加密技术：从理论到实践

第1卷：算法篇

荣 亮 (著)

序言

TBD

荣亮

2021年April月于苏州

目录

第1章　环游密码世界

随着物联网和智能家居的兴起，网络信息安全已经渗透到我们日常生活的方方面面。密码技术作为信息安全的基石，为网络通信提供了安全和可靠的技术手段。本章，我们先整体了解一下密码世界，看看各种密码技术如何为我们的信息安全保驾护航。

1.1　密码学中的基本概念

信息在人与人、人与机器、机器与机器之间交互的过程中存在被第三方（人或计算机）窃取的风险，密码技术提供了信息在通信双方交互的过程中免遭第三方窃取并破解，以及确保通信任何一方不被欺骗的一系列算法。

首先，我们来了解一下与密码技术有关的角色：

- *发送者* (sender)：消息的发送方。
- *接收者* (receiver)：消息的接收方。
- *窃听者* (eavesdropper)：监听在消息传送的通道上，窃取消息的恶意攻击方。
- *破译者* (cryptanalyst)：为研究密码强度而工作的密码破译人员或密码学研究者，要注意和窃听者的本质区别。

如图 1.1所示，如果发送者不对要发送出去的消息进行任何处理，很容易被窃听者窃取并获知消息的内容。为确保消息的机密性 (confidentiality)，发送者在发送消息之前需要对其进行加密 (encryption)。消息可以是任何类型的数据，例如，邮件、文档和交易等。通常，我们把加密前的消息称之为*明文* (plaintext)，加密之后的消息称之为*密文* (ciphertext)。接收者收到密文后将其恢复回明文的过程称为解密 (decryption)。

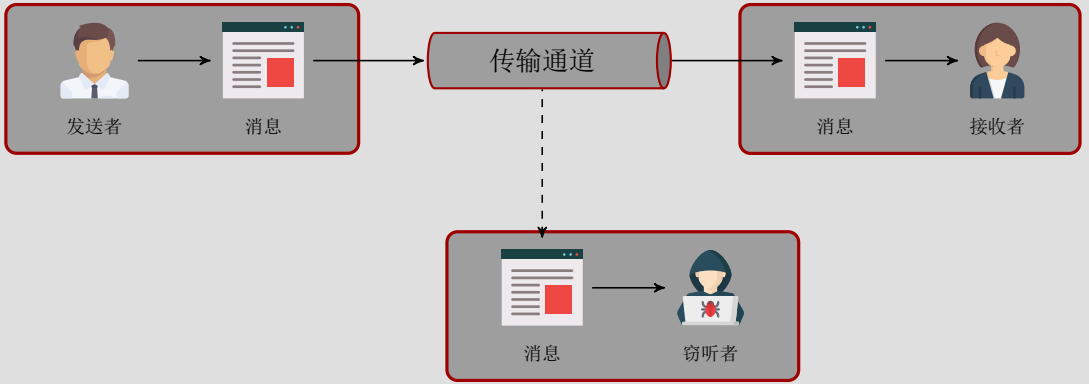


图1.1　消息的发送、接收和窃听

因为加密和解密需要相应的密钥才能完成，当窃听者窃取到加密后的密文之后，因为没有密钥，也就无法还原出原始的明文。这就相当于，我们在把重要的机密文件传给接收人之前，先把机密文件锁在保险柜里面，然后把保险柜交给物流公司帮忙交付给接收人，接收人收到之后用保险柜的钥匙打开取出该机密文件。在物流运输的过程中，保险柜有可能面临被丢失的风险，如果有人偷盗了保险柜，因为没有钥匙也无法取出里面的文件。整个过程可以用图 1.2来描述。

1.2　对称加密算法和非对称加密算法

在上面的例子中，如果保险箱在运输的过程中被坏人偷盗，坏人有可能使用物理破坏的方法撬开保险箱，取出里面的机密文件。那么，显而易见，保险箱越坚固，坏人就越难破坏保险箱拿到里面的机密文件。所以，保险箱的坚固程度就决定了里面机密文件的安全性到底有多高。

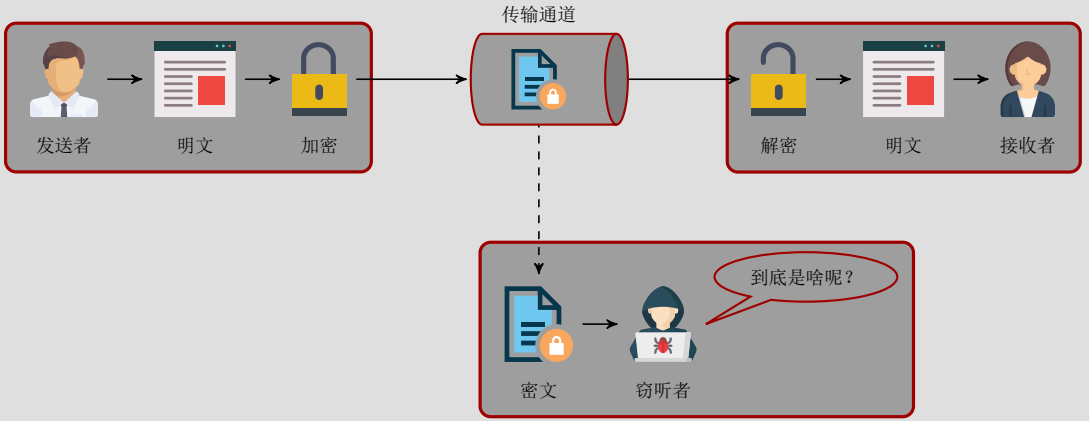


图1.2　消息加密和解密的过程

在网络通信领域，网络传输通道是极不可靠的，信息加密后的密文在传输的过程中，存在被窃听者窃取的风险，加密算法也要确保即使密文被窃取也不能在现实的时间内被破解，这也正是加密算法的魅力所在。

从原理上，加密算法被分成两大类，即*对称加密算法* (symmetric encryption algorithm) 和*非对称加密算法* (asymmetric encryption algorithm)。它们最主要的区别在于密钥 (key) 的使用方式不同。区别于现实生活中的“钥匙”，密码算法中的密钥是一串很长的看起来非常杂乱无章的字符序列。

对称加密算法在加密和解密时使用了相同的密钥，因为加密方和解密方使用了相同的密钥，任何人拿到了密钥就能解密加密后的消息从而获取到原始的数据。因此，对称加密算法的密钥必须在加密方和解密方两者同时妥善保管，不能泄露给任何未经授权的第三方。

相反，非对称加密算法则在加密和解密时使用了不同的密钥。而且，在非对称加密算法下，加密密钥通常被公开，但解密密钥需要由接收者私自妥善保管。据此特点，非对称加密算法又常常被称为*公钥加密算法* (public key encryption)。

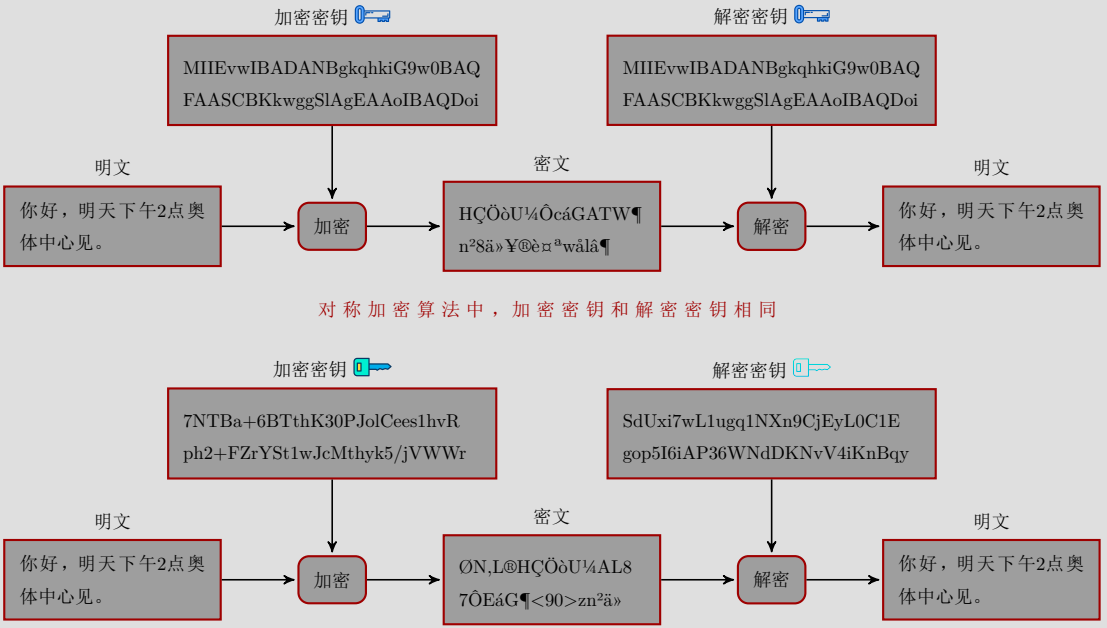


图1.3　对称加密和非对称加密

非对称加密算法是在1976年，由狄菲（Whitfield Diffie）与赫尔曼（Martin Hellman）两位学者以单向函数与单向暗门函数为基础，提出了“非对称密码体制即公开密钥密码体制”的概念，开创了密码学研究的新方向。现代计算机和互联网中的安全体系，很大程度上都依赖于公钥加密算法。

1.3　其他密码技术

加密算法为消息提供了机密性，但信息安全远不止于此，还有更多的问题需要解决。例如，如何保证数据的一致性，确保数据没有被恶意篡改过；如何对信息的来源进行判断，能对伪造来源的信息进行甄别。本节，我们来初步了解一下密码学工具箱中，除加密算法以外的的其他几种密码技术。

1.3.1　单向散列函数

有时候，接收者希望能够验证消息在传递的过程中，没有被篡改过，即入侵者不会用假消息冒充合法消息而达到某些非法的目的。

我们经常会发现，在互联网上下载免费软件的时候，有安全意识的软件发布者会在发布软件的同时发布该软件的散列值 (hash)。散列值就是用*单向散列函数* (one-way hash function) 计算出来的。这样，下载该软件的人可以自行计算所下载文件的散列值与发布者所发布的散列值进行比较。如果两个散列值一致，就说明下载的软件与发布者所发布的软件是相同的。软件发布者通过发布散列值的方法，可以防止有人在软件里植入一些恶意程序来侵害下载该软件的人的计算机系统。

单向散列函数所保证的并不是机密性，而是完整性 (integrity)。散列值通常又称为哈希值、校验和 (checksum)、指纹 (fingerprint) 或消息摘要 (message digest)。

1.3.2　消息认证码

为了确认消息是否来源于所期望的对象，可以使用*消息认证码* (message authentication code) 技术。通过消息认证码，不但能够确认消息是否被篡改，而且能够确认消息是否来自于所期望的通信对象。也就是说，消息认证码不仅能够保证完整性，还能够提供认证机制。

1.3.3　数字签名

我们先来看一个例子：供应商给采购方发来邮件，内容是“该商品的采购价格是10万元”。由于这封邮件涉及到数额巨大的交易，如果你是采购人员，肯定会特别小心，一定要核实该邮件确实来自你联系的供应商。仅仅靠邮件发送者的Email地址是不足以判断这封邮件的实际来源，因为邮件的发送者很容易被伪装 (spoofing)。

另一方面，还有这样一种可能，这封邮件确实是来自于采购方所联系的供应商。但是，供应商后来又反悔想提高采购价格，于是便谎称 “我当时根本就没发送过那封邮件”。像这样事后否认自己做过某件事情的行为，称为抵赖 (repudiation)。现代商战中，大量充斥着这种案例。

当然，还有一种风险，就是供应商发给采购方的邮件在传输过程中，被别有用心的人篡改，将采购费改成了20万元。数字签名是一项能够同时防止伪装、抵赖和篡改等威胁的密码技术。当供应商对邮件的内容加上数字签名之后再通过邮件一起发送，采购方则可以通过对数字签名 (digital signature) 进行验证 (verify) 来检测出邮件是否被伪装和篡改，还能够防止供应商事后抵赖。

1.3.4　伪随机数生成器

伪随机数生成器 (Pseudo Random Number Generator, PRNG) 用于在系统需要随机数的时候，通过一系列种子值计算出来的伪随机数。因为生成一个真正意义上的“随机数”对于计算机来说是不可能的，伪随机数也只是尽可能地接近其应具有的随机性，但是因为有“种子值”，所以伪随机数在一定程度上是可控可预测的。随机数在密码技术中承担了重要的职责，例如在访问HTTPS加密站点时进行的TLS通信，会生成一个仅用于当前通信的临时密钥（即会话密钥），这个密钥就是基于伪随机数生成器产生的。如果生成的随机数的算法不够好，窃听者就有可能推测出密钥，从而带来通信机密性下降的风险。

1.4　信息安全所面临的威胁及对策

回顾一下，我们前面初步介绍了六种密码技术：

- 对称加密算法
- 非对称加密算法（公钥加密算法）
- 单向散列函数
- 消息认证码
- 数字签名
- 伪随机数生成器

我们同时讨论了每种技术所解决的具体问题，这里把前面的内容再梳理一遍，用图 1.4所示的思维导视图总结了信息安全所面临的潜在威胁以及针对各种安全威胁所能采用的密码技术及对策。我们没有把伪随机数生成器画在图里面，是因为它通常渗透在其他五种密码技术中使用，发挥了非常重要的作用。我们把这六种密码技术统称为*密码学家的工具箱*。

从图 1.4中，我们可以看到，有些密码技术可以用来解决信息安全中的多种威胁，例如，数字签名可以防止篡改、伪装和抵赖，但不提供保密。对于某些面临的威胁，也可能存在多种应对的密码技术，例如为了防止窃听导致信息被泄露，可以使用对称加密算法或非对称加密算法。但是每种密码技术都有着各自的特点，适用于不同的场景。后面章节，我们会对这些密码技术进行深入的探讨，逐个揭开它们的神秘面纱。

1.5　密码与信息安全常识

随着信息技术的飞速发展，计算机的计算能力和存储能力正在以惊人的速度不断提升，我们所熟知的摩尔定律到目前为止仍然成立。大数据和物联网应用正在渗透到社会组织的每一个细胞，几乎对所有行业产生颠覆性和革命性的影响。产业的发展环境逐步成熟，网络基础设施支撑能力大幅提升，网络通信的数据量正在呈现指数级爆炸式增长。在人们的生活越来越依赖互联网的时代，信息安全在网络通信中发挥的作用尤为重要，密码技

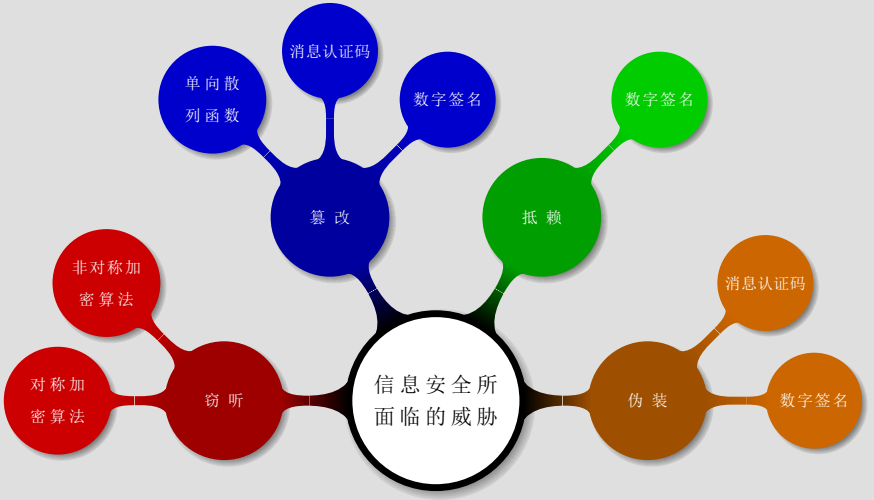


图1.4　信息安全所面临的威胁及其相应的密码技术对策思维导视图

术为保障信息安全提供了全方位的技术支持。本小节从最佳实践的角度阐述我们应该怎么合理地利用密码技术来保障信息的安全。

1.5.1　任何时候不要尝试发明新的加密算法

刚接触密码技术的软件开发人员，经常会出现这样的想法：我自己设计一个不对外公开的密码算法不就可以保障信息的机密性了吗？这种想法是绝对错误的。加密系统的保密性只应建立在对密钥的保密上，不应该取决于加密算法的保密，这是密码学中的金科玉律。任何时候，我们都不要尝试自己去发明新的加密算法，因为对加密算法的保密是困难的。对手可以用窃取、购买的方法来取得算法、加密器件或者程序。如果得到的是加密器件或者程序，可以对它们进行反向工程而最终获得加密算法。如果只是密钥失密，那么失密的只是和此密钥有

关的情报，日后通讯的保密性可以通过更换密钥来补救；但如果是加密算法失密，而整个系统的保密性又建立在算法的秘密性上，那么所有由此算法加密的信息就会全部暴露。

1.5.2　不要使用低强度的密码

很多人对密码的使用有这么一个误区：就算密码强度再低也比不用密码更安全吧。其实，这种想法是非常危险的。与其使用低强度的密码，还不如从一开始就不使用密码。这主要源于用户容易通过“密码”这个词获得一种“错误的安全感”。“信息被加密了”这一事实并不能和信息安全划上句号。攻击者使用暴力穷举（brute-force）等攻击方法就可能破解低强度的密码。

1.5.3　信息安全也是一门社会性课题

有了密码技术，信息安全就能完全得到保证吗？答案是否定的。密码技术只是信息安全的一部分，在信息安全的背景下，社会工程学（social engineering）攻击是一种操纵相关人员泄露出机密信息的攻击方法，建立在使人决断产生认知偏差的基础上，有时候这些偏差被称为“人类硬件漏洞”。犯罪分子利用社会工程学的手法进行诱骗，使受害者不会意识到被利用来攻击网络。当人们没有意识到他们拥有的信息的价值的时候，并不会特意地保护他们所得知的信息，社会工程学正是利用了这一点。

本书不会详细讨论社会工程学攻击，但是为了让大家提高安全意识，防患于未然，特列举以下一些流行的社会工程学攻击：

- 伪装：犯罪分子通过伪装成各种角色来骗取访问权限。例如，伪装成一个看门人、雇员或者客户来获取物理访问权限；冒充贵宾、高层经理或者其他有权或进入计算机系统并察看文件的人。
- 偷窥：通过偷窥方式在他人输入密码时收集他的密码。甚至寻找在垃圾箱中记录密码的纸、电脑打印的文件、快递信息等，往往也可以找到有用的信息。

- 钓鱼：钓鱼涉及虚假邮件、聊天记录或网站设计，模拟与捕捉真正目标系统的敏感数据。比如伪造一条上来自银行或其他金融机构的需要“验证”您登陆信息的消息，来冒充一条合法的登陆页面来骗取你的登录密码。

- 引诱：攻击者可能使用能勾起你欲望的东西引诱你去点击，可能是一场音乐会或一部电影的下载链接，也有可能是你“中奖” 需要兑换礼品的链接，或者是商品大力打折的促销链接。一旦点击了这些链接，你的计算机设备或网络就会感染恶意软件以便于犯罪分子进入你的系统。

上面提到的这些攻击手段，都与密码的强度毫无关系。信息安全是一个复杂的系统性工程，其安全程度往往取决于系统中最薄弱的环节。通常，最薄弱的环节不是密码，而是人类自己。“道高一尺魔高一丈”，信息安全上的漏洞和人性上脆弱的环节也不断被不法分子发掘，我们唯有不断的增强自己的安全意识和时刻保持清醒才能更好地防患于未然。

1.6　本章小结

本章，我们初步了解了密码世界里常用的密码技术，并介绍了使用哪种密码技术来应对信息安全中存在的威胁。我们后在后续章节中更详细地介绍每种密码技术的细节，并为应用开发者介绍怎么在工程中使用各种密码技术。

第2章　密码的历史典故

从密码学发展历程来看，可分为古典密码和现代密码两类。古典密码有着悠久的历史，是以字符为基本加密单元的密码。而现代密码则以信息块为基本的加密单元。古典密码和现代密码的分水岭大致就是在计算机问世的时候。本章将重点回顾古典密码的发展历史并分享一些和密码相关的有趣典故和历史事件。

2.1　中国古代人民怎么加密？

早期加密算法主要使用在军事中，中国历史上最早关于加密算法的记载出自于周朝兵书《六韬 · 龙韬》中的《阴符》和《阴书》。其中《阴符》记载了：

> 太公曰：“主与将，有阴符，凡八等。有大胜克敌之符，长一尺。破军擒将之符，长九寸。降城得邑之符，长八寸。却敌报远之符，长七寸。警众坚守之符，长六寸。请粮益兵之符，长五寸。败军亡将之符，长四寸。失利亡士之符，长三寸。诸奉使行符，稽留，若符事闻，泄告者，皆诛之。八符者，主将秘闻，所以阴通言语，不泄中外相知之术。敌虽圣智，莫之能识。”

简单来说，阴符是以八等长度的符来表达不同的消息和指令，属于密码学中的替代法（图 2.1），在应用中是把信息转变成敌人看不懂的符号，但知情者知道这些符号代表的含义。

阴符只能表述最关键的八种信号，无法表达丰富的含义和传递更具体的消息。所以，《阴书》又作了补充：

> 武王问太公曰：“引兵深入诸侯之地，主将欲合兵，行无穷之变，图不测之利，其事烦多，符不能明；相去辽远，言语不通。为之奈何？” 太公曰：“诸有阴事大虑，当用书，不用符。主以书遗将，将以书问主。书皆一合而再离，三发而一知。再离者，分书为三部。三发而一知者，言三人，人操一分，相参而不相知情也。此谓阴书。敌虽圣智，莫之能识。”

阴书作为阴符的补充，所有密谋大计，都应当用阴书，而不用阴符。国君用阴书向主将传达指示，主将用阴书向国君请示问题，这种阴书都是一合而再离（把一封书信分为三个部分）、三发而一知（派三个人送信，每人负

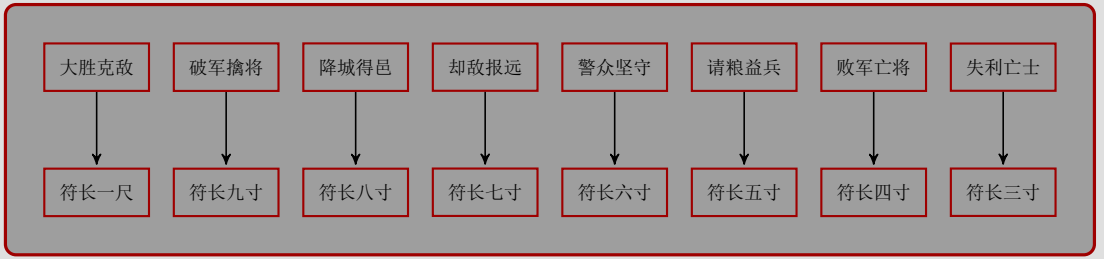


图2.1　阴符所蕴含的加密原理：替换法

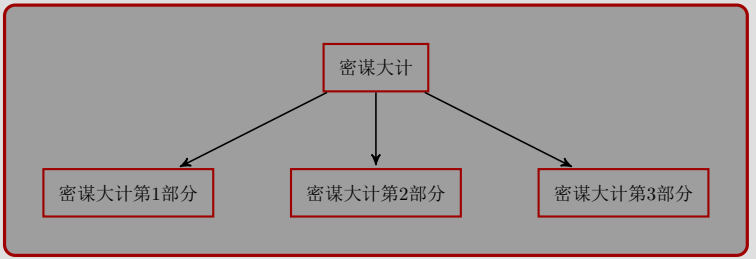


图2.2　阴书所蕴含的加密原理：文字分拆法

责其中的一部分）。阴书运用了文字拆分法直接把一份文字拆成三份（图 2.2），由三种渠道发送到目标方手中。敌人只有同时截获三份内容才可能破解阴书上写的内容。

无论是阴符，还是阴书，都有着一定的局限性。一是有可能被对方截获而难以达到传递消息的目的，二是有可能被对方破译内容并被对方将计就计加以利用。因此，并不是“敌虽圣智，莫之能识”。张献忠袭取襄阳就说明了这一点。

崇祯十三年七月，张献忠率领起义军突破明军防线，进入四川，杨嗣昌亦率明军十万尾随追击。面对强敌，张献忠挥师东进，于次年二月进入湖北兴山、当阳。在东进途中，起义军活捉了由襄阳（今湖北襄樊市）回四川的杨嗣昌的军使。张献忠从其口中得知杨嗣昌大营所在地襄阳城防空虚，决定奔袭襄阳。他杀掉使者，搜出所携带的兵符，挑选了二十八名起义军战士，换上明军的衣服，持兵符先行。张献忠自己则亲率二千精骑，随后跟进，一昼夜急行三百里，直扑襄阳。伪装成明军的起义军士兵到达襄阳时正是夜间，他们自称是督师杨嗣昌派来调运军械的，并出示兵符。守城明军用小筐吊上兵符，细心查验，完全吻合，才命开门放入。城门刚打开，二十八名起义军战士一涌而入，挥刀砍杀守门明军，占领城门。张献忠率领的后续部队恰好赶到，顺利入城。一时杀声震天，明军惊慌失措，被迫投降。起义军杀死襄王朱翊铭，降俘明军数千人，占领襄阳，杨嗣昌闻讯呕血而死。此战表明，无论是阴符还是阴书，都不是万无一失的。

2.2　凯撒密码

我们把视角切换到世界历史的长河中，看看古罗马时期凯撒大帝是怎么使用加密算法对军事信息进行加密的。根据罗马早期纪传体作者盖乌斯·苏维托尼乌斯的记载，恺撒大帝的加密策略很简单，就是把字母按照字母表顺序向后移动几位，但是偏移量（offset）只有他和将军知道，如果移动后超过了字母表中的最后一个字母（对于英文字母表而言就是Z），就回到字母表的第一个字母重新开始下一轮。以英文字母表为例，在偏移量为3的情况下，A将会替换为D，B将会被替换为E， W会被替换为Z，X会被替换为A；明文HELLO会被转换为密文KHOOR。这种加密方法又被称为移位加密。

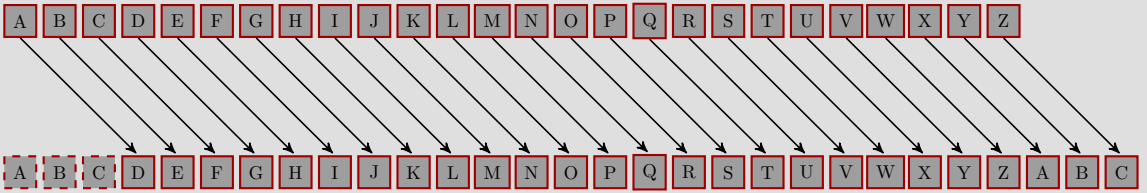


图2.3　凯撒密码的加密原理：替换法

凯撒密码的解密方法也很一目了然，只需要将密文中的每个字母向相反方向平移规定的偏移量便可解密出明文。恺撒密码的加密、解密算法还能够通过同余的数学方法进行计算。首先将字母表中的字母按顺序用数字代替，$A=0$，$B=1$，...，$Z=25$。此时偏移量为k的加密算法的数学公式即为：

$$E_k(x) = (x + k) \bmod 26$$

解密算法的数学公式可以表示为：

$$D_k(x) = (x + 26 - k) \bmod 26$$

从加密和解密数学公式可以看出，当偏移量$k=13$时（字母表内所有字母数量的一半），凯撒密码加密和解密算法的公式完全相同，这是一种特殊的凯撒密码的变种算法，被称为ROT13。ROT13在英文网络论坛常常用作隐藏八卦、妙句、谜题解答以及某些脏话的工具，目的是逃过版主或管理员的匆匆一瞥。因为ROT13的加密和解密计算公式完全相同，很明显，文字经过两次 ROT13加密之后，会恢复成原来的文字。

凯撒密码中的偏移量就相当于加密算法中的密钥。这个偏移量必须由发送者和接收者事先约定好。那么，当接收者以外的人窃取到用凯撒密码加密后的密文之后，是不是就无法破解这个密文了呢？或者换句话说，凯撒密码能够被破解吗？

破解密码的复杂度很大程度上取决于密钥空间（keyspace）的大小，所谓密钥空间是指密钥的取值范围到底有多大。凯撒密码中的密钥是偏移量k，其取值范围为0至25的整数，共26种可能的取值，密钥空间非常有限。攻击者往往可以采用暴力破解（brute-force attack）的方法就可以轻而易举地破解凯撒密码。

假设发送者和接收者之间约定的偏移量为3，那么明文CRYPTOGRAPHY加密后的密文则为FUBSWRJUDSKB。当第三方窃听到密文之后，由于凯撒密码的密钥空间只有26种可能的取值，窃听者可以使用穷举搜索（exhausitive search）的方法对每种可能的密钥取值尝试一遍：

```
k =  0: FUBSWRJUDSKB => FUBSWRJUDSKB
k =  1: FUBSWRJUDSKB => ETARVQITCRJA
k =  2: FUBSWRJUDSKB => DSZQUPHSBQIZ
k =  3: FUBSWRJUDSKB => CRYPTOGRAPHY
```

```
k =  4: FUBSWRJUDSKB => BQXOSNFQZOGX
k =  5: FUBSWRJUDSKB => APWNRMEPYNFW
k =  6: FUBSWRJUDSKB => ZOVMQLDOXMEV
k =  7: FUBSWRJUDSKB => YNULPKCNWLDU
k =  8: FUBSWRJUDSKB => XMTKOJBMVKCT
k =  9: FUBSWRJUDSKB => WLSJNIALUJBS
k = 10: FUBSWRJUDSKB => VKRIMHZKTIAR
k = 11: FUBSWRJUDSKB => UJQHLGYJSHZQ
k = 12: FUBSWRJUDSKB => TIPGKFXIRGYP
k = 13: FUBSWRJUDSKB => SHOFJEWHQFXO
k = 14: FUBSWRJUDSKB => RGNEIDVGPEWN
k = 15: FUBSWRJUDSKB => QFMDHCUFODVM
k = 16: FUBSWRJUDSKB => PELCGBTENCUL
k = 17: FUBSWRJUDSKB => ODKBFASDMBTK
k = 18: FUBSWRJUDSKB => NCJAEZRCLASJ
k = 19: FUBSWRJUDSKB => MBIZDYQBKZRI
k = 20: FUBSWRJUDSKB => LAHYCXPAJYQH
k = 21: FUBSWRJUDSKB => KZGXBWOZIXPG
k = 22: FUBSWRJUDSKB => JYFWAVNYHWOF
k = 23: FUBSWRJUDSKB => IXEVZUMXGVNE
k = 24: FUBSWRJUDSKB => HWDUYTLWFUMD
k = 25: FUBSWRJUDSKB => GVCTXSKVETLC
```

纵览所有尝试的破解，就会发现只有当$k = 3$的时候，密文FUBSWRJUDSKB才可以解密出有意义的字符序列CRYPTOGRAPHY，即“密码学”的英文单词。因此，凯撒密码是一种极其不安全的加密方法，可以被攻击者在很快的时间内破解，无法保护重要的秘密。

2.3　简单替换密码

2.3.1　什么是简单替换密码

凯撒密码通过将明文中的每个字符按照在字符表中的顺序平移固定数量的字符数来生成密文。由于字符偏移量的取值空间极其有限，致使凯撒密码能被轻而易举地破解。我们也提到了密钥空间这个概念，凯撒密码就是因为过小的密钥空间可以被攻击者使用暴力破解的方法在非常快的时间内被破解。你可能意识到凯撒密码这种通过平移字符来实现字符替换的方法过于公式化，如果把这种映射用随机化的方式打乱，是不是就完美了呢？这就是我们接下来要讨论的简单替换密码。

简单替换密码将字母表中的26个字母，分别与其他字母建立一一映射的关系，这种映射关系不像凯撒密码那样通过平移字符这种线性化的方法，而是用一个映射表来描述明文字符和密文字符之间的映射关系，这种映射表也称为字符替换表。为了更直观地展示字符之间的映射关系，我们把明文中的字符都用小写字母表示，密文中的字符都用大写字母表示。图 2.4 就是一个简单的字符替换表。

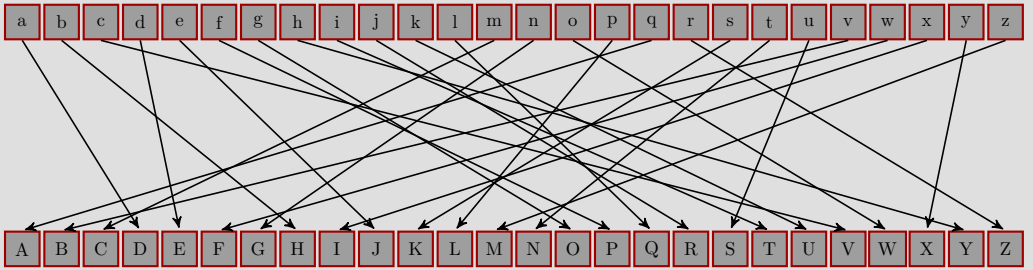


图2.4　简单替换密码的映射表

显然，图 2.4表示的字符替换关系不像凯撒密码那么有规律，明文字符和密文字符之间的映射看起来是无章可循的。可以说，凯撒密码是简单字符替换密码的一个特例。为了更好地展示明文字符和密文字符之间的替换关系，我们对图 2.4稍作转换，图 2.5，但仍然保持字符之间原来的映射关系。

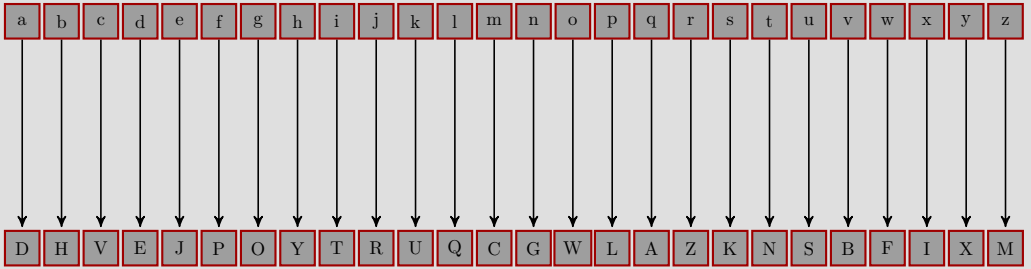


图2.5　变换后的简单替换密码的映射表

凯撒密码可以用暴力破解来破译，但简单替换密码则不然。前面，我们提到，密码算法被破译的困难程度取决于密钥空间的大小。我们来看看简单替换密码的密钥空间。明文字母中的a可以对应A，B，...，Z这26个字母中的任意一个（26种），b可以对应除了a所对应的字母以外的剩余25个字母中的任意一个（25种）。以此类推，我们可以计算出简单替换密码的密钥空间大小是：

$$26 \times 25 \times 24 \times \cdots \times 1 = 403291461126605635584000000$$

这个数字约等于400×10^{24}，密钥的数量如此巨大，用暴力破解进行穷举搜索就非常困难了。我们假设以当前（2018年11月）排名第一的Summit超级计算机在峰值性能下每秒约200亿次浮点计算的速度来遍历密钥的话，要遍历完所有的密钥也需要花费超过6千万年的时间。这还是用我们当前最顶级的超级计算机的峰值计算速度来遍历的，普通的家用计算机将要花费几百亿年的时间才能遍历完所有的密钥。由此可见，简单替换密码的密钥空间是足够大的。

2.3.2　用频率分析的方法破解简单替换密码

超大的密钥空间让破译简单替换密码看起来变得不可能，但密码破译工作者发现用频率分析的密码破译方法，使破译简单替换密码成为可能了。

在任何一种书面语言中，不同的字母或字母组合出现的频率各不相同。而且，对于以这种语言写的任意一段文本，都具有大致相同的特征字母分布。比如，在英语中，字母e出现的频率很高，而x出现的较少。类似地，字母组合st、 ng、th以及qu等双字母组合出现的频率非常高，nz、qj组合则极少。表 2.1是人们从大量的英文文章中统计出的字母频率。

字母	频率	字母	频率
e	11.1607%	m	3.0129%
a	8.4966%	h	3.0034%
r	7.5809%	g	2.4705%
i	7.5448%	b	2.0720%
o	7.1635%	f	1.8121%
t	6.9509%	y	1.7779%
n	6.6544%	w	1.2899%
s	5.7351%	k	1.1016%
l	5.4893%	v	1.0074%
c	4.5388%	x	0.2902%
u	3.6308%	z	0.2722%
d	3.3844%	j	0.1965%
p	3.1671%	q	0.1962%

表2.1　英文字母出现的频率表

简单替换密码的密钥空间如此巨大，但它的弱点也是显而易见的，就是明文中相同的字母在转换为密文后总是被同一个字母所替换。我们参考这个英文字母频率表来实际尝试破译一段密文。现在，假设我们得到下面一段经过简单替换密码加密过后的密文，其明文都是小写英文字母。

EAOQASBAEEAOQECPENFECQMRQFENABDCQECQSENFBAOZQSNBECQUNBLEAFRKKQSECQFZNBVFPBLPSSADFAKAR
ESPVQARFKASERBQASEAEPWQPSUFPVPNBFEPFQPAKESAROZQFPBLOIAGGAFNBVQBLECQUEALNQEAFZQQGBAUAS
QPBLOIPFZQQGEAFPIDQQBLECQCQPSEPTCQPBLECQECARFPBLBPERSPZFCATWFECPEKZQFCNFCQNSEAENFPTAB
FRUUPENABLQHAREZIEAOQDNFCLEALNQEAFZQQGEAFZQQGGQSTCPBTQEALSQPUPIECQSQFECQSROKASNBECPEF
ZQQGAKLQPECDCPELSQPUFUPITAUQDCQBDQCPHQFCRKKZQLAKKECNFUASEPZTANZURFEVNHQRFGPRFQECQSQFE
CQSQFGQTEECPEUPWQFTPZPUNEIAKFAZABVZNKQKASDCADARZLOQPSECQDCNGFPBLFTASBFAKENUQECAGGSQFF
ASFDSABVECQGSARLUPBFTABERUQZIECQGPBVFAKLNFGSNXLZAHQECQZPDFLQZPIECQNBFAZQBTQAKAKKNTQPB
LECQFGRSBFECPEGPENQBEUQSNEAKECRBDASECIEPWQFDCQBCQCNUFQZKUNVCECNFMRNQERFUPWQDNECPOPSQO
ALWNBDCADARZLKPSLQZFOQPSEAVSRBEPBLFDQPERBLQSPDQPSIZNKQOREECPEECQLSQPLAKFAUQECNBVPKEQS
LQPECECQRBLNFTAHQSQLTARBESIKSAUDCAFQOARSBBAESPHQZZQSSQERSBFGRXXZQFECQDNZZPBLUPWQFRFSP
ECQSOQPSECAFQNZZFDQCPHQECPBKZIEAAECQSFECPEDQWBADBAEAKECRFTABFTNQBTQLAQFUPWQTADPSLFAKR
FPZZPBLECRFECQBPENHQCRQAKSQFAZRENABNFFNTWZNQLAQSDNECECQGPZQTPFEAKECARVCEPBLQBEQSGSNFQ
FAKVSQPEGNETCPBLUAUQBEDNECECNFSQVPSLECQNSTRSSQBEFERSBPDSIPBLZAFQECQBPUQAKPTENAB

首先，我们统计这段密文中各个字母出现的次数和频率，结果如**表 2.2**所示。

根据密码研究工作者总结出来的字母频率**表 2.1**，字母e的出现频率远高于其他字母。经统计后，我们发现密文中字母Q的出现频率最高。我们先暂且假设字母Q就是由e变换而来的，这样，我们把密文中的Q替换回e，就得到下面的字母序列。

EAOeASBAEEAOeECPENFECeMReFENABDCeECeSENFBAOZeSNBECeUNBLEAFRKKeSECeFZNBVFPBLPSSADFAKAR
ESPVeARFKASERBeASEAEPWePSUFPVPNBFEPFePAKESAROZeFPBLOIAGGAFNBVeBLECeUEALNeEAFZeeGBAUAS
ePBLOIPFZeeGEAFPIDeeBLECeCePSEPTCePBLECeECARFPBLBPERSPZFCATWFECPEKZeFCNFCeNSEAENFPTAB
FRUUPENABLeHAREZIEAOeDNFCLEALNeEAFZeeGEAFZeeGGeSTCPBTeEALSePUPIECeSeFECeSROKASNBECPEF
ZeeGAKLePECDCPELSePUFUPITAUeDCeBDeCPHeFCRKKZeLAKKECNFUASEPZTANZURFEVNHeRFGPRFeECeSeFE
CeSeFGeTEECPEUPWeFTPZPUNEIAKFAZABVZNKeKASDCADARZLOePSECeDCNGFPBLFTASBFAKENUeECAGGSeFF
ASFDSABVECeGSARLUPBFTABERUeZIECeGPBVFAKLNFGSNXLZAHeECeZPDFLeZPIECeNBFAZeBTeAKAKKNTePB
LECeFGRSBFECPEGPENeBEUeSNEAKECRBDASECIEPWeFDCeBCeCNUFeZKUNVCECNFMRNeERFUPWeDNECPOPSeO

字母	次数	频率	字母	次数	频率
Q	137	12.47%	K	34	3.09%
E	117	10.65%	D	28	2.55%
A	93	8.46%	U	28	2.55%
P	84	7.64%	T	24	2.18%
F	82	7.46%	G	22	2.00%
C	75	6.82%	O	15	1.36%
S	68	6.19%	I	14	1.27%
B	65	5.91%	V	14	1.27%
N	53	4.82%	W	10	0.91%
L	42	3.82%	H	8	0.73%
Z	41	3.73%	X	3	0.27%
R	40	3.64%	M	2	0.18%

表2.2　密文中各英文字母出现的次数和频率

```
ALWNBDCADARZLKPSLeZFOePSEAVSRBEPBLFDePERBLeSPDePSIZNKeOREECPEECeLSePLAKFAUeECNBVPKEeS
LePECECeRBLNFTAHeSeLTARBESIKSAUDCAFeOARSBBAESPHeZZeSSeERSBFGRXXZeFECeDNZZPBLUPWeFRFSP
ECeSOePSECAFeNZZFDeCPHeECPBKZIEAAECeSFECPEDeWBADBAEAKECRFTABFTNeBTeLAeFUPWeTADPSLFAKR
FPZZPBLECRFECeBPENHeCReAKSeFAZRENABNFFNTWZNeLAeSDNECECeGPZeTPFEAKECARVCEPBLeBEeSGSNFe
FAKVSePEGNETCPBLUAUeBEDNECECNFSeVPSLECeNSTRSSeBEFERSBPDSIPBLZAFeECeBPUeAKPTENAB
```

英文文章中，以字母e结尾的单词，the的出现频率极高，对上面的这段字符序列，进一步统计发现ECe出现了27次，远远高于其他以e结尾的包含3个字母的字符串的出现次数。我们进一步假定t被替换成了E，h被替换成了C，于是，我们继续将上面字符序列中E和C分别替换回t和h，得到：

tAOeASBAttAOethPtNFtheMReFtNABDhetheStNFBAOZeSNBtheUNBLtAFRKKeStheFZNBVFPBLPSSADFAKAR
tSPVeARFKAStRBeAStAtPWePSUFPVPNBFtPFePAKtSAROZeFPBLOIAGGAFNBVeBLtheUtALNetAFZeeGBAUAS
ePBLOIPFZeeGtAFPIDeeBLthehePStPThePBLthethARFPBLBPtRSPZFhATWFthPtKZeFhNFheNStAtNFPTAB
FRUUPtNABLeHARtZItAOeDNFhLtALNetAFZeeGtAFZeeGGeSThPBTetALSePUPItheSeFtheSROKASNBthPtF
ZeeGAKLePthDhPtLSePUFUPITAUeDheBDehPHeFhRKKZeLAKKthNFUAStPZTANZURFtVNHeRFGPRFetheSeFt
heSeFGeTtthPtUPWeFTPZPUNtIAKFAZABVZNKeKASDhADARZLOePStheDhNGFPBLFTASBFAKtNUethAGGSeFF
ASFDSABVtheGSARLUPBFTABtRUeZItheGPBVFAKLNFGSNXLZAHetheZPDFLeZPItheNBFAZeBTeAKAKKNTePB
LtheFGRSBFthPtGPtNeBtUeSNtAKthRBDASthItPWeFDheBhehNUFeZKUNVhthNFMRNetRFUPWeDNthPOPSeO
ALWNBDhADARZLKPSLeZFOePStAVSRBtPBLFDePtRBLeSPDePSIZNKeORtthPttheLSePLAKFAUethNBVPKteS
LePththeRBLNFTAHeSeLTARBtSIKSAUDhAFeOARSBBAtSPHeZZeSSetRSBFGRXXZeFtheDNZZPBLUPWeFRFSP
theSOePSthAFeNZZFDehPHethPBKZItAAtheSFthPtDeWBADBAtAKthRFTABFTNeBTeLAeFUPWeTADPSLFAKR
FPZZPBLthRFtheBPtNHehReAKSeFAZRtNABNFFNTWZNeLAeSDNththeGPZeTPFtAKthARVhtPBLeBteSGSNFe
FAKVSePtGNtThPBLUAUeBtDNththNFSeVPSLtheNSTRSSeBtFtRSBPDSIPBLZAFetheBPUeAKPTtNAB

进一步分析，我们发现thPt也多次出现，英文中单词that出现的频率也是特别高的。同时，我们发现P在这段密文中出现的频率也是极高的，我们几乎可以不假思索地猜测a被替换成了P。把P替换回a，我们得到：

tAOeASBAttAOethatNFtheMReFtNABDhetheStNFBAOZeSNBtheUNBLtAFRKKeStheFZNBVFaBLaSSADFAKAR
tSaVeARFKAStRBeAStAtaWeaSUFaVaNBFtaFeaAKtSAROZeFaBLOIAGGAFNBVeBLtheUtALNetAFZeeGBAUAS
eaBLOIaFZeeGtAFaIDeeBLtheheaStaTheaBLthethARFaBLBatRSaZFhATWFthatKZeFhNFheNStAtNFaTAB
FRUUatNABLeHARtZItAOeDNFhLtALNetAFZeeGtAFZeeGGeSThaBTetALSeaUaItheSeFtheSROKASNBthatF
ZeeGAKLeathDhatLSeaUFUaITAUeDheBDehaHeFhRKKZeLAKKthNFUAStaZTANZURFtVNHeRFGaRFetheSeFt
heSeFGeTtthatUaWeFTaZaUNtIAKFAZABVZNKeKASDhADARZLOeaStheDhNGFaBLFTASBFAKtNUethAGGSeFF
ASFDSABVtheGSARLUaBFTABtRUeZItheGaBVFAKLNFGSNXLZAHetheZaDFLeZaItheNBFAZeBTeAKAKKNTeaB
LtheFGRSBFthatGatNeBtUeSNtAKthRBDASthItaWeFDheBhehNUFeZKUNVhthNFMRNetRFUaWeDNthaOaSeO
ALWNBDhADARZLKaSLeZFOeaStAVSRBtaBLFDeatRBLeSaDeaSIZNKeORtthattheLSeaLAKFAUethNBVaKteS
LeaththeRBLNFTAHeSeLTARBtSIKSAUDhAFeOARSBBAtSaHeZZeSSetRSBFGRXXZeFtheDNZZaBLUaWeFRFSa
theSOeaSthAFeNZZFDehaHethaBKZItAAtheSFthatDeWBADBAtAKthRFTABFTNeBTeLAeFUaWeTADaSLFAKR

FaZZaBLthRFtheBatNHehReAKSeFAZRtNABNFFNTWZNeLAeSDNththeGaZeTaFtAKthARVhtaBLeBteSGSNFe
FAKVSeatGNtThaBLUAUeBtDNththNFSeVaSLtheNSTRSSeBtFtRSBaDSIaBLZAFetheBaUeAKaTtNAB

继续猜测，theSe会不会是there呢，Leath会不会是death呢，于是，我们用r和d分别替换回S和L，得到：

tAOeArBAttAOethatNFtheMReFtNABDhethertNFBAOZerNBtheUNBdtAFRKKertheFZNBVFaBdarrADFAKAR
traVeARFKArtRBeArtAtaWearUFaVaNBFtaFeaAKtrAROZeFaBdOIAGGAFNBVeBdtheUtAdNetAFZeeGBAUAr
eaBdOIaFZeeGtAFaIDeeBdtheheartaTheaBdthethARFaBdBatRraZFhATWFthatKZeFhNFheNrtAtNFaTAB
FRUUatNABdeHARtZItAOeDNFhdtAdNetAFZeeGtAFZeeGGerThaBTetAdreaUaIthereFtherROKArNBthatF
ZeeGAKdeathDhatdreaUFUaITAUeDheBDehaHeFhRKKZedAKKthNFUArtaZTANZURFtVNHeRFGaRFethereFt
hereFGeTtthatUaWeFTaZaUNtIAKFAZABVZNKeKArDhADARZdOeartheDhNGFaBdFTArBFAKtNUethAGGreFF
ArFDrABVtheGrARdUaBFTABtRUeZItheGaBVFAKdNFGrNXdZAHetheZaDFdeZaItheNBFAZeBTeAKAKKNTeaB
dtheFGRrBFthatGatNeBtUerNtAKthRBDArthItaWeFDheBhehNUFeZKUNVhthNFMRNetRFUaWeDNthaOareO
AdWNBDhADARZdKardeZFOeartAVrRBtaBdFDeatRBderaDearIZNKeORtthatthedreadAKFAUethNBVaKter
deaththeRBdNFTAHeredTARBtrIKrAUDhAFeOARrBBAtraHeZZerretRrBFGRXXZeFtheDNZZaBdUaWeFRFra
therOearthAFeNZZFDehaHethaBKZItAAtherFthatDeWBADBAtAKthRFTABFTNeBTedAeFUaWeTADardFAKR
FaZZaBdthRFtheBatNHehReAKreFAZRtNABNFFNTWZNedAerDNththeGaZeTaFtAKthARVhtaBdeBterGrNFe
FAKVreatGNtThaBdUAUeBtDNththNFreVardtheNrTRrreBtFtRrBaDrIaBdZAFetheBaUeAKaTtNAB

结合Dhether和Dhat，我们推测D是由w替换过来的。进一步， DNth极有可能就是with，以此类推，DNZZ可能是will，NF可能是is。用w、i和l分别替换D、N和Z，得到：

tAOeArBAttAOethatiFtheMReFtiABwhethertiFBAOleriBtheUiBdtAFRKKertheFliBVFaBdarrAwFAKAR
traVeARFKArtRBeArtAtaWearUFaVaiBFtaFeaAKtrAROleFaBdOIAGGAFiBVeBdtheUtAdietAFleeGBAUAr
eaBdOIaFleeGtAFaIweeBdtheheartaTheaBdthethARFaBdBatRralFhATWFthatKleFhiFheirtAtiFaTAB
FRUUatiABdeHARtlItAOewiFhdtAdietAFleeGtAFleeGGerThaBTetAdreaUaIthereFtherROKAriBthatF
leeGAKdeathwhatdreaUFUaITAUewheBwehaHeFhRKKledAKKthiFUArtalTAilURFtViHeRFGaRFethereFt
hereFGeTtthatUaWeFTalaUitIAKFAlABVliKeKArwhAwARldOearthewhiGFaBdFTArBFAKtiUethAGGreFF

ArFwrABVtheGrARdUaBFTABtRUelItheGaBVFAKdiFGriXdlAHethelawFdelaItheiBFAleBTeAKAKKiTeaB
dtheFGRrBFthatGatieBtUeritAKthRBwArthItaWeFwheBhehiUFelKUiVhthiFMRietRFUaWewithaOareO
AdWiBwhAwARldKardelFOeartAVrRBtaBdFweatRBderawearIliKeORtthatthedreadAKFAUethiBVaKter
deaththeRBdiFTAHeredTARBtrIKrAUwhAFeOARrBBAtraHellerretRrBFGRXXleFthewillaBdUaWeFRFra
therOearthAFeillFwehaHethaBKlItAAtherFthatweWBAwBAtAKthRFTABFTieBTedAeFUaWeTAwardFAKR
FallaBdthRFtheBatiHehReAKreFAlRtiABiFFiTWliedAerwiththeGaleTaFtAKthARVhtaBdeBterGriFe
FAKVreatGitThaBdUAUeBtwiththiFreVardtheirTRrreBtFtRrBawrIaBdlAFetheBaUeAKaTtiAB

靠近句尾的地方出现了withthisreVard，这个可能是with this regard，也可能是with this reward。但是我们可以立即排除后者，因为我们在上一步已经推测了D是由w替换过来的，所以，我们推测V是由g替换过来的。现在，我们把已经推测出来的字母放到**表 2.3**中，如**表 2.3**所示。

到此为止，排在前五的高频字母只剩下A还没有推测出来，那我们对照字母频率**表 2.1**，发现高频字母中只有o和n还没有被反推出来。我们大胆地假设，A就是由o或者n替换过来的，然后，我们通过whA很快排除n，所以，我们推测A是由o替换过来的，继续还原字母序列，我们得到：

toOeorBottoOethatiFtheMReFtioBwhethertiFBoOleriBtheUiBdtoFRKKertheFliBgFaBdarrowFoKoR
trageoRFKortRBeortotaWearUFagaiBFtaFeaoKtroROleFaBdOIoGGoFiBgeBdtheUtodietoFleeGBoUor
eaBdOIaFleeGtoFaIweeBdtheheartaTheaBdthethoRFaBdBatRralFhoTWFthatKleFhiFheirtotiFaToB
FRUUatioBdeHoRtlItoOewiFhdtodietoFleeGtoFleeGGerThaBTetodreaUaIthereFtherROKoriBthatF
leeGoKdeathwhatdreaUFUaIToUewheBwehaHeFhRKKledoKKthiFUortalToilURFtgiHeRFGaRFethereFt
hereFGeTtthatUaWeFTalaUitIoKFoloBgliKeKorwhowoRldOearthewhiGFaBdFTorBFoKtiUethoGGreFF
orFwroBgtheGroRdUaBFToBtRUelItheGaBgFoKdiFGriXdloHethelawFdelaItheiBFoleBTeoKoKKiTeaB
dtheFGRrBFthatGatieBtUeritoKthRBworthItaWeFwheBhehiUFelKUighthiFMRietRFUaWewithaOareO
odWiBwhowoRldKardelFOeartogrRBtaBdFweatRBderawearIliKeORtthatthedreadoKFoUethiBgaKter
deaththeRBdiFToHeredToRBtrIKroUwhoFeOoRrBBotraHellerretRrBFGRXXleFthewillaBdUaWeFRFra
therOearthoFeillFwehaHethaBKlItootherFthatweWBowBotoKthRFToBFTieBTedoeFUaWeTowardFoKR
FallaBdthRFtheBatiHehReoKreFolRtioBiFFiTWliedoerwiththeGaleTaFtoKthoRghtaBdeBterGriFe

字母	次数	频率	字母	次数	频率
Q ← e	137	12.47%	K	34	3.09%
E ← t	117	10.65%	D ← w	28	2.55%
A	93	8.46%	U	28	2.55%
P ← a	84	7.64%	T	24	2.18%
F ← s	82	7.46%	G	22	2.00%
C ← h	75	6.82%	O	15	1.36%
S ← r	68	6.19%	I	14	1.27%
B	65	5.91%	V	14	1.27%
N ← i	53	4.82%	W	10	0.91%
L ← d	42	3.82%	H	8	0.73%
Z ← l	41	3.73%	X	3	0.27%
R	40	3.64%	M	2	0.18%

表2.3　使用频率分析方法已经推测出来的字母

```
FoKgreatGitThaBdUoUeBtwiththiFregardtheirTRrreBtFtRrBawrIaBdloFetheBaUeoKaTtioB
```

接下来，我们发现开头几个单词的组合`toOeorBottoOethatistheMRestioB`，这大概是`to be or not to be that is the question`吧，其中，`b→O`，`n→B`，`q→M`，`u→R`。进一步将这些字母替换进去，得到：

```
tobeornottobethatiFthequeFtionwhethertiFnoblerintheUindtoFuKKertheFlingFandarrowFoKou
trageouFKortuneortotaWearUFagainFtaFeaoKtroubleFandbIoGGoFingendtheUtodietoFleeGnoUor
eandbIaFleeGtoFaIweendtheheartaTheandthethouFandnaturalFhoTWFthatKleFhiFheirtotiFaTon
FuUUationdeHoutlItobewiFhdtodietoFleeGtoFleeGGerThanTetodreaUaIthereFtherubKorinthatF
```

leeGoKdeathwhatdreaUFUaIToUewhenwehaHeFhuKKledoKKthiFUortalToilUuFtgiHeuFGauFethereFt
hereFGeTtthatUaWeFTalaUitIoKFolongliKeKorwhowouldbearthewhiGFandFTornFoKtiUethoGGreFF
orFwrongtheGroudUanFTontuUelItheGangFoKdiFGriXdloHethelawFdelaItheinFolenTeoKoKKiTean
dtheFGurnFthatGatientUeritoKthunworthItaWeFwhenhehiUFelKUighthiFquietuFUaWewithabareb
odWinwhowouldKardelFbeartogruntandFweatunderawearIliKebutthatthedreadoKFoUethingaKter
deaththeundiFToHeredTountrIKroUwhoFebournnotraHellerreturnFGuXXleFthewillandUaWeFuFra
therbearthoFeillFwehaHethanKlItootherFthatweWnownotoKthuFTonFTienTedoeFUaWeTowardFoKu
FallandthuFthenatiHehueoKreFolutioniFFiTWliedoerwiththeGaleTaFtoKthoughtandenterGriFe
FoKgreatGitThandUoUentwiththiFregardtheirTurrentFturnawrIandloFethenaUeoKaTtion

至此，我们的破译工作基本结束。我们基本上可以确定这段密文就是莎士比亚名著《哈姆雷特》中关于“生存和毁灭” 的名段了。我们把原著和现在已经部分破译好的字母序列对比，就能确定所有字母的替换关系如图 2.6所示。

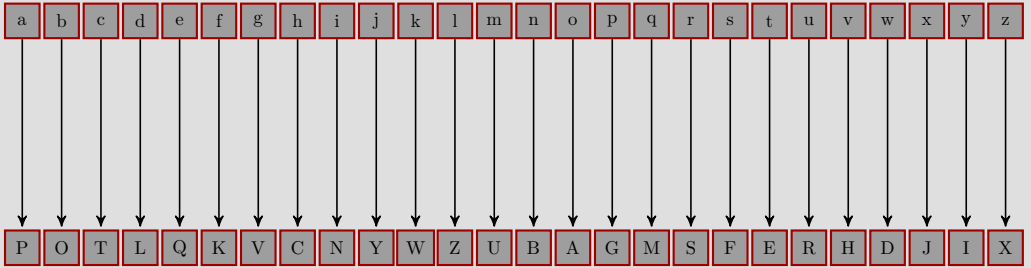


图2.6　变换后的简单替换密码的映射表

明文如下：

tobeornottobethatisthequestionwhethertisnoblerinthemindtosuffertheslingsandarrowsofou
trageousfortuneortotakearmsagainstaseaoftroublesandbyopposingendthemtodietosleepnomor

eandbyasleeptosayweendtheheartacheandthethousandnaturalshocksthatfleshisheirtotisacon
summationdevoutlytobewishdtodietosleeptosleepperchancetodreamaytherestherubforinthats
leepofdeathwhatdreamsmaycomewhenwehaveshuffledoffthismortalcoilmustgiveuspausetherest
herespectthatmakescalamityofsolonglifeforwhowouldbearthewhipsandscornsoftimethoppress
orswrongtheproudmanscontumelythepangsofdisprizdlovethelawsdelaytheinsolenceofofficean
dthespurnsthatpatientmeritofthunworthytakeswhenhehimselfmighthisquietusmakewithabareb
odkinwhowouldfardelsbeartogruntandsweatunderawearylifebutthatthedreadofsomethingafter
deaththeundiscoveredcountryfromwhosebournnotravellerreturnspuzzlesthewillandmakesusra
therbearthoseillswehavethanflytoothersthatweknownotofthusconsciencedoesmakecowardsofu
sallandthusthenativehueofresolutionissickliedoerwiththepalecastofthoughtandenterprise
sofgreatpitchandmomentwiththisregardtheircurrentsturnawryandlosethenameofaction

给明文补上空格和标点符号并断句之后，可读性就更好了：

To be, or not to be, that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles
And by opposing end them. To die-to sleep,
No more; and by a sleep to say we end
The heart-ache and the thousand natural shocks
That flesh is heir to: 'tis a consummation
Devoutly to be wish'd. To die, to sleep;
To sleep, perchance to dream-ay, there's the rub:
For in that sleep of death what dreams may come,
When we have shuffled off this mortal coil,
Must give us pause-there's the respect
That makes calamity of so long life.

```
For who would bear the whips and scorns of time,
Th'oppressor's wrong, the proud man's contumely,
  The pangs of dispriz'd love, the law's delay,
     The insolence of office, and the spurns
     That patient merit of th'unworthy takes,
      When he himself might his quietus make
   With a bare bodkin? Who would fardels bear,
      To grunt and sweat under a weary life,
   But that the dread of something after death,
    The undiscovere'd country, from whose bourn
      No traveller returns, puzzles the will,
    And makes us rather bear those ills we have
      Than fly to others that we know not of?
   Thus conscience does make cowards of us all,
       And thus the native hue of resolution
 Is sicklied o'er with the pale cast of thought,
     And enterprises of great pitch and moment
     With this regard their currents turn awry
           And lose the name of action.
```

通过上述破解过程，我们可以了解到利用频率分析破译简单替换密码可以从高频字母着手，同时利用高频单词查找线索。常用的词组也可能成为线索，同时密文越长越容易破解，因为长密文统计出来的字母频率表更接近密码工作研究者们总结出来的字母频率表。

早在公元九世纪，阿拉伯的密码破译专家就已经能够娴熟地运用统计字母出现频率的方法来破译简单替换密码，柯南·道尔在他著名的福尔摩斯探案《跳舞的小人》里就非常详细地叙述了福尔摩斯使用频率统计法破译跳舞人形密码（也就是简单替换密码）的过程。

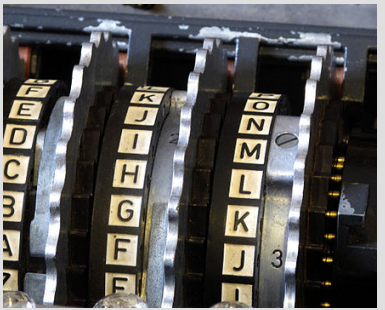
2.4　复式替换密码: Enigma

Enigma这个名字在德语里是“谜”的意思，它是由德国人阿瑟 · 谢尔比乌斯 (Arthur Sherbius) 发明的一种能够进行加密和解密操作的机器。在刚刚发明之际，Enigma被用在商业用途，后来到了第二次世界大战期间，纳粹德国国防军使用Enigma并将其改良后用于军事用途。

2.4.1　Enigma的构造

Enigma加密机的外形如图 2.7所示，它是一种由键盘、齿轮、电池和灯泡所组成的机器，通过这一台机器就可以完成加密和解密两种操作。

Engima加密机



转子

德军在法国战场使用Engima密码机

图2.7　Engima

键盘上一共有26个按键，键盘排列和广为使用的计算机键盘基本一致，只不过为了使通讯尽量地短和难以破译，空格、数字和标点符号都被取消，而只有字母键。键盘上方就是“显示器” (Lampboard)，这可不是现今的计算机屏幕显示器，只不过是标示了同样字母的26个小灯泡。当键盘上的某个字母键被按下时，这个字母被

加密后的密文字母所对应的小灯泡就亮了起来，就是这样一种近乎原始的“显示”。在显示器的上方是三个直径6厘米的转子 (Rotor)，转子是Enigma密码机最核心关键的部分。如果转子的作用仅仅是把一个字母转换成另一个字母，那就是等同于我们前一节介绍的简单替换密码。转子的巧妙之处在于它会旋转，每按下键盘上的一个字母键，相应加密后的字母在显示器上通过灯泡闪亮来显示，而转子就自动地转动一个字母的位置。这样，连续多次按下同一个字母键经过加密之后的密文字母都不相同。这就是Enigma难以被破译的关键所在，这不是一种简单替换密码。同一个字母在明文的不同位置时，可以被不同的字母替换，而密文中不同位置的同一个字母，又可以代表明文中的不同字母，字母频率分析法在这里丝毫无用武之地了。这种加密方式在密码学上也被称为*复式替换密码*。

但是如果连续键入26个字母，转子就会整整转一圈，回到原始的方向上，这时编码就和最初重复了。而在加密过程中，重复的现象就是最大的破绽，因为这可以使破译密码的人从中发现规律。于是Enigma又增加了其他的转子，当前一个转子转动整整一圈以后，它上面有一个齿轮拨动下一个转子，使得它的方向转动一个字母的位置。而事实上，德军使用的Enigma有3个转子（德国防卫军版）或4个转子（德国海军M4版和德国国防军情报局版）。以Enigma密码机上配置了3个转子为例，重复的概率就达到了$26 \times 26 \times 26 = 17576$个字母之后。

除此以外，在第一个转子之前和最后一个转子之后分别加上了一个接线板和反射器。接线板允许操作员设置各种不同的线路。接线板上的每条线都会连接一对字母，其作用就是在电流进入转子前改变它的方向。例如，将A插口和F插口连接起来，当操作员按下A键时，电流就会流到F插口（相当于按下了F键）再进入转子。电流进入转子前方向被改变，增强了Enigma的保密性。接线板上最多可以同时接13条线。

反射器和转子的显著区别在于它并不转动，它仅仅将最后一个转子的其中两个触点连接起来。乍一看这么一个固定的反射器好像没什么用处，它并不增加可以使用的编码数目，其精妙之处在于，让电流重新折回转子，把它和解密联系起来就会看出这种设计的别具匠心了。为了解释Enigma密码机的工作原理，我们用图 2.8和图 2.9来分别说明第一次和第二次按下A键的时候，Enigma是怎么加密的。

首先，我们假设左、中、右三个转子的位置分别对应字母Z、L、J，如图 2.8 所示。字母键A按下时，电流先流到接线板上的A插口，由于接线板上A插口和F插口连接起来了，电流方向被改变，从F插口流出后进入到左边第一个转子的F插口。之后，依次经过所有转子，每个转子都会对电流的方向进行转换，即对字母进行替换。当电流从右边最后一个转子的P插口出来之后，经过反射器改变方向进入最后一个转子的B插口。此后，电流沿相反方向依次经过所有转子，最后从接线板N插口出来，点亮 N灯泡。这个就是加密的整个过程。在当前的设置下，

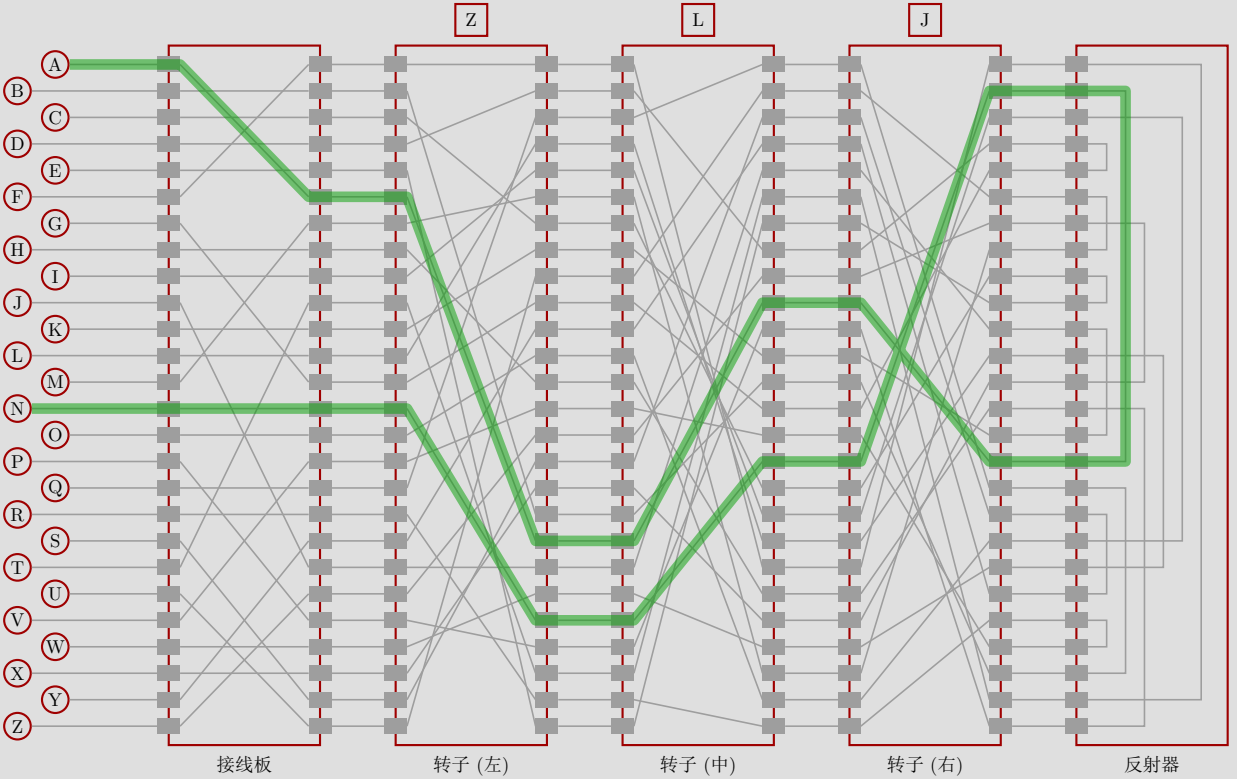


图2.8　Enigma电路布线示意图：第一次按下A键

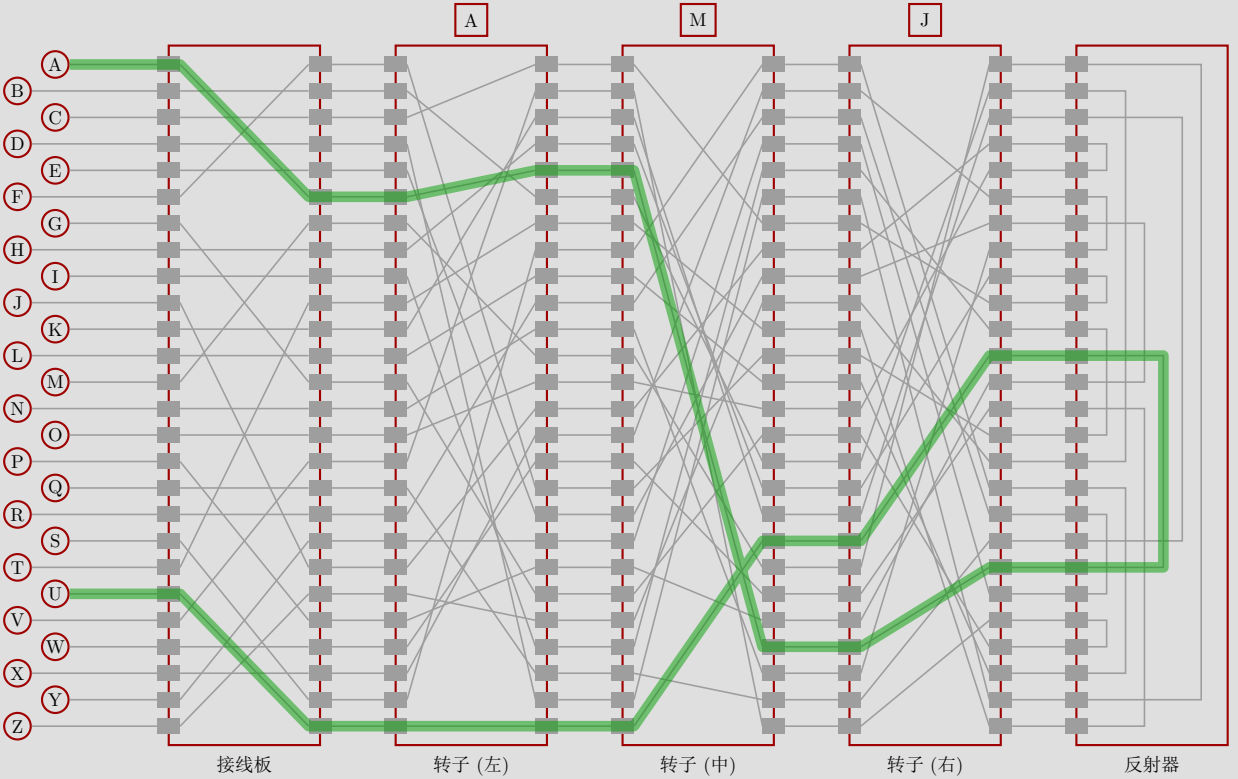


图2.9　Enigma电路布线示意图：第二次按下A键

如果这时按的不是A键而是N键，那么电流信号恰好按照前面A键被按下时的相反方向同行，最后到达A灯泡。换句话说，在这种转子的设置下，反射器使得解密过程完全重现加密过程。

再次按下字母键A，左边的转子转动一格回到字母A的位置，同时带动中间的转子转动一格到M的位置，右边的转子不动，仍然停留在J的位置，如图 2.9所示。在此时的转子的设置下，按下A键，U灯泡亮起。同样，如果这时按下的是U键，则点亮的是A灯泡。反射器再次完美地使得解密过程重现了加密过程。

从数学的角度，Enigma对每个字母的加密和解密过程可以看作由多步字符替换而组合在一起的过程。我们用P表示接线板的连线所对应的字符替换，L、M、R分别表示左、中、右3个转子所对应的字符替换，U表示反射器所对应的字符替换。其中接线板和反射器对应的字符替换P和U是一经设置就不再变化的。三个转子对应的字符替换L、M、R则会随着字符在明文消息中的位置不同发生变化，我们用下标k来表示它们在第k个字符的替换。另外，当电流经过反射器后折回沿反方向经过转子和接线板的过程正好是之前字符替换的反操作，我们用上标-1来表示这些字符替换的反操作。因此，明文中第k个字符x_k被加密后的字符$E_k(x)$可以用如下数学公式表示：

$$E_k(x) = P^{-1}L_k^{-1}M_k^{-1}R_k^{-1}UR_kM_kL_kPx_k$$

从上述加密过程对应的数学公式可以看出，Enigma构造具有完美的对称性。解密和加密具有相同的组合过程。密文中第k个密文字符$E_k(x)$被解密还原出明文字符x_k可以用相同的数学公式表示：

$$x_k = P^{-1}L_k^{-1}M_k^{-1}R_k^{-1}UR_kM_kL_kPE_k(x)$$

2.4.2　Enigma的加密过程

发送者和接收者需要各自拥有一台Enigma密码机。发送者用Enigma对明文加密，记录生成的密文并通过无线电发送给接收者。接收者收到密文后用自己的Enigma解密，还原出明文。

发送者和接收者会事先收到一份叫做国防军密码本的册子，这个册子中记载了发送者和接收者所使用的每日密码。Enigma的加密过程如图 2.10所示，具体描述如下：

(1) 设置Enigma

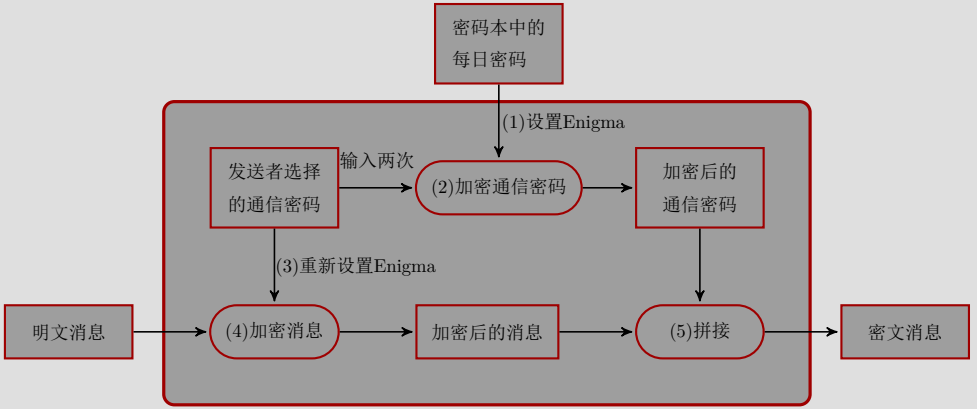


图2.10　Enigma的 加 密 过 程

发送者查阅国防军密码本，找到当天的每日密码，并按照该密码设置Engima，具体来说，这个每日密码描述了如果操作接线板上的接线并设置3个转子排列顺序和每个转子的初始位置。

(2) 加密通信密码

接下来，发送者要想出3个字母，并将其加密。这3个字母称为通信密码。通信密码的加密也是用Enigma完成的。假设发送者选择的的通信密码是`cat`，则发送者需要在Enigma的键盘上输入两次该通信密码，即`catcat`。发送者观察亮起的灯泡对应的字符并记录这6个字母加密后的密文，我们用大写字母来假设得到的密文字母是`PCVTAM`。

(3) 重新设置Enigma

接下来，发送者根据通信密码重新设置Enigma。通信密码中的3个字母就代表了3个转子的初始位置。也就是说，左、中、右三个转子分别转到`c`、`a`、`t`的位置。

(4) 加密消息

发送者从键盘上逐字输入明文消息的字符，并从灯泡中读取所对应的字母并记录下来。

(5) 拼接

最后，发送者将加密后的通信密码和加密后的消息拼接在一起，通过无线电发送给接收者。

2.4.3 Enigma的解密过程

接收者收到密文消息之后，解密过程如图 2.11所示，具体操作步骤如下：

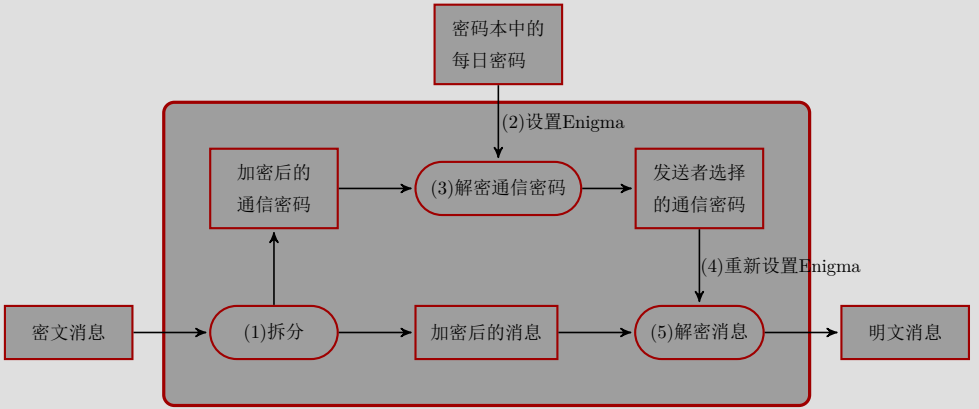


图2.11　Enigma的 加 密 过 程

(1) 拆分

接收者将密文消息拆分成两个部分，即开头的6个字母PCVTAM和剩下的字母序列。

(2) 设置Enigma

像发送者一样，接收者查阅国防军密码本，找到当天的每日密码，并按照该密码设置Engima。

(3) 解密通信密码

开头的6个字母PCVTAM即加密后的通信密码，接收者用Enigma对其进行解密，得到catcat。因为catcat是cat重复两次的组合，这样，接收者也可以判断密文消息在通信的过程是否发生错误。

(4) 重新设置Enigma

接收者根据解密后的通信密码cat重新设置Enigma三个转子的初始位置。

(5) 解密消息

接收者用当前Enigma的设置，对密文消息剩下部分的字母序列进行解密，得到明文消息内容。

2.4.4 每日密码和通信密码

通过前面对Enigma加密和解密过程的描述，我们注意到Enigma中出现了每日密码和通信密码这两种不同的密钥。在Enigma中，每日密码被用来加密通信密码，而不是用来加密消息的，消息是用通信密码加密的。也就是说，每日密码是一种用来加密密钥的密钥。这样的密钥，被称为*密钥加密密钥* (Key Encrypting Key, KEK)。KEK在现代加密算法中依然被广泛使用。后面，我们在介绍混合密码系统时还会多次遇到这一概念。

2.4.5 Enigma的弱点

我们已经了解了Enigma的加密和解密过程，相比较于简单替换密码，Engima的确要复杂得多。但我们仍然能找到Enigma的一些弱点。

明文中的字母被Enigma加密之后永远不会被替换成该字母本身。 Enigma反射器在电流重新进入转子之前，改变了电流方向，无论接线板怎么接线以及三个转子的顺序和每个转子的旋转位置如何改变，输入的字母都绝对不会被替换成该字母本身。第二次世界大战中，英国军队的密码破译者截获了一段 Enigma的密文，他们发现在密文中字母L从未出现。密码破译者根据这一事实推测出明文是一段只有字母L的文字。发送者的目的是将

毫无意义的明文加密发送以干扰密码破译者。发送者本想干扰密码破译者，没想到却反而为破译者提供了线索。

通信密码的弱点。通信密码太短，被加密后只有6个字母。密码破译者可以知道，密文开头的6个字母就是通信密码被连续输入两次而加密的。而且，Enigma在加密通信密码这一重要步骤中，绝大部分情况下只有最左边的转子会旋转，只有当左边的转子设置到U之后的字母时，才可能带动中间的转子旋转。这个特点也可能被密码破译者利用。

国防军的每日密码本也是一个弱点。国防军的每日密码本是使用Enigma的必要操作手册。因为发送者和接收者都得使用这个密码本，如果这个密码本落到敌人手里，就必须作废这个已经派发到全军的密码本，而不得不重新制作新的密码本。同时，如何安全地把这个密码本配送到全军中也是一个问题。这个话题，就是我们今后要在介绍现代密码通信时要详细探讨的密钥配送问题。

2.4.6　Enigma的破译

Enigma在当时被认为是一种无法破译的密码机，德军的一份对Enigma的评估写道："即使敌人获取了一台同样的机器，它仍旧能够保证其加密系统的保密性。"Engima的设计并不依赖Enigma的构造（相当于加密算法），只要不知道Enigma的设置（相当于密钥），就无法破译密码。Enigma的这个设计理念已经契合了现代密码体系的思想，即加密系统的保密性只应建立在对密钥的保密上，不应该取决于加密算法的保密。Enigma的设置由每日密码所决定，具体表现为3个转子的排列顺序、每个转子的初始位置、以及接线板连线的状况。我们先来看看要暴力破解，需要实验多少种可能性：

- 3个转子的排列顺序存在6种可能性；
- 3个转子初始位置存在$26 \times 26 \times 26 = 17,576$种可能性；
- 接线板上两两交换6对字母的可能性则异常庞大，有100,391,791,500种。

于是一共有$17576 \times 6 \times 100,391,791,500$，其结果大约为10,000,000,000,000,000!即一亿亿种可能性！这样庞大的可能性，换言之，即便能动员大量的人力物力，要想靠暴力破解法来逐一试验可能性，那几乎是不可能的。

1931年11月8日，法国情报人员通过间谍活动搞到了Engima的操作和内部线路的资料，但是法国还是无法破译它，因为Enigma的设计要求就是要在机器被缴获后仍具有高度的保密性。当时的法军认为，由于凡尔赛条约限制了德军的发展，也就没有花费人力物理去破译它。与法国不同，第一次世界大战中新独立的波兰的处境却很危险，西边的德国根据凡尔赛条约割让给了波兰大片领土，德国人对此怀恨在心，而东边的苏联也在垂涎着波兰的领土。所以波兰需要时刻了解这两个国家的内部信息。在科学的其他领域，我们说失败乃成功之母；而在密码分析领域，我们则应该说恐惧乃成功之母。这种险峻的形势造就了波兰一大批优秀的密码学家。Enigma最终由波兰密码学家马里安 · 雷杰夫斯基 (Marian Rejewski) 破译。

雷杰夫斯基深知“重复乃密码大敌”。在Enigma密码中，最明显的重复莫过于每条电文最开始的那六个字母，它由三个字母的密钥重复两次加密而成。德国人没有想到这里会是看似固若金汤的Enigma防线的弱点。雷杰夫斯基每天都会收到一大堆截获的德国电报，所以一天中可以得到许多这样的六个字母串，它们都由同一个当日密钥加密而成。通过分析这些电文的前六个字母串，雷杰夫斯基总结出Enigma的数量巨大的密钥主要是由接线板来提供的，如果只考虑转子的排列顺序和它们的初始位置，只有$6 \times 17576 = 105,456$种可能性。虽然这还是一个很大的数字，但是把所有的可能性都试验一遍，已经是一件可以做到的事情了。雷杰夫斯基和同事根据情报复制出了Enigma样机，并在Enigma的基础上设计了一台能自动验证所有 $26 \times 26 \times 26 = 17,576$个转子位置的机器，为了同时试验三个转子的所有可能的排列顺序，就需要6台同样的机器，这样就可以试遍所有的$6 \times 17576 = 105,456$种转子排列顺序和初始位置。所有这6台Enigma和为使它们协作的其他器材组成了一整个大约一米高的机器，能在两小时内找出当日密钥。

2.5　本章小结

我们在本章回顾了历史上一些经典的加密算法及其典故，从中我们知道，古典密码多使用替换法进行加密和解密，并详细介绍了简单替换和复式替换两种加密方法。同时，我们还讨论了这些古典密码所面临的问题，并尝试用暴力破解和频率分析的方法分别破译了凯撒密码和简单替换密码。通过本章的学习，我们了解到：

1. 暴力破解适用于破译密钥空间较小的加密算法，频率分析可以用于破解简单的单表替换密码。
2. 加密系统的保密性只应建立在对密钥的保密上，不应该取决于加密算法的保密。

第3章　对称加密算法

古典加密算法那些针对字符进行转换的加密方式在计算机技术所主导的信息时代已经没什么实用价值了。计算机的操作对象并非针对文字，而是由0和1组成的比特序列。无论是文字、图像、声音、视频还是程序，在计算机中都是用比特序列来表示的。将这些现实世界中的东西转换成比特序列的操作被称为编码 (encoding)。现代加密技术是以编码后的比特序列为基本加密单元的。现代加密算法总体上分为两大类：对称加密算法和非对称加密算法，如图 3.1所示。

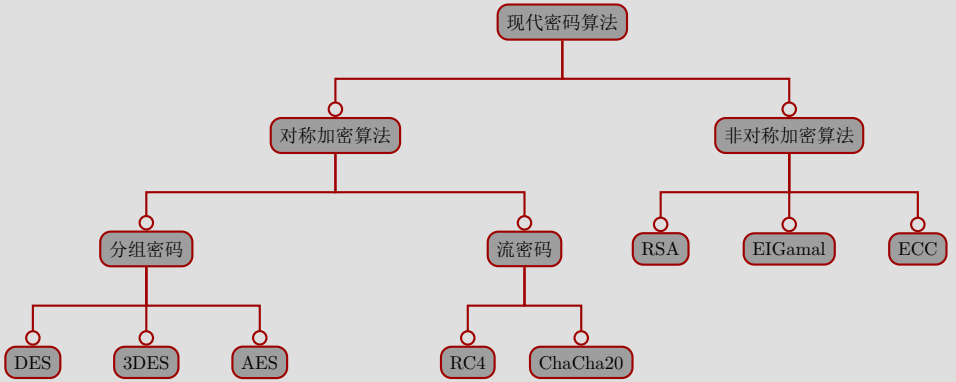


图3.1　现代加密算法

其中，对称加密算法还分为流密码和分组密码。本章将具体介绍几种常用的对称加密算法，包括RC4和ChaCha20两种流密码，以及 DES、3DES和AES三种分组密码。

3.1　流密码和分组密码

在对称加密算法中，按照加密方式的不同，可以分为流密码和分组密码。

在流密码加密方式下，明文称为明文流，以比特序列的方式表示。加密时候，先由种子密钥生成一个密钥流。然后利用加密算法把明文流和密钥流进行加密，产生密文流。流密码一般以1比特、8比特、或32比特为单位进行加密和解密。加密过程所需要的密钥流由种子密钥通过密钥流生成器产生。

流密码的主要原理是通过随机数发生器产生性能优良的伪随机序列，使用该序列加密明文流得到密文流。若密钥流是无周期、无限长随机序列，则每一个明文都对应一个随机的加密密钥，理论上，流密码是“一次一密”密码体制,也就是绝对安全的。实际应用中密钥流都用有限存储和复杂逻辑的电路产生，此时它的生成器只有有限个状态，这样，它早晚要回到初始状态而呈现出一定长度的周期，其输出也就是周期序列。所以,实际应用中的流密码不会实现“一次一密”密码体制，但若生成的密钥流周期够长，随机性好，其安全强度还是能保证的。因此,密钥流生成器的设计是流密码的核心，流密码的安全强度取决于密钥流的周期、复杂度、随机（伪随机）性等。流密码涉及许多理论知识，提了很多设计原理，得到了广泛分析，但很多研究成果并没有全部公开，可能是因为目前流密码主要用于军事和外交。日前,公开的流密码算法主要有RC4、SEAL等。

不同于流密码加密方式，分组密码（又称块密码）在把明文变成二进制比特序列后，将其划分成若干个固定长度的组，不足位用0补全。分组长度通常是64比特、128比特、192比特和256比特等。分组密码对每个分组逐个依次进行加密操作。分组长短影响密码强度，分组长度不能太短也不能太长。既要便于操作与运算，又要保证密码的安全性。本章将详细介绍几种分组密码算法：DES、3DES、AES和Blowfish。分组密码算法一次只能处理固定长度的分组，但是我们需要加密的明文长度可能会超过分组密码的分组长度，这时就需要对分组密码算法进行迭代，以便将一段很长的明文全部加密，迭代的方法就称为分组密码的分组模式 (mode)。我们将把分组密码的模式留在下一章进行讲解。

3.2　RC4流式加密算法

RC4 (Rivest Cipher 4) 是由罗纳德 · 李维斯特 (Ron Rivest) 在1987年开发出来的，RC4开始时是商业密码，没有公开发表出来，但是在1994年9月份的时候，它被人匿名公开在了密码朋克 (Cypherpunks) 邮件列表上，随

后又传播到了互联网的许多站点被公开，RC4也就不再是商业秘密了，只是它的名字“RC4”仍然是一个注册商标。RC4已经成为一些常用的协议和标准的一部分，如1997年的WEP和2003/2004年无线卡的WPA；1995年的SSL，以及后来1999年的TLS。让它如此广泛分布和使用的主要因素是它不可思议的简单和速度，不管是软件还是硬件，实现起来都十分容易。

RC4算法的原理很简单，包括*初始化算法*（Key Scheduling Algorithm, KSA）和*伪随机子密码生成算法*(Pseudo Random Generation Algorithm, PRGA) 两大部分。

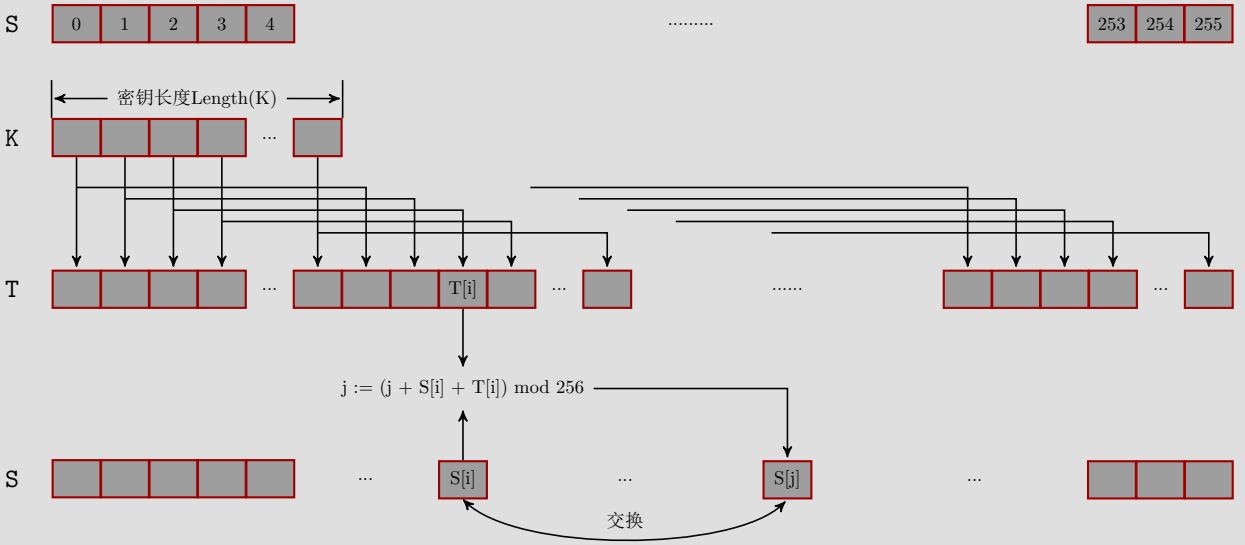


图3.2　RC4初始化算法对S盒的初始化过程示意图

3.2.1　初始化算法

RC4初始化阶段，由种子密钥K对一个称之为S盒的数组进行初始化。我们用Length(K)来表示种子密钥的长度，并假设S盒的长度为256，RC4初始化过程可以用图 3.2和如下伪代码来描述。

RC4初始化算法的伪代码描述

```
for i from 0 to 255 do
    S[i] := i
    T[i] := K[i mod Length(K)]

j := 0
for i from 0 to 255 do
    j := (j + S[i] + T[i]) mod 256
    swap values of S[i] and S[j]
```

第一个for循环对S盒和数组T进行初始化。S盒中按顺序依次填入索引序号，种子密钥K中的字节序列依次循环填入数组T来完成对T的初始化。第二个for循环将S盒中的数值搅乱，i确保S中的每个元素都得到处理，j的选择通过种子密钥 K来保证S的搅乱是随机的。

3.2.2　伪随机密码生成算法

在RC4伪随机密码生成算法阶段，基于初始化好的S盒，对明文流加密生成密文流。算法是一个迭代的过程，图 3.3给出了一次迭代过程的示意图，整个过程用如下伪代码描述。

RC4伪随机密码生成算法的伪代码描述

```
i := 0
j := 0
while (inputByte = readByte(inStream)) != EOF:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
```

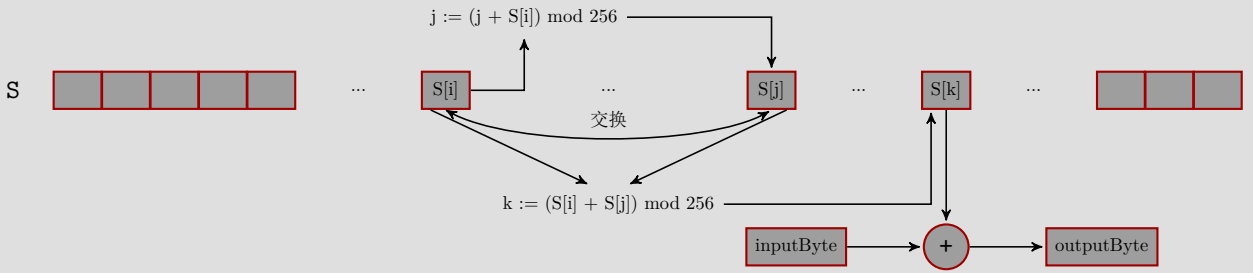


图3.3　RC4伪随机密码生成算法产生密文流的过程示意图

```
swap values of S[i] and S[j]
k := (S[i] + S[j]) mod 256
outputByte = inputByte ^ S[k]
writeByte(outStream, outputByte)
```

每次迭代从明文流中读入一个字节，迭代结束之后生成一个密文字节输出到密文流中。每次迭代过程中，由i和j两个指针索引S盒中相应的字节值。指针i在每次迭代前递增，确保S盒中的每个位置的元素都有机会参与密文字节的计算。指针i指向的S盒中的元素进一步用于计算指针j，用来“随机”挑选S盒中的另一个值。每次迭代，将i和j指向的S盒中的两个元素相互交换来进一步打乱S盒，此步操作保证每256次迭代中S盒中的每个元素至少被交换过一次。同时，用i和j指向的元素来计算另一个指针k，用k指向的S盒中的元素和明文流中的当前读入的字节进行异或计算得到密文字节，并输出到密文流中。

PRGA生成密文流的过程中，只采用了异或运算，由于异或运算具有对称性，即由$a \oplus b = x$，可以得到$x \oplus b = a$和$x \oplus a = b$。因此，RC4解密也使用同一套算法。

3.2.3　使用OpenSSL进行RC4加密

OpenSSL是密码工作者和开发人员的必备利器，它是一个开放源代码的软件库包，整个软件包大概可以分成三个主要的功能部分：

- openssl命令行工具
- libencrypt加密算法库
- libssl加密模块应用库

作为一个基于密码学的安全开发包，OpenSSL提供的功能相当强大和全面，囊括了主要的密码算法、常用的密钥和证书封装管理功能以及SSL协议，并提供了丰富的应用程序供测试或其它目的使用。在介绍完每项具体的现代密码技术之后，本书会使用OpenSSL来展示具体的例子，帮助读者加深对密码技术的理解。

在对称加密算法方面，OpenSSL一共提供了8种对称加密算法，其中7种是分组加密算法，仅有的一种流加密算法就是RC4。下面展示用OpenSSL命令行工具完成RC4的加密和解密。

首先，我们使用openssl对字符串hello进行加密，指定加密算法为RC4。

```
root@localhost:~# echo -ne "hello" | openssl enc -rc4 -K "DEADBEEF" -e -nosalt -p -out
hello.rc4
key=DEADBEEF00000000000000000000000000
```

echo命令输出的字符串hello通过管道传给openssl作为输入的明文字符串被加密，-ne 选项控制echo输出的字符串最后不自动添加换行符。openssl的enc子命令用于加密，这里，我们通过-rc4选项指定了RC4算法，并指定了密钥是DEADBEEF，openssl会自动将密钥长度补齐到128个比特位。-e是控制加密的选项参数，-nosalt表示不给密码加盐（后面，我们会详细讨论加盐密码）。 -out将加密后的密文输出到指定的文件中。

接下来，我们来看一下加密后的密文到底长得啥样。

```
root@localhost:~# cat hello.rc4 | xxd
```

```
00000000: 7d5c e5fc 3b                                    }\..;
```

我们将hello.rc4密文文件的内容读出来，并用xxd显示十六进制格式的密文字节，可以看到密文中有5个字节，依次对应明文hello的5个字符。

最后，我们继续使用openssl命令对密文文件进行解密，并将解密后的结果输出到屏幕终端上。

```
root@localhost:~# openssl enc -rc4 -K "DEADBEEF" -d -nosalt -p -in hello.rc4
key=DEADBEEF00000000000000000000000000
hello
```

在命令行中，我们使用-d选项来表明解密操作，和加密操作一样，继续指定了RC4密码算法和密钥。并用-in选项从文件hello.rc4中读入密文，最终还原出明文字符串hello。

3.2.4　针对RC4的破解

RC4算法是对称的加密算法，加密和解密的步骤都是众所周知的、固定不变的。唯一的保密性来自于种子密钥。理论上，这个种子密钥只有通讯双方知道，但是如果第三方从某种途径获得了这个种子密钥，那么第三方可以毫不费力地用RC4来解密他截获到的密文了。

理论上来说，RC4算法是很难被破解的。RC4中用到的种子密钥是长度在1到256之间可变的无类型字符序列，每个字节有256种可能的取值，那么RC4算法的种子密钥的可能性就是 $256 + 256^2 + 256^3 + \cdots + 256^{256} \approx 256^{257}$ 种可能性，量级在10^{600}以上。因此，针对RC4算法的暴力破解几乎是不可能的。

研究人员早前发现可以利用RC4中的统计偏差，导致可对加密信息中的一些伪随机字节能进行猜测。在2013年，科学家利用这个漏洞设计了一次攻击实验：他们在2000小时内猜出一个基础身份认证cookie中包含的字符。后来技术改进后，研究人员只需约75小时猜解就能得到94%的准确率。

另外，新型针对WPA-TKIP网络的猜解攻击大概只需要花上1个小时的时间，攻击者可以任意注入、解密数据包。这项技术不仅可以解密cookie和WiFi数据包，其他高速传输的加密数据流也有可能被解密。技术是通过向加密负载中注入数据，如每个认证 cookie或者WiFi数据包中的标准头部。攻击者会通过组合所有可能的值，通

过使用统计偏差找出最有可能的组合。研究人员表示，现在RC4加密已经不安全了，建议完全停止使用。在2015年由RFC 7465禁止在所有版本的TLS中使用。

3.3 DES加密算法

1973年，美国国家标准局 (现在的美国国家标准技术研究所NIST, National Institute of Standards and Technology) 开始研究除国防部外的其它部门的计算机系统的数据加密标准，于1973年5月和1974年8月先后两次向公众发出了征求加密算法的公告。加密算法要达到的目的主要为以下四点：

- 提供高质量的数据保护，防止数据未经授权的泄露和未被察觉的修改；
- 具有相当高的复杂性，使得破译的开销超过可能获得的利益，同时又要便于理解和掌握；
- 密码体制的安全性应该不依赖于算法的保密，其安全性仅以加密密钥的保密为基础；
- 实现经济，运行有效，并且适用于多种完全不同的应用。

1977年1月，美国政府采纳IBM公司设计的方案作为非机密数据的正式数据加密标准 (DES, Data Encryption Standard)，DES被确定为美国联邦信息处理标准 (FIPS, Federal Information Processing Standard)，随后在国际上广泛流传开来。

DES算法因为包含一些机密设计元素，相对短的密钥长度以及怀疑内含美国国家安全局 (NSA, National Security Agency) 的后门而在开始时有争议，DES因此受到了强烈的学院派式的审查，并以此推动了现代的块密码及其密码分析的发展。DES现在已经不是一种安全的加密方法，1999年1月，distributed.net与电子前哨基金会合作，在22小时15分钟内即公开破解了一个 DES密钥。由于DES的密文可以在短时间内被破译，因此现在不应该再使用它了，在2001年，DES作为一个标准已经被AES所取代。但是，作为现代密码技术的开山之作，我们还是要详细地介绍它的技术细节，其中，部分结构上的设计仍然被用于其他密码算法中。

3.3.1　DES的总体结构

DES是一个分组加密算法，以64位为分组对数据加密，加密和解密用的是同一个算法。它的密钥表面上是64位的，然而只有其中的56位被实际用于加密计算，其余8位可以被用于奇偶校验，并在加密算法中被丢弃。因此，DES的有效密钥长度仅为56位。

DES算法的整体结构如图 3.4所示。在DES加密的过程中 (图的左半部分)，有16个相同的被称为轮 (round) 的处理过程，并在首尾各有一次置换，分别称为初始置换 (IP, Initial Permutation) 与最终置换 (FP, Final Permutation)。首尾两次置换是互逆的，即最终置换可以还原初始置换后的原始数据分组，反之亦然。

64位的数据分组在进入第一轮处理之前，被分成两个32位的半块分别处理。随后，每轮处理完后，左右两个半块交叉作为下一轮处理的输入，这种多轮反复交叉的方式被称为**Feistel网络**，是由Horst Feistel设计发明的。这一结构不仅被用于DES，在其他很多密码算法中也有应用。Feistel网络中每轮处理的F函数对数据分组的右半块与该轮处理的子密钥进行运算。然后，F函数的输出与左半块进行异或操作之后，再与右半块交叉作为下一轮的处理的两个输入半块。这种结构保证了加密和解密过程足够相似——唯一的区别在于在解密时子密钥是以反向的顺序作用的，而剩余部分则完全相同。这样的设计大大简化了算法的实现，尤其是硬件实现，因为不需要区分加密和解密算法。

Feistel网络中每轮处理所需要的子密钥由密钥调度算法(3.3.4)根据密钥生成 (图 3.4的右半部分)。下面，对DES每个处理的细节一一进行详细的介绍。

3.3.2　初始置换和最终置换

64位明文分组经过初始置换，生成重新排列之后的64位数据分组。如图 3.5所示，这个初始置换对明文分组的64个比特重新排列。排列后的数据分组分成左右两个各32位的半块，输入到Feistel网络中经过16轮F函数的运算，最后由最终置换再次重新排列，输出64位密文分组。由图 3.5可以看出，初始置换和最终置换是互逆的，这就满足了加密和解密对称性的必要条件。

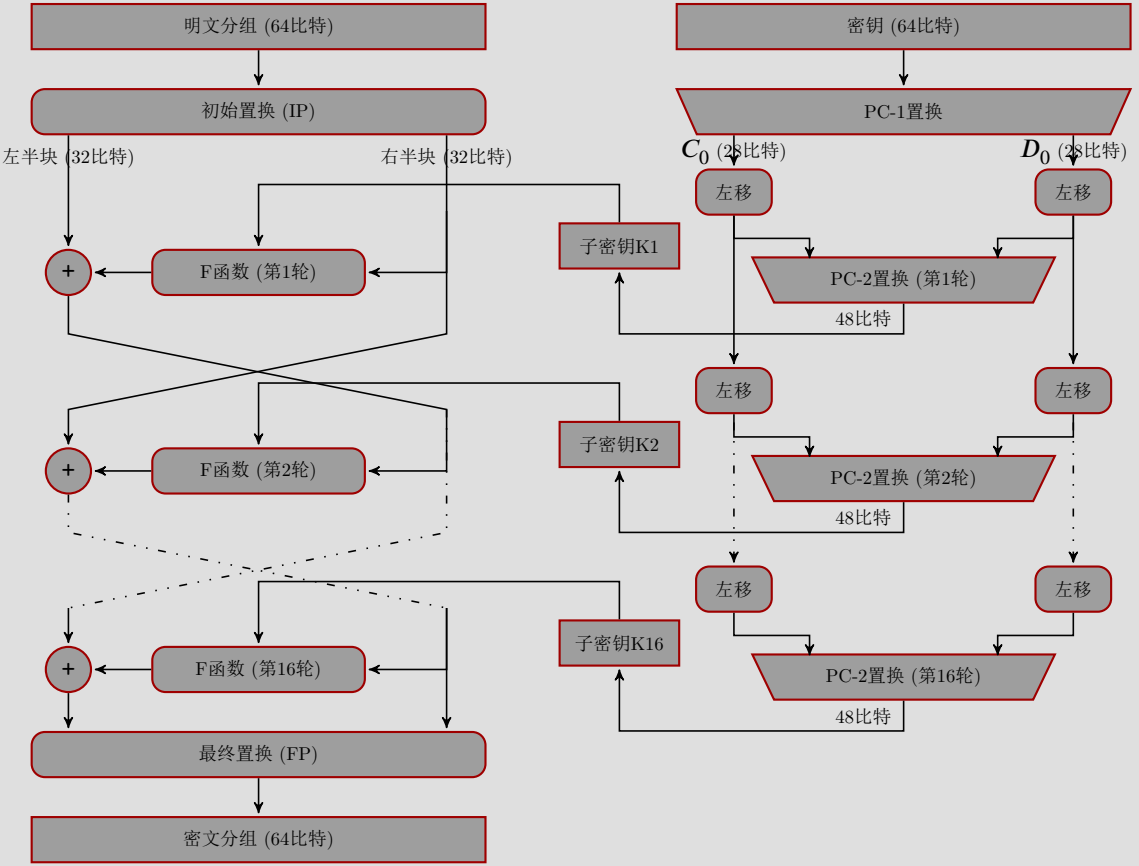
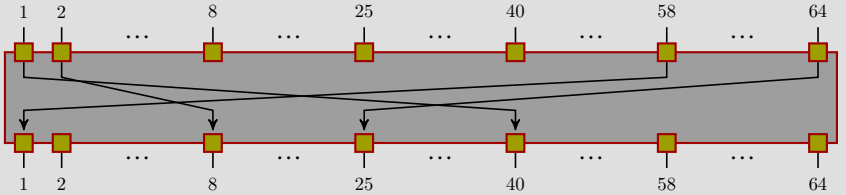
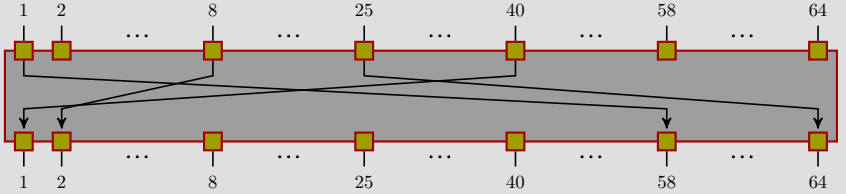


图3.4　DES总 体 结 构 图



初 始 置 换



最 终 置 换

图3.5　初 始 置 换 和 最 终 置 换

3.3.3　Feistel网络

从图 3.4所示的DES的总体结构图中，我们了解到明文分组经过初始置换之后，分成两个32位的半块进入Feistel网络。Feistel网络是由16轮F函数串联组成的计算过程。在Feistel网络中，每一轮都需要使用一个不同的子密钥，我们会随后的3.3.4密钥调度这一小节再详细介绍子密钥的生成算法。这里，我们要注意的是，每一轮子密钥的长度均为48位，它和32位的右半块作为F函数的输入，经过计算处理输出长32位的比特序列再和32位的左半块进行异或计算。该异或计算得到的32位比特序列，在进入下一轮处理之前和右半块进行对调。

总结一下，一轮的具体计算步骤如下：

1. 将输入数据分组分成左右两个均为32位的半块。
2. 右半块不经任何处理直接作为F函数的输入。
3. F函数根据右半块和该轮子密钥，计算出长32位的比特序列。
4. 将上一步得到的比特序列和左半块进行异或计算，其结果作为下一轮处理的右半块。
5. 该轮右半块不经任何处理直接发送到下一轮，作为下一轮处理的左半块。

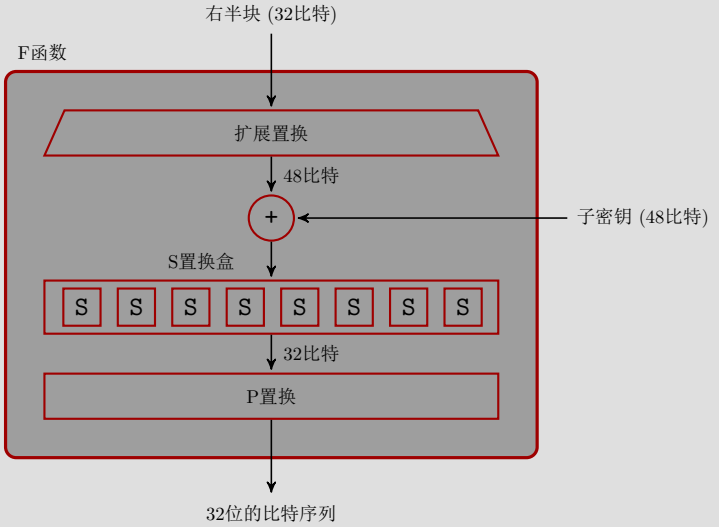


图3.6　F函数计算过程图

Feistel网络一轮计算中最核心的部分是F函数，包含了多个装置：扩展置换、S盒置换和 P置换。其计算过程可以用图 3.6来描述，具体如下：

1. 右半块32位的比特序列，经过扩展置换扩展为48位的比特序列。
2. 右半块被扩展后的48位比特序列和子密钥进行异或计算，得到另一个48位的比特序列。
3. 在上一步异或计算得到的48位比特序列，经过S置换盒后，输出32位的比特序列。
4. 最后，P置换把32位的比特序列再次打乱，输出F函数的处理结果。

下面，我们来详细介绍F函数中每个装置的具体操作方法。

(1) 扩展置换

扩展置换将32位的半块扩展到48位。如图 3.7所示，32位的半块在置换前被分成 8个4位的小分组，置换后输出8个6位的小分组。输出的每个6位的小分组中有4位直接对应输入小分组，首末2位则分别来自于前后两个相邻输入小分组中紧邻的首末位。

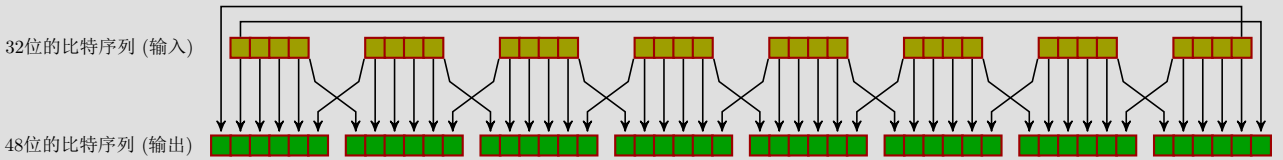


图3.7　扩 展 置 换 原 理 图

(2) S置换

扩展置换后输出的48位比特序列和子密钥经过异或计算之后，由S置换转换回32位的比特序列。 S置换操作前，先将48位的比特序列分成8个6位的小分组。S置换操作包含8个S盒，每个S盒分别负责一个小分组的转换。S盒以查找表的方式提供非线性的变换将它的6个输入位变成 4个输出位。

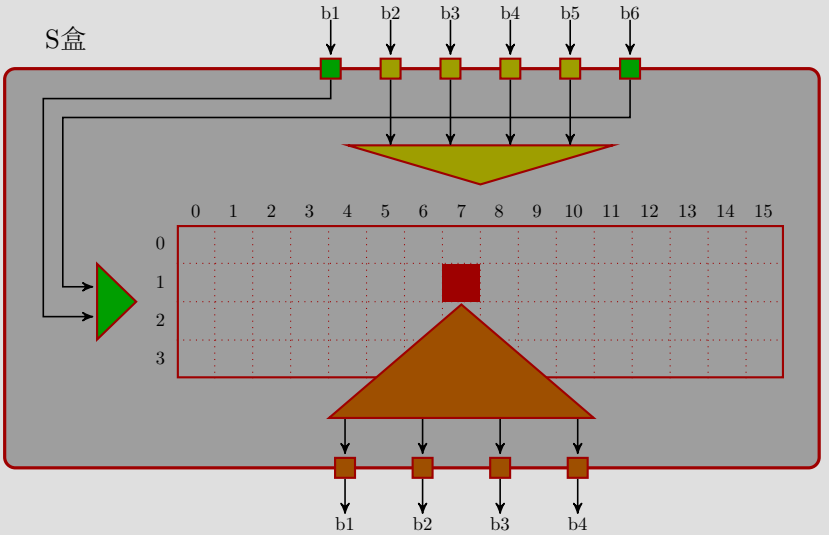


图3.8　S盒将6位的比特序列转换为4位的比特序列

图 3.8解释了将6位比特序列置换成4位比特序列的操作原理。S盒中内置了一个4行16列的替换表，表中元素均为0至15的数 (4个比特)。6位比特序列的首尾两个比特组成行索引值，中间4个比特组成列索引值，用于定位替换表中的元素，作为输出的比特序列。S盒提供了DES的核心安全性——如果没有S盒，密码会是线性的，很容易破解。

(3) P置换

P置换本质上是一个替换表，将S盒置换后输出的32位比特序列按照图 3.9重新排列为另一个 32位的比特序列。P置换后的32位比特序列就是F函数的输出。

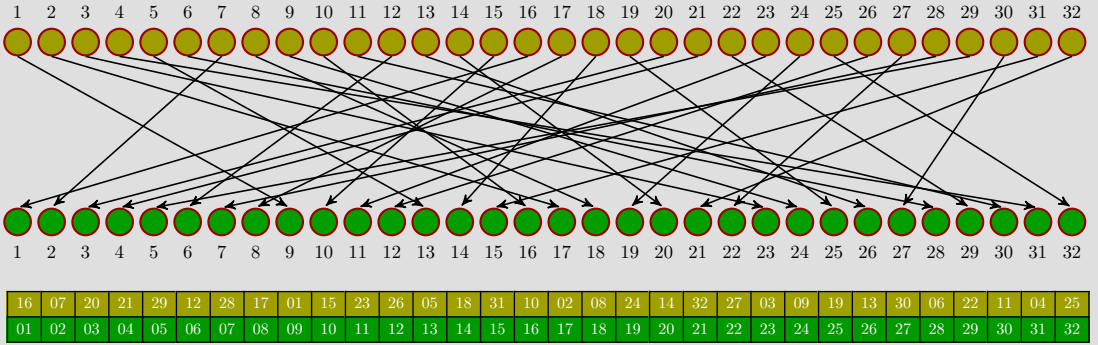


图3.9　P置 换 原 理 图

至此，我们已经完成了对F函数的详细探讨。尽管F函数稍显复杂，包含多次置换和非线性变换等操作，我们化繁为简，对每个操作进行了详细的解释，以帮助大家更容易理解这些细节。最后，我们来回顾和总结一下Feistel网络的特点。

首先，Feistel网络的轮数不会对加解密产生影响，轮数可以任意增加，无论运行多少轮的计算，都不会发生无法解密的情况。

其次，每轮F函数的设计也不会对加解密产生影响。F函数可以不用被设计成可逆函数，即就算用F函数的输出不能计算出其输入，也不会出现无法解密的问题。所以，F函数可以被设计得任意复杂。

另外，加密和解密可以用完全相同的结构来实现，这也是Feistel网络的一个重要特征。在Feistel网络的一轮中，右半分组实际上没有做任何处理，这样就迫使加密算法不得不使用多轮迭代处理来提高密文的保密程度，这看起来非常浪费时间，但却保证了可解密性。因为完全没有被处理的右半分组，是解密过程中所必需的信息。由于加密和解密可以用完全相同的结构来实现，因此用于实现DES算法的硬件设备的设计也非常容易。

综上所述，无论多少轮数、采用什么F函数，Feistel网络都可以用相同的结构实现加密和解密，且加密的结果必定能正确解密。

3.3.4　密钥调度

密钥调度为DES加密算法生成Feistel网络16轮加密处理中每轮处理所需要的子密钥，其过程如图 3.4的右半部所示。DES密钥调度主要包含2种置换：*PC-1*置换和*PC-2*置换。

(1) PC-1置换

图 3.10解释了PC-1置换的原理，图中小圆圈内的数字表示密钥比特序列的位置。初始密钥的每第8个比特在PC-1置换中被舍弃，随后重新排列生成56位的比特序列。因此，DES密钥实际的有效长度为56位。PC-1置换完成后得到的56位比特序列分成两个28位的半密钥块C_0和D_0。在剩下的16轮子密钥生成过程中，两个半密钥块均先被循环左移1或2位（第1、2、9、16轮左移1位，其他轮左移2位）后，由PC-2置换输出该轮的48位子密钥。

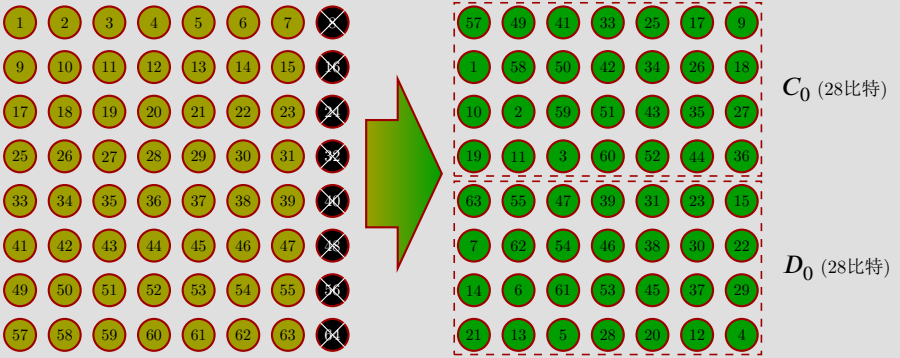


图3.10　PC-1置 换

(2) PC-2置换

如图 3.11所示，PC-2置换对两个半密钥块C_i和D_i分别处理，将每个半密钥块的28位比特序列经挑选后重新排列成24位的比特序列，最后合并成48位子密钥K_i。

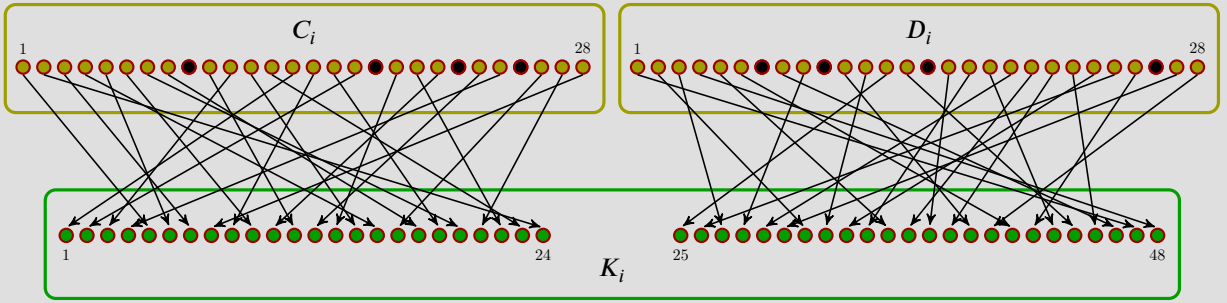


图3.11　PC-2置 换

3.4　3DES加密算法

由于计算机运算能力不断增强，DES密码的密钥长度（56位）变得容易被暴力破解。3DES (Triple DES, 三重DES) 为了增强DES密码的强度，将DES重复三次来增加DES的密钥长度以避免被暴力破解。因此，3DES并不是一种全新的分组密码算法。

3.4.1　3DES的总体结构

3DES加密算法的总体结构如图 3.12所示。在加密和解密过程中，都需要经过3次DES处理。3DES使用 “密钥包”，其包含3个DES密钥，K1，K2和K3，均为56位（除去奇偶校验位）。

在加密过程中，明文分组经过3次DES处理，即DES → DES^{-1} → DES，这里，我们用DES^{-1}表示DES的解密。所以，在3DES加密过程中，实际的运算是：DES加密 → DES解密 → DES加密。这看起来有点不可思议，但实际上却是有意为之。这个设计是IBM提出来的，目的是为了让3DES能够向下兼容DES。当3DES的所有密钥K1、K2、K3都相同时，3DES也就等同于普通的DES了。这是因为前两次 DES处理（DES加密 → DES解密）完成后的结果就是最初的明文分组。

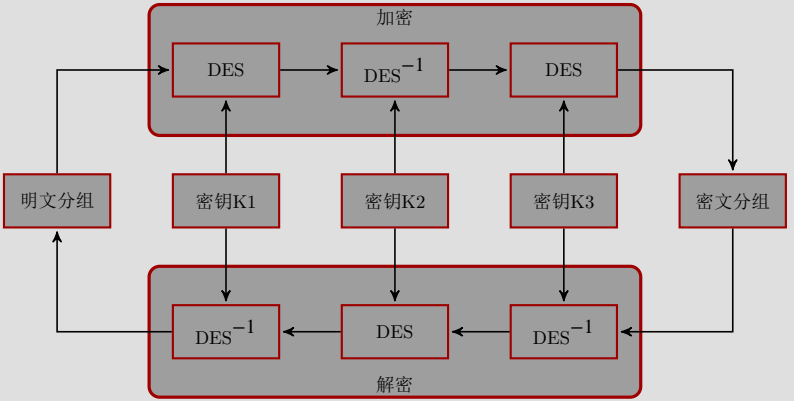


图3.12　3DES密码算法总体结构

用E表示DES加密函数，D表示DES解密函数，那么3DES的加密算法可以表示为：

$$ciphertext = E_{K3}(D_{K2}(E_{K1}(plaintext)))$$

也就是说，使用密钥K1进行DES加密，再用密钥K2进行DES“解密”，最后以密钥K3进行DES加密。

而解密算法则和加密过程正好相反，是以密钥K3、密钥K2、密钥K1的顺序执行的，形式上可以表示为：

$$plaintext = D_{K1}(E_{K2}(D_{K3}(ciphertext)))$$

即以密钥K3解密，以密钥K2“加密”，最后以密钥K1解密。无论是加密还是解密，中间一步都是前后两步的逆。

3.4.2　密钥选项

3DES标准定义了三种密钥选项：

- 密钥选项1：三个密钥是独立的，即$K1 \neq K2 \neq K3$。
- 密钥选项2：$K1$和$K2$是独立的，而$K3 = K1$。
- 密钥选项3：三个密钥均相等，即$K1 = K2 = K3$。

密钥选项1的强度最高，拥有$3 \times 56 = 168$个独立的密钥位。这种3DES密码算法称为DES-EDE3，EDE是加密 (Encryption) – 解密 (Decrpytion) – 加密 (Encryption) 3个单词首字母的缩写。

密钥选项2的安全性稍低，拥有$2 \times 56 = 112$个独立的密钥位。该选项比简单的应用DES两次的强度较高，即使用K1和K2，因为它可以防御中途相遇攻击 (Meet-in-the-Middle Attack)。这种3DES密码算法称为DES-EDE2。

密钥选项3等同于普通DES，只有56个密钥位。这个选项提供了与DES的兼容性，因为第1和第2次DES操作相互抵消了。

3.4.3　3DES密码现状

银行机构目前还在普遍使用3DES，并持续开发和宣传基于3DES的标准，例如EMV。EMV三个字母分别代表Europay、 MasterCard与Visa，是制定该标准最初的三家公司。EMV是国际金融业界对于智慧支付卡与可使用晶片卡的POS终端机及自动柜员机 (ATM) 等所制定的标准。EMV智慧卡（也称为IC卡）的信息储存在集成电路中而非过去的磁条里，但大部分EMV卡背也有可以向下兼容的磁条。芯片可以和插入式读卡器交换数据，非接触式智能卡还可以使用射频识别（RFID）技术在一定距离范围内交换数据。符合EMV标准的支付卡叫做芯片卡。

此外，Microsoft OneNote和Microsoft Outlook 2007使用3DES密码保护用户数据。

尽管3DES还有在使用，但是其两个独立密钥的变种DES-EDE2已经在2015年正式退出。在经过一系列安全分析和针对 3DES的实例攻击演示之后，2017年11月，NIST建议应当限制3DES用于对数量在2^{20}个以上的64位

分组（8MB）的数据进行加密，因此，TLS、IPSec和大文件加密不应该再使用3DES。目前，NIST正在推动2030年彻底废除3DES的使用，但是，随着密码研究揭露的越来越多的漏洞，3DES可能会被加速退出历史舞台。

3.5　AES加密算法

AES的全称是Advanced Encryption Standard，意思是高级加密标准。它的出现主要是为了取代DES加密算法的，我们已经知道DES算法的有效密钥长度是56位，因此算法的理论安全强度是2^{56}。但二十世纪中后期计算机飞速发展，元器件制造工艺的进步使得计算机的处理能力越来越强，虽然出现了3DES的加密方法，但由于它的加密时间是DES算法的3倍多， 64位的分组大小相对较小，所以还是不能满足对信息安全的要求。于是1997年1月2号，美国国家标准技术研究所（NIST）宣布希望征集高级加密标准，用以取代DES。在密码标准征集中，所有AES候选提交方案都必须满足以下标准：

- 分组大小为128位的分组密码。
- 必须支持三种密码标准：128位、192位和256位。
- 比提交的其他算法更安全。
- 在软件和硬件实现上都很高效。

AES也得到了全世界很多密码工作者的响应，先后有很多人提交了自己设计的算法。有5个候选算法进入最后一轮：Rijndael， Serpent，Twofish，RC6和MARS。最终经过安全性分析、软硬件性能评估等严格的步骤，Rijndael算法获胜。

3.5.1　Rijndael算法

Rijndael算法是由比利时学者Joan Daemen和Vincent Rijmen所提出的，算法的名字就由两位作者的名字组合而成。 Rijndael的优势在于集安全性、性能、效率、可实现性及灵活性与一体。不同于DES，Rijndael使用的是代换—置换网络（SPN，Substitution-Permutation Network），而非Feistel结构。

严格地说，AES和Rijndael加密算法并不完全一样（虽然在实际应用中两者可以互换）。Rijndael算法支持更大范围的分组和密钥长度，介于128-256之间所有32的倍数均可，最小支持128位，最大256位，共25种组合。而AES标准支持的分组大小固定为128位，密钥长度有3种选择：128位、192位及256位，分别为AES-128、AES-192和AES-256。

3.5.2 Rijndael的加密

不同于DES加密算法的Feistel网络结构，Rijndael算法是基于代换—置换网络的迭代算法。但是，和DES加密算法类似，Rijndael加密过程中，明文数据也需要经过多个轮次的转换后方能生成密文，每个轮次的转换操作由轮函数定义。同时，Rijndael也包括一个密钥调度算法来根据种子密钥（用户密钥）编排生成多组轮密钥。

根据Rinjdael算法的定义，加密轮数会针对不同的分组及不同的密钥长度选择不同的数值。以word（字长）为单位，1 word等于 4个字节，用N_b表示分组的长度，N_k表示密钥的长度，那么，$N_b = 4$就是$4 \times 4 \times 8 = 128$比特，以此类推。 Rinjdael给出的加密轮数$N_r$的选择如表 3.1所示。

N_r	$N_b = 4$ (AES)	$N_b = 6$	$N_b = 8$
$N_k = 4$	10 (AES-128)	12	14
$N_k = 6$	12 (AES-192)	12	14
$N_k = 8$	14 (AES-256)	14	14

表3.1　Rinjdael加密轮数N_r

由于AES标准中分组大小固定为128位（$N_b = 4$），以128位密钥长度（$N_k = 4$）为例，即AES-128算法，加密的总体结构如图 3.13所示，AES-128需要10轮加密，图中左半部分给出了AES-128加密的过程，右半部分为轮密钥的生成过程。加密过程中，第0轮相当于预处理，不计算在轮数内。除最后一轮之外，第1至9轮，每轮的加密轮函数会根据轮密钥对数据依次进行如下 4种运算：

- `SubBytes`（字节代换）
- `ShiftRows`（行移位）

- MixColumns（列混淆）
- AddRoundKey（轮密钥加）

图 3.13中小方格中的数字代表该字节在分组中的位置。这16字节的明文分组以列为主序排列为一个4×4 的状态矩阵，矩阵中的每个元素就是一个字节。由明文分组生成的状态矩阵称为初始状态。初始状态矩阵经过第0轮的 AddRoundKey运算后的状态矩阵，进一步经过10轮迭代处理生成密文分组。前一轮处理完后的输出状态作为下一轮处理的输入状态，第10轮加密完后的输出状态以列为主序排列为密文分组。

以下，对Rijndael算法中使用的4种运算进行详细的介绍。

AddRoundKey（轮密钥加）

在每次的加密循环中，都会由种子密钥产生一把轮密钥（通过Rijndael密钥调度算法产生）。在AddRoundKey运算中，轮密钥与状态矩阵中的每个字节元素依次相“加”，这里的加法运算实际上是逻辑运算中的异或计算，如图 3.14 所示。

SubByte（字节代换）

在SubBytes步骤中，状态矩阵中的各个字节通过一个8比特的S盒进行替换，如图 3.15所示。字节代换是可逆的非线性变换，也是AES运算组中唯一的非线性变换。S盒是事先设计好的替换查询表。其设计不是随意的，而是根据设计原则严格计算求得，不然无法保证算法的安全性。

ShiftRows（行移位）

ShiftRows描述了作用在矩阵的行上的移位操作，如图 3.16。在此运算中，每一行都向左循环位移某个偏移量。在AES-128中，第1行维持不变，第2行里面的每个元素都向左循环移动一格。同理，第3行及第4行向左循环位移的偏移量就分别是2格和3格。

MixColumns（列混淆）

列混淆是Rijndael算法中最复杂的一步，其理论基础为近世代数的域论，实质是在有限域（亦称伽罗瓦域，Galois Field）$GF(2^8)$上的多项式乘法运算。伽罗瓦是法国的数学家，群论的创立者。用群论彻底解决了根

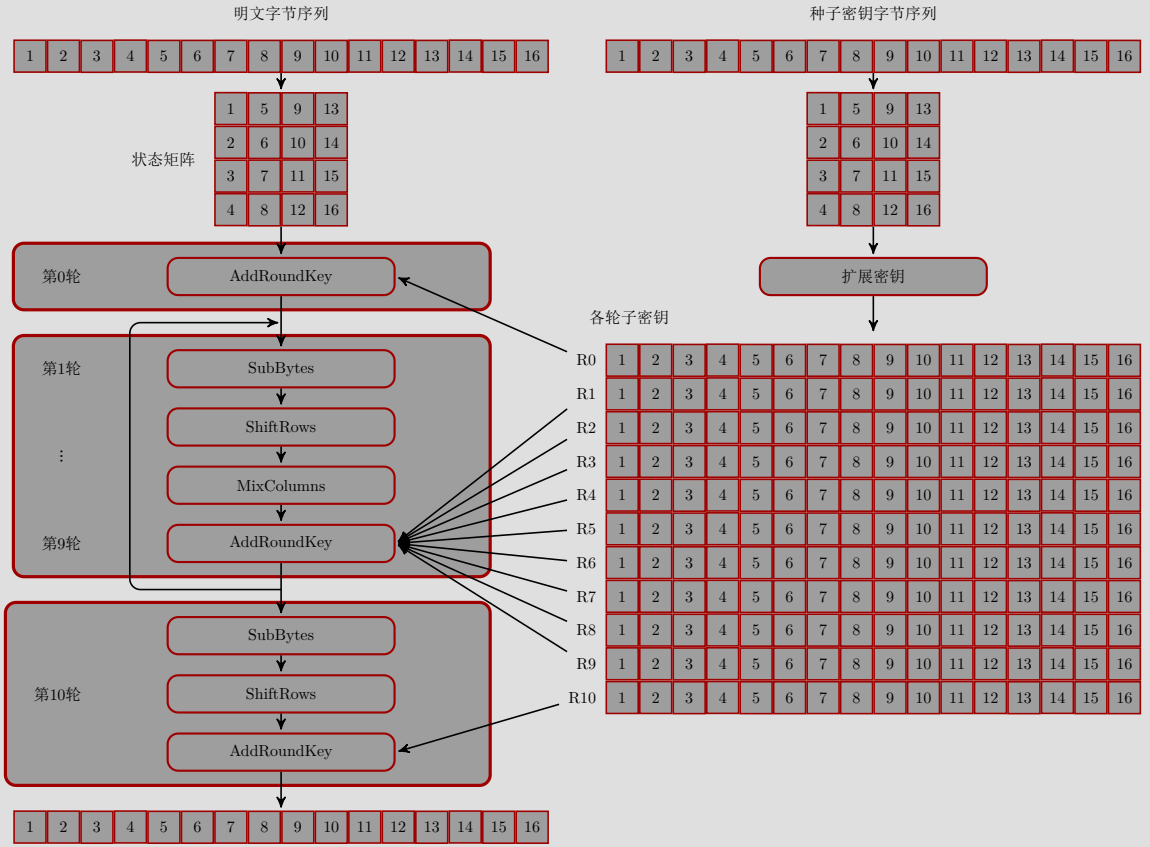


图3.13　AES算 法 总 体 结 构

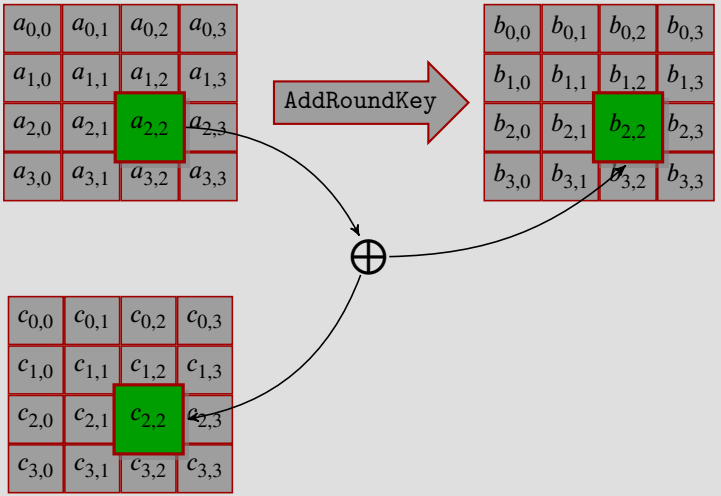


图3.14　AddRoundKey运 算

式求解代数方程的问题，而且由此发展了一整套关于群和域的理论。在解释列混淆运算之前，我们有必要先简单了解一下域的概念和域中元素的四则运算。

一组元素的集合，以及在集合上的四则运算，构成一个域。其中加法和乘法必须满足交换、结合和分配的规律，这看起来跟我们平时算术中的乘法和加法运算非常类似。但是，域的一个非常重要的性质是加法和乘法具有封闭性，即加法和乘法的结果仍然是域中的元素。

伽罗瓦的重大发现在于有限域，也就是仅含有限多个元素的域。所以当我们说伽罗瓦域的时候，就是指有限域。$GF(2^w)$表示含有2^w个元素的有限域。

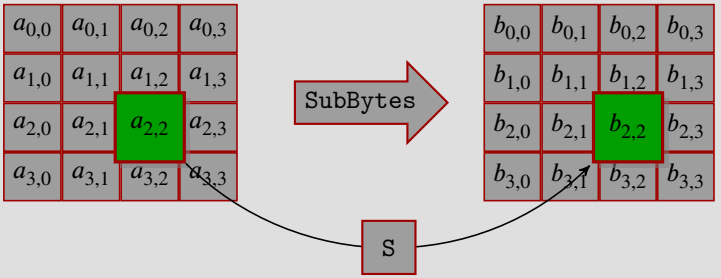


图3.15　SubBytes运 算

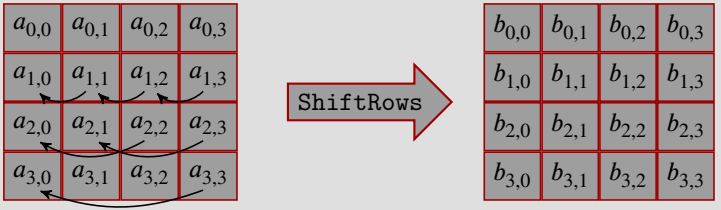


图3.16　ShiftRows运 算

$GF(2^8)$由一组从0x00到0xff的256个值组成，加上加法和乘法运算。$GF(2^8)$加法就是异或（XOR）操作。然而，$GF(2^8)$的乘法有点复杂。AES的MixColumns运算及其逆运算需要知道怎样只用七个常量0x01、0x02、0x03、0x09、 0x0b、0x0d和0x0e来相乘。所以这里不全面介绍$GF(2^8)$的乘法，而只是针对这七种特殊情况进行说明。

首先，在$GF(2^8)$中用0x01的乘法是特殊的；它相当于普通算术中用1做乘法并且结果也同样—任何值乘0x01等于其自身。

现在让我们看看用0x02做乘法。和加法的情况相同，虽然理论是深奥的，但最终结果十分简单。只要被乘的值小于0x80，这时乘法的结果就是该值左移1比特位。如果被乘的值大于或等于0x80，这时乘法的结果就是左移1比特位再用值0x1b异或。它防止了“域溢出”并保持乘法的乘积在范围以内。

一旦你在$GF(2^8)$中用0x02建立了加法和乘法，你就可以用任何常量去定义乘法。用0x03 做乘法时，你可以将0x03分解为2的幂之和。为了用0x03乘以任意字节b，因为 $0x03 = 0x02 \oplus 0x01$，因此：

$$\begin{aligned} b \otimes 0x03 &= b \otimes (0x02 \oplus 0x01) \\ &= (b \otimes 0x02) \oplus (b \otimes 0x01) \end{aligned}$$

同理，用0x0d去乘以任意字节b可以这样做：

$$\begin{aligned} b \otimes 0x0d &= b \otimes (0x08 \oplus 0x04 \oplus 0x01) \\ &= (b \otimes 0x08) \oplus (b \otimes 0x04) \oplus (b \otimes 0x01) \\ &= (b \otimes 0x02 \otimes 0x02 \otimes 0x02) \oplus (b \otimes 0x02 \otimes 0x02) \oplus (b \otimes 0x01) \end{aligned}$$

MixColumns运算的其它乘法遵循大体相同的模式，例如：

$$\begin{aligned} b \otimes 0x09 &= b \otimes (0x08 \oplus 0x01) \\ &= (b \otimes 0x02 \otimes 0x02 \otimes 0x02) \oplus (b \otimes 0x01) \\ b \otimes 0x0b &= b \otimes (0x08 \oplus 0x02 \oplus 0x01) \\ &= (b \otimes 0x02 \otimes 0x02 \otimes 0x02) \oplus (b \otimes 0x02) \oplus (b \otimes 0x01) \\ b \otimes 0x0e &= b \otimes (0x08 \oplus 0x04 \oplus 0x02) \\ &= (b \otimes 0x02 \otimes 0x02 \otimes 0x02) \oplus (b \otimes 0x02 \otimes 0x02) \oplus (b \otimes 0x02) \end{aligned}$$

基于以上关于有限域$GF(2^8)$上的加法和乘法运算，图 3.17说明了MixColumns的原理，用矩阵乘法公式可以表示为：

$$\begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix} \quad 0 \le j \le 3$$

由于矩阵乘法将一列中的各个字节利用有限域$GF(2^8)$上的乘法运算和加法运算组合起来，输出状态矩阵相应列的每个字节都受来自输入列四个字节的影响。同样，每个输入列单个字节都会对输出列四个字节造成影响。因此，ShiftRows和MixColumns两步骤为这个密码系统提供了扩散性。

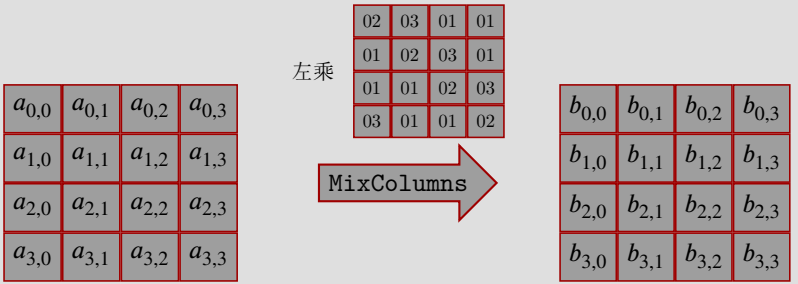


图3.17　MixColumns运 算

3.5.3　Rinjdael密钥扩展算法

密钥扩展算法是Rijndael的密钥编排实现算法，其目的是根据种子密钥（用户密钥）生成多组轮密钥。轮密钥用于每轮（包括第0轮）AddRoundKey运算中和状态矩阵进行按字节顺序的异或计算。我们继续

用N_b、N_k、N_r来表示分组和密钥长度（以word为单位），以及加密轮数（参见**表 3.1**）。Rijndael需要扩展出N_r+1个子密钥参与每轮的`AddRoundKey`运算。由于每轮子密钥的长度和分组长度N_b一致，所以，需要扩展的所有子密钥的总长度$N_w = N_b \times (N_r+1)$个字。因为AES算法固定了分组长度为 128位，即4个word，因此，AES算法中$N_b = 4$。那么，AES加密算法需要根据种子密钥扩展的子密钥总长度N_w如**表 3.2**所示。

	N_b	N_k	N_r	N_w
AES-128	4	4	10	**44**
AES-192	4	6	12	**52**
AES-256	4	8	14	**64**

表3.2　AES扩展子密钥的word数N_w

将每轮子密钥以word为单位排列成一个序列，用w_i $(0 \le i \le N_w - 1)$表示子密钥序列中的第i个元素，那么，AES算法的密钥扩展就是计算每个w_i的值。w_i的计算可以用如下数学公式描述：

$$w_i = \begin{cases} [k_{4i}\ k_{4i+1}\ k_{4i+2}\ k_{4i+3}], & \text{if } 0 \le i < N_k; \\ w_{i-N_k} \oplus \mathrm{G}(w_{i-1}), & \text{if } i \ge N_k \text{ and } i \bmod N_k = 0; \\ w_{i-N_k} \oplus \mathrm{SubWord}(w_{i-1}), & \text{if } i \ge N_k,\ N_k > 6, \text{ and } i \bmod N_k = 4; \\ w_{i-N_k} \oplus w_{i-1}, & \text{Otherwise.} \end{cases}$$

公式略显复杂，但是如果我们用**图 3.18**来解释AES-128的密钥扩展过程，则该计算就一目了然了。

首先，从公式中的第1个条件可知，前N_k个w元素由种子密钥初始化而来。对于AES-128，$N_k = 4$，将种子密钥每4个字节依序赋值给w_0、w_1、w_2和w_3。将每N_k个w元素排成一行，根据公式中的第2个条件，每行的第一个w元素由上一行的最后一个w元素经过`G`函数转换后与上一行的第一个w元素进行异或计算而来。当$N_k > 6$时，即AES-128和AES-192，根据第4个条件，每行的其他w元素均由该行前一个w元素和该列上一行的w元素直接异或计算而来。第3个条件则适用于AES-256，也就是说，在AES-256中，每行的第5个w元素则由该行的前一个元素经过`SubWord`替换后与该列上一行的w元素异或计算而来。

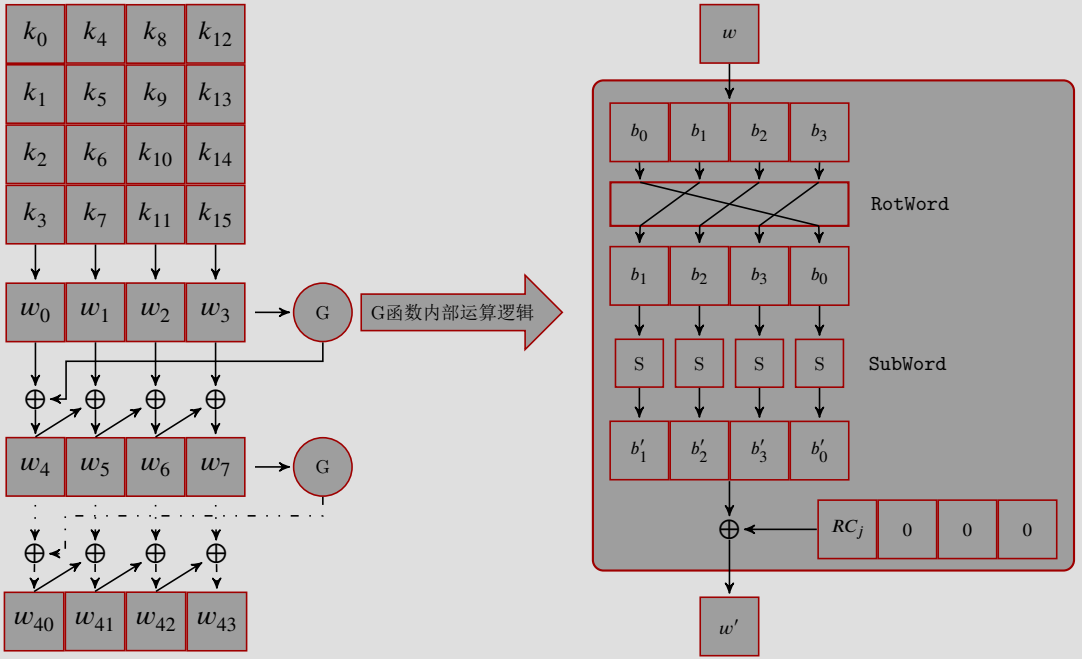


图3.18　AES-128密钥扩展原理图

在AES子密钥生成过程中，每行的最后一个w元素都要经过G函数转换后再和该行第一个w元素“相加”计算出下一行的第一个w元素。图 3.18的右半部分描述了G函数的计算过程，主要包括3个运算，分别为：RotWord、SubWord和一次异或计算。RotWord将w元素中的4个字节进行循环置换（以字节为单位循环右移）。SubWord则是根据加密过程中SubByte运算中的S盒对每个字节进行替换操作。而最后一步的异或计算则

稍微繁琐，其复杂之处在于轮常量RC_j $(1 \le j \le N_r)$的计算。RC_j由定义在有限域$GF(2^8)$ 上的$0x02^{j-1}$的值计算而来，根据前面讨论有限域$GF(2^8)$上的乘法运算的规则，我们不难得出每轮的轮常量RC_j值如下表所示。

j	1	2	3	4	5	6	7	8	9	10
RC_j	0x01	0x02	0x04	0x08	0x10	0x20	0x40	0x80	0x1b	0x36

表3.3　AES密钥扩展G函数中轮常量RC_j的值

3.5.4　Rinjdael的解密

上一节，我们已经对Rinjdael的总体加密过程以及加密过程中所涉及的各种运算逐个进行了详细的说明。下面，我们来了解 Rinjdael的解密。Rinjdael的解密过程就是将图 3.13所示的加密过程颠倒过来。加密轮函数中的四种运算对应的逆运算分别为：`InvSubBytes`（图 3.19）、`InvShiftRows`（图 3.20）、`InvMixColumns`（图 3.21）和`AddRoundKey`。

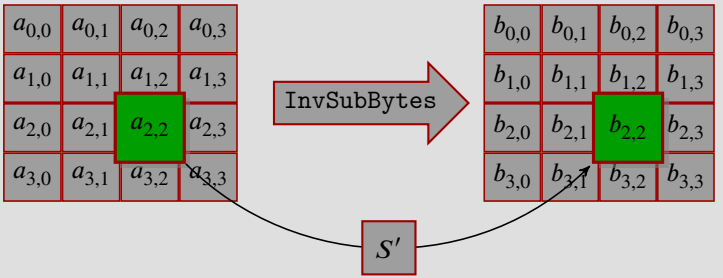


图3.19　InvSubBytes运 算

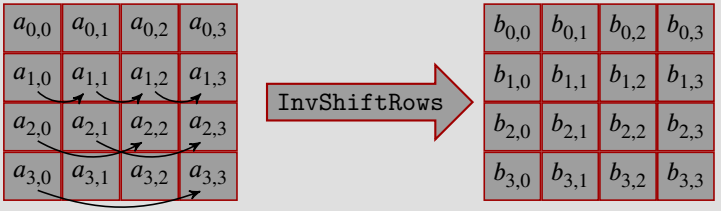


图3.20　InvShiftRows运 算

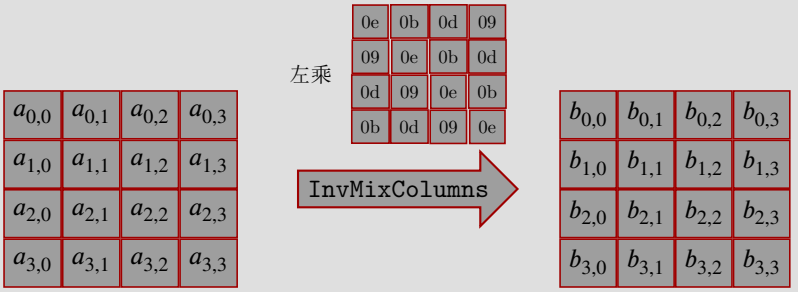


图3.21　InvMixColumns运 算

在AddRoundKey运算中使用了异或计算，由于异或计算本身具有自反性，即$a \oplus b \oplus b = a$，因此AddRoundKey的逆运算即是其自身。

InvSubBytes是SubBytes的逆运算。在InvSubBytes运算中用到的S'盒恰好是 SubBytes中S盒的反向字符替换表。

InvShiftRows是ShiftRows的逆运算。在InvShiftRows运算中作用在状态矩阵上的移位操作往反方向上移动相应的格数便实现了ShiftRows的逆运算。

InvMixColumns是MixColumns的逆运算。在有限域$GF(2^8)$上，由于

$$\begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} = \begin{bmatrix} 01 & 00 & 00 & 00 \\ 00 & 01 & 00 & 00 \\ 00 & 00 & 01 & 00 \\ 00 & 00 & 00 & 01 \end{bmatrix}$$

说明，这两个矩阵在有限域$GF(2^8)$上互逆，因此，左乘逆矩阵便可实现MixColumns的逆运算。

3.5.5 Rijndael的安全性

AES是目前被广泛采用的对称加密算法。AES加密算法（使用128，192，和256位密钥的版本）的安全性，在设计结构及密钥的长度上都已到达保护机密信息的标准。最高机密信息的传递，则至少需要192或256位的密钥长度。

在密码学的意义上，只要存在一个方法，比穷举法还要更有效率，就能被视为一种“破解”。从应用的角度来看，这种程度的破解依然太不切实际，但理论上的突破有时仍可提供对漏洞模式的深入了解。

由于密钥长度每增加1位，密钥空间将增加2倍，并且如果密钥的每个可能值的机会均相等，则意味着平均蛮力密钥搜索时间将增加一倍。这意味着蛮力搜索的工作量随着密钥长度的增加而呈指数增长。密钥长度本身并不意味着针对攻击的安全性，因为已经发现具有很长密钥的密码容易受到攻击。

针对AES最大的争议则在于AES有一个相当井然有序的代数框架的数学结构。尽管相关的代数攻击尚未出现，但有许多学者认为，把安全性创建于未经透彻研究过的结构上是有风险的。不过，这也只是一种假设而已。到目前为止，尚无已知的实际攻击方法。不过，根据斯诺登的文件，美国国家安全局（NSA）正在研究基于 τ统计信息的加密攻击是否可能有助于破坏AES。

3.6　Chacha20加密算法

ChaCha20是一种流式加密算法，它源自于Salsa20加密算法。2004年，ECRYPT启动了eSTREAM流密码计划的研究项目。丹尼尔·J·伯恩斯坦（Daniel J. Bernstein）在2005年设计出了Salsa20流式加密算法，被eSTREAM选中，成为最终胜出的7个算法之一。其特点是结构简单、易于实现，很安全（支持128比特或256比特的密钥），速度很快，也适合在嵌入式系统上实现。此外，Salsa20不受专利的约束。随后，在2008年，伯恩斯坦在Salsa基础上做了一些改动，发布了变种算法ChaCha，其目的是在不牺牲性能的前提下，增加每一轮的扩散。

ChaCha20由Google公司率先在Andriod移动平台中的Chrome浏览器和Google网站间的HTTPS通讯时代替RC4使用。 ChaCha20通常和Poly1305消息认证码（Message Authentication Code, MAC）算法一起配合使用，合称为 ChaCha20-Poly1305算法。在密码学上，这种把加密算法和MAC算法的组合称为关联数据的认证加密（Authenticated Encryption with Aassociated Data， AEAD）。比如，AES-GCM就是另外一种AEAD算法。

AEAD是一种同时具备保密性，完整性和可认证性的加密形式。AEAD产生的原因很简单，单纯的对称加密算法，其解密步骤是无法确认密钥是否正确的。也就是说，加密后的数据可以用任何密钥执行解密运算，得到一组疑似原始数据，而不确认密钥是否是正确的时候，也就也不知道解密出来的原始数据是否正确。因此，需要在单纯的加密算法之上，加上一层验证手段，来确认解密步骤是否正确。这里，我们先对AEAD有个初步的认知，后面，我们在介绍到MAC算法的时候，会重点讨论AEAD。本小节，我们主要介绍ChaCha20加密算法。

3.6.1　ARX运算和操作函数

Salsa20的核心是一个建立在基于Add-Rotate-XOR（ARX）操作的Hash函数之上，将一个64字节的输入序列转换为另一个64字节的输出序列。这个Hash函数产生的伪随机字节流和明文字节流进行异或运算生成密文字节流。这使得 Salsa20具有了不同寻常的优势，它可以在恒定时间内定位到输出流中的任何位置。

前面我们都是采用自顶向下的方法介绍各个加密算法的，即先介绍算法的总体结构，然后深入到算法运算的细节。现在，我们使用自底向上的方法来介绍Salsa20加密算法，先介绍其中采用的各种运算，再介绍salsa20是如何将这些运算组合起来的。

ARX运算

在Salsa20算法内部，采用的3个基本运算是：*32*位模加、循环移位和异或计算。异或运算自然不必多说了，普遍应用在密码算法中。我们用符号$\boxplus$表示模加运算，w_1和w_2两个word的32位模加运算则可以表示为：

$$w_1 \boxplus w_2 = (w_1 + w_2) \bmod 2^{32}$$

Salsa中的循环移位为循环左移操作，我们用R^m表示循环左移m位的运算，$R^m(w)$表示将32位的word w循环左移m位。

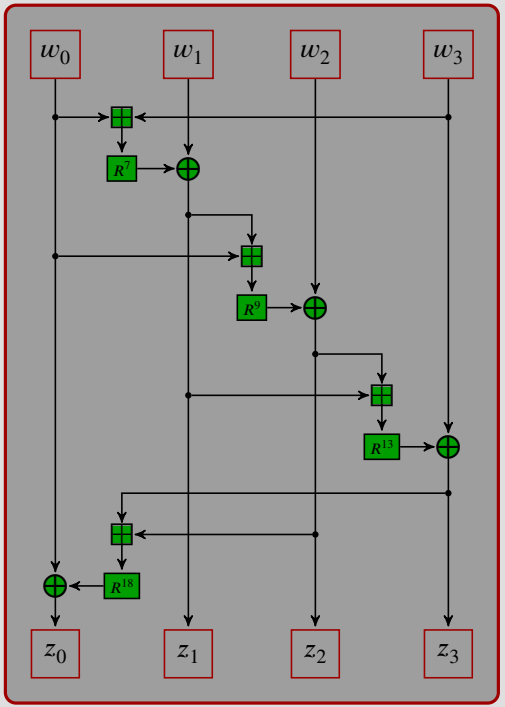
QuarterRound函数

用$w = (w_0, w_1, w_2, w_3)$表示由4个word组成的序列，那么$\texttt{QuarterRound}(w)$输出另外一个由4个word组成的序列$z = (z_0, z_1, z_2, z_3) = \texttt{QuarterRound}(w)$。Salsa算法中定义的 `QuarterRound`函数如下：

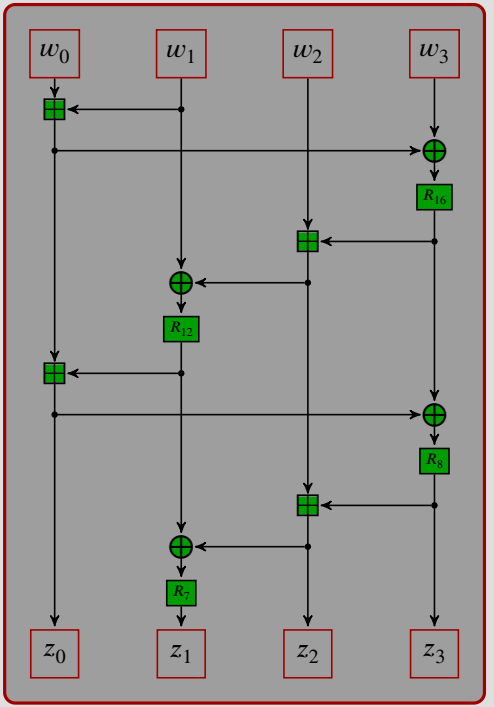
$$z_1 = w_1 \oplus (R^7(w_0 \boxplus w_3))$$
$$z_2 = w_2 \oplus (R^9(z_1 \boxplus w_0))$$
$$z_3 = w_3 \oplus (R^{13}(z_2 \boxplus w_1))$$
$$z_0 = w_0 \oplus (R^{18}(z_3 \boxplus w_2))$$

随后，在2008年，伯恩斯坦发布的ChaCha密码家族对Salsa的`QuarterRound`函数进行了修改，其目的是增加每一轮的扩散以实现相同或稍微提升的性能。ChaCha算法中的`QuarterRound`函数如下：

$$w_0 = w_0 \boxplus w_1; \quad w_3 = w_3 \oplus w_0; \quad w_3 = R^{16}(w_3);$$
$$w_2 = w_2 \boxplus w_3; \quad w_1 = w_1 \oplus w_2; \quad w_2 = R^{12}(w_2);$$
$$w_0 = w_0 \boxplus w_1; \quad w_3 = w_3 \oplus w_0; \quad w_3 = R^{8}(w_3);$$
$$w_2 = w_2 \boxplus w_3; \quad w_1 = w_1 \oplus w_2; \quad w_2 = R^{7}(w_2);$$
$$(z_0, z_1, z_2, z_3) = (w_0, w_1, w_2, w_3)$$



Salsa的QuarterRound运 算



ChaCha的QuarterRound运 算

图3.22　QuarterRound的 运 算 过 程

图 3.22展 示 了Salsa和ChaCha算 法 的QuarterRound运 算 过 程 。 经 过 比 较 ， 不 难 发 现ChaCha算 法 中 的QuarterRound运算对每个word更新2次，增加了扩散性。

ChaCha状态

ChaCha状态是一个以行为主序的4×4的矩阵（如图 3.23所示），矩阵中的每个元素是一个word。QuarterRound运算每次从状态矩阵上选取4个元素进行转换运算。为简化表示，我们用QuarterRound(a,b,c,d)来表示选取状态矩阵中元素下标分别为a、b、c和d的word进行 QuarterRound运算，即QuarterRound$(1,5,9,13)$表示对word序列(w_1,w_5,w_9,w_{13})进行 QuarterRound运算。

w_0	w_1	w_2	w_3
w_4	w_5	w_6	w_7
w_8	w_9	w_{10}	w_{11}
w_{12}	w_{13}	w_{14}	w_{15}

图3.23　ChaCha状态矩阵

如果选取的4个元素属于状态矩阵的某一列，就被称为ColumnRound；如果选取的4个元素排列在状态矩阵的某一条对角线上，则被称之为DiagnoalRound。例如，QuarterRound$(1,5,9,13)$就是ColumnRound，QuarterRound$(0,5,10,15)$和QuarterRound$(2,7,8,13)$都属于DiagnoalRound。

LittleEndian函数

在计算机内存中的字节排序分为两类：Big-Endian（大端）和Little-Endian（小端）。标准的Big-Endian和Little-Endian的定义如下：

- Little-Endian就是低位字节排放在内存的低地址端，高位字节排放在内存的高地址端。
- Big-Endian就是高位字节排放在内存的低地址端，低位字节排放在内存的高地址端。

根据上述定义，对于字节序列$b=(b_0,b_1,b_2,b_3)$，经过LittleEndian函数运算之后的word值可以表示为：

$$\texttt{LittleEndian}(b) = 2^{24}b_3 + 2^{16}b_2 + 2^8 b_1 + b_0$$

3.6.2 ChaCha20分组函数

ChaCha20分组函数通过对如下的输入数据形成状态矩阵，并对状态矩阵进行多轮QuarterRound运算后，计算出密钥流的分组。

- *256*位的密钥：32个字节被划分成由8个32位的LittleEndian整数排列组成的word序列；
- *96*位的随机常量（*nonce*）：12个字节被划分成3个32位的LittleEndian整数排列组成的word序列；
- *32*位的分组计数器：4个字节被LittleEndian函数转换成一个word。

首先，根据图 3.24所示的过程初始化一个4×4的状态矩阵，共16个word、64个字节。以上三个输入数据的长度加起来共48个字节，缺少的16个字节由常量字符串“expand 32-byte k”中每个字母对应的ASCII编码后的字节序列补上。

由图 3.24可知，ChaCha状态的初始化过程如下：

- 初始状态的前4个word，由常量字符串“expand 32-byte k”对应的ASCII编码后的字节序列，经LittleEndian运算而来。
- 256位的密钥对应的字节序列经LittleEndian运算后，得到初始状态矩阵中的第5至12个字节，即w_4 — $w_1 1$。
- 4字节的分组计数器的LittleEndian值填充到第13个字节。
- 96位的随机常量nonce所对应的字节序列经LittleEndian转换后，填充到初始状态矩阵的最后3个字节。

由此可知，一旦密钥和随机常量确定后，初始状态就取决于分组计数器的值了。在早期的ChaCha版本中，还存在64位分组计数器和 64位随机常量组合的方案，但在RFC8439中，已经舍弃了这种组合，确定了32位分组计数器的方案。这就意味着当密钥和随机常量确定后，最多有2^{32}个状态（分组），把每个64字节的状态当做512位的比特流，那么最多可以形成 $2^{32} \times 512 \div 8\ bytes = 256GB$不重复的密钥流，这在很多应用场合下，已经够用

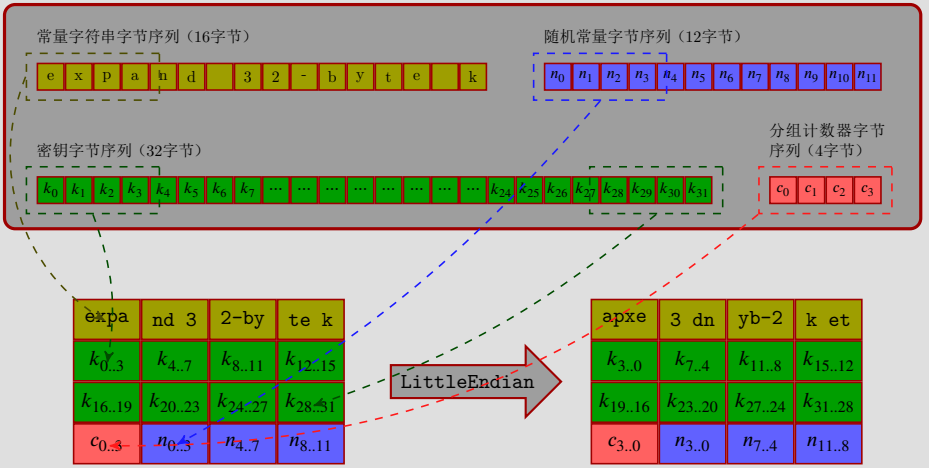


图3.24　ChaCha状 态 矩 阵 初 始 化 过 程

了。对于需要加密超过 256GB数据的应用，比如文件或磁盘加密等，推荐使用早期的64位分组计数器和64位随机常量组合的方案。

在确定了初始状态矩阵之后，ChaCha20算法执行20轮由`ColumnRound`和`DiagnoalRound`交替进行的运算。每轮运算执行4次`QuarterRound`计算。在轮计算为`ColumnRound`时，执行如下的4次`QuarterRound` 计算，确保状态矩阵中的每个word都替换掉。

QuarterRound(0,4,8,12)

QuarterRound(1,5,9,13)

QuarterRound(2,6,10,14)

QuarterRound(3,7,11,15)

在轮计算为DiagnoalRound时，执行的QuarterRound计算如下。4次QuarterRound计算同样确保状态矩阵中的每个word都被替换。

QuarterRound(0,5,10,15)

QuarterRound(1,6,11,12)

QuarterRound(2,7,8,13)

QuarterRound(3,4,9,14)

在20轮计算以后，也就是经过对上述8个QuarterRound运算进行10次迭代之后输出的状态矩阵再和初始状态矩阵按元素进行32位模加计算所得到的结果状态矩阵，进一步使用LittleEndian序列化成64字节的分组。整个过程参见示意图 3.25。

3.6.3 ChaCha20加解密过程

ChaCha20是一种流式加密算法，通过连续调用分组函数来生成连续的密钥流。每次调用分组函数的种子密钥和随机常量nonce值都是固定不变的，只对分组计数器进行递增。这样，每次调用分组函数生成了不同的512比特的密钥流片段。分组函数的调用次数取决于明文数据的长度。明文数据字节序列和密钥流字节序列按位置顺序依次进行异或计算便得到了加密后的密文数据。由于异或计算具有自反性，解密和加密过程相同，将密文数据字节序列和分组函数生成的密钥流进行异或后，解密出明文数据。

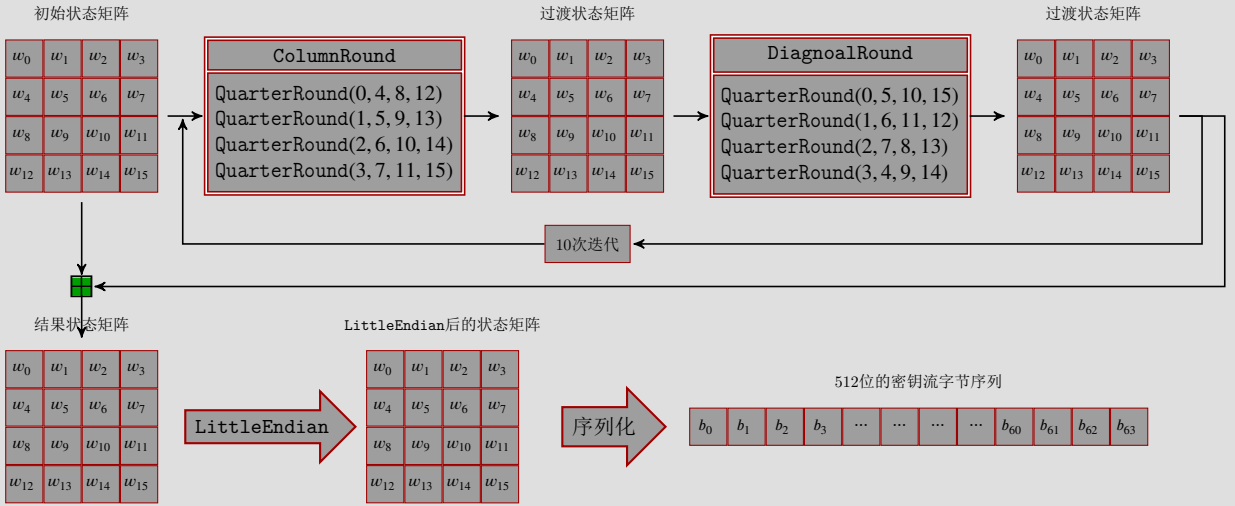


图3.25　ChaCha分组函数计算过程图

3.6.4　ChaCha20的应用

在Google采用了ChaCha20-Poly1305来替代RC4完成互联网的安全通信之后，ChaCha20-Poly1305算法也以`chacha20-poly1305@openssh.com`成为OpenSSH中的一个新密码套件。

ChaCha20-Poly1305在IKE和IPsec中的使用已在RFC 7634中标准化。在RFC 7905中，ChaCha20-Poly1305已经被加入TLS扩展标准。

若CPU不支持AES指令集，ChaCha20可提供比AES更好的性能。因此，ChaCha20通常被广泛应用在基于ARM CPU的移动设备上。

3.7　本章小结

本章，我们介绍了对称加密，同时介绍了对流密码和分组密码两种加密方式。进而，详细阐述了RC4、DES、3DES、AES和 ChaCha20密码算法的加解密原理及过程。

然而，用对称加密进行通信时，也有着其无法逾越的弱点，也就是密钥的配送问题，即如何将密钥安全地发送给通信对方。非对称加密（即公钥密码）的技术就完美地解决了密钥配送问题。我们将在下一章对密钥配送问题和非对称加密进行详细的探讨。

另外，本章所介绍的分组密码算法，都需要对明文按照固定的长度划分的分组进行加密。当需要加密的明文长度超过分组长度时，就需要对多个分组进行迭代，我们将在第？章探讨分组密码的分组模式。当明文长度不为分组长度的倍数时，对明文按照分组长度划分数据块之后，最后一个数据块的长度就不够分组的长度，我们也将在第？章讨论分组密码的填充模式。

第4章　非对称加密算法

上一章，我们介绍了对称加密，并具体介绍了RC4、AES、DES和ChaCha20多种对称加密算法。那么有了对称加密算法，通信双方是否就可以安全地进行通信了呢？本章，我们会先讨论对称加密遇到的密钥配送问题。然后，我们来看非对称加密是如何完美地解决密钥配送问题的。

4.1　密钥配送问题

我们先假设一个情景：发送方需要将一封非常重要的秘密邮件发送给接收方。发送方想到使用某个对称加密算法对信件内容进行加密后发送给发送方。但是，他立马意识到一个问题，接收方必须知道加密算法的密钥才能解密出邮件的真实内容。于是，怎么把密钥传递给接收方就变成了一个必须直面解决的问题。但是，怎么才能安全地将密钥从发送方传递给接收方呢？下面我们介绍一下解决密钥配送问题的几个方法。

- **事先共享密钥**

  解决密钥配送问题的最简单方法就是事先共享密钥，也就是发送方提前将密钥告诉接收方。如果他们两个离得很近，那没有问题，直接私下见面沟通就可以了。倘若他们分隔两地那就麻烦了。因为邮寄或者远程传输的过程中，密钥可能会被劫持。

  退一步说，即便能够安全有效地共享密钥，也会存在一个密钥保存的问题，因为每两个人之间进行通信都需要一个完全不同的密钥。假设邮件是发送方发给他的客户的商务合同，当客户数量巨大时，则需要保存一个相当大数量的密钥库。实际操作起来非常不便，也容易出错。

- **密钥中心分配密钥**

  为了解决保存大数量的密钥的问题。可以考虑采用密钥中心来对密钥进行集中管理。我们可以将密钥中心看成是一个服务器，它里面保存了每一个人的密钥信息，下面我们看一下具体的通信流程：

1. 发送方和接收方需要进行通信；
2. 密钥中心随机生成一个密钥，这个密钥将会是发送方和接收方本次通信要使用的临时密钥；
3. 密钥中心取出保存好的本次通信要使用的密钥；
4. 密钥中心将临时密钥使用发送方的密钥加密后，发给发送方；
5. 密钥中心将临时密钥使用接收方的密钥加密后，发给接收方；
6. 发送方收到加密后的数据，使用自己的密钥解密后，得到临时密钥；
7. 接收方收到加密后的数据，使用自己的密钥解密后，也得到临时密钥；
8. 发送方和接收方可以使用这个临时密钥自由通信了。

特别要注意的是，这里的临时密钥的使用方法很巧妙，后面我们讲到大家最常用的HTTPS通信协议时，会再次领会到这个临时密钥的巧妙使用。

密钥中心很好，但是也有缺点，首先，密钥中心的密钥是集中管理的，一旦被攻破，所有人的密钥都会暴露。其次，所有的通信都要经过密钥中心，可能会造成性能瓶颈。

- **使用DH密钥交换**

  DH（Diffie-Hellman）密钥交换算法能够让通信双方在不安全的信道上通过交互一些信息来生成相同的密钥。算法的具体细节我们将在**DH密钥交换**一节中讲到。

- **使用非对称加密中的私钥**

  密码配送的原因就在于对称加密使用的密钥是相同的。如果有种加密算法能够使用不同的密钥加密和解密，那么密钥配送是不是就不是问题了呢？

在非对称加密算法中，需要生成两个密钥，一个是公开密钥（public key），另外一个是私有密钥（private key）。公钥用作加密，私钥用作解密。使用公钥对明文加密后的密文，只能用相对应的私钥才能解密还原出真实的明文。由于加密和解密需要两个不同的密钥，故被称为非对称加密。不同于对称加密中加密和解密都使用同一个密钥，非对称加密算法中的公钥可以公开发布，但私钥必须由用户自行秘密保管，因此，又被称为公钥加密算法。

回到刚才发送秘密邮件的问题，如果接收方事先用非对称加密算法生成了公钥和私钥，并把公钥发给了发送方，则发送方可以将邮件使用公钥进行加密，然后发给接收方，这个邮件只有接收方才能解密。即使公钥和加密后的邮件被任何第三方截获了也无法解密出邮件内容。

当然这里也有一个问题，就是邮件发送方要确保生成的公钥的确是邮件接收方公开发布的。这个问题的解决方法我们会在后面的章节再深入讨论。非对称加密还有一个问题就是加密性能大约只有对称加密算法的几百分之一，这使得它不太适合大批量数据的加密。

基于公开密钥加密的特性，它还能提供数字签名的功能，使电子文件可以得到如同在纸本文件上亲笔签署的效果。关于数字签名，我们将在后面的章节中进行详细的阐述。

4.2　必备的数学知识

在讲解DH密钥交换算法和RSA加密算法之前，有必要了解一下一些必备的数论知识。数论是纯粹数学的分支之一，主要研究整数的性质，我们先来了解如下几个基本的概念。

- **质数**

  质数是指在一个大于1的自然数中，除了1和该整数自身外，不能被其他自然数整除的数。这个概念，我们在上小学的时候都学过了，这里就不再过多解释了。

- **互质数**

  公约数只有1的两个数叫做互质数。很显然，两个不同的质数一定是互质数。常用的判断两个数是否互质的算法是辗转相除法，又称欧几里得算法（Euclidean algorithm）。辗转相除法首次出现于欧几里得的《几何

原本》，而在中国则可以追溯至东汉出现的《九章算术》。辗转相除法用于计算两个整数的最大公约数。如果两个整数的最大公约数为1，则说明它们必为互质数。

辗转相除法基于如下原理：两个整数的最大公约数等于其中较小的数和较大数相除余数的最大公约数。例如，252和105的最大公约数是21（$252 = 21 \times 12; 105 = 21 \times 5$）；因为$252 - 105 = 21 \times (12 - 5) = 147$，所以147和105的最大公约数也是21。在这个过程中，较大的数缩小了，所以继续进行同样的计算可以不断缩小这两个数直至其中一个变成0。这时，所剩下的还没有变成0的数就是两数的最大公约数。辗转相除法是一种递归算法，每一步计算的输出值就是下一步计算时的输入值。

假设，我们要计算a和b（$a \geq b$）两个数的最大公约数$\gcd(a, b)$，第i步带余除法的商为q_i，余数为r_{i+1}，则辗转相除法可以写成如下形式：

$$\begin{aligned} r_0 &= a \\ r_1 &= b \\ &\vdots \\ r_{i+1} &= r_{i-1} - q_i r_i \quad (0 \leq r_{i+1} < |r_i|) \\ &\vdots \end{aligned}$$

当某步得到的$r_{i+1} = 0$时，计算结束。上一步得到的r_i即为a、b的最大公约数。

- **模运算**

模运算即求余运算。和模运算紧密相关的一个概念是“同余”。数学上，当两个整数除以同一个正整数，若得相同余数，则二整数同余。两个整数a、b，若它们除以正整数m所得的余数相等，则称a和b对于模m同余，记作：$a \equiv b \pmod{m}$；读作：a同余于b模m，或者a与b关于模m同余。例如：$26 \equiv 14 \pmod{12}$。

模运算具有如下一些基本性质：

★ 整除性

$$a \equiv b \pmod{m} \Rightarrow k \cdot m = a - b, k \in \mathbb{Z}.$$

k为整数集$\mathbb{Z}$中的某个整数。也就是说，a和b之差是m的倍数。

★ 传递性

$$\left.\begin{array}{ll} a \equiv b & (\mathrm{mod}\ m) \\ b \equiv c & (\mathrm{mod}\ m) \end{array}\right\} \Rightarrow a \equiv c \pmod{m}.$$

★ 保持基本运算

$$\left.\begin{array}{ll} a \equiv b & (\mathrm{mod}\ m) \\ c \equiv d & (\mathrm{mod}\ m) \end{array}\right\} \Rightarrow \left\{\begin{array}{ll} a \pm c \equiv b \pm d & (\mathrm{mod}\ m) \\ ac \equiv bd & (\mathrm{mod}\ m) \end{array}\right.$$

这个性质可以进一步引申出：

$$a \equiv b \pmod{m} \Rightarrow \left\{\begin{array}{ll} an \equiv bn & (\mathrm{mod}\ m), \forall n \in \mathbb{Z} \\ a^n \equiv b^n & (\mathrm{mod}\ m), \forall n \in \mathbb{N}^0 \end{array}\right. \tag{4.1}$$

★ 除法原理

如果m_1和m_2都可以整除$(a-b)$，那么m_1和m_2的最小公倍数$\mathrm{lcm}(m_1, m_2)$必定可以整除$(a-b)$，记为$a \equiv b \pmod{\mathrm{lcm}(m_1, m_2)}$。这可以推广成以下性质：

$$\left.\begin{array}{ll} a \equiv b & (\mathrm{mod}\ m_1) \\ a \equiv b & (\mathrm{mod}\ m_2) \\ & \vdots \\ a \equiv b & (\mathrm{mod}\ m_n) \end{array}\right\} \Rightarrow a \equiv b \pmod{\mathrm{lcm}(m_1, m_2, \cdots, m_n)}, \quad (n \geq 2). \tag{4.2}$$

- **模反元素**

  两个正整数a和m互质，那么一定可以找到整数b，使得 $ab-1$被m整除，即$ab \equiv 1 \pmod{m}$，这时，b就叫做a的模反元素，也称为模倒数或者模逆元。有时，也用a^{-1}来表示a的模反元素。

- **扩展欧几里得算法求模反元素**

  求模反元素是RSA加密算法中获得所需公钥、私钥的必要步骤。扩展欧几里得算法是欧几里得算法（辗转相除法）的扩展，可以在求得a、b的最大公约数的同时，能找到整数x、y（其中一个很可能是负数），使它们满足如下的贝祖等式：

$$ax + by = \gcd(a, b)$$

  扩展欧几里得算法在辗转相除法的基础上每步增加了s_i和t_i两组序列，并令$s_0 = 1, s_1 = 0$和$t_0 = 0, t_1 = 1$，在辗转相除法每步计算r_{i+1}之外额外计算 $s_{i+1} = s_{i-1} - q_i s_i$和$t_{i+1} = t_{i-1} - q_i t_i$，亦即：

$$\begin{array}{lll} r_0 = a, & s_0 = 1, & t_0 = 0; \\ r_1 = b, & s_1 = 0, & t_1 = 1; \\ \vdots & \vdots & \vdots \\ r_{i+1} = r_{i-1} - q_i r_i, & s_{i+1} = s_{i-1} - q_i s_i, & t_{i+1} = t_{i-1} - q_i t_i; \quad (0 \le r_{i+1} < |r_i|) \\ \vdots & \vdots & \vdots \end{array} \tag{4.3}$$

  算法结束条件与辗转相除法一致，也是$r_{i+1} = 0$，此时所得的s_i和t_i即满足等式$\gcd(a, b) = r_i = as_i + bt_i$。

  下表以$a = 240$、$b = 46$为例演示了扩展欧几里得算法。所得的最大公约数是2，所得贝祖等式为$\gcd(240, 46) = 2 = -9 \times 240 + 47 \times 46$。

序号i	商q_{i-1}	余数r_i	s_i	t_i
0		240	1	0
1		46	0	1
2	$240 \div 46 = 5$	$240 - 5 \times 46 = 10$	$1 - 5 \times 0 = 1$	$0 - 5 \times 1 = -5$
3	$46 \div 10 = 4$	$46 - 4 \times 10 = 6$	$0 - 4 \times 1 = -4$	$1 - 4 \times (-5) = 21$
4	$10 \div 6 = 1$	$10 - 1 \times 6 = 4$	$1 - 1 \times (-4) = 5$	$-5 - 1 \times 21 = -26$
5	$6 \div 4 = 1$	$6 - 1 \times 4 = 2$	$-4 - 1 \times 5 = -9$	$21 - 1 \times (-26) = 47$
6	$4 \div 2 = 2$	$4 - 2 \times 2 = 0$	$5 - 2 \times (-9) = 23$	$-26 - 2 \times 47 = -120$

- **欧拉函数**

在数论中，对正整数n，欧拉函数$\varphi(n)$是小于或等于n的正整数中与n互质的数的数目。此函数以其首名研究者欧拉命名，它又称为φ函数或是欧拉总计函数（totient function）。例如$\varphi(8) = 4$，因为1、3、5、7均和8互质。

当n为质数时，$\varphi(n) = n - 1$。欧拉函数是积性函数，即当m和n互质时，$\varphi(mn) = \varphi(m)\varphi(n)$。

- **欧拉定理**

欧拉定理证明当a、n为两个互质的正整数时，则有

$$a^{\varphi(n)} \equiv 1 \pmod{n} \tag{4.4}$$

其中$\varphi(n)$为欧拉函数。欧拉定理的证明过程非常复杂，这里，我们只要记住这个结论就可以了。

上述结果可分解为$a^{\varphi(n)} = a \cdot a^{\varphi(n)-1} \equiv 1 \pmod{n}$，这样，$a^{\varphi(n)-1}$即为$a$关于模$n$的模反元素。

当n为质数p时，结合欧拉函数，就有：$a^{p-1} \equiv 1 \pmod{p}$，这就是著名的费马小定理，是欧拉定理的特例。

4.3　DH密钥交换

DH（Diffie-Hellman）密钥交换算法是第一个实用的在不安全的信道上创建共享密钥的方法，解决了密钥传送问题。这个共享密钥可以在后续的通信中作为对称加密算法的密钥来加密通信内容。DH密钥交换算法由惠特菲尔德·迪菲（Whitfield Diffie）和马丁·赫尔曼（Martin Hellman）在1976年首次发表，这个方法被发明后不久出现了RSA公钥加密算法。

4.3.1　DH密钥交换算法的原理

DH密钥交换算法为通信双方在不安全的信道上建立的共享密钥可以作为双方在公共网络上使用对称加密算法加密数据的密钥。假设甲乙双方需要传递加密算法的密钥，我们看DH密钥交换算法怎么实现在一个不安全的信道上共享密钥的，具体的算法步骤描述如下。

1. 首先，甲乙双方协定使用质数p和底数g。
2. 甲选择一个秘密整数a，计算$A = g^a \pmod p$，将A发送给乙。
3. 乙也选择一个秘密整数b，计算$B = g^b \pmod p$，将B发送给甲。
4. 甲计算$e = B^a \pmod p$。
5. 乙计算$e = A^b \pmod p$。

密钥交换过程结束后，甲乙双方均计算得到了相同的密钥e。DH密钥交换过程所用到的计算非常巧妙，它利用了我们在前面介绍模运算时所引申出来的性质（公式 4.1），最终让通信双方协商出相同的密钥。我们只要证明$B^a \equiv A^b \pmod p$，就可以推断甲和乙计算出的密钥e是相等的。根据模运算公式 4.1，我们可以得到：

$$A = g^a \pmod p \Rightarrow A^b \equiv g^{ab} \pmod p$$
$$B = g^b \pmod p \Rightarrow B^a \equiv g^{ab} \pmod p$$

进而，根据模运算的传递性质便得到：$B^a \equiv A^b \pmod{p}$。

我们也可以用图 4.1来形象地解释DH密钥交换算法的原理和过程。

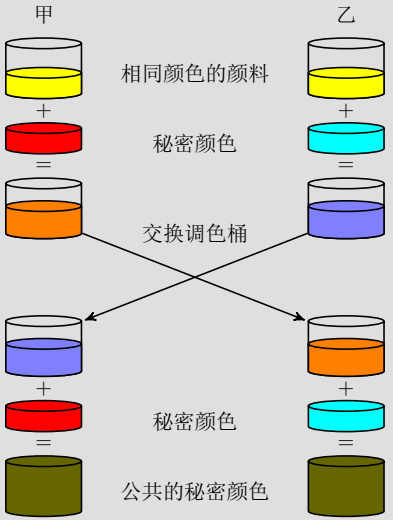


图4.1　图解DH密钥交换的概念

1. 甲乙双方协商好一个共同的颜色□作为颜料桶的底色。【甲乙双方协定相同的质数p和底数g。】
2. 甲选择一个秘密颜色■。【甲选择一个秘密整数a。】
3. 乙也选择一个秘密颜色■。【乙也选择一个秘密整数b。】
4. 甲将秘密颜色添加到颜料桶中，混合后得到过渡颜色■。【甲计算$A = g^a \pmod{p}$。】

5. 乙也将秘密颜色添加到颜料桶中，混合后得到过渡颜色🟦。【乙计算$B = g^b \pmod p$。】

6. 甲乙双方交换调色桶。【甲将A发送给乙，乙将B发送给甲。】

7. 甲再次把秘密颜色添加到颜料桶中，混合后得到最终的秘密颜色🟫。【甲计算$e = B^a \pmod p$。】

8. 乙也再次把秘密颜色添加到颜料桶中，混合后得到最终的秘密颜色🟫。【乙计算$e = A^b \pmod p$。】

4.3.2 DH密钥交换的安全性分析

在DH密钥交换过程中，甲乙通信双方各自生成一个只有自己知道的秘密整数a和b，这两个秘密整数需要他们各自安全保管，不能相互透露，也不会公开给任何第三方。除了这两个秘密整数之外，其它的参数要么是公开的，要么可能被第三方截获。我们定义这些整数的含义如下：

- a：甲的私钥
- b：乙的私钥
- A：甲的公钥
- B：乙的公钥

现在，我们假设不怀好意的丙监听了甲和乙的通信，他截获了甲和乙协商好的质数p和底数g，他还可能监听到甲和乙交换的公钥A和B。但是，由于丙不知道甲和乙的私钥a和b，所以，他就无法直接计算出他们最终的共享密钥。然而，丙在得知甲和乙交换的公钥A和B的情况下，能否反推出他们的私钥a和b呢？这就是离散对数问题。

在实数域中，对数的定义$\log_b a$是指对于给定的a和b，有一个数x，使得$b^x = a$。相应地，在模运算的情况下，当模m有原根时，设b为模m的一个原根，则当$x \equiv b^k \pmod m$时：

$$\mathrm{ind}_b x \equiv k \pmod m$$

这里，$\mathrm{ind}_b x$是x以整数b为底，关于模m的离散对数值。离散对数在一些特殊情况下可以快速计算。然而，当整数非常大时，通常没有非常高效的方法来计算它们。离散对数求解的困难性就奠定了DH密钥交换算法的安全性基础。除DH密钥交换算法之外，公钥密码学中的多个重要算法均是建立在寻找离散对数的问题解的困难性的基础之上。

4.4　RSA加密算法

本节我们要介绍的是RSA非对称加密算法。RSA是由罗纳德·李维斯特（Ron Rivest）、阿迪·萨莫尔（Adi Shamir）和伦纳德·阿德曼（Leonard Adleman）在1977年一起提出的。当时他们三人都在麻省理工学院工作。RSA就是他们三人姓氏开头字母拼在一起组成的。

4.4.1　RSA公钥和私钥的生成

假设消息的接收方需要通过一个不可靠的通道接收某个发送方的消息，接收方可以用以下的方式来产生一个公钥和一个私钥：

1. 任意选择两个特别大的质数p和q $(p \neq q)$，计算$N = pq$；
2. 根据欧拉函数，求得$r = \varphi(N) = \varphi(p)\varphi(q) = (p-1)(q-1)$；
3. 选择一个小于r的整数e，使e与r互质。并求得e关于r的模反元素，命名为d，那么，$ed \equiv 1 \pmod{r}$。
4. 将p和q的记录销毁。

(N, e)是公钥，(N, d)是私钥。消息接收方将他的公钥(N, e)传给发送方，而将他的私钥(N, d)秘密保存起来。

4.4.2　RSA加密和解密

加密消息

假设发送方要给接收方发送一个消息M，他知道接收方产生的N和e。他使用事先和接收方约好的格式将消息M转换为一个小于N的非负整数m，比如，他可以将每一个字转换为这个字的Unicode码，然后将这些Unicode码连接在一起组成 Unicode码串。假如他的信息非常长超过了N的位数，他可以将这个信息分为多个段，然后将每一段转换为m。用下面这个公式他可以将每一段m加密为c：

$$c \equiv m^e \pmod{N}$$

计算c并不复杂。发送方算出c后就可以将它传递给接收方。

解密消息

接收方收到发送方的消息c之后，就可以利用他的私钥d来解码。他可以用下面这个公式将c解密还原回m：

$$m \equiv c^d \pmod{N}$$

接收方利用上面的公式对来自发送方的每一段c解密得到所有的m后，便可以复原消息M。

正确性证明

我们只要证明$m^{ed} \equiv m \pmod{N}$对所有的m ($m < N$)都成立，便能证明解密公式的正确性。

首先，将模运算的基本性质（公式 4.1）应用在加密公式上，可知

$$c^d \equiv m^{e \cdot d} \pmod{N}$$

已知$ed \equiv 1 \pmod{r}$，即 $ed = 1 + h\varphi(N), h \in \mathbb{Z}$。那么有

$$m^{ed} = m^{1+h\varphi(N)} = m \cdot m^{h\varphi(N)} = m\left(m^{\varphi(N)}\right)^h$$

下面，我们分别从m与N互质和不互质两种情况来证明$m^{ed} \equiv m \pmod N$。

1. 若m与N互质，则由欧拉定理（公式 4.4）可得：

$$m^{ed} \equiv m\left(m^{\varphi(N)}\right)^h \equiv m(1)^h \equiv m \pmod N$$

2. 若m与N不互质，那么m必定为质数p或者q的倍数。不失一般性考虑$m = ph$，以及$ed - 1 = k(q-1)$，得：

$$m^{ed} = (ph)^{ed} \equiv 0 \equiv ph \equiv m \pmod p$$

$$m^{ed} = m^{ed-1}m = m^{k(q-1)}m = (m^{q-1})^k m \equiv 1^k m \equiv m \pmod q$$

因为$N = pq$，根据模运算性质公式 4.2，可得$m^{ed} \equiv m \pmod N$。

因此，不论m与N是否互质，都有$m^{ed} \equiv m \pmod N$，解密公式的正确性得证。

示例

我们用如下的例子再来解释RSA加密和解密。为简化计算和说明，我们特意用了较小的的质数。

1. 选择两个不同的质数，$p = 61$、$q = 53$，计算得到$N = 61 \times 53 = 3233$。
2. 根据欧拉函数，计算$r = \varphi(N) = \varphi(p)\varphi(q) = (p-1)(q-1) = 3120$。
3. 随机选择一个整数e，使其满足条件$1 < e < \varphi(N)$，并且e与$\varphi(N)$互质。为了简化计算，在1到3120之间，我们选择$e = 19$。
4. 计算e关于$\varphi(N)$的模反元素d。根据模反元素的定义，可知$ed \equiv 1 \pmod{\varphi(N)} \Rightarrow ed - 1 = k\varphi(N)$，于是，找$e$关于$\varphi(N)$的模反元素$d$，实质上就是对下面的这个二元一次方程求解：

$$ex + \varphi(N)y = 1$$

已知，$e = 19$、$\varphi(N) = 3120$，那么，该二元一次方程可以确定为：

$$19x + 3120y = 1$$

根据扩展欧几里得算法（公式 4.3），令$a = \varphi(N) = 3120$、$b = e = 19$，如下表格展示了扩展欧几里得算法求解的过程。

序号i	商q_{i-1}	余数r_i	s_i	t_i
0		3120	1	0
1		19	0	1
2	$3120 \div 19 = 164$	$3120 - 164 \times 19 = 4$	$1 - 164 \times 0 = 1$	$0 - 164 \times 1 = -164$
3	$19 \div 4 = 4$	$19 - 4 \times 4 = 3$	$0 - 4 \times 1 = -4$	$1 - 4 \times -164 = 657$
4	$4 \div 3 = 1$	$4 - 1 \times 3 = 1$	$1 - 1 \times -4 = 5$	$-164 - 1 \times 657 = -821$
5	$3 \div 1 = 3$	$3 - 3 \times 1 = 0$	$-4 - 3 \times 5 = -19$	$657 - 3 \times -821 = 3120$

最终，求得

$$3120 \times 5 + 19 \times (-821) = 1$$

据此，我们得出-821是19关于3120的模反元素。由于，我们期望得到一个正的模反元素，我们可以对上式稍作变化：

$$3120 \times 5 + 19 \times (-821) = 3210 \times 5 - 3120 \times 19 + 19 \times (-821) + 3120 \times 19 = 3120 \times (-14) + 19 \times 2299 = 1$$

所以，计算得到19关于3120的模反元素为2299，即$d = 2299$。

5. 将N和e封装成公钥，N和d封装成私钥。在该例中，$N = 3233$、$e = 19$、$d = 2299$，所以，公钥就是$(3233, 19)$，私钥就是$(3233, 2299)$。

现在，用大写字母A（ASCII编码值为65）来验算。对A加密后的数值为：

$$c = 65^{19} \pmod{3233} = 278839166712846014151954650878906 25 \pmod{3233} = 232$$

加密后的c经过解密公式计算后，可以还原回A对应的ASCII编码值。

$$m = 232^{2299} \pmod{3233} = 65$$

因为Python支持大数计算，我们使用Python解释器对解密公式的计算进行了验证，如下：

```
>>> pow(232, 2299)
180666347328265653258383894933004371487240092743537919407108211261805084212220403841284148846624524015919076278935762369
566193343373349827864820080189518110480760747411699792950837852232292376913816411843615809211304474825053467587778518242
705796906398756110666492666972719706854587962788746728592081235628180351761034041597084051469628142945625665210401870986
442721134645212335726733132166369845454237582598087434795873554555260338178625720110806025557360972874808487522190662 10
619113008297797669429153077700180340012046587925451990211178787025745543474937384533184941240981443646810905632560017797
164576088730314607384211004327617327635178100260842037549674043154025693004064517193174075606873030274879212886640256718
665968747046854756558556199498208619743014219982493267989161266257659462813102634476691108885295057325167573493051735583
673050390800019703579110868646618431152738053248378892584841525864881331330558804557033686702663256220319720619864147462
457892300319645383317856427729293119136561330755943202534086697378417807270552100178840044134499523362403674966693591336
577177957298225146756165001934902583109171687460847444019203743457164082131742194034466322608427604603401435949150255353
167890669197496031009024840083448661115491646464303377615784066236199296533834394865384197075062745634267534418168640817
424468556772942925280897253355617717434342285080762099998828522129417566010479204644104292099767595547713995184596564770
846187113210570630262758496094801761926960834360049939434211153760029990028088059951335925931689078224504960623918651079
688626485351471887329228750521678743769719298330078395342693680820677607978860075472603284591081978845348401933010582961
715134552244237772732343129376084801065135826082799548514816580275002572759105226991583897249209302400084882484902231414
679276530977345493022958064891113046805422308317132979749963467378921937799748997220709775693863201849843276213035220170
545609986454363649688738924003242283233840217150129748682588956582747963087606885500046294948362290586345709620961276928 9
374979551812079424992612524776964098670315062860833982712896496580209652265818710052177749601018268634694056192881801971
412595297556191324528937998152904330753015249551827717505770160953381451044451543347782430157121255731279700992538585215
932515665528805150133600611761358997307194927012520356077213596602633684699034284488910849840539184412861752193259315452
246176738190057730572995438895207000236623099243884076505984398862702283169615494664457971770570987245194733806323377122
286109671358484163490956375346459888738581962269779143849006354996175630784722798139982235293336772432981439666083041949
844566275218153109084134275113846364694249405528268547453378279436208678184219574972752175517012613535399534905666380302
020953429975260331393956656812841803685754758104089627323030398750742429645165556546169115553269759219431315130255562 46
120840048465833578351161205816617445169310771103115190663602507183841536120873692101731581646749526290557141549117250 30
425814161105559997027124793318828459960085593871242398252958038699416980182447786555363244184674123757645842637024065381
```

```
596935308564528073626647114113643070580514753787312750381977711286475543663087935357863164168758014129308963875253198862
654595257149816784182103395744978482681259217580815579619074400671791525472006042878557602774594655191057318536471996771
844829549301435988658254276688766710499692948849907504681233890302120690615021874955575989889213131060422977831670810 41
187756858587622362659710874999186306351606161881751791533071144356243503072548236578126339032301833932214768243450446076
546215503531642936182239633636385811688670571778861864160991173020990886148880382426313463334638563928251329292293440632
566524085488682556020848138573214990137866674087324317521837880861474015052622836241792933113303883752085594517898768295
821375387986639834878495672170462777967186971519909015894898616734179333521581001342356401303003610109565476104109591222
517044401265721639725369721984091454411669690674505627715443041918517320236802640545564447298478565733204333722951402934
602220878907076431174092629244853045388609058583374019347120706157193268810219917567810835286250340074435861653660982469
633473651169328479699632556730006394457047838662896045675565354054866829326251395536821826889945701382282055743315085377
558076704519355427234037456315229107129234520458141890558027526137706513643583784798607060872176775829019661560018081487
961909663707872025542811874710646966968336608251065322703223428064237047055508341249916565471793771656314815573207685823
368749182280730888919083522561541442756713823316443382258971741072627198193854577691742068876264446763965940106466936027
020366123173011942389948399002949797996104367980534999386508969071033488551861821644023744260577425996826859987231024113
767835439286860962584280821382521198271684888688163305338394177651569643021607505665913530800289662003939322128227107953
913460648991579219516942778577488855889952914922420300636158048865264323480272073464067440235372581890145511615641221513
616092969911983441384427415821010492563014085693561040698558019006431856499211494470260645913068840992705561721182382665
255004593624625699359469285979121572481215011787152301618492896251213797373595407083525551981643590692024635498114713236
366557323067311457655290559131727494540203647963662930161610554325430764210413722454942260442855947985932505535235434067
774719812532475107993524856251238842368
>>> pow(232, 2299) % 3233
65
```

4.4.3　RSA算法的安全性

回顾RSA密钥生成的过程，一共出现了6个数字：质数p和q、N、$\varphi(N)$、e以及d。这6个数字中，N和e组成了公钥，是对外公开的。其余4个数字都不公开，最关键的是d，因为它是私钥的一部分。那么，有没有可能在知道N和e的情况下推导出d呢？

已知$ed \equiv 1 \pmod{\varphi(N)}$，再加上$N$和$e$是公开的，如果能够计算出$\varphi(N)$，那么就能根据扩展欧几里得算法计算出$d$。而$\varphi(N) = (p-1)(q-1)$，只有知道$p$和$q$，才能计算出$\varphi(N)$。同时，我们也知道$N = pq$，因此，在已知$N$和$e$来推导$d$的最终目标就落在对$N$进行质因数分解了。所以，我们可以得到这样的结论：如果N可以被质因数分解，我们就可以根据公钥推导出私钥。

你也许觉得解决这个问题挺简单的，平时都写过对整数做因数分解的小程序。可是，对于大整数的因数分解，迄今为止还是一件非常困难的事情。目前，除了暴力破解之外，还没有发现别的有效方法。对极大整数做因数分解的难度决定了RSA算法的可靠性。换言之，对一极大整数做因数分解愈困难，RSA算法愈可靠。如果哪一天有人找到一种快速因数分解的算法，那么RSA的安全性就崩塌了。但找到这样的算法的可能性是非常小的。今天只有短的RSA密钥才可能被暴力破解。到目前为止，还没有任何可靠的攻击RSA算法的方式。

举例来说，你可以轻而易举地对3233进行因数分解（$3233 = 61 \times 53$），但是对下面这个整数进行因数分解就已经特别困难了。

$$
\begin{array}{l}
1230186684530117755130494958384962720772853569595334\\
7921973224521517264005072636575187452021997864693899\\
5647494277406384592519255732630345373154826850791702\\
6122142913461670429214311602222124047927473779408066535141959745985690214341 3
\end{array}
=
\begin{array}{l}
3347807169895689878604416984821269081770479498371376\\
8568912431388982883793878022876147116525317430877378149467999489
\end{array}
\times
\begin{array}{l}
3674604366679959042824463379962795263227915816434308\\
7642676032283815739666511279233373417143396810270092798736308917
\end{array}
$$

事实上，这大概是人类已经分解的最大整数（232个十进制位，768个二进制位）。比它更大的因数分解，还没有被报道过，因此目前被破解的最长RSA密钥就是768位。NIST建议的RSA密钥长度为至少2048位。

4.5　椭圆曲线密码学

椭圆曲线密码学（Elliptic Curve Cryptography，ECC）是一种基于椭圆曲线数学的公开密钥加密算法。椭圆曲线在密码学中的使用是在1985年由Neal Koblitz和Victor Miller分别独立提出的。ECC的主要优势是它相比RSA加密算法使用较小的密钥长度并提供相当等级的安全性。

4.5.1 群

在介绍椭圆曲线之前，有必要先来了解一下群（group）的概念。在数学上，群是由一种集合 $\mathbb{G}$以及一个二元运算（比如用符号+表示的“加法”运算）所组成的，并且符合包含下述四个性质的代数结构：

1. 封闭性（closure）：$a \in \mathbb{G}, b \in \mathbb{G} \Rightarrow a + b \in \mathbb{G}$。

2. 结合律（associativity）：$a \in \mathbb{G}, b \in \mathbb{G}, c \in \mathbb{G} \Rightarrow (a + b) + c = a + (b + c)$。

3. 单位元（identity element）：存在一个单位元用0表示，使得$a + 0 = 0 + a = a$。单位元是与任意元素运算不改变其值的元素。

4. 逆元（inverse）：对于$\mathbb{G}$中的每个元素a，存在$\mathbb{G}$中的一个元素b，使得$a + b = b + a = 0$。

群运算的次序很重要，把元素a与元素b进行二元运算，所得到的结果不一定与把元素b与元素a进行二元运算的结果相同，亦即，$a + b = b + a$（交换律，commutativity）不一定恒成立。如果把交换律作为第5个性质的话，我们把同时满足这五个性质的群称为阿贝尔群（abelian group）。

从我们通常的加法概念来看，整数集$\mathbb{Z}$是一个阿贝尔群。自然数集$\mathbb{N}$不是一个群，因为它不满足第4条性质。

4.5.2 实数域上的椭圆曲线

一条椭圆曲线就是一组由如下形式的方程定义的点集。

$$y^2 = x^3 + ax + b, \quad 4a^3 + 27b^2 \neq 0$$

椭圆曲线的定义要求曲线是非奇异的。几何上来说，这意味着图像里面没有尖点、自相交或孤立点。其中，$4a^3 + 27b^2 \neq 0$这个限定条件是为了保证曲线不包含奇点（singularity）。

图 4.2给出了a从-2到0，b从-1到3变化时对椭圆曲线形状的影响。从图中，可以看到随着a和b的变化，椭圆曲线也会在平面上呈现出不同的形状，但有一点是很容易辨认的，椭圆曲线始终是关于x轴对称的。我们观察到，

当$a=0,b=0$时，曲线存在尖点，不满足椭圆曲线规定的限定条件。另外，我们还可以找出曲线自相交和存在孤立点的情况，如图 4.3。这些特殊点，我们都称之为奇点，此时a和b的值不满足椭圆曲线的限定条件。

另外，我们还需要一个无穷处的点（point at infinity/ideal point）作为曲线的一部分，从现在开始，我们将用0这个符号表示无穷处的点。如果我们将无穷处的点也考虑进来的话，那么椭圆曲线的表达式精炼为：

$$\left\{(x,y)\in\mathbb{R}^2 \mid y^2=x^3+ax+b, 4a^3+27b^2\neq 0\right\}\cup\{0\}$$

椭圆曲线上的群论

我们可以在椭圆曲线上定义一个群：

- 群中的元素就是椭圆曲线上的点。
- 单位元就是无穷处的点0。
- 点P的逆元是关于X轴对称的另一边的点，记作$-P$。
- 二元运算规则定义如下：取一条直线和椭圆曲线相交的三点P、Q、R（皆非单位元），他们的总和等于单位元0，即$P+Q+R=0$。

请注意最后一条规则，我们仅仅说了需要三个在一条直线上的点，并没有规定它们的顺序。这就意味着，如果P、Q、R在一条直线上的话，它们满足

$$P+(Q+R)=Q+(P+R)=R+(P+Q)=\cdots=0$$

这样，我们可以直观地证明：定义在椭圆曲线上的加法运算是符合交换律和结合律的，这是一个阿贝尔群。

几何加法

由于椭圆曲线的点集属于一个阿贝尔群，所以，我们可以将$P+Q+R=0$写成$P+Q=-R$。这个方程式让我们派生出了一个用几何方法去计算两个点P和Q的和：当我们画一条直线通过P、Q，这条线将会和椭圆曲线相交

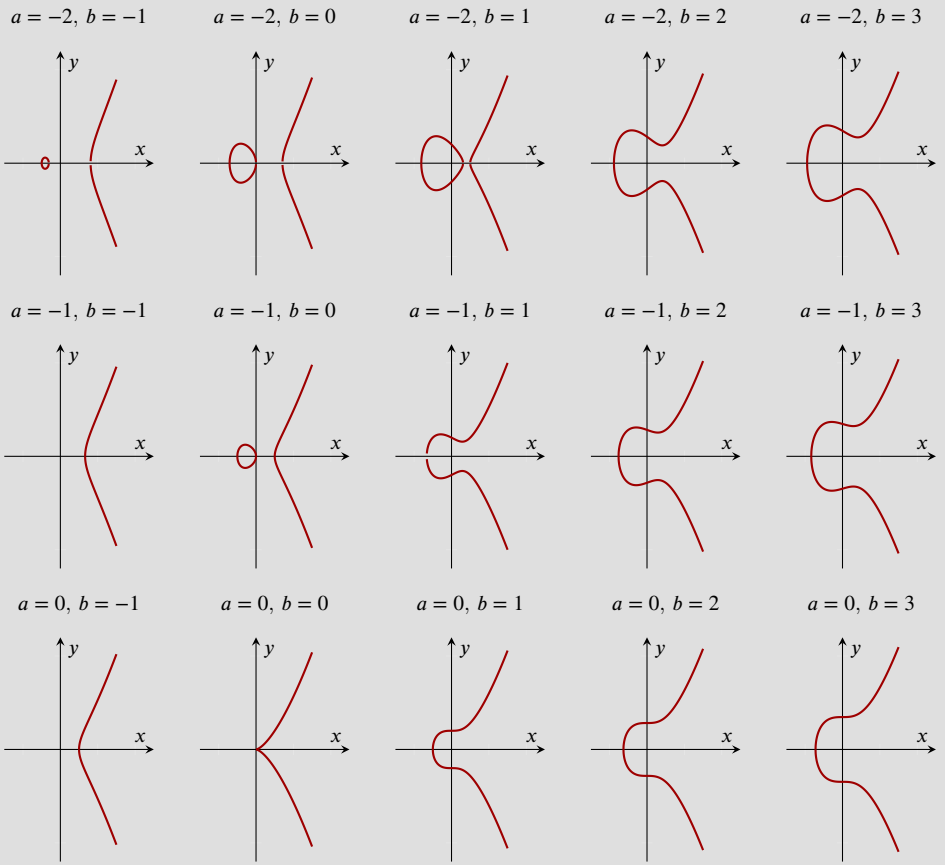


图4.2　椭圆曲线形状随参数a、b的变化

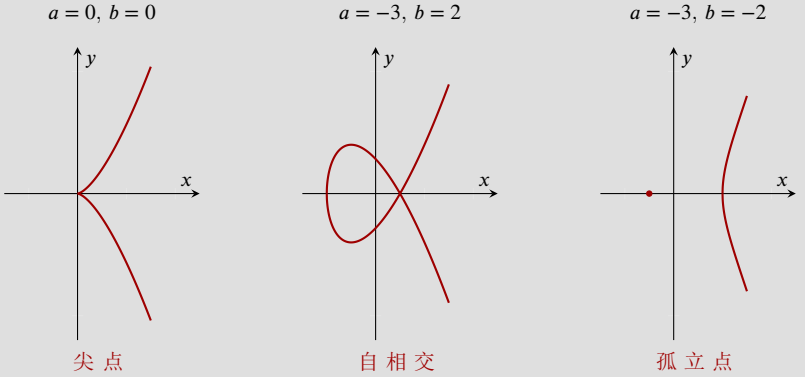


图4.3　当限定条件不满足时，椭圆曲线存在尖点、自相交或孤立点

于第三个点R（这就意味着 P、Q、R三点是在一条直线上的）。如果我们取它相反的点$-R$，我们就可以找到$P+Q$的结果，如图 4.4所示。

这个几何方法非常有用但是还需要再考虑以下几种情况（图 4.5画出了每种情况的一个例子）：

- **情形1：$P=-Q$**

  在这种情况下，穿过两点的直线是和x轴垂直的，和曲线没有相交的第三个点。此时，Q是P的逆元，从逆元的定义可以得到$P+Q=P+(-P)=0$。

- **情形2：$P=Q$**

  在这种情况下，有无数条线会经过这个点。我们假设一个点$Q' \neq P$. 当Q'越来越接近P并和P重合的时候，穿过P和Q'两点的这条线最终会成为曲线的一条切线，这条切线与曲线相交的另一点就是R，也就有$P+P+R=0$，或者写成$P+P=-R$，R是曲线和切线的交点，P是切点。

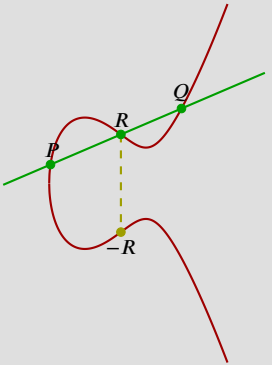


图4.4　穿过P和Q的直线与曲线相交的第三点R关于x轴对称的点$-R$就是$P+Q$的结果

- **情形3：$P \neq Q$但是经过P和Q的直线与椭圆曲线没有第三个交点**

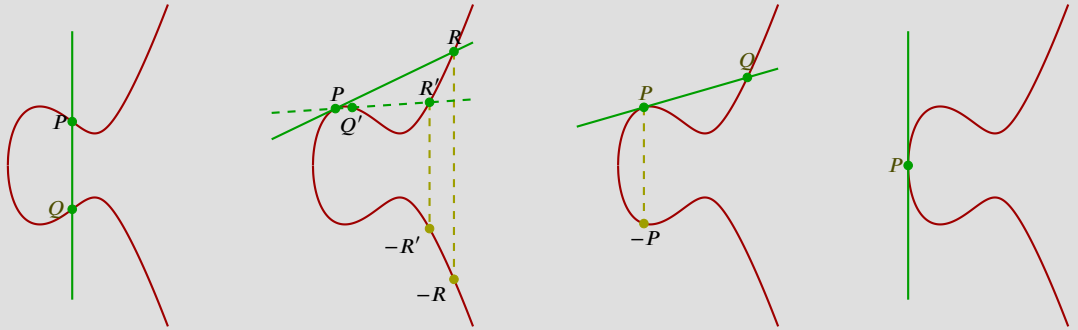
  这种情况与上一种情况非常相似。事实上，这种情况就是一条直线穿过P和Q与曲线相切。我们可以假设P是切点，在上一个情况下，我们已经说明了$P+P=-Q$，这个方程现在可以写成：$P+Q=-P$。

- **情形4：$P=Q$且$P=-Q$**

  这种情况是上述情形2或情形3的一个特例。此时，P为曲线和x轴的交点。经过P点的切线垂直于x轴和曲线没有相交的其他点。P的逆元为其自身，同时，$P+Q=P+P+0=0$。

- **情形5：$P=0$或者$Q=0$**

  很明显，这样我们是画不出线的，无穷远点0不在xy平面上。但是我们已经定义了0作为单位元。$P+0=P$和$Q+0=Q$，对于任意的P和Q都适用，单位元的作用就是与任意元素运算不改变其值的元素。



(1) $P+Q+0=0$　(2) $P+P+R=0$　(3) $P+Q+P=0$　(4) $P+P+0=0$

图4.5　椭圆曲线上几何加法的几种特殊情况

代数加法

如果我们想要一台计算机能够运行点的加法运算，那我们就需要把几何方法转换成代数方法。将一些规则转换成一系列的方程式看上去是非常直观的，但是实际上是很枯燥的，因为要算三次方程。出于这个原因，这里我只放结果。

首先，我们先去掉一些特殊情况，只会考虑两个非无穷处点$P(x_P, y_P)$、$Q(x_Q, y_Q)$。我们针对P和Q是否对称这两种情况分别考虑。

- 先假设P和Q不对称，即$x_P \neq x_Q$

  此时，经过P和Q的直线的斜率为

$$k = \frac{y_P - y_Q}{x_P - x_Q}$$

令该直线的方程为$y = kx + d$，直线与椭圆曲线相交，则有：

$$(kx + d)^2 = x^3 + ax + b \Rightarrow x^3 - k^2x^2 + (a - 2kd)x + (b - d^2) = 0$$

因为直线与椭圆曲线相交于第三点R，P、Q、R为直线与曲线的交点，即上述方程的解，有：

$$(x - x_P)(x - x_Q)(x - x_R) = x^3 - (x_P + x_Q + x_R)x^2 + (x_Px_Q + x_Px_R + x_Qx_R)x - x_Px_Qx_R$$

替换x^2的系数后，得到$x_P + x_Q + x_R = k^2$，这样，我们便能求得R的横坐标：

$$x_R = k^2 - x_P - x_Q \tag{4.5}$$

进而，通过斜率，我们接下来求得R的纵坐标：

$$y_R = y_P + k(x_R - x_P)$$

于是，$(x_P, y_P) + (x_Q, y_Q)$的结果为$(x_R, -y_R)$（请注意符号的变化，并且记住：$P + Q = -R$）。

- 若P和Q对称，即$x_P = x_Q$

  进一步考虑以下两种情况：

  1. 若$y_P = -y_Q$，即P和Q关于x轴对称，此时，$P + Q = 0$。
  2. 若$y_P = y_Q$，则P和Q重合，曲线在P点的切线斜率为下式的一阶导数。

     $$y_P = \pm\sqrt{x_P^3 + ax_P + b}$$

     根据一阶导数的计算方法，可以得到曲线在P点的切线斜率为：

$$k = \frac{3x_P^2 + a}{2y_P}$$

因此，根据公式 4.5，可以计算得到：

$$x_R = k^2 - 2x_P$$
$$y_R = y_P + k(x_R - x_P)$$

标量乘法和对数

除了加法，我们还需定义另一个运算：标量乘法，如下：

$$nP = \underbrace{P + P + \cdots + P}_{\text{n times}}$$

根据上述标量乘法的定义和几何加法的定义，$2P = P + P$的结果为经过P的切线与曲线的交点关于x轴的对称点，$3P = P + 2P$为经过P和$2P$与曲线的交点关于x轴的对称点，以此类推。图 4.6给出了标量乘法的过程图。

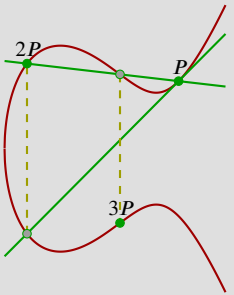


图4.6　椭圆曲线上点的标量乘法

从标量乘法的定义，也可以看出计算nP需要做n次加法运算。如果n有k位二进制的话，我们的算法时间复杂度是$O(2^k)$，当k特别大的时候，这不是一个好的结果。幸好还有一个被称作*快速幂算法*的方法，它的原理可以用如下例子解释。假设$n = 151$，二进制表示为10010111_2，这个二进制也可以表示成幂次加之和：

$$\begin{aligned} 115 &= 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \\ &= 2^7 + 2^4 + 2^2 + 2^1 + 2^0 \end{aligned}$$

由此，椭圆曲线上的点的标量乘法可以简化为：

$$151 \cdot P = 2^7 \cdot P + 2^4 \cdot P + 2^2 \cdot P + 2^1 \cdot P + 2^0 \cdot P$$

快速幂算法告诉我们的是，在该例中，只需要7次倍乘和4次加法操作就可以计算出$151P$。7次倍乘依次计算出$2^1 \cdot P$、$2^2 \cdot P$、$\cdots$、$2^7 \cdot P$。每次倍乘都在前一次倍乘结果的基础上乘以2。最后，将不需要的中间倍乘结果舍弃掉，把剩下来需要的倍乘结果通过4次加法操作就可以计算出$151P$的最终结果。

倍乘和加法都是时间复杂度为常数的运算，那么，这个算法的时间复杂度是$O(\log n)$。（或者是$O(k)$，如果我们考虑到比特长度的话），这个结果还是不错的。

对于给定的n和P，我们现在至少可以利用快速幂算法在$O(\log n)$的时间复杂度级别计算出$Q = nP$。那么反过来呢？如果我们已知Q和P，如何找到n呢？这个问题就是著名的对数问题（logarithm problem）。到目前为止，没有发现比穷举试探方法快太多的算法，于是椭圆曲线加密所依赖的数学难题就这么诞生了。

4.5.3　有限域的椭圆曲线

前面，我们已经了解了在实数域$\mathbb{R}$的椭圆曲线可以用来定义一个群。我们在实数域上面的椭圆曲线定义了一个点的加法运算，并对加法运算的几何方法和代数方法进行了详细的阐述。

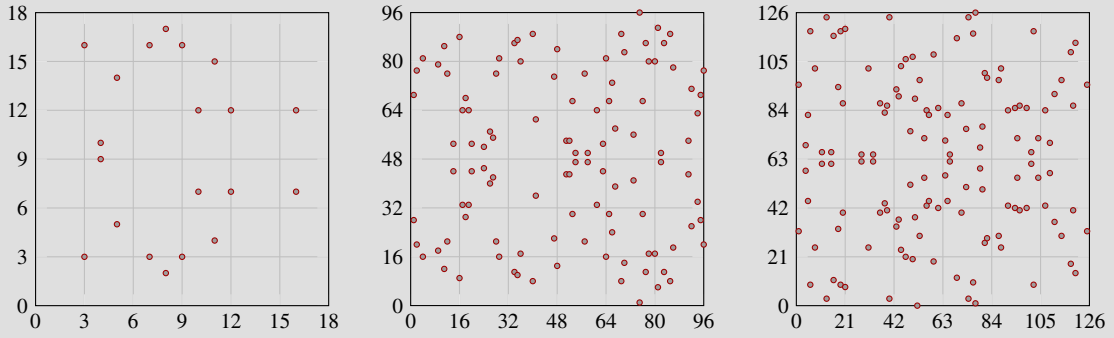
接下来，我们将椭圆曲线限定在有限域内，然后看看会有什么变化。我们对有限域的概念应该不感到陌生了，在介绍AES算法的 `MixColumn`运算的时候，我们曾经介绍过有限域（或伽罗瓦域）的概念。这里，我们用$\mathbb{F}_p$表

示有p个元素的有限域。回顾一下有限域上的加法和乘法运算，两者都满足封闭性、交换律和结合律，乘法相对加法还满足分配律，即 $x \times (y+z) = x \times y + x \times z$。

现在，我们对椭圆曲线在有限域$\mathbb{F}_p$上的定义如下：

$$E = \left\{(x,y) \in (\mathbb{F}_p)^2 | y^2 \equiv x^3 + ax + b \pmod{p}, \quad 4a^3 + 27b^2 \not\equiv 0 \pmod{p}\right\} \cup \{0\}$$

这里的0仍然是无穷处的点，a和b是$\mathbb{F}_p$上的两个整数。图 4.7给出了椭圆曲线 $y^2 = x^3 - 3x + 10$在有限域$\mathbb{F}_{19}$、$\mathbb{F}_{97}$和$\mathbb{F}_{127}$上的图形。从几何的角度，图形则从连续的曲线变成xy平面上的离散点的集合。即便对于定义域进行了限制，$\mathbb{F}_p$域上的椭圆曲线依然可以组成一个阿贝尔群。



$\mathbb{F}_{19}$　　$\mathbb{F}_{97}$　　$\mathbb{F}_{127}$

图4.7　有限域上的椭圆曲线$y^2 = x^3 - 3x + 10$

有限域上椭圆曲线的点加法

显然，为了使得点加法在$\mathbb{F}_p$域上依然有效，我们需要对$\mathbb{F}_p$域上三点共线的定义作一些小小的修改。在实数域，三点共线意味着能够找到一条直线将三个点连在一起。当然，在$\mathbb{F}_p$域中的直线，与实数域中的是有所不同的。不太严谨地说，$\mathbb{F}_p$中的直线是满足方程$y \equiv kx + d \pmod{p}$的点(x, y)的集合。

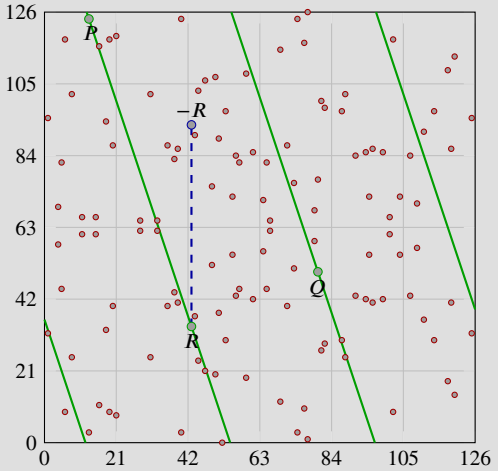


图4.8　椭圆曲线$y^2 = x^3 - 3x + 10$在有限域$\mathbb{F}_{127}$上的点加法

有限域上的椭圆曲线点加法保留了所有我们已知的特性：

- 单位元的定义：$P+0=0+P=P$。
- 逆元的定义：$P+(-P)=0$。
- 对于一个非0的点P，逆元$-P$是横坐标相同但是纵坐标相反的点。或者还有一种方式，$-P=(x_P,-y_P \pmod p)$。举个例子，如果曲线在$\mathbb{F}_{127}$上有一个点$P=(2,5)$，逆元是$-P=(2,-5 \pmod{127})=(2,122)$。

图 4.8展示了有限域$\mathbb{F}_{127}$上$y^2\equiv x^3-3x+10 \pmod{127}$的所有点。请注意，连接点$P=(13,124)$和$Q=(80,50)$的直线$y\equiv -3x+36 \pmod{127}$在图中多次重复，这是因为对127取模的原因。方程将P、Q连接上之后，和域中的$R=(43,34)$“相交”，R的逆元为$-R=(43,-34 \pmod{127})=(43,93)$，也就有$(13,124)+(80,50)=(43,93)$。

有限域上椭圆曲线的代数加法

除了在每一个表达式后面加上一个关于p的模运算以外，其它与代数加法一节中所描述的步骤都相同。因此，令$P=(x_P,y_P)$、$Q=(x_Q,y_Q)$、$R=(x_R,y_R)$，我们可以按如下方程计算$P+Q=-R$：

$$\begin{aligned} x_R &\equiv (k^2-x_P-x_Q) \pmod p \\ y_R &\equiv [y_P+k(x_R-x_P)] \pmod p \\ &\equiv [y_Q+k(x_R-x_Q)] \pmod p \end{aligned}$$

分以下两种情形计算斜率k。

- 如果$P\neq Q$，斜率k的形式如下：

$$k\equiv (y_P-y_Q)(x_P-x_Q)^{-1} \pmod p$$

  这里，$(x_P-x_Q)^{-1}$为x_P-x_Q关于p的模反元素。模反元素的计算可以使用扩展欧几里得算法求得。

- 如果$P=Q$，斜率k为：

$$k \equiv (3x_P^2 + a)(2y_P)^{-1} \pmod{p}$$

这里，你会发现数学的美妙之处：把椭圆曲线从实数域转换到有限域之后，点的几何加法和代数加法的公式表现出惊人的相似。

数乘和循环子群

在有限域$\mathbb{F}_p$上的椭圆曲线的乘法有个很有意思的属性。取一个曲线：$y^2 \equiv x^3 + 2x + 3 \pmod{97}$和点$P = (3, 6)$，现在来计算$P$的所有倍数：

$$0P = 0$$
$$1P = (3, 6)$$
$$2P = (80, 10)$$
$$3P = (80, 87)$$
$$4P = (3, 91)$$
$$5P = 0$$
$$6P = (3, 6)$$
$$7P = (80, 10)$$
$$\dots$$

到此，我们发现了两个规律：（1）P的倍乘只有5个取值，永远不会出现第6个，如图 4.9。（2）这些取值们是循环重复着的。我们可以写成这样：

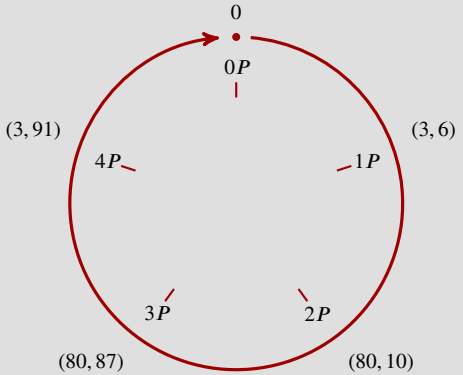


图4.9　$P = (3, 6)$的所有倍乘的取值只有5个点然后不停地循环重复

$$
\begin{aligned}
5kP &= 0 \\
(5k+1)P &= P \\
(5k+2)P &= 2P \\
(5k+3)P &= 3P \\
(5k+4)P &= 4P \\
&\cdots
\end{aligned}
$$

也可以把这5个式子“压缩”成一个：$kP = (k \pmod 5)P$。这个规则同样适用于所有的点，不仅仅是对$P = (3, 6)$。事实上，对于任意的P：

$$nP + mP = \underbrace{P + P + \cdots + P}_{\text{n times}} + \underbrace{P + P + \cdots + P}_{\text{m times}} = (n+m)P$$

这意味着：如果我们将P的倍乘进行相加，我们获得的仍然是P的倍数。这个性质非常重要，它证明了**nP的集合是椭圆曲线形成的群里的一个具有循环性质的子群**。这里的点P叫做循环子群的基点（base point）。

子群的阶

首先，我们要定义一下在一个群有多少个点就叫做这个群的“阶”（order）。穷举椭圆曲线在有限域$\mathbb{F}_p$中所有可能的值的时间复杂度为$O(p)$。当p很大的时候，这算下来就很慢很慢，有些不太可行。好在有一个更快的算法来计算阶—Schoof算法。这里，我们不对Schoof算法的细节进行展开，只需要知道它的复杂度是多项式时间。

在循环的子群里我们可以下一个新的、与前面的定义相等的定义，由P生成子群的阶是满足条件$nP = 0$的最小的正整数n。前面的例子中，$5P = 0$，那由$P = (3,6)$生成的子群的阶就等于5。由P生成的子群的阶不能使用Schoof的算法，因为这个算法只能在整个椭圆曲线上生效，在子群上无效。

由P生成的子群的阶和椭圆曲线是有联系的，*群论中的拉格朗日定理告诉我们，子群的阶是父群的阶的因子*。换句话说，如果一个椭圆曲线包含N个点，它的一个子群包含n个点，那么n是N的因子。

举个例子，在$\mathbb{F}_{37}$上的曲线$y^2 = x^3 - x + 3$的阶是$N = 42$。它的子群的阶则可能是$n = 1, 2, 3, 6, 7, 14, 21, 42$中的一个。如果我们代入曲线上的点$P = (2,3)$，我们可以发现$P \neq 0$、$2P \neq 0$、…、$7P = 0$。因此，由$P$生成的子群的阶是7。

找基点和离散对数

在ECC算法中，我们想找到一个阶数比较大的子群。所以通常呢，我们会选择一条椭圆曲线，然后去计算它的阶N，选择一个以较大的因子作为子群的阶n，最终，依此找到一个合适的基点。也就是说，我们的计算步骤并不是先选择一个基点然后去计算它的阶，而是反过来操作的。

首先，根据群论上的拉格朗日定理，我们知道，$h = N/n$里的h永远是一个整数（因为n整除N）。我们把h叫做辅因子（cofactor of the subgroup）。

现在，思考一下对于椭圆曲线中的每一个点，我们有$NP = 0$，且$N = hn$。因此，我们可以写成：$n(hP) = 0$。假设n是质数，这个方程式告诉我们：如果$G = hP \neq 0$，则生成了一个阶为n的子群。若$G = hP = 0$，则子群的阶是1，需要重新另外选择一个P。

现在我们总结一下寻找基点的算法：

1. 计算椭圆曲线的阶N。
2. 选择一个阶为n的子群，n必须是质数且必须是N的因子。
3. 计算辅因子$h = N/n$。
4. 在曲线上选择一个随机的点P。
5. 计算$G = hP$。
6. 若$G = 0$，那么回到步骤4。否则，我们已经找到了阶为n和辅因子是h的子群的生成器或基点。

请注意，上面这个算法仅仅适用于n是质数的情况下。如果n不是质数，那么G的阶可以是n的任何一个因子。

在标量乘法和对数一节中，我们已经引出了椭圆曲线的对数问题。现在，回到有限域上的椭圆曲线，我们可以提出相同的问题：如果我们已知P和G，怎么计算h呢？这个问题，就是有限域上椭圆曲线的离散对数问题。到目前为止，没有找到一个能在多项式时间内解出来的算法。这样一来，这个数学难题就奠定了椭圆曲线加密的安全基础。

我们知道在有限域上计算点的数乘是一个容易的过程，但是离散对数问题却是非常难的，接下来，我们就来看看这些理论是如何应用在密码学上的。

4.5.4　椭圆曲线加密算法

椭圆曲线加密算法建立在有限域上的椭圆曲线所形成的循环子群上，因此，我们的算法需要以下几个参数：

- 质数p：用于确定有限域的范围；
- 椭圆曲线方程中的a和b；
- 用于生成子群的的基点G；
- 子群的阶n；
- 子群的辅助因子h。

通常，我们使用六元组(p, a, b, G, n, h)来定义这些参数。

密钥对的生成

消息的接收方按照如下过程生成密钥对，自己保管私钥，将公钥发送给信息的发送方。

1. 选取椭圆曲线的参数p、a、b，并寻找椭圆曲线上一点作为基点G。
2. 在$\{1, \ldots, n-1\}$范围内随机选择整数d作为私钥。
3. 计算$H = dG$，H即为公钥。
4. 把椭圆曲线参数p、a、b，基点G，以及公钥H传给发送方。

接收方知道了私钥d和基点G（还有主要参数中的其他参数），求得公钥H是很容易的。相反，发送方知道公钥H和基点G，想要求得私钥d却是很困难的，因为这要求解决离散对数问题。

加解密过程

消息的发送方使用公钥对消息进行加密生成密文，接收方使用私钥对密文进行解密。

1. 发送方选择随机数r，将明文消息编码到椭圆曲线上的点M。
2. 发送方生成密文C，该密文是一个点对，$C = \{rG, M + rH\}$。
3. 发送方将密文C传给接收方。
4. 接收方收到密文C后进行解密计算：$M + rH - d(rG) = M + r(dG) - d(rG) = M$。
5. 接收方对M解码还原出明文消息。

4.5.5　基于椭圆曲线扩展的加密算法

一些基于离散对数问题的加密算法或数字签名算法，通过扩展椭圆曲线之后，增强了安全性。这些算法包括：

- 椭圆曲线迪菲-赫尔曼密钥交换（ECDH）
- MQV密钥协商算法（ECMQV）
- ElGamal离散对数密码体制（ECElGamal）
- 椭圆曲线数字签名算法（ECDSA）

对于采用椭圆曲线加密的算法来说，完成算法所必须的群操作比同样大小的质因数分解或模整数离散对数系统要慢。因此使用椭圆曲线的加密算法能用小得多的密钥长度来提供同等的安全，在这方面来说它确实比例如RSA之类的更快。 ECC被广泛认为是在给定密钥长度的情况下最强大的非对称加密算法，因此在对带宽要求十分紧的连接中会十分有用。

椭圆加密算法的应用范围很广，如 TLS、OpenPGP以及SSH都在使用，在比特币以及其他加密数字货币中也得到广泛使用。另外我国重点推广的国密SM2算法也正是基于椭圆曲线算法。

4.6 ECDH

ECDH是**DH密钥交换**在使用椭圆曲线加密方法后的变种。DH算法由于计算性能不佳，因为需要做大量的乘法，为了提升DHE算法的性能，所以就出现了现在广泛用于密钥交换算法—ECDH算法。ECDH算法是在DH算法的基础上利用了ECC椭圆曲线特性，可以用更少的计算量计算出公钥，以及最终的会话密钥。甲乙通信双方使用ECDH密钥交换算法的过程：

- 甲乙双方事先确定好椭圆曲线的参数a、b和有限域参数p，以及曲线上的基点G，这些参数都是公开的。
- 甲乙双方各自生成自己的私钥和公钥。甲方的私钥为d_A，公钥为$H_A = d_A G$；乙方的私钥为d_B，公钥为$H_B = d_B G$。
- 甲乙双方交换各自的公钥H_A和H_B。
- 最后，甲方计算$S = d_A H_B$（用自己的私钥点乘乙的公钥）；同样，乙方计算$S = d_B H_A$（用自己的私钥点乘甲的公钥）。双方求得的S是一致的，因为$d_A H_B = d_A(d_B G) = d_B(d_A G) = d_B H_A$。

这个过程中，如果双方的私钥都是随机、临时生成的，我们将之称为*ECDHE算法*，E表示短暂的（ephemeral），指的是交换的密钥是暂时的动态的，而不是固定的静态的。

静态的DH或者ECDH算法里有一方的私钥在每次密钥协商的时候都是固定不变的，通常是服务器方固定私钥不变，客户端的私钥则是随机生成的。于是，在静态模式下，DH或ECDH交换密钥时就只有客户端的公钥是变化，而服务端公钥是不变的，那么随着时间延长，黑客就会截获海量的密钥协商过程的数据，黑客就可以依据这些数据暴力破解出服务器的私钥，然后就可以计算出会话密钥了。这样一来所有的历史会话中传输的数据以及后续会话中传输的数据都将被黑客破解。所以，静态模式的DH或ECDH算法不具备前向安全性。

现在常用的是ECDHE交换算法。在这种模式下，每次会话的的私钥随机生成的，没有任何关系。黑客所截获的密钥协商过程的数据只在某一次会话中使用，他只能通过解决离散对数问题来破解私钥。退一步说，即便有个厉害的黑客破解了某一次通信过程的私钥，其他通信过程的私钥仍然是安全的，因为每个通信过程的私钥都是独立的，这样就保证了前向安全。

4.7　本章小结

在这一章，我们首先引出了对称加密算法的密钥配送问题，紧接着讨论了解决密钥配送问题的方法，并详细介绍了DH密钥交换算法、以及RSA和ECC两种非对称加密算法（公钥加密算法）。对于公钥加密算法中所涉及的数学数论知识和椭圆曲线进行了详细的解释。通过本章的学习，我们了解到：

1. DH密钥交换算法能够让通信双方在不安全的信道上创建出相同的共享密钥，这个共享密钥可以作为对称加密算法的密钥来加密通信内容，解决了密钥传送问题。
2. 受DH密钥交换算法的启发，RSA开创了非对称加密算法的先河，这种新的加密模式有两把密钥，一把是公钥，还有一把是私钥。公钥和私钥一一对应，有一把公钥就必然有一把与之对应的、独一无二的私钥，反之亦成立。公钥是公开的，任何人都可以拿到公钥，但是私钥需要妥善保管。通常，信息发送方用公钥对消息进行加密，接收方用私钥可以解开公钥加密的消息。由于只有接收方有私钥，这样一来，就不存在密钥配送的问题了。RSA加密算法的安全性建立在大数的质因数分解的困难性之上。
3. ECC是一种基于椭圆曲线数学的非对称加密算法。我们首先对实数域上的椭圆曲线进行了详细的描述，然后扩展到有限域上的椭圆曲线，并讨论了有限域上的计算点的数乘，随后引出了离散对数问题。离散对数问题奠定了椭圆曲线安全性的基础。和RSA相比， ECC的主要优势是可以使用较小的密钥长度提供相当等级的安全性。基于椭圆曲线加密的算法有ECDH、ECMQV、ECDSA等。