个扇区同样大小的伪随机分组。（例如，运行 OFB 模式下的某个算法可以做到这个。）为了加密任何扇区，先与该伪随机分组相异或，然后用 ECB 模式正常地加密。该方法被叫做“ECB+OFB”（参见 15.4 节）。

由于 CBC 和 CFB 模式是错误恢复模式，所以你可以用扇区上除了第一和第二个分组的

所有数据为该扇区产生一个 IV。例如，扇区 3001 的 IV 可以是扇区除了前 128 比特的所有数据的 Hash 函数。产生该 IV 后，可以正常的在 CBC 模式下进行加密了。为了对该扇区解密，把扇区的第二个 64-比特分组作为一个 IV，然后对扇区的其余部分进行解密，接着，利用解密的数据，重新生成一个 IV ，再对第一个 128 比特解密。

你可以使用有很大分组尺寸的分组密码算法，这样就可以一次对整个扇区完成加密。Crab（参见 14.6 节）是一个例子。

表 10.5　文件级加密与驱动器级加密的比较

	文件级加密	驱动器加密
好处	易实现,易使用 相对小的不利结果 用户可以在不同机器间无问题的移动文件 用户可无问题的对文件做备份	暂时文件,工作文件等可在安全驱动器里存储 在这种系统里很难忘记重复加密
安全问题	通过无安全意识程序存在潜在泄露量(如,为了暂存将文件写入磁盘) 差劲的程序实现会对同一个口令用相同的密钥重复加密	用一个设备驱动器或抗内存程序会使一些事情出错 差劲的程序实现允许了选择明文或选择密文攻击 如果一个系统用一个口令加密，丢失它意味着攻击者得到了任何一切 很有限的密码算法可以使用，如，OFB 序列密码就不能工作
可用性	用户必须计算怎样去做 对不同的文件可能有不同的口令 所选文件的手工加密是唯一的存取控制	存在性能缺陷 该驱动器可在 Windows，OS/2 DOS，设备驱动器等间神奇地相互作用

10.5 硬件加密与软件加密

硬件

直到最近，所有加密产品都是特定的硬件形式。这些加/解密盒子被嵌入到通信线路中，然后对所有通过的数据进行加密。虽然软件加密在今天正变得很流行，硬件仍是商业和军事应用的主要选择。例如，NSA 只对硬件加密授权使用。为什么这样是有原因的：

首先是速度。正如我们在第三部分看到的那样，加密算法含有很多对明文比特的复杂运算，没有哪类这样的操作能在一般的计算机上进行。两种最常见的加密算法，DES 和 RSA 在普通用途的微处理器上运行没有效率可言。尽管一些密码设计者不断尝试使他们的算法更

适合软件实现，但特殊的硬件将一直获得速度之胜利。

另外，加密常常是高强度的计算任务。计算机微处理器对此效率不高，将加密移到芯片上，即使那个芯片仅是另一个处理器，也会使整个系统速度加快。

硬件流行的第二个原因是安全性。对运行在一般的、没有物理保护的计算机上的某个加密算法，Mallory 可以用各种跟踪工具秘密修改算法而使任何人都不知道。硬件加密设备可以安全地封装起来，以避免此类事情发生，防窜改盒能防止别人修改硬件加密设备。特殊目的的 VLSI 芯片可以覆盖一层化学物质，使得任何企图对它们内部进行访问都将导致芯片逻辑的破坏。美国政府的 Clipper 和 Capstone 芯片（参见 24.16 和 24.17 节）都被设计成防窜改，芯片设计成这样就使 Mallory 不可能读到未加密的密钥。

IBM 发明了一种用来加密主机数据和通信的加密系统[515，1027]。它包括用防窜改模块保存密钥，这个系统在 24.1 节中讨论。

电磁幅射有时会暴露电子设备内正在处理的东西。可以将加密盒子屏蔽起来，使得信息不致泄露。通用计算机也可以屏蔽，但却是个复杂得多的问题。美军称这类操作为 TEMPEST，这个课题远远超出本书范围。

硬件流行的最后一个原因是易于安装。大多数加密应用与普通计算机无关。多数人希望加密他们的电话会话、传真或数据链路。将专用加密硬件放在电话、传真机和调制解调器中比放在微处理器或软件中便宜得多。

当加密数据来自计算机时，安装一个专用加密设备也比修改计算机系统软件更容易。加密应该是不可见的，它不应该妨碍用户。对于软件要做到这点的唯一办法是将加密程序写在操作系统软件的深处，这很不容易。另一方面，就是初学者也能将加密盒插在他们的计算机和外接调制解调器之间。

目前，市场上有三类基本的加密硬件：自带加密模块（可完成一些如银行口令确认和密钥管理等功能），用于通信链路的专用加密盒以及可插入个人计算机的插卡。

一些加密盒是为一些具体的通信链路设计的，如 T—1 加密盒设计成不加密同步比特。用于同步或异步通信链路的加密盒是不同的。较新的一些加密盒趋向于处理更高的比特率和高通用性。

即使如此 ，许多加密设备也有一些不相容问题，购买者应该小心注意这些差别，并了解他们的特殊用处，避免自己购买的加密设备不能满足要求。特别要注意，硬件类型、操作系统、应用软件、网络等方面的限制。

PC-板加密器通常将所有写到硬盘上的东西进行加密，并且可以配置以将写到软盘和串口的东西都加密。并不为这些板卡屏蔽电磁辐射或物理干扰，因为如果计算机不受影响，保护这些板卡是没有意义的。

越来越多的公司开始将加密硬件设备安装到他们的通信设备上。保密电话、传真机和调制解调器都可买到。

虽然有多少种设备、就有多少种不同的解决方案，但这些设备的内部密钥管理通常是安全的。一些方案在一种场合比在另一场合更合适，购买者应该懂得哪类密钥管理与加密盒相结合，哪类是自己所期望的。

软件

任何加密算法都可以用软件实现。软件实现的不利之处是速度、开销、和易于改动（或操作）。有利之处是灵活性和可移植性，易使用，易升级。本书末尾采用 C 语言写的算法，稍作修改便可以在任何计算机上实现。可以不花一分钱将他们容易地复制下来，并安装在许多机器上。他们也能和大型应用如通信或字处理程序相结合。

软件加密程序很大众化，并可以用于大多数操作系统。这些是用于保护个人文件；用户通常必须手工加解密文件。密钥管理方案的安全性是重要的：密钥不应当储存在磁盘的任何一处（甚至也不应该写在处理器与磁盘交换数据的内存中）。密钥和未加密文件在加密后应删除，许多程序对这点都很草率，但用户必须仔细选择。

当然，Mallory 可以一直用无用的东西替换软件加密算法，但对大多数用户来说，这不是什么问题。如果 Mallory 能够潜入办公室将加密程序修改掉，也能将一个隐形摄象机置于墙中，搭线窃听电话线路，或者将一台 TEMPEST 检测仪放于墙下。如果 Mallory 确实比一般用户更强有力的话，那么用户早在游戏开始之前就输掉了。

10.6 压缩、编码、加密

将数据压缩算法跟加密算法一起使用是很有意义的，有两个原因：

——如果密码分析依靠明文中的冗余，那么压缩将使文件在加密之前将冗余减少。

——加密是耗时的，在加密之前压缩文件可以提高整个处理过程的速度。

需要重点记住的是压缩要在加密之前进行。如果加密算法真的很好，那么密文是不可压缩的，它看上去就象随机数据。（这个可作为一种检验加密算法的测试方法，如果密文是可以压缩的，这说明所使用的算法不是很好。）

如果你想加进某种传输编码或错误检测和恢复，记住要在加密之后。如果在通信信道有噪声存在，解密的错误扩散特性会使得噪音更糟。图 10.3 总结了这些步骤。

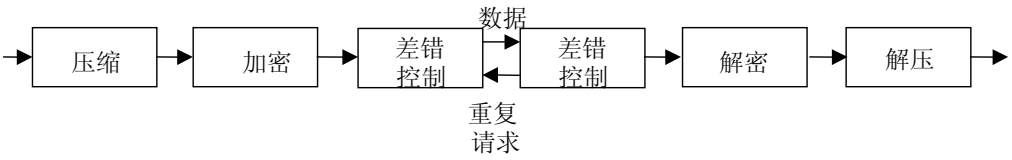

图10.3

10.7 检测加密

Eve 怎样检测到一个加密的文件呢？Eve 从事间谍行业，所以这是一个重要的问题，设想一下她在网络上偷听向四方高速传播的信息，她必须挑选出有意义的那些。加密文件确实是有趣的，但她是怎样知道它们是被加密的呢？

通常地讲，她依赖于这个事实：大多数流行的加密程序都有定义良好的报头。由于这个原因，用 PEM 和 PGP（参见 24.10 和 24.12 节）加密的电子邮件消息可以很容易地被辨别出来。

其它加密器（包括软件）也只产生貌似随机比特流的密文文件。她是怎样从其它貌似随机的比特流中将其区分出来的呢？没有确定的方法，但 Eve 可以试着做大量的事情：

——检查文件。ASCII 文本很容易认出。其它的文件格式如，TIFF，T_eX，C，信末附言，G3 传真，微软 Excel 等都有标准的辨别特性。执行代码同样也是可以检测到的。UNIX 文件通常有“幻数”可供检测。

——用大多数压缩算法试着对文件进行解压。如果文件被压缩（并没有加密），她就可以恢复出原始文件。

——试着压缩文件。如果文件是密文（且算法良好），那么文件被通用压缩程序有效压缩的可能性会很小。（“有效”是指超过 1-2%）如果是别的文件（例如，二进制图象，二进制数据文件等），都可能被压缩。

任何不能被压缩且没被压缩过的文件很可能就是密文。（当然，也可以特别制造可被压缩的密文。）对算法进行识别更加困难。如果算法是好的，你不可能识别。如果算法有某种轻微的偏差，那就有可能在文件中辨别出这些偏差。然而，偏差必须有明显的意义，或文件必须足够大才能很好的进行。

10.8 密文中隐藏密文

在过去几年里，Alice 和 Bob 一直是互相用密文发送消息。Eve 将所有的这些消息全部收集起来，但对它们无可奈何。最后，秘密警察为这些不可读的密文烦透了，于是就把他俩抓了起来，“把密钥交出来!”，他们命令道。Alice 和 Bob 拒绝了，但随后他们就找到了办法。他们怎样去做呢？

加密一个文件而有两种解密方法，每一种用各自不同的密钥，这不是很好。Alice 可以用他们的共享密钥加密发给 Bob 的真正消息，再用别的密钥加密一些无关紧要的消息。如果 Alice 被捕，她可以供出加密无关紧要消息的密钥，而对真正的密钥守口如瓶。

做到这点最简单的方法是使用一次一密乱码本。假设 P 是明文，D 是虚假明文，C 是密文，K 是真正的密钥，K'是虚假密钥。Alice 加密 P：

$$P \oplus K=C$$

Alice 和 Bob 共享密钥 K，所以 Bob 可以解密 C：

$$C \oplus K=P$$

如果秘密警察强迫他们供出密钥，他们并不会供出密钥 K，而是：

$$K'=C \oplus D$$

秘密警察就用它恢复出了明文——虚假明文：

$$C \oplus K'=D$$

因为使用的是一次一密乱码本，且 K 是完全随机的，所以没有任何方法证实 K'不是真正密钥。为了使情况更有信服力，Alice 和 Bob 可以编造一些轻度不犯法的虚假消息代替根本没有犯法的真正消息。一对以色列间谍曾经这样做过。

Alice 可以用她喜爱的算法和密钥 K 加密 P 得到 C。然后将 C 跟一些世俗的明文——譬如《傲慢与偏见》来得到 K'。她将 C 和这些异或运算储藏在她的硬盘上。现在，当安全局

审讯她时，她可以解释她是一个业余的密码爱好者，K′只不过是 C 的一次一密乱码本。安全局可能要怀疑某些东西，但除非他们已知密钥 K，否则他们无法证实 Alice 的解释不合理。

另一个方法是用对称算法的密钥 K 对 P 加密，用密钥 K′对 D 进行加密。打乱密文的比特（或字节）产生最后的密文。如果安全局需要密钥，Alice 就将 K′交出并说了打乱的比特（或字节）是为挫败密码分析而设计的随机的噪音。麻烦就出在她的解释太令人相信了，以至于安全局很可能不信她（仔细考虑一下，本书的建议）。

对 Alice 来说一个更好的方法是生成一个虚假消息 D，将明文 P 和 D 级联，然后压缩后使大小如同 D 一般。令级联为 P′。Alice 接着用她跟 Bob 共享的算法加密 P′得到 C，并将它传给 Bob，Bob 解密得到 P′，P，D。然后两人同时计算 C⊕D= K′。 当安全局叩开他们的门时 K′就变成了虚假一次一密乱码本。Alice 必须把 D 传送出去，这样两人就伪造了现场。

对 Alice 来说其它方法是利用无关紧要的消息，并通过某些纠错编码运作它。然后引入一些与秘密的加密消息有关的错误。在接收端，Bob 析取出错误，恢复秘密消息并解密。Alice 和 Bob 或许被强迫去解释为什么在无噪声的计算机网络上怎会一直有 30%的错码率，但在别的环境中该方案可以实施。

最后，Alice 和 Bob 可以在他们的数字签名算法中使用阈下信道（参见 4.2 和 23.3 节）。该方法是检测不到的，可以很好的使用，但它有一个缺点是每个签名的无关紧要的消息只允许 20 左右的阈下文本字符。它并不适合传送密钥。

10.9 销毁信息

在大多数的计算机上删除一个文件时，该文件并不会真的被删除。删除掉的唯一东西就是磁盘索引文件中的入口，磁盘索引文件用来告诉机器磁盘上的数据在哪里。许多软件供应商不失时机的出售文件恢复软件，它们可以在文件被删除后将其恢复。

还有别的方面的担忧：虚拟内存意味着你的计算机可以随时将内存读写到你的磁盘。即使你没有保存它，你永远也不知道你正在运行的一个敏感的文件是什么时候写到磁盘上的。这就是说，即使你从来未保存过的明文，计算机也可以替你做了。并且如 Stacker 和 DoubleSpace 这样的驱动器级的压缩程序会使得预测数据是怎样存到磁盘上，且存到哪里更加困难。

为了删除某个文件，让文件恢复软件都不能读，必须对磁盘上文件的所有比特进行物理写覆盖。根据国家计算机安全中心[1148]：

写覆盖就是将不涉及安全的数据写到以前曾存放敏感数据的储存位置……为了彻底清除……储存介质，DoD 要求先用一种格式进行写覆盖，然后用该格式的补码，最后用另一种格式。例如，先用 0011 0101，接着用 1100 1010，在接着用 1001 0111。写覆盖的次数根据储存介质而定，有时依赖信息的敏感程度，有时对不同的 DoD 部分要求。无论怎样，在最后没有用不涉及安全的数据写覆盖之前，彻底清除就没有完成。

你可能必须删除某个文件或清除整个驱动器，你也应当清除磁盘上所有没有用的空间。

大多数商用程序声称实现了 DoD 标准覆盖三次：首先用全 1；接着用全 0；最后用 1-0 格式重复进行。按照我的一般的偏执狂级别，我建议覆盖一个被删除的文件需要 7 次：首先全 1；其次全 0；其余 5 次用密码学安全的伪随机序列。最近国家标准和技术研究所对电子隧道显微镜的研究表明即使这样也是不够的。说实话，如果你的数据的确有足够大的价值，还是相信从磁性介质上完全清除数据是不可能的吧！将介质烧掉或切碎；买张新磁盘要比丢失你的秘密便宜得多。