第 5 章

密码学

ZHQM ZMGM ZMFM
– G 朱利叶斯凯撒

KXJEY UREBE ZWEHE WRYTU HEYFS KREHE GOYFI
WTTTU OLKSY CAJPO BOTEI ZONTX BYBWT GONEY
CUZWR GDSON SXBOU YWRHE BAAHY USEDQ
约翰·肯尼迪

5.1 简介

密码学是安全工程与数学相遇的地方。它为我们提供了构成大多数现代安全协议基础的工具。它是保护分布式系统的关键技术，但要做好却出奇的难。正如我们在第 4 章“协议”中已经看到的，密码学经常被用来保护错误的东西，或者以错误的方式保护它们。不幸的是，可用的加密工具并不总是非常有用。

但是没有安全工程师可以忽视密码学。一位学医的朋友曾告诉我，她年轻时曾在海外工作，因为经济原因，他们缩短了医学学位，并专注于尽快培养专家。一天，一名双肾切除并等待移植的患者需要重做透析分流术。

外科医生以没有尿液分析存档为由，将患者从手术室送回。只是他没有想到，一个没有肾脏的病人是不能产生尿液的。

正如医生需要了解生理学和外科手术一样，安全工程师至少需要熟悉加密的基础知识（以及许多其他知识）。从广义上讲，人们可以通过三个层次来接触加密货币。第一个由潜在的直觉组成；我们用来澄清这些直觉的第二个数学，在可能的情况下提供安全证明，并整理导致最混乱的结构；第三个是密码工程 我们常用的工具，以及

他们可能出问题的经验。在本章中,我假设您没有接受过加密方面的培训,并着手解释基本的直觉。我用工程学来说明它们,并绘制足够多的数学图,以帮助您在需要时访问文献。您需要一些加密知识的原因之一是许多常见的结构令人困惑,而且许多工具提供不安全的默认值。例如,Microsoft 的 Crypto API (CAPI) 推动工程师使用电子密码本模式;到本章结束时,你应该明白那是什么,为什么不好,以及你应该怎么做。

许多加密货币教科书都假设他们的读者是纯数学专业的毕业生,所以让我从非数学定义开始 o 。密码学是指设计密码的科学和艺术;破解它们的科学和艺术的密码分析;而密码学,通常简称为密码学,是对两者的研究。加密过程的输入通常称为明文或明文,输出称为密文。此后,事情变得有些复杂。有许多基本构建块,例如块密码、流密码和哈希函数。分组密码可能有一个密钥用于加密和解密,在这种情况下它们被称为共享密钥(也称为秘密密钥或对称),或者有单独的密钥用于加密和解密,在这种情况下它们被称为公共-密钥或不对称。数字签名方案是一种特殊类型的非对称加密原语。

我将首先举一些历史例子来说明基本概念。然后,我将通过介绍密码学家使用的安全模型来微调定义,包括完美保密、具体安全、不可区分性和随机预言机模型。最后,我将展示更重要的加密算法的实际工作原理,以及它们如何用于保护数据。在途中,我将举例说明人们如何破解弱密码,以及如何使用强密码破解弱结构。

5.2 历史背景

Suetonius 告诉我们,Julius Caesar 通过将“A”写成“D”,将“B”写成“E”等来加密他的电报 [1843]。当奥古斯都凯撒登基时,他改变了帝国密码系统,现在“C”代表“A”,“D”代表“B”等等。在现代术语中,我们会说他将调从“D”更改为“C”。值得注意的是,Bernardo Provenzano 使用了类似的代码,据称是西西里黑手党的 capo di tutti capi,他用“4”代表“a”,“5”代表“b”等等。这直接导致他在 2006 年被意大利警方拦截并破译了他的一些信息 [1535]。

阿拉伯人将这个想法概括为单字母表替换,其中使用关键字来排列密码字母表。我们将明文写成小写字母,密文写成大写字母,如图5.1所示:

```
abcdefghijklmnopqrstuvwxyz
安全ABDFGHJKLMNOPQVWXZ
```

图 5.1 – 单字母代换密码

OYAN RWSGKFR AN AH RHTFANY MSOYRM OYSH SMSEAC NCMAKO;但破解这种密码是纸笔拼图。诀窍在于某些字母和字母组合比其他字母更常见;在英语中,最常见的字母依次是 e、t、a、i、o、n、s、h、r、d、l、u。人工智能研究人员已经试验了解决单字母替换问题的程序。仅使用字母和二元字母（字母对)频率,他们通常需要大约 600 个字母的密文;更聪明的策略,例如猜测可能的词,可以将其减少到大约 150 个字母;使用神经网络并接近人类分析师能力的最先进系统也在破译 Ugaritic 和 Linear B 等古代文字上进行了测试 [1194]。

基本上有两种方法可以生成更强的密码 流密码和块密码。在前者中,您使加密规则取决于明文符号在明文符号流中的位置,而在后者中,您在一个块中同时加密多个明文符号。

5.2.1 早期的流密码 Vigen`ere

这种早期的流密码通常被认为是法国人 Blaise de Vigen`ere 的发明,他是一位曾为国王查理九世服务的外交官。它的工作原理是使用“A”= 0、“B”= 1、...、“Z”= 25 的约定将密钥重复添加到明文中,并且加法以 26 为模执行 也就是说,如果结果大于 25,我们减去 26 的倍数以使其进入范围 [0, ..., 25],即 [A, ..., Z数学家把它写成

$$C = P + K \bmod 26$$

因此,例如,当我们将 P (15) 添加到 U (20) 时,我们得到 35,我们通过减去 26 将其减少到 9。9 对应于 J,因此 P 在密钥 U 下的加密（以及 U 在密钥 U 下的加密)键 P) 是 J,或更简单地说是 U + P = J。在这种表示法中,朱利叶斯凯撒的系统使用固定键 K = D,而奥古斯都凯撒使用 K = C,维吉尼亚使用重复键,也称为运行键。开发了技术来快速做到这一点,从印刷表格到黄铜密码轮。无论采用何种技术,使用重复关键字作为密钥的加密看起来如图 5.2 所示：

平心而论这是个问题
钥匙 跑跑跑跑跑跑跑跑跑跑
密码 KIOVIEEIGKIOVNURNVJNUVKHVMGZIA

图 5.2 – Vigen`ere（多字母替换密码）

从好色之徒贾科莫·卡萨诺瓦 (Giacomo Casanova) 到计算先驱查尔斯·巴贝奇 (Charles Babbage),许多人似乎已经找到了解决多字母密码的方法。但第一个公布的解决方案是 1863 年由普鲁士步兵军官弗里德里希·卡西斯基 [1020] 提出的。他注意到,给定一段足够长的密文,重复的模式将以关键字长度的倍数出现。

例如,在图 5.2 中,我们看到在九个字母后重复“KIOV”,在六个字母后重复“NU”。由于三整除六和九,我们可能会猜测三个字母的关键字。然后密文字母一、四、七等都用同一个密钥加密;所以我们可以使用频率分析技术来猜测这个字母最有可能的值,然后对密钥的其余字母重复该过程。

5.2.2 一次性一密本

使这种类型的流密码抵御攻击的一种方法是使密钥序列与明文一样长,并且永不重复。这被称为一次一密,由 Gilbert Vernam 在第一次世界大战期间提出 [1001];给定任何密文和任何相同长度的明文,有一个密钥可以将密文解密为明文。因此,无论对手可以进行多少计算,他们都不会更聪明,因为给定任何密文,该长度的所有可能明文都是同样可能的。因此,该系统具有完美的保密性。

这是一个例子。假设您截获了一条来自战时德国特工的消息,您知道该消息以“希特勒万岁”开头,并且密文的前十个字母是 DGTYI BWPJA。所以一次一密的前十个字母是wclnb tdefj,如图5.3所示:

平凡的黑希特勒
密钥 wclnbtdefj
密码 DGTYIBWPJA

图 5.3 – 间谍的消息

但是一旦他用他的关键材料烧掉了那块丝绸,间谍就可以声称他实际上是地下抵抗组织的成员,而这条信息实际上是说“绞死希特勒”。这也是可能的,因为密钥材料很容易就是 wggsb tdefj,如图 5.4 所示:

密码 DGTYIBWPJA
密钥 wggsbtdefj
普通吊带

图 5.4 – 间谍可以声称他说了什么

现在我们很少在密码学中不劳而获,一次一密的完美保密性的代价是它无法完全保护消息的完整性。所以如果你想让这个间谍陷入困境,你可以将密文更改为 DCYTI BWPJA（图 5.4）：

密码 DCYTIBWPJA
密钥 wclnbtdefj
普通吊带

图 5.5 – 操纵消息以诱捕间谍

Leo Marks 在第二次世界大战特种作战执行官中关于密码学的引人入胜的书 [1224] 讲述了如何将一次性密钥材料印在丝绸上,特工们可以将其隐藏在他们的衣服里;每当使用过钥匙时,它就会被撕裂和烧毁。事实上,在战争期间,克劳德·香农 (Claude Shannon) 证明了当且仅当可能的密钥与可能的明文一样多时,密码具有完美的保密性,并且每个密钥的可能性都相等;所以一次一密是唯一一种提供完美保密的系统。 1948 年 [1714 年,1715 年],他终于获准发表这篇文章。

一次性磁带从二战后期开始用于双方的高层通信,然后用于北约盟国之间的战略通信,1963年开始用于美苏热线。总共生产了数千台机器,使用纸张关键材料的磁带,直到它们最终被 1980 年代中期的计算机所取代1。但是这样的密码学对于大多数应用程序来说太昂贵了,因为它消耗的密钥材料与 trac 一样多。

流密码更常见的是使用伪随机数生成器将短密钥扩展为长密钥流。然后通过将密钥流（一次一个符号)与数据组合来对数据进行加密。从通过标准统计随机性测试的意义上来说,密钥流出现“随机”是不够的:它还必须具有这样的特性,即使对手拿到了相当多的密钥流符号,也不应该能够预测任何更多。

一个早期的例子是转子机器,机械流密码设备产生很长的伪随机状态序列 2并将它们与明文组合以获得密文。这些机器是由 20 年代的许多人独立发明的,其中许多人试图将它们出售给银行业。银行一般不感兴趣,原因我们将在下面讨论,但转子机器在第二次世界大战中被战斗人员广泛用于加密无线电通信,而盟军为破译德国通信所做的努力包括艾伦·图灵 (Alan Turing) 和其他人在巨像 (Colossus) 上的工作,这在战后帮助启动了计算机行业。

流密码已广泛用于必须最小化门数以节省功率的硬件应用中。然而,分组密码更灵活,并且在现在设计的系统中更常见,所以让我们接下来看看它们。

5.2.3 早期的分组密码 Playfair

Playfair 密码由查尔斯·惠斯通爵士于 1854 年发明,他是一位电报先驱,同时还发明了手风琴和惠斯通电桥。它不被称为惠斯通密码的原因是他向政客 Baron Playfair 展示了它;晚饭后,Playfair 又在餐巾纸上向阿尔伯特亲王和帕默斯顿子爵（后来的首相)展示了它。

这个密码使用 5 x 5 的网格,我们在其中放置字母表,按

1关于机器的信息可以在加密博物馆看到,https://www.cryptomuseum.com。

2个字母在美国使用的Hagelin机器的情况下,在情况下的排列德国的 Enigma 和英国的 Typex

关键字,并省略字母“J”（见图 5.6）：

图 5.6 – Playfair 加密表

明文首先通过用 I 替换 J 来调节,然后将其分成字母对,通过用 x 分隔它们来防止成对出现双字母,最后在必要时添加 z 完成最后一个字母对。 Playfair 在他的餐巾纸上写的例子是“Lord Granville s letter”,它变成了“lo rd gr an vi lx le sl et terz”。

然后使用以下规则一次加密两个字母：

- 如果这两个字母在同一行或同一列,则它们被后面的字母替换。例如， am 加密为 LE
- 否则这两个字母位于表格中一个矩形的两个角,我们将它们替换为该矩形另外两个角的字母。例如，“lo”加密为“MT”。

我们现在可以加密我们的样本文本如下：

Plain lord gr an vi lx le sl et terz
密码 MT TB BN ES WH TL MP TA LN NL NV

图 5.7 – Playfair 加密示例

这种密码的变体在第一次世界大战中被英国军队用作野战密码,在第二次世界大战中被美国人和德国人使用。这是对 Vigen`ere 的重大改进,因为分析师可以收集的统计数据是二合字母（字母对)而不是单个字母,因此分布更加平坦,攻击需要更多的密文。

同样,分组密码的输出仅仅看起来直观“随机”是不够的。 Playfair 密文看起来是随机的；但是它们有一个特性,如果你改变一对明文中的一个字母,那么通常只有一个密文字母会改变。因此使用图 5.7 中的密钥,rd 加密为 TB,而 rf 加密为 OB,rg 加密为 NB。一个结果是给定足够的密文或一些可能的词,可以重建表（或等效的表)[740]。事实上,本章开头的引述是未来总统杰克·肯尼迪发送的一条 Playfair 加密信息,当时他还是一名年轻的中尉,在他的机动鱼雷艇在海浪中沉没后与其他十名幸存者躲在一个小岛上。与日本驱逐舰相撞。

如果日本人拦截了它,他们可能已经解密了,历史可能会有所不同 。对于更强的密码,我们需要的效果

密码输入的微小变化会完全分散到输出中。
平均而言,改变一个输入位会导致一半的输出位发生改变。我们将在下一节中加强这些想法。

通过选择比两个字符更长的块长度,也可以大大提高块密码的安全性。例如,在支付系统中广泛使用的数据加密标准 (DES) 的块长度为 64 位,而在大多数其他应用中已取代它的高级加密标准 (AES) 的块长度为两倍这个。我在下面讨论 DES 和 AES 的内部细节;目前,我只想说我们需要的不仅仅是足够的块大小。

例如,如果银行帐号总是出现在交易中的同一位置,那么每次涉及它的交易都使用相同的密钥加密时,它很可能会产生相同的密文。这可能允许对手剪切和粘贴两个不同的密文的一部分,以产生有效但未经授权的交易。假设一个骗子在一家银行的电话公司工作,并监控了一笔他知道“支付 IBM 10,000,000 美元”的加密交易。他可能会向他的兄弟汇款 1,000 美元,导致银行计算机插入另一笔交易“支付约翰史密斯 1,000 美元”,拦截该指令,并从解密为“支付约翰史密斯 10,000,000 美元”的两个密文中组成一个错误指令。因此,除非密码块与消息一样大,否则密文将包含多个块,我们将需要某种方式将这些块绑定在一起。

5.2.4 哈希函数

第三种经典类型的密码是散列函数。这种演变是为了保护消息的完整性和真实性,我们不希望有人能够以这种方式操纵密文,从而导致明文发生可预测的变化。

19 世纪中叶发明电报后,银行迅速成为电报的主要用户,并开发了电子转账系统。什么是 有线 是付款指令,例如:

> 到伦敦的伦巴第银行。请从我们的帐户中付款。 1234567890 1000 英镑给切斯特顿路 456 号的约翰·史密斯,他在剑桥汇丰银行有一个账户。 301234 4567890123,并通知他这是“多琳史密斯的结婚礼物”。来自美国加利福尼亚州圣巴巴拉市第一牛仔银行。费用由我们支付。

由于电报信息是由人类从一个地方传递到另一个地方的
运营商,运营商有可能操纵支付信息。

在 19 世纪,银行、电报公司和航运公司开发了密码本,不仅可以保护交易,还可以缩短交易时间 考虑到当时国际电报的成本,这一点很重要。密码本本质上是一种块密码,它将单词或短语映射到固定长度的字母或数字组。所以“请从我们的

没有跟你交代。”可能会变成“AFVCT”。有时密码也被加密。

银行意识到流密码和密码本都不能保护消息的真实性。例如,如果“1000”的代码字是“淡紫色”,而“1,000,000”的代码字是“品红色”,那么可以将编码的 trac 与已知交易进行比较的歪曲的电报员应该能够弄清楚这一点并将一个替换为另一个。

对银行而言,关键的创新是使用密码本,但通过将代码组添加到一个称为测试密钥的数字中来实现单向编码。 (现代密码学家会将其描述为散列值或消息验证码,我稍后会更仔细地定义这些术语。)

这是一个简单的例子。假设银行有一本带表的密码本
付款金额对应的数字如图 5.8 所示:

	0 1		2个	3个	4个	5个	6个	7	8个	
9 x 1000 14 22 40 87 66 29 40 12 31 05 87 94	87 66 29 40 12 31 05 87 94									

图 5.8 – 一个简单的测试密钥系统

现在,为了验证一笔 376,514 英镑的交易,我们可能会将 53(不是百万)、54(300,000)、29(70,000)和 71(6,000)相加,忽略不重要的数字。这给了我们一个测试密钥 207。

大多数真实系统都比这更复杂;他们通常有货币代码、日期甚至收款人帐号的表格。在更好的系统中,代码组的长度是四位数而不是两位,为了让攻击者更难重建表,测试密钥被压缩:“7549”的密钥可能会通过添加变成“23”第一和第二位数字,第三和第四位数字,忽略进位。

这使得这样的测试密钥系统变成了单向函数,因为虽然可以从一条消息中计算出一个测试,但如果知道密钥,就不可能逆转这个过程并从一个单独的系统中恢复一条消息或一个密钥。测试 测试只是没有包含足够的信息。事实上,单向函数至少从 17 世纪就已经存在。科学家罗伯特·胡克 (Robert Hooke) 于 1678 年发表了经过排序的变位词“ceiiinosssttuu”,并在两年后透露它源自“Ut tensio sic uis” “力随着张力而变化”,或者我们现在所说的弹簧胡克定律。 (目标是确定这个想法的优先级,同时给他时间做更多的工作。)

按照现代密码学标准,银行测试密钥并不强大。
根据设计细节,给定几十到几百条经过测试的消息,耐心的分析师可以重建足够多的表格来伪造交易。通过同谋将几条精心挑选的信息插入银行系统,这就更容易了。但银行却侥幸逃脱:测试密钥从 19 世纪末到 19 世纪末都运行良好

80 年代。作为银行安全顾问工作了几年,午餐时听年长的审计员讲故事,我只听说过两起利用它进行欺诈的案例:一次是涉及密码分析的外部尝试,但由于攻击者不了解银行程序而失败,以及一个成功但小的欺诈,涉及一名不正当的工作人员 。我将在银行和簿记一章中讨论取代测试密钥的系统。

但是,测试密钥是我们用于身份验证的代数函数的历史示例。他们在核武器的指挥和控制中使用的认证代码以及现代分组密码中都有重要的现代后代。每种情况下的想法都是相同的:如果您可以使用唯一的密钥来验证每条消息,那么简单的代数就可以为您提供理想的安全性。

假设你有一个任意长度的消息 M 并且想要计算一个 128 位长的认证码 A,并且你想要的属性是没有人应该能够找到一个不同的消息M0 ,其认证码在同一密钥下也将是 A,除非他们知道密钥,除非幸运地猜测概率为 2128。您可以简单地选择一个 128 位素数 p 并计算 A = k1M + k2 (mod p),其中密钥由两个 128 -位数k1 和k2。

这是安全的,原因与一次性一密相同:给定任何其他消息M0 ,您可以找到另一个密钥（k0 1， k0 2）,该密钥向 A 验证M0。因此,在不知道密钥的情况下,攻击者无法创建有效伪造的信息。由于有 256 位的密钥和只有 128 位的标签,即使对于具有无限计算能力的对手来说也是如此:这样的对手可以很容易地找到每对消息和标签的2128个可能的密钥,但无法在它们之间进行选择.我将在下面讨论这个通用散列函数如何与块密码一起使用,以及它如何在第 2 部分的核命令和控制中使用。

5.2.5 非对称原语

最后,一些现代密码系统是不对称的,因为不同的密钥用于加密和解密。因此,例如,如今大多数网站都有一个包含公钥的证书,人们可以使用该公钥使用称为 TLS 的协议加密他们的会话;网页的所有者可以使用相应的私钥解密trac。我们稍后会详细介绍。

也有一些计算机前的例子;也许最好的是邮政服务。您可以给我发私信,给我写个地址,然后投进邮箱。完成后,我将是唯一能够阅读它的人。当然,很多事情都可能出错:您可能会弄错我的地址（无论是由于错误还是由于欺骗）;警察可能会得到授权打开我的邮件;信件可能被不诚实的邮递员偷走;欺诈者可能会在我不知情的情况下重定向我的邮件;否则小偷可能会从我的擦鞋垫上偷走这封信。公钥密码学也可能出现类似的问题:假公钥可以插入系统,计算机可以被黑,人们可以被胁迫等等。我们将在后面的章节中更详细地研究这些问题。

密码学的另一个非对称应用是数字签名。这

这里的想法是,我可以使用私有签名密钥对消息进行签名,然后任何人都可以使用我的公共签名验证密钥来检查它。同样,还有手稿签名和印章形式的计算机前类似物;再一次,无论是旧方法还是新方法,都有一连串可能出错的事情非常相似。

5.3 安全模型

在深入研究现代密码的详细设计之前,我想更仔细地了解各种类型的密码以及我们可以推断其安全性的方式。

安全模型试图将密码“好”的想法形式化。我们已经看到了完美保密的模型:给定任何密文,该长度的所有可能的明文都是同样可能的。类似地,可以设计只使用一次密钥的身份验证方案,以便对其进行最好的伪造攻击是随机猜测,通过选择足够长的标签,可以将其成功概率降低到我们想要的最低水平。

第二个模型是具体的安全性,我们想知道对手必须做多少实际工作。在撰写本文时,需要现存最强大的对手 比特币矿工社区,消耗与丹麦国家一样多的电力 大约需要十分钟才能解决 68 位密码难题并挖掘新区块。因此,一个 80 位的密钥将花费他们212倍的时间,或者大约一个月; 128 位密钥(现代系统中的默认密钥)再难 248倍。因此,即使在 1000 年内,偶然找到正确密钥的概率也是235或十亿分之一。一般而言,如果在时间 t 工作的对手以至多 的概率成功破解密码,则系统是 (t,)-安全的。

第三个模型,现在许多理论家称之为标准模型,是关于不可区分性的。这使我们能够推断我们关心的密码的特定属性关系。例如,大多数密码系统不会隐藏消息的长度,因此我们不能仅仅要求对手不能区分对应于两条消息的密文来定义密码是安全的;我们必须更明确地要求对手不能区分两个相同长度的消息 M1 和 M2。这是通过让密码学家和密码分析师玩一个游戏来正式化的,在这个游戏中,分析师通过找到一个她不应该能够以超过可忽略的概率来区分的东西的有效鉴别器来获胜。如果密码不具有完美的安全性,这可能是渐近的,我们通常希望努力比安全参数 n 的任何多项式函数增长得更快 比如密钥的长度(以位为单位)。安全证明通常包括一个缩减,我们表明如果存在一个随机(即概率)算法在 n 的时间多项式中运行,它学习信息它不应该以不可忽略的概率,那么这将为一个有效的鉴别器我们已经信任的底层加密原语。最后,如果明文没有有效的区分符,无论分析师可能掌握的关于它的任何辅助信息,都可以说一个结构具有语义安全性;即使她只知道一点点,即使她能解密任何内容

其他密文,她无法从目标密文中学到更多信息。这会跳过相当多的数学细节,您可以在 Katz 和 Lindell [1022] 等标准文本中找到这些细节。

第四种模型是随机预言机模型,它不像标准模型那样具有通用性,但通常会导致更高效的构造。如果没有有效的方法将其与该类型的随机函数区分开来,并且特别是它通过了我们应用的所有统计和其他随机性测试,则我们将其称为密码原始伪随机。当然,密码原语实际上是一种算法,实现为硬件门阵列或软件程序;但是输出应该“看起来是随机的”,因为在给定我们的计算模型允许的测试类型和测试数量的情况下,它们与合适的随机预言机无法区分。

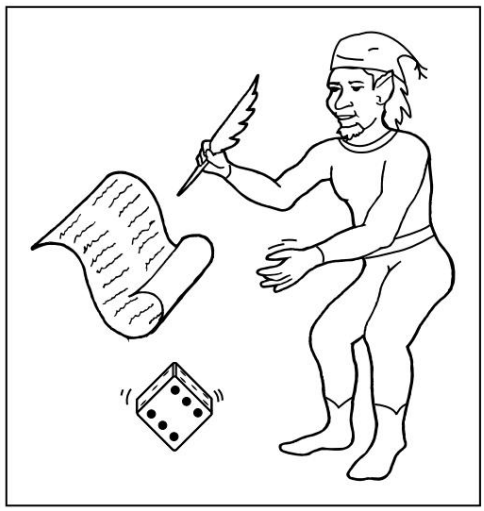


图 5.9：- 随机预言机

为了想象一个随机的神谕,我们可以想象一个小精灵坐在一个黑盒子里,黑盒子里有物理随机源和一些存储方式（见图 5.9） 在我们的图片中用骰子和卷轴表示。小精灵将接受某种类型的输入,然后在卷轴中查看此查询是否曾被回答过。如果是这样,它将给出在那里找到的答案;如果没有,它会通过掷骰子随机生成一个答案,并记录下来以备将来参考。我们将进一步假设带宽有限 小精灵每秒只会回答这么多的查询。更重要的是,我们的预言机可以根据几个不同的规则运行。

5.3.1 随机函数 散列函数

第一种随机预言机是随机函数。随机函数接受任意长度的输入字符串并输出固定长度的字符串,比如 n 位长。相同的输入给出相同的输出,但输出的集合看起来是随机的。所以小精灵只有一个简单的输入和输出列表,它会随着工作稳定增长。

随机函数是我们的加密哈希函数模型。这些在 1960 年代首次在计算机系统中用于密码的单向加密，并且在今天有更多用途。例如，如果警察没收了你的笔记本电脑，标准取证工具将计算所有文件的校验和，以确定哪些文件是已知的（如系统文件），哪些是新文件（如用户数据）。如果文件损坏，这些哈希值将发生变化，因此可以向法庭保证警方没有篡改证据。如果我们想要证明我们在特定日期之前拥有给定电子文档的证据，我们可能会将其提交给在线时间戳服务或将其挖掘到比特币区块链中。但是，如果文档仍然是机密的 例如我们想要为其确定优先权日期的发明 那么我们不会上传整个文档，而只是上传消息哈希。这是我们在上面第 5.2.4 节中讨论的胡克字谜的现代等价物。

5.3.1.1 属性

随机函数的第一个主要属性是单向性。给定输入 x 的知识，我们可以轻松计算哈希值 h(x)，但是如果这样的输入未知，则给定 h(x) 很难找到 x。 （小精灵只会为给定的输入选择输出，而不是相反。）由于输出是随机的，攻击者要反转随机函数所能做的最好的事情就是继续输入更多的输入，直到他幸运为止；对于 n 位输出，这平均需要大约2n1 次猜测。伪随机函数将具有相同的属性，或者它们可用于将其与随机函数区分开来，这与我们的定义相反。因此，伪随机函数也将是单向函数，前提是有太多可能的输出，对手无法偶然猜测具有所需目标输出的输入。这意味着选择 n 以便对手不能在接近2n 次计算时做任何事情。例如，如果我们声称 SHA256 是一个伪随机函数，那么我们就是说没有实用的方法可以找到散列为给定 256 位值的输入，除非您已经知道它并使用它来计算该值.

伪随机函数的第二个特性是输出不会给出任何关于输入的任何部分的信息。因此，我们可以通过将值 x 与密钥 k 连接并计算 h(x, k) 来获得值 x 的单向加密。但是，如果哈希函数不够随机，以这种方式使用它进行单向加密是自找麻烦。 （我将在 22.2.1 节稍后讨论一个例子：许多电话公司在 1990 年代和 2000 年代初期用于验证手机用户的哈希函数不够随机，这导致了攻击。）

具有足够长输出的伪随机函数的第三个特性是很难发现冲突，即不同消息M1 6= M2且 h(M1) = h(M2)。除非对手可以找到捷径攻击（这意味着该函数不是伪随机的），否则发现碰撞的最佳方法是收集大量消息Mi及其对应的哈希值h(Mi)，对哈希值进行排序，然后寻找比赛。如果散列函数的输出是一个 n 位数，那么就有2n个可能的散列值，那么敌人在期望找到匹配之前需要计算的散列数大约是这个的平方根，即2n/2 个哈希值。这个事实是非常重要的

安全工程,所以让我们更仔细地看看它。

5.3.1.2 生日定理

生日定理得名于以下问题。一位数学老师问一班 30 名学生,他们认为其中两人生日相同的概率是多少。大多数学生凭直觉认为这不太可能,然后数学老师让学生们陆续说出自己的生日。一旦召集了 23 名学生,比赛的几率就会超过 50%。由于这让大多数人感到惊讶,它也被称为“生日悖论”。

生日定理在 1930 年代首次用于计算鱼的数量,因此也被称为捕获-再捕获统计 [1665]。假设一个湖里有 N 条鱼,你抓了 m 条鱼,把它们圈起来然后扔回去,那么当你第一次抓到一条你已经圈起来的鱼时,m 应该是“大约”N 的平方根。

这成立的直观原因是,一旦你有 p N 个样本,每个样本都可能匹配任何其他样本,因此可能匹配的数量大约是 p N xp N 或 N,这就是你需要的3。

这个定理对安全工程师有很多应用。例如,如果我们有一个生物识别系统,可以以百万分之一的概率验证一个人的身份声明,那么两个随机选择的对象将被错误地识别为同一个人,这并不意味着我们可以使用它作为拥有两万名教职员工和学生的大学的可靠身份识别手段。这是因为将有近两亿对可能。事实上,一旦注册人数超过 1000 人,您就希望找到第一次碰撞 第一对可能被系统误认为彼此的人。然而,使用它来验证所声称的身份可能很好(尽管许多其他事情可能会出错;请参阅第 2 部分中有关生物识别的章节进行讨论)。

在某些应用程序中,碰撞搜索攻击不是问题,例如在挑战-响应协议中,攻击者必须找到刚刚发出的挑战的答案,并且您可以在其中防止重复挑战。

例如,在敌我识别 (IFF) 系统中,常见设备的响应长度为 48 到 80 位。你不能再买太多了,因为它会降低雷达的精度。

但是在其他应用程序中,碰撞是不可接受的。当我们设计数字签名系统时,我们通常首先通过加密散列函数传递消息 M,然后对散列 h(M) 进行签名,出于稍后讨论的多种原因。在这样的应用程序中,如果可以找到 h(M1) = h(M2)但M1 6= M2 的碰撞,那么 Mafia 拥有的书店的网站可能会预先计算合适的M1、 M2 对,让你签署一个M1说诸如“我特此以 32.95 美元的价格订购 Rubber Fetish 第 7 卷的副本”,然后出示签名和 M2 ,如“我特此以 75,000 美元抵押我的房子,请将资金发送给百慕大黑手党控股公司。”

出于这个原因,与数字签名方案一起使用的散列函数有 n

3更准确地说,从 N 条鱼中随机选择的 m 条鱼不同的概率是 = N(N 1) ... (N m+1)/Nm由 N m2/2log(1/) [1037]。

大到足以使它们无碰撞。从历史上看,两种最常见的哈希函数是 MD5,它具有 128 位输出,因此最多需要264次计算才能破解,而 SHA1 具有 160 位输出,密码分析员的工作因数最多为280。然而,碰撞搜索最多给出哈希函数强度的上限,结果这两个特定函数都令人失望,我将在 5.6.2 节稍后描述的密码分析攻击。

总结一下:如果你需要一个加密哈希函数来抗碰撞,那么你最好选择一个输出至少为 256 位的函数,比如 SHA-2 或 SHA-3。但是,如果您只需要确保没有人会为现有的、外部给定的散列找到第二个原像,那么您也许可以用更少的东西来凑合。

5.3.2 随机生成器 流密码

第二个基本密码原语是随机生成器,也称为密钥流生成器或流密码。这也是一个随机函数,但它与哈希函数相反,因为它具有短输入和长输出。如果我们有一个良好的伪随机函数,其输入和输出足够长,我们可以通过丢弃输出的几百位将其变成散列函数,并通过填充除了几百位以外的所有位将其变成流密码具有常量的输入位,并将输出用作密钥流。

它可以用来保护我们备份数据的机密性,如下所示:我们转到密钥流生成器,输入一个密钥,得到一个随机位的长文件,并将它与我们的明文数据异或得到密文,然后我们发送到我们在云端的备份服务。 (这也称为加法流密码,因为它是异或加法模 2)复制在他的卷轴上以供参考,以防他再次给出相同的输入键。如果我们需要恢复数据,我们回到生成器,输入相同的密钥,获得相同的密钥流,并与我们的密文进行异或运算,以再次取回我们的明文。其他有权访问密钥流生成器的人将无法生成相同的密钥流,除非他们知道密钥。请注意,这不会给我们任何文件完整性的保证;正如我们在一次一密的讨论中看到的那样,向明文添加密钥流可以保护机密性,但它无法检测文件的修改。为此,我们可能会对文件进行哈希处理并将其保存在安全的地方。保护散列不被修改可能比保护整个文件更容易。

一次一密系统非常适合我们的理论模型,除了它们用于保护跨空间而不是跨时间的通信:两个通信方已经提前共享了密钥流的副本。 Vernam 最初的电报密码机使用的是穿孔纸带; Marks 描述了 SOE 代理人的丝质钥匙是如何在牛津由退休的女士们制造的。我们将在物理安全一章中讨论现代硬件随机数生成器。

密钥流生成器的一个真正问题是防止相同的密钥流

被多次使用,无论是加密多于一个备份磁带还是加密多于一条在通信通道上发送的消息。第二次世界大战期间,俄罗斯的外交通讯量超过了他们事先分发给大使馆的一次性磁带的数量,因此被重复使用。但是如果M1 +K= C1， M2 +K= C2,那么对手就可以将这两个密文结合起来,得到两条消息的组合： C1 C2 = M1 M2,如果消息Mi有足够的冗余度,则可以恢复。

事实上,短信确实包含了足够的冗余,可以恢复很多东西;在俄罗斯 trac 的情况下,这导致了 Venona 项目,在该项目中,美国和英国从 1943 年开始解密了大量战时俄罗斯的 trac,并打破了一些俄罗斯间谍网络。用一位前美国国家安全局首席科学家的话来说,它变成了“两次录音带”。

为避免这种情况,正常的工程实践不仅要有密钥,还要有种子（也称为初始化向量或 IV）,因此我们每次都在不同的地方启动密钥流。种子 N 可以是序列号,或者以更复杂的方式从协议生成。在这里,您需要确保双方在正确的工作密钥上同步,即使存在可能试图让您重用旧密钥流的对手。

5.3.3 随机排列 分组密码

第三种原语,也是现代密码学中最重要的原语,是分组密码,我们将其建模为随机排列。这里,函数是可逆的,输入明文和输出密文都是固定大小的。

使用 Playfair,输入和输出都是两个字符;对于 DES,它们都是 64 位的位串。无论符号的数量和底层字母表有多少,加密都会作用于固定长度的块。 （所以如果你想加密一个较短的输入,你必须像我们的 Playfair 示例中的最后一个 z 一样填充它。）

我们可以将块加密可视化如下。和以前一样,我们在一个盒子里有一个小精灵,盒子里有骰子和卷轴。这在左边有一列明文,在右边有一列密文。当我们要求小精灵加密一条消息时,它会检查左侧栏以查看它是否有一条记录。如果不是,它掷骰子生成一个适当大小的随机密文（并且还没有出现在卷轴的右手栏中）,然后在卷轴中写下明文/密文对。如果它确实找到了一条记录,它会从右侧的列中为我们提供相应的密文。

当被要求解密时,小精灵做同样的事情,但各列的功能相反:他获取输入的密文,在右边的卷轴上寻找它,如果他找到它,他会给出之前与之关联的消息.如果没有,他会随机生成一条新消息,记下并提供给我们。

分组密码是伪随机排列的键控族。对于每个键,我们都有一个独立于所有其他排列的排列。我们可以认为每个键对应一个不同的卷轴。直观的想法是,密码机应该在给定明文和密钥的情况下输出密文,在给定密文和密钥的情况下输出明文,但只给定明文和密文应该什么都不输出。此外,没有人

应该能够推断出关于它尚未产生的明文或密文的任何信息。

我们将使用为加密建立的符号编写分组密码
关于协议的章节：

$$C = \{M\}K$$

随机排列模型还允许我们定义不同类型的分组密码攻击。在已知的明文攻击中，对手只是从对应于目标密钥的 oracle 中随机选择了一些输入和输出。在选择明文攻击中，允许对手进行一定数量的明文查询并得到相应的密文。在选择的密文攻击中，他可以进行许多密文查询。在选择的明文/密文攻击中，他被允许进行任何一种类型的查询。

最后，在相关密钥攻击中，他可以使用与目标密钥 K 相关的密钥进行查询，例如 K + 1 和 K + 2。

在每种情况下，攻击者的目标可能是推断出他尚未提出的查询的答案（伪造攻击），或者恢复密钥（不出所料，称为密钥恢复攻击）。

这种关于攻击的精确性很重要。当有人发现密码原语中的漏洞时，它可能与您的应用程序相关，也可能不相关。通常不会，但会被媒体大肆宣传 所以你需要能够向你的老板和你的客户清楚地解释为什么这不是问题。所以你得仔细看，弄清楚究竟发现了什么样的攻击，参数是什么。例如，针对数据加密标准算法（差分密码分析）宣布的第一次主要攻击需要247个选定的明文来恢复密钥，而下一个主要攻击（线性密码分析）将其改进为243个已知明文。虽然这些攻击具有巨大的科学重要性，但它们的实际工程效果为零，因为没有任何实际系统可以向攻击者提供如此多的已知文本（更不用说选择的文本）。这种不切实际的攻击通常被称为认证攻击，因为它们影响密码的安全认证，而不是提供实际的利用。不过，它们可能会产生商业影响：对 DES 的攻击破坏了信心并开始将人们转移到其他密码。在其他一些情况下，以 o 作为认证开始的攻击已被后来的想法发展成漏洞利用。

您应该担心哪种类型的攻击取决于您的应用程序。例如，对于广播娱乐系统，黑客可以购买解码器，观看大量电影并将其与加密的广播信号进行比较；因此，已知明文攻击可能是主要威胁。但令人惊讶的是，有许多应用程序可能会发生选择明文攻击。一个历史性的例子来自第二次世界大战，当时美国分析家了解到日本打算建造一座岛屿“AF”，他们怀疑这意味着中途岛。于是他们安排中途岛的指挥官发送了一条未加密的消息，报告其淡水冷凝器出现问题，然后截获了一份日本报告“AF 缺水”。知道中途岛是日本的目标，切斯特尼米兹海军上将正在等待他们并击沉了四艘日本航母，扭转了战争的潮流[1001]。

其他攻击更专业。选择明文/密文攻击可能令人担忧,因为威胁是午餐时间攻击:有人在授权用户外出时临时访问加密设备,并使用他们选择的数据尝试所有允许的操作一段时间.相关密钥攻击是一个问题,其中块密码被用作构建哈希函数（我们将在下面讨论)的构建块。

如上所述,为了排除所有此类攻击,目标是语义安全;密码不应允许推断未经授权的信息（无论是明文、密文还是密钥）,除非概率可忽略不计。

5.3.4 公钥加密和陷门单向突变

公钥加密算法是一种特殊的块密码,其中小精灵将为任何请求它的人执行与特定密钥相对应的加密,但只会为密钥的所有者执行解密操作。继续我们的类比,用户可能给卷轴一个只有她和小精灵知道的秘密名称,使用小精灵的公共单向函数计算这个秘密名称的散列,发布散列,并指示小精灵为任何引用此散列的人执行加密操作。这意味着委托人,比如爱丽丝,可以发布一个密钥,如果鲍勃愿意,他现在可以加密一条消息并将其发送给她,即使他们从未见过面。所需要的只是他们可以访问 oracle。

最简单的变体是陷门单向排列。这是一种任何人都可以执行的计算,但只有知道暗门（例如密钥)的人才能将其逆转。该模型类似于加密散列函数的“单向函数”模型。尽管如此,让我们正式声明一下:公钥加密原语由一个函数组成,给定随机输入 R 将返回两个密钥,KR（公钥)和 KR1 （私钥）,具有以下属性:

1.给定KR,计算KR1是不可行的（所以也不可能计算R）;

2. 有一个加密函数 {...},它使用加密密钥 KR 应用于消息 M,将产生密文 C = {M}KR;和

3. 有一个解密函数,它使用解密密钥KR1应用于密文 C ,将产生原始消息 M = {C}KR1 。

出于实际目的,我们希望在通信通道的两端复制预言机,这意味着要么使用防篡改硬件,要么（更常见的）使用数学而不是金属来实现其功能。

在大多数真实系统中,加密是随机的,因此每次有人使用相同的公钥加密相同的消息时,答案都是不同的;这对于语义安全是必要的,因此对手无法检查对给定密文的明文猜测是否正确。有

甚至比这要求更高的模型,例如在对手可以解密他们选择的密文（目标密文除外)的情况下分析安全性。但这暂时可以。

5.3.5 数字签名

我们将在此处定义的最终加密原语是数字签名。基本思想是消息上的签名只能由一个主体创建,但任何人都可以检查。因此,它可以在电子世界中执行与普通签名在纸质世界中相同的功能。应用程序包括对软件更新进行签名,以便 PC 可以判断 Windows 的更新确实是由 Microsoft 而不是外国情报机构制作的。

签名方案也可以是确定性的或随机的:在第一种情况下,计算消息上的签名总是会给出相同的结果,而在第二种情况下,它会给出不同的结果。 （后者更像是手写签名;没有两个是完全相同的,但银行有办法确定给定样本是真实的还是伪造的。)此外,签名方案可能支持也可能不支持消息恢复。如果他们这样做了,那么根据签名,任何人都可以恢复生成它的消息;如果他们不这样做,那么验证者在执行验证之前需要知道或猜测消息。

形式上,签名方案,如公钥加密方案,具有密钥对生成函数,给定随机输入 R 将返回两个密钥,

R（私人签名密钥)和 VR（公共签名验证密钥)与
的属性

1.给定公开签名验证密钥VR,计算不可行
私人签名密钥 R;

2. 有一个数字签名函数,给定一个消息M和一个签名私钥R,将产生一个签名SigR{M};和

3. 有一个验证函数,给定签名SigR{M}和公共签名验证密钥 VR,如果使用 R 正确计算签名,则输出 TRUE,
否则输出 FALSE。

在我们不需要消息恢复的地方,我们可以将一个简单的数字签名自然算法建模为一个随机函数,该函数将任何输入消息减少为固定长度的单向哈希值,然后是一种特殊的块密码,其中 elf将仅对一个委托人执行一个方向的操作,称为签名。另一方面,它将为任何人执行验证。

对于这个简单的方案,签名验证意味着elf（或签名验证算法)只根据签名是否良好输出TRUE或FALSE。但是在消息恢复的方案中,任何人都可以输入一个签名并取回它对应的消息。在我们的精灵模型中,这意味着如果精灵之前见过签名,它会在滚动条上给出对应的消息,否则会给出一个随机值（并将输入和随机输出记录为签名和消息

一对）。这有时是可取的：当通过低带宽通道发送短消息时，如果只需要发送签名而不是签名加上消息，则可以节省空间。使用消息恢复的应用程序是机印邮票或邮戳：邮票由二维条形码和邮政计价器制作的数字签名组成，其中包含价值、日期和发件人姓名等信息收件人的邮政编码。我们将在 16.3.2 节末尾讨论这个问题。

一般情况下我们不需要消息恢复；待签名的消息可能是任意长度的，所以我们先通过一个哈希函数，然后对哈希值进行签名。我们需要哈希函数不仅是单向的，而且是抗碰撞的。

5.4 对称密码算法

现在我们已经整理了定义，我们将深入了解如何在实践中实施它们。虽然大多数解释都是针对研究生数学专业的学生，但我在这里给出的介绍是基于我多年来与计算机科学本科生一起开发的，以帮助非专业人士掌握要点。事实上，即使在研究层面，大部分密码学与数学一样都是计算机科学：现代对密码的攻击是从猜测位、搜索模式、排序可能的结果等组合在一起的，需要的是聪明才智和毅力，而不是任何东西特别高调。

5.4.1 SP 网络

克劳德·香农 (Claude Shannon) 在 1940 年代提出，可以通过反复组合替换和换位来构建强密码。例如，可以将一些关键材料添加到输入文本块中，然后整理输入的子集，并以这种方式继续多次。他将密码的属性描述为混淆和扩散 添加未知密钥值会使攻击者混淆明文符号的值，而扩散意味着通过密文传播明文信息。块密码需要扩散和混淆。

最早的块密码是结合了替换和置换电路的简单网络，因此被称为 SP 网络 [1009]。图 5.10 显示了一个具有 16 个输入的 SP 网络，我们可以将其想象为一个 16 位数字的位，以及两层四位可逆替换框（或 S 盒），每个都可以可视化为包含数字 0 到 15 的一些排列的查找表。

这种安排的要点是，如果我们要在数字逻辑中实现任意 16 位到 16 位功能，我们将需要220位内存 每个输出位对应一个216位的查找表。那是数百

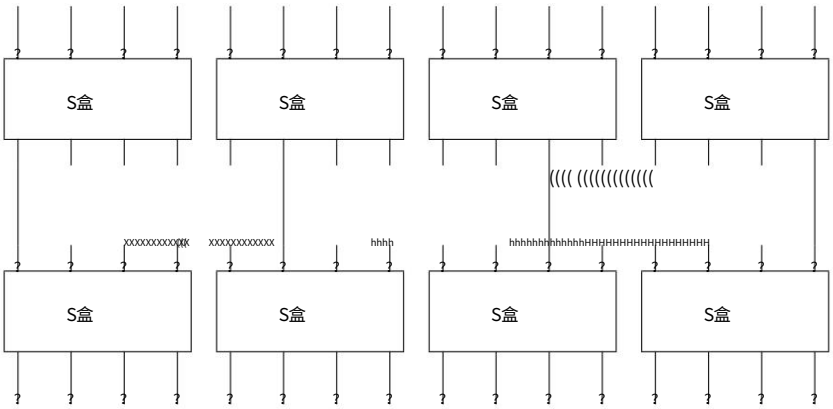


图 5.10：- 一个简单的 16 位 SP 网络分组密码

数千个门,而一个四位到四位的函数只需要 4 x 24或 64 位内存。人们可能希望通过选择合适的参数,对于不知道密钥值的对手来说,通过迭代这个简单结构产生的函数将无法与随机的 16 位到 16 位函数区分开来。密钥可能由一些四位 S-box 的选择组成,或者它可能在每一轮添加以提供混淆,并且通过 S-box 提供的结果文本提供 diusion。

要使这样的设计安全,需要做三件事：

1.密码需要足够“宽”

2.它需要有足够的轮数,并且

3. the S-boxes need to be suitably chosen.

5.4.1.1 区块大小

首先,在 16 位块上运行的块密码相当有限,因为对手只能根据观察到的明文和密文块构建字典。生日定理告诉我们,即使输入的明文是随机的,他也希望在看到几百个块后立即找到匹配项。所以一个实用的分组密码通常会处理64位、128位甚至更多位的明文和密文。因此,如果我们使用四位到四位 S 盒,我们可能有 16 个（对于 64 位块大小)或 32 个（对于 128 位块大小）。

5.4.1.2 回合数

其次,我们必须有足够的回合。图 5.10 中的两轮完全不够,因为对手可以通过以适当的模式调整输入位来推断 S 盒的值。例如,他可以保持最右边的 12 位不变并尝试调整最左边的四位,以推导出左上角 S-box 中的值。 （攻击比

这一点,因为有时对 S-box 的输入位进行调整不会导致任何输出位发生变化,因此我们必须更改其其他输入之一并再次进行调整。
但它仍然是一项基本的学生练习。）

我们需要的轮数取决于数据通过密码扩散的速度。在我们的简单示例中,扩散非常慢,因为一轮 S-box 的每个输出位仅连接到下一轮的一个输入位。与其简单地排列导线,不如进行线性变换更有效,其中一轮中的每个输入位都是前一轮中几个输出位的异或。如果块密码要用于解密和加密,则此线性变换必须是可逆的。我们将在下面有关 AES 和 DES 的部分中看到一些具体示例。

5.4.1.3 S盒的选择

S-box 的设计也会影响安全所需的轮数,研究错误的选择可以让我们进入更深层次的分组密码理论。假设 S-box 是将输入 (0,1,2,...,15) 映射到输出 (5,7,0,2,4,3,1,6,8,10, 15,12,9,11,14,13）。然后输入的最高有效位将作为输出的最高有效位不变地通过。如果在上述密码的两轮中使用相同的 S 盒,则输入的最高有效位将通过成为输出的最高有效位。我们当然不能声称我们的密码是伪随机的。

5.4.1.4 线性密码分析

对真实分组密码的攻击通常比这个例子更难发现,但它们使用了相同的想法。结果可能是 S-box 具有输入的第一位等于输出的第二位加上输出的第四位的属性;更常见的是,S-box 的线性近似值以一定的概率成立。线性密码分析 [895, 1244] 通过收集许多关系来进行,例如“第一个 S-box 输入的第 2 位加上第 5 位等于输出的第 1 位加上第 8 位,概率为 13/16”,然后寻找方法将它们粘合在一起,形成输入位、输出位和密钥位之间的代数关系,该关系以不同于一半的概率成立。如果我们能找到一个以概率 p = 0.5+1/M 保持整个密码的线性关系,那么根据概率论中的采样定理,一旦我们有大约M2个已知文本,我们就可以期望开始恢复密钥位。如果最佳线性关系的M2值大于已知文本的可能总数（即2n ,其中输入和输出为 n 位宽） ,那么我们认为密码对于线性密码分析是安全的。

5.4.1.5 差分密码分析

差分密码分析 [245, 895] 类似,但基于 S 盒输入的给定变化将导致输出发生特定变化的概率。对 8 位 S-box 的典型观察可能是“如果我们

一次翻转输入位 2、3 和 7,那么将翻转的输出位只有 0 和 1 的概率为 11/16”。事实上,对于任何非线性布尔函数,调整输入位的某些组合将导致输出位的某些组合以不同于一半的概率 发生变化。分析过程是查看所有可能的输入差异模式并寻找那些值i,这样输入变化将产生具有特别高（或低)概率的输出变化。

。 o

与线性密码分析一样,我们然后寻找将事物连接起来的方法,以便我们可以输入密码的输入差异 erence 将产生已知的输出差异 erence,并在多轮中具有有用的概率。给定足够多的选择输入,我们将看到预期的输出并能够对密钥进行推论。与线性密码分析一样,如果攻击所需的文本数量大于该密钥可能的不同文本总数 ,则通常认为密码是安全的。 （我们必须小心病态情况,例如,如果你有一个带有 32 位块的密码和一个具有差分攻击的 128 位密钥,在给定一对的情况下成功概率为240。给定大量文本在许多键下,我们最终会解决当前键。）

这两个主题有很多变化。例如,我们可以寻找不可能发生（或很少发生)的差异,而不是寻找高概率差异。这有一个迷人的名字不可能的密码分析,但它在许多系统上绝对是可能的 [242]4。

分组密码设计涉及许多权衡取舍。例如,我们可以通过仔细设计轮数来减少每轮信息泄漏,从而减少所需的轮数。但是一个复杂的设计在软件上可能会很慢,或者在硬件上需要很多门,所以使用简单的循环但更多的循环可能会更好。简单的回合也可能更容易分析。

一个谨慎的设计者也会使用比绝对必要的更多的轮数来阻止今天已知的攻击,以便提供一些安全余量,因为攻击只会变得更好。但是,虽然我们可能能够证明一个密码可以抵抗我们所知道的所有攻击,并且有一定的安全余量,但这并不能说明它是否能抵抗新型攻击。 （分组密码的一般安全性证明似乎暗示了诸如 P 6= NP 之类的结果,这将彻底改变计算机科学。）

5.4.2 高级加密标准（AES）

高级加密标准 (AES) 是一种算法,最初以其发明者 Vincent Rijmen 和 Joan Daemen [507] 的名字命名为 Rijndael。它作用于 128 位块,可以使用长度为 128、192 或 256 位的密钥。它是一个 SP 网络;为了指定它,我们需要修复 S 盒、轮次之间的线性变换以及将密钥添加到计算中的方式。

AES 使用单个 S-box 作用于字节输入以给出字节输出。出于实现的目的,它可以被简单地视为一个查找表

4 这可能在第二次世界大战期间首先在 Bletchley 使用,在那里对打破德国的 Enigma 机器是没有任何字母能被自己加密。

256 字节； it is actually defined by the equation S(x) = M(1/x) + b over the field GF(28) where M is a suitably chosen matrix and b is a constant.这种结构给出了严格的差分和线性边界。

线性变换是基于将被加密的值的 16 个字节排列成一个正方形,然后进行按字节排序和混合操作。第一步是 shu e,其中最上面一行四个字节保持不变,第二行向左移动一位,第三行向左移动两位,第四行向左移动三位。第二步是列混合步骤,其中一列中的四个字节使用矩阵乘法进行混合。这在图 5.11 中进行了说明,该图作为示例显示了如何传播第一列中第三个字节的值的变化。

这种组合的效果是,密码输入的变化可能会在仅仅两轮之后影响所有输出 雪崩效应使线性和差分攻击都变得更加困难。

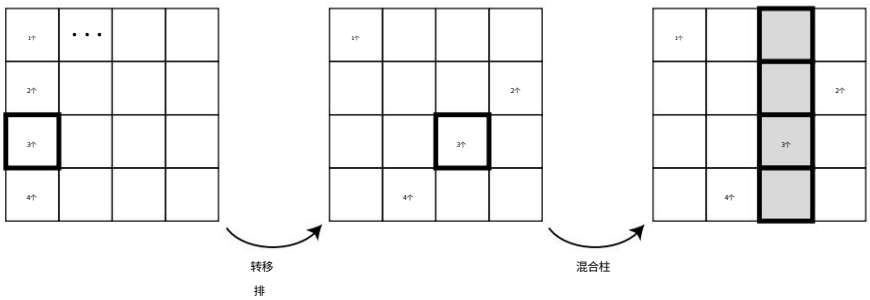


图 5.11：- AES 线性变换,通过它对输入字节 3 的影响来说明

密钥材料经过线性变换后逐字节添加。这意味着每轮需要 16 个字节的密钥材料;它们是通过递归关系从用户提供的密钥材料中派生出来的。

该算法使用 10 轮 128 位密钥、12 轮 192 位密钥和 14 轮 256 位密钥。这些足以提供实用的安全性,但不是认证的安全性 正如我们在 AES 竞赛时所期望的那样,正如我在本章的早期版本中所描述的那样。第一次密钥恢复攻击使用一种称为 biclique 密码分析的技术,于 2009 年由 Andrey Bogdanov、Dmitry Khovratovich 和 Christian Rechberger [273] 发现;它们只提供了很小的优势,现在估计128 位 AES 的复杂度为2126,256位 AES 的复杂度为2254.3 ,而2127和2255

蛮力搜索。在我们有相关键的情况下,更快的快捷方式攻击是众所周知的。但这些攻击在实践中都没有任何区别,因为它们需要不可行的大量文本或相关键的非常特殊的组合。

我们应该相信 AES 吗?俄罗斯、中国和日本政府试图让公司改用本地密码,日本产品 Camellia 与 AES 和另一个 AES 竞赛决赛入围者 Bruce Schneier 的 Twofish 一起出现在许多加密库中。 （Camellia 是由一个团队设计的,该团队自己的 AES 候选人在第一轮就被淘汰了。）阴谋论者

请注意,美国政府选择了 AES 竞赛决赛入围的五种算法中最弱的一种。好吧,我是 AES 决赛选手 Serpent [94] 的设计师之一,它在比赛中排名第二:获胜者 Rijndael 获得 86 票,Serpent 59 票,Twofish 31 票,RC6 23 票和 MARS 13 票。 Serpent 具有易于分析的简单结构 图 5.10 的结构,但修改得足够宽且具有足够的轮数 并且设计为比 Rijndael 具有更大的安全余量,以应对现在已经发生的攻击出现了。然而一个简单的事实是,虽然 Serpent 更安全,但 Rijndael 更快;行业和加密研究人员在上一届 AES 会议上投票支持它,NIST 批准它作为标准。

我参与了整个过程,并在 1990 年代的大部分时间从事共享密钥密码的分析和设计,因此我非常有信心 AES 能够抵御基于数学密码分析的实际攻击。尽管 AES 不如 Serpent 安全,但实际的安全性完全取决于实施,我们现在在实施 AES 方面拥有丰富的经验。实际攻击包括时序分析和功率分析。

在前者中,主要风险是对手观察到缓存未命中并使用它们来计算密钥。在后者中,对手使用设备进行加密时所消耗的电流的测量值 想想客户在黑手党拥有的商店的终端中放置的银行智能卡。我在第 2 部分排放安全一章中详细讨论了这两个问题;对策包括在许多 CPU 中执行 AES 的特殊操作,这些操作之所以可用,正是因为该算法现在已成为标准。实现 Serpent 也没有意义,“以防万一 AES 被破坏”:具有可交换算法被称为可插入密码术,但算法协商协议中出现致命错误的风险比任何人都会对 AES 进行生产攻击。 (稍后我们将看到许多示例,其中使用多种算法导致某些东西严重崩溃。)

背后的故事是,早在 1970 年代,美国国家安全局操纵了以前的标准分组密码的选择和参数,即数据加密标准 (DES),以提供一种足以满足当时美国工业界需求的密码。时间,同时让外国政府相信它是不安全的,所以他们改用自己的弱设计。一旦我描述了 DES 的设计,我将在下面更详细地讨论这个问题。 AES 似乎遵循了这个剧本;通过选择一种仅在数学上足够强大并且其安全实施需要技巧和谨慎的算法,美国政府确保俄罗斯、中国、日本和其他地方的公司最终将使用不太安全的系统,因为技巧和安全性较低 ort 已投入实施。然而,这可能是运气而不是马基雅维利式的狡猾:NIST 的相关委员会将不得不有很大的勇气来无视投票并选择另一种算法。哦,自 2005 年以来,NSA 已批准使用 128 位密钥的 AES 来保护最高机密的信息,并使用 192 位或 256 位密钥来保护绝密信息。所以我建议你使用 AES 而不是 GOST,或者 Camellia,甚至 Serpent。 AES 的最终规范是联邦信息处理标准 197,它的发明者已经写了一本书来详细描述它的设计 [507]。

5.4.3 Feistel 密码

许多分组密码使用更复杂的结构,这是 Feistel 和他的团队在 1950 年代末和 1960 年代初开发 Mark XII IFF 时发明的。 Feistel 随后转到 IBM,并成立了一个研究小组,该小组开发了数据加密标准 (DES) 算法,该算法仍然是支付系统安全性的支柱。

Feistel 密码具有图 5.12 所示的梯形结构。输入被分成两个块,左半部分和右半部分。计算左半部分的轮函数f1并使用异或（不带进位的二进制加法)与右半部分组合,尽管在某些 Feistel 密码中也使用带进位的加法。 （我们使用异或符号。)然后,计算右半部分的函数f2并将其与左半部分组合,依此类推。最后（如果轮数是偶数)交换左半边和右半边。

您可能会看到 Feistel 密码的符号是 (f, g, h, ...),其中 f, g, h, ... 是连续的轮函数。在这种表示法下,上述密码是(f1, f2, ...f2k1, f2k)。使我们能够解密 Feistel 密码的基本结果　实际上也是他设计的全部要点　是:

$$1(f1, f2, ..., f2k1, f2k) = (f2k, f2k1, ..., f2, f1)$$

换句话说,要解密,我们只需按相反的顺序使用轮函数即可。因此轮函数fi不必是可逆的,并且 Feistel 结构让我们可以将任何单向函数转换为分组密码。这意味着我们在尝试选择具有良好扩散和混淆属性的轮函数时受到的限制较少,并且它还满足任何其他设计约束,例如代码大小、软件速度或硬件门数。

5.4.3.1 Luby-Racko　结果

1988 年 Mike Luby 和 Charlie Racko　证明了 Feistel 密码的关键理论结果。他们表明如果fi是随机函数,则(f1, f2, f3)在选择明文攻击下与随机排列无法区分,这个结果很快被扩展到表　明(f1, f2, f3, f4)在选择的明文/密文攻击下是无法区分的　换句话说,它是一个伪随机排列。 （我省略了一些技术细节。）

在工程术语中,效果是给定一个非常好的轮函数,四轮 Feistel 就足够了。因此,如果我们有一个我们有信心的散列函数,就可以直接从中构造分组密码:在 Feistel 网络中使用四轮密钥散列。

5.4.3.2 DES

DES 算法广泛用于银行和其他支付应用程序。
广泛部署的“杀手级应用”是 ATM 网络;从那里

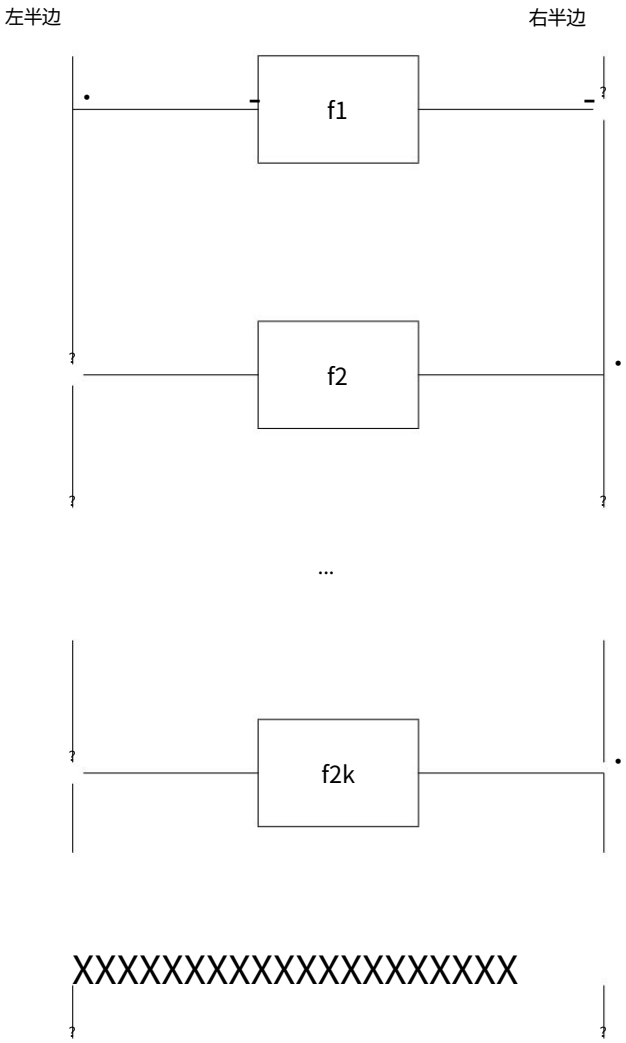


图 5.12：- Feistel 密码结构

它扩展到预付费计价器、交通票和许多其他领域。在其经典形式中,它是一种 Feistel 密码,具有 64 位块和 56 位密钥。它的 round 函数在 32 位半块上运行,由三个操作组成:

- 首先,块从32 位扩展到48 位;
- 接下来,使用异或混合48位的轮密钥。
- 结果通过一行八个 S-box,每个 S-box 接受一个六位输入并提供四位输出;
- 最后,输出的位根据固定模式进行置换。

扩展、密钥混合和 S 盒的效果如图 5.13 所示:

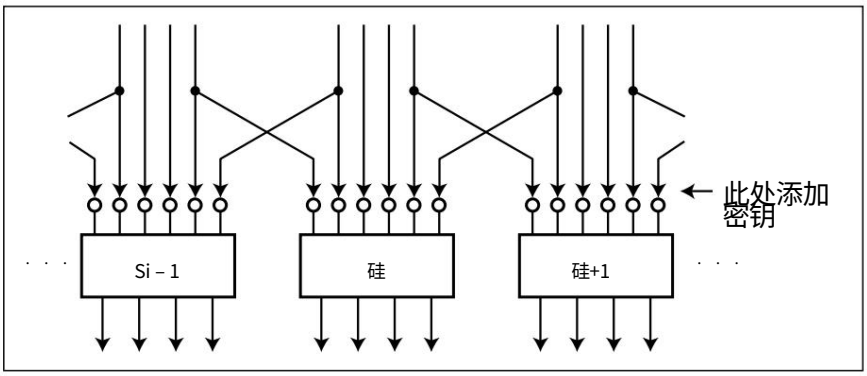


图 5.13：- DES 轮函数

轮密钥是通过根据略微不规则的模式在十二个不同的轮次中使用每个用户密钥位从用户提供的密钥派生的。 [1397] 中给出了 DES 的完整规范。

DES 于 1974 年推出,立即引起争议。最明显的批评是钥匙太短了。想要使用蛮力找到 56 位密钥的人,即通过尝试所有可能的密钥,将有256次加密的总耗尽时间和一半的平均解决时间,即255次加密。 Whit Die 和 Martin Hellman 在 1977 年争论说 DES 密钥搜索机器可以用一百万个芯片构建,每个芯片每秒测试一百万个密钥;因为一百万大约是220,所以平均需要215秒或 9 个多小时才能找到密钥。他们争辩说,在 1977 年可以用 2000 万美元制造这样一台机器 [557]。 IBM 的科学家发明了 DES,IBM 反驳说他们将向美国政府收取 2 亿美元来建造这样的机器。 （事后看来,他们都是对的。）

在 1980 年代,一直有各种情报机构正在建造 DES 密钥搜索机器的谣言,但第一次成功的公钥搜索攻击发生在 1997 年。在通过网络组织的分布式努力中,14,000 台 PC 花费了四个多月的时间找到挑战的关键。 1998 年,电子前沿基金会 (EFF) 以不到 25 万美元的价格建造了一台名为 Deep Crack 的 DES 密钥搜索机器,它在 3 天内破解了 DES 挑战。它包含 1,536 个以 40MHz 频率运行的芯片,每个芯片包含 24 个搜索单元

每个需要 16 个周期来进行测试解密。因此,搜索速率为每个搜索单元每秒 250 万次测试解密,或每个芯片每秒 6000 万个密钥。破解器的设计是公开的,可以在 [619] 中找到。到 2006 年,波鸿大学和基尔大学的 Sandeep Kumar 及其同事使用 120 个 FPGA 构建了一台机器,耗资 10,000 美元,平均可以在 7 天内破解 DES [1108]。一个拥有 100,000 台机器的现代僵尸网络需要几个小时。所以现在单个 DES 的密钥长度是不够的。

对 DES 的另一个批评是,由于 IBM 应美国政府的要求对其设计原则保密,因此可能有一个“活板门”可以让他们轻松访问。然而,设计原则是在 1992 年差分密码分析被发明和发表后发表的 [473]。故事是 IBM 在 1972 年发现了这些技术,而美国国家安全局 (NSA) 甚至更早。 IBM 应美国国家安全局的要求对设计细节保密。我们将在 26.2.7.1 中讨论所有这些的政治方面。

我们现在对 DES 有了相当透彻的分析。最著名的捷径攻击,即涉及比密钥搜索更少计算的密码分析攻击,是使用242 个已知文本的线性攻击。 DES 在 20 轮以上时是安全的,但出于实际目的,它的安全性受到其密钥长度的限制。我不知道在任何实际应用程序中,攻击者可能会获得甚至240条已知文本。所以已知的捷径攻击不是问题。然而,它对关键字搜索的脆弱性使得单个 DES 在大多数应用程序中无法使用。与AES一样,也有基于时序分析和功耗分析的攻击。

处理 DES 密钥长度问题的通常方法是使用不同的密钥多次使用该算法。银行网络已在很大程度上转向三重 DES,这是 1999 年以来的标准 [1397]。 Triple-DES 进行加密,然后解密,然后进一步加密,所有这些都使用独立的密钥。正式地:

$$3DES(k0, k1, k2; M) = DES(k2; DES1(k1; DES(k0; M)))$$

通过将三个密钥设置为相等,您可以获得与单个 DES 加密相同的结果,从而提供与旧设备的向后兼容模式。
（一些银行系统使用双密钥三重 DES,设置k2 = k0;这在单重 DES 和三重 DES 之间提供了一个中间步骤。）大多数新系统使用 AES 作为默认选择,但许多银行系统致力于使用分组密码一个八字节的块,因为 ATM、销售点终端和银行网络相互通信的许多协议中使用的消息格式,并且因为使用块密码来生成和保护客户 PIN（这我在有关银行业务和簿记的章节中进行了讨论）。

在可预见的未来,三重 DES 是用于此类目的的完美可用的分组密码。

另一种防止关键字搜索（并使功率分析更加困难)的方法是白化。除了 56 位密钥,比如k0,我们还选择了两个 64 位白化密钥k1和k2,第一个与加密前的明文异或,第二个与加密后的输出相异或,得到加密后的密文。这种复合密码称为 DESX。正式地,

$$DESX(k0, k1, k2; M) = DES(k0; M \oplus k1) \oplus k2$$

可以证明,在合理的假设下,DESX 具有您期望的属性;它继承了 DES 的不同强度,但它对关键字搜索的抵抗力随着白化量的增加而增加 [1047]。白化块密码用于某些应用程序,最具体的是我在下面讨论的 XTS 操作模式。如今,它通常与 AES 一起使用,AESX 的定义也类似,使用白化密钥使每个块加密操作都是唯一的我们将在下面的 5.5.7 节中看到。

5.5 操作模式

一个常见的失败是加密库允许甚至鼓励开发人员使用不适当的操作模式。这指定了如何将具有固定块大小（DES 为 8 个字节,AES 为 16 个字节)的块密码扩展为处理任意长度的消息。

在多个块上使用块密码有几种标准操作模式[1404]。了解它们至关重要,这样您就可以为工作选择合适的工具,尤其是一些常用工具默认提供较弱的工具。这种弱模式就是我们接下来要讨论的电子密码本（ECB）模式。

5.5.1 如何不使用分组密码

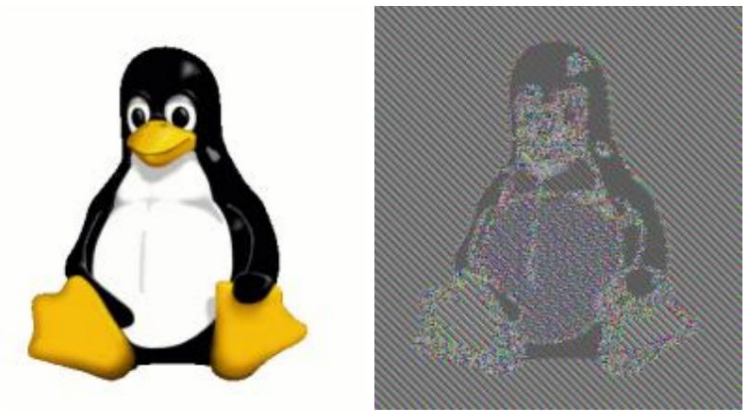
在电子密码本模式下,我们只是用我们的块密码加密每个后续的明文块以获得密文,就像上面的 Playfair 示例一样。这对于使用单块的协议来说已经足够了,例如挑战-响应和一些关键管理任务;它还用于加密取款机系统中的 PIN。

但如果我们用它来加密冗余数据,模式就会显示出来,向对手提供有关明文的信息。例如,图 5.14 显示了在 ECB 模式下使用 DES 加密卡通图像时发生的情况。重复的明文块都加密为相同的密文,使图像易于识别。

在上个世纪的一个流行的企业电子邮件系统中,使用的加密是 DES ECB,密钥来自八个字符的密码。如果您查看该系统生成的密文,您会发现某个块比其他块更常见 对应于空明文的块。这给出了有史以来对现场 DES 加密系统最简单的攻击之一:只需用字典中的每个密码加密一个空块,然后对答案进行排序。现在,您可以立即破解其密码为字典中密码之一的任何密文。

此外,使用 ECB 模式加密需要真实性的超过一个块长度的消息（例如银行支付消息)是特别愚蠢的,因为它使您容易受到沿块边界的剪切和拼接攻击。例如,如果银行消息说“请向帐号 X 支付总金额 Y,他们的参考号是 Z”,那么攻击者可能会发起一项旨在将 X 的某些数字替换为 Z 的某些数字的支付。

,



(a) 明文 (b) ECB密文

图 5.14:Linux 企鹅,明文和 ECB 加密（来自维基百科,源自 Larry Ewing 创建的图像）。

5.5.2 密码块链接

大多数加密多个块的商业应用程序过去使用密码块链接或 CBC 模式。与 ECB 一样,这是使用 DES 标准化的原始操作模式之一。在其中,我们在加密之前将前一个密文块与当前明文块进行异或运算（见图 5.15）。

这种模式隐藏了明文中的模式:每个块的加密取决于所有先前的块。输入初始化向量 (IV) 通过加密成相同的密文来确保定型明文消息头不会泄露信息,就像使用流密码一样。

然而,知道一些明文的对手可能能够剪切和拼接一条消息（或使用同一密钥加密的几条消息的一部分）。事实上,如果在密文中插入一个错误,它只会影响解密时的两块明文,因此如果明文没有任何完整性保护,敌人可以插入两块乱码的随机数据他们选择的地点。因此,CBC 加密通常必须与单独的身份验证代码一起使用。

更微妙的事情也可能出错;系统必须将明文填充为块大小的倍数,并且如果解密消息并发现不正确填充的服务器表明了这一事实,无论是通过返回“无效填充”消息还是只需要更长的时间来响应,那么这将打开一个填充 oracle 攻击,其中攻击者调整输入密文,一次一个字节,观察错误消息,并最终能够解密整个消息。这是 Serge Vaudenay 在 2002 年发现的;直到 2016 年 [1949],它的变体才被用于对抗 SSL、IPSEC 和 TLS。

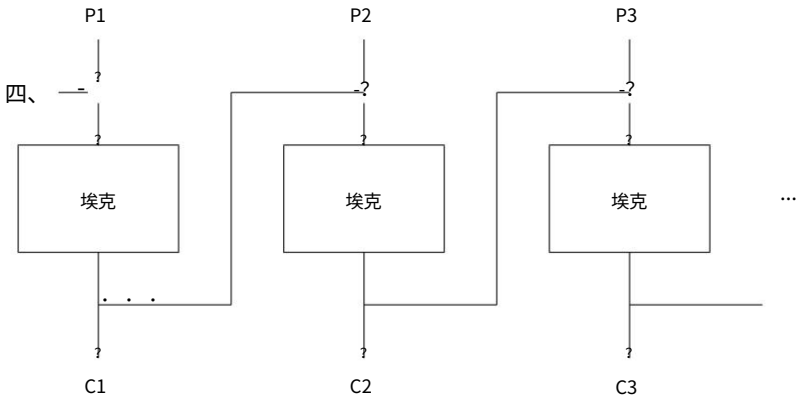


图 5.15：- 密码块链接 (CBC) 模式

5.5.3 反加密

块密码加密的反馈模式正在逐渐淡出,而不仅仅是因为密码问题。它们很难并行化。使用 CBC，必须在每个块输入和每个块输出之间计算整个密码块。这在高速应用中可能会带来不便,例如保护骨干链路上的流量。由于硅很便宜,我们宁愿流水线加密芯片,以便它在尽可能少的时钟滴答中加密一个新块（或生成一个新的密钥流块）。

最简单的解决方案是使用 AES 作为流密码。我们通过加密从初始化向量开始的计数器来生成密钥流： Ki = {IV + i}K,从而将密钥 K 扩展为密钥流的长块Ki流,通常与消息Mi us的块组合ing exclusive-or 给出密文Ci = Mi Ki。

正如我们在上面的 5.2.2 节中提到的,附加流密码有两个系统漏洞。第一种是深度攻击:如果使用相同的密钥流两次,则两个密文的异或就是两个明文的异或,通常可以从中推导出明文，就像 Venona 一样。第二个是它们无法保护消息的完整性。假设流密码用于加密资金转移消息。这些消息是高度结构化的;例如,您可能知道字节 37–42 包含正在传输的金额。

然后,您可以使来自本地银行的数据流通过您的计算机,例如通过 SS7 攻击。你走进银行,把 500 美元寄给同谋。密文Ci = Mi Ki准时到达您的机器。你知道字节 37-42 的Mi ,所以你可以恢复Ki并构建一个修改后的消息,指示接收银行支付 500,000 美元而不是 500 美元!这是深度攻击的一个例子;这不仅是我们从一次一密中获得的完美保密的代价,也是更不起眼的流密码的代价。

处理这种情况的通常方法是添加一个验证码,最常见的标准使用一种称为 Galois 计数器模式的技术,我将在后面介绍。

5.5.4 传统流密码模式

您可能会发现两种旧的流密码操作模式,即输出反馈模式 (OFB) 和不太常见的密文反馈模式 (CFB)。

输出反馈模式包括重复加密初始值并将其用作流密码中的密钥流。为初始化向量写 IV,我们将有 K1 = {IV }K和 Ki = {IV }K(i1)。然而,OFB 模式下的 n 位块密码通常具有2n/2个块的循环长度,之后生日定理将确保我们循环回到 IV。因此,如果我们在高速链路上使用 64 位块密码(例如三重 DES),我们可能会遇到周期长度问题:一旦我们调用了232个多一点的 64 位伪随机值,匹配的可能性就大了.(在 CBC 模式下,生日定理也确保在大约2n/2个块之后,我们将开始看到重复。)但是,计数器模式加密的保证周期长度为2n而不是2n/2,正如我们所指出的以上很容易并行化。尽管这种 OFB 仍在使用,因为计数器模式在 2002 年才成为 NIST 标准。

密码反馈模式是另一种流密码,设计用于必须抗干扰的无线电系统。它被设计成自同步的,因为即使我们遇到突发错误并丢失几位,系统也会在一个块长度后恢复同步。这是通过使用我们的块密码加密密文的最后 n 位,将最后一个输出位添加到下一个明文位,并将密文移动一位来实现的。

但这会花费每位一个块密码操作,并且具有非常糟糕的错误放大特性;现在人们倾向于使用专用的链路层协议来进行同步和纠错,而不是试图将它们与 trac 层的密码学相结合。

5.5.5 消息认证码

块密码的另一种社会操作模式不用于加密数据,而是用于保护其完整性和真实性。这是消息验证码,或 MAC。为了使用分组密码计算消息的 MAC,我们使用 CBC 模式对其进行加密,并丢弃除最后一个以外的所有输出密文块;最后一个块是 MAC。 (中间结果是保密的,以防止剪接攻击。)

这种结构使 MAC 依赖于所有明文块以及密钥。只要消息长度是固定的,它就是安全的; Mihir Bellare、Joe Kilian 和 Philip Rogaway 证明,在这种情况下对 MAC 的任何攻击都会对底层分组密码进行攻击 [211]。

如果消息长度是可变的,你必须确保在一个字符串上计算的 MAC 不能用作在不同字符串上计算 MAC 的 IV,这样对手就不能通过获取 MAC 来作弊两者的组成

字符串。为了解决这个问题,NIST 标准化了 CMAC,其中密钥的变体在最后一次加密之前被异或运算[1405]。 （CMAC 基于 Tetsu Iwata 和 Kaoru Kurosawa [965] 的提议。）您可能会看到遗留系统,其中 MAC 仅包含最后一个输出块的一半,另一半被丢弃或用于其他机制。

还有其他可能的 MAC 构造:最常见的一种是 HMAC,它使用带有密钥的散列函数;我们将在 5.6.2 节中对其进行描述。

5.5.6 伽罗瓦计数器模式

上述模式都是在 70 年代和 80 年代为 DES 开发的（尽管计数器模式直到 2002 年才成为美国政府的官方标准）。它们不适用于需要保护完整性和机密性的批量加密;如果您使用 CBC 模式或计数器模式来加密您的数据并使用 CBC-MAC 或 CMAC 来保护其完整性,那么您为您处理的每个数据块调用分组密码两次,并且该操作无法并行化。

现代方法是使用专为经过验证的加密而设计的操作模式。自 2007 年被 NIST 批准以来,Galois 计数器模式 (GCM) 已成为默认模式 [1407]。它对每个文本块仅使用一次块密码调用,并且它是可并行化的,因此您可以在快速数据链路上以低成本和低延迟获得高吞吐量。加密以计数器模式的变体执行;生成的密文也用作多项式的系数,该多项式在2128个元素的伽罗华域上的关键依赖点处进行评估,以提供验证器标签。标签电脑

tation 是我在第 5.2.4 节中描述的那种通用哈希函数,只要密钥永远不会被重复使用,它就可以证明是安全的。提供的密钥与随机 IV 一起使用以生成唯一的消息密钥和唯一的身份验证器密钥。因此,输出是与明文长度相同的密文,加上一个 IV 和一个通常各为 128 位的标签。

GCM 也有一个有趣的增量属性:可以计算一个新的验证器和密文,其工作量与更改的位数成正比。GCM 是由 Cisco 的 David McGrew 和 John Viega 发明的;他们的目标是创建一种适用于高性能网络硬件的有效认证加密模式 [1268]。 GCM 是经过身份验证的批量内容加密的合理默认设置。 （有一个更早的复合模式,CCM,你会发现它在蓝牙 4.0 及更高版本中使用;它结合了计数器模式和 CBC-MAC,因此它的计算成本大约是它的两倍,并且不能并行化或增量地重新计算[1406]。）

5.5.7 XTS

GCM 和其他经过身份验证的加密模式通过添加消息密钥和身份验证器标记来扩展明文。这在硬盘加密等应用程序中非常不方便,我们更喜欢保留明文长度的操作模式。磁盘加密系统过去使用 CBC,扇区号提供 IV,但自 Windows 10 以来,Microsoft 一直在使用新的

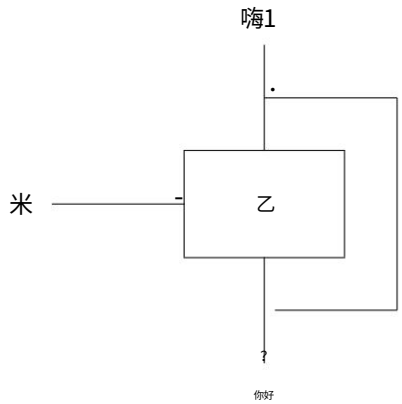


图 5.16：- 前馈模式（散列函数）

操作模式,XTS-AES,受 GCM 启发并于 2007 年标准化。这是一种密码本模式,但明文由从磁盘扇区派生的调整密钥增白。形式上,在块 j 处用密钥 K 加密的消息Mi是

$$AESX(KTj , K, KTj ; M)$$

其中调整密钥 KTj是通过使用不同的密钥加密 IV,然后将其与合适的常数重复相乘,从而为每个块提供不同的增白器而得出的。这意味着如果攻击者交换两个加密块,所有 256 位将解密为随机错误值。您仍然需要更高层的机制来检测密文操纵,但简单的校验和就足够了。

5.6 哈希函数

在 5.4.3.1 节中,我展示了 Luby-Racko 定理如何使我们能够从散列函数构造分组密码。也可以从块密码构造散列函数5。诀窍是一次将一个消息块提供给我们的块密码的密钥输入,并使用它来更新哈希值（从 o 开始,比如H0 = 0）。为了使这个操作不可逆,我们添加前馈:第 (i 1) 个哈希值与第 i 轮的输出异或。这个 Davies-Meyer 构造给出了分组密码的最终操作模式（图 5.16）。

5事实上,我们还可以从流密码构造散列函数和块密码 因此,根据我将在下一节中讨论的一些注意事项,给定这三个原语中的任何一个,我们可以构造另外两个。

生日定理在这里再次出现,因为如果哈希函数 h 是使用 n 位块密码构建的,则有可能找到两条消息M1 6= M2 ,其中 h(M1) = h(M2)大约为 2n/2 e ort（散列比Mi多的消息并寻找匹配项）。所以 64 位块密码是不够的,因为伪造一条消息将花费232条消息的数量,这太容易了。像 AES 这样的 128 位密码过去就足够了,事实上,蓝光 DVD 中的 AACS 内容保护机制使用“AES-H”,即以这种方式从 AES 派生的哈希函数。

5.6.1 常用哈希函数

1990 年代和 2000 年代最常用的散列函数演变为具有 512 位密钥和块大小从 128 位增加到 512 位的块密码的变体。前两个由 Ron Rivest 设计,其他两个由 NSA 设计：

- MD4 有三轮和 128 位哈希值,发现有冲突
  1998 年 [568]；
- MD5 有四轮和 128 位哈希值,发现冲突
  2004 年[1979 年,1981 年]；
- SHA-1,于 1995 年发布,有五轮和 160 位散列值。 2017 年发现了碰撞 [1828],2020 年发现了更强大的攻击版本 [1146]；
- SHA-2 于 2002 年取代它,有 256 位和 512 位版本
  （称为 SHA256 和 SHA512)加上许多变体。

这些散列函数背后的块密码是相似的:它们的轮函数是 32 位处理器上可用的寄存器操作的复杂混合 [1667]。密码分析稳步推进。 MD4 于 1998 年被 Hans Dobbertin 破解[568]； MD5 于 2004 年被 Xiaoyun Wang 及其同事破解 [1979, 1981];现在可以很容易地发现冲突,即使是在包含有意义的文本并遵循消息格式（例如用于数字证书的格式)的字符串之间也是如此。 Wang 在次年严重削弱了 SHA-1,提供了一种仅需269步即可发现碰撞的算法 [1980];现在需要大约260 次计算。 2017 年 2 月,来自阿姆斯特丹和谷歌的科学家发表了一篇文章,以证明这一点并帮助说服人们转向更强大的哈希函数,例如 SHA-2 [1828]（以及从早期版本的 TLS 到 TLS 1.3）。 2020 年,Ga¨etan Leurent 和 Thomas Peyrin 开发了一种改进的攻击,可以计算选择的前缀冲突,以数万美元的成本实现证书伪造 [1146]。

2007 年,美国国家标准与技术研究院 (NIST) 组织了一场寻找替代哈希函数族的竞赛 [1409]。获胜者 Keccak 具有完全不同的内部结构,并在 2015 年被标准化为 SHA-3。因此我们现在可以选择 SHA-2 和 SHA-3 作为标准哈希函数。

许多已部署的系统仍然使用散列函数,例如 MD5,其中有一个简单的碰撞搜索算法。碰撞是否会破坏任何给定的应用程序可能是一个复杂的问题。我已经提到了取证系统,它在没收的计算机上保存文件的哈希值,以向法庭保证警方没有篡改证据;哈希冲突只会表明有人试图篡改,无论是警察还是被告,并引发更仔细的调查。如果银行系统真的接受了一条由客户写成“支付 X 金额 Y”的消息,对其进行哈希处理并签名,那么骗子就可以找到两条消息“支付 X 金额 Y”和“支付 X 金额 Z”散列为相同的值,得到一个签名,然后将其交换为另一个。

但是银行系统不是这样运作的。他们通常在实际交易中使用 MAC 而不是数字签名,并且交易的所有各方都保存日志,因此不容易潜入冲突对中的一个。在这两种情况下,您可能都必须找到现有哈希值的原像,这比查找冲突更难进行密码分析任务。

5.6.2 哈希函数应用 HMAC、承诺和更新

但是,即使今天可能很少有应用程序可以通过碰撞发现算法让坏人窃取真钱,但漏洞能力的存在仍然会破坏系统的价值。一些从事取证工作的人继续使用 MD5,因为他们已经使用了多年,而且它的冲突不会提供有用的攻击。这可能是一个错误。 2005 年,一名被指控在澳大利亚悉尼超速行驶的驾车者在新南威尔士州道路和交通管理局未能找到专家证明 MD5 在该应用程序中是安全的之后被宣告无罪。法官“在毫无疑问的情况下不满意照片自拍摄以来 [没有] 被更改过”,并宣告司机无罪;第二年 [1432],他的奇怪裁决在上诉中得到维持。因此,即使漏洞不构成工程威胁,它仍然可以构成认证威胁。

哈希函数还有许多其他用途。其中之一是计算 MAC。一个朴素的方法是用一个密钥对消息进行哈希处理： MACk(M) = h(k,M)。
然而,公认的执行此操作的方法称为 HMAC,它使用一个额外的步骤,在该步骤中再次对计算结果进行哈希处理。这两个散列操作是使用密钥的变体完成的,通过用两个不同的常量对它们进行异或运算得出。因此 HMACk(M) = h(k B, h(k A, M))。 A 是根据需要多次重复字节 0x36 构造的,B 也类似地从字节 0x5C 构造。如果哈希函数处于弱侧,则这种构造会使可利用的冲突更难被发现 [1089]。 HMAC 现在是 FIPS 198-1。

散列函数的另一个用途是做出稍后会显示的承诺。例如,我可能希望为数字文档加上时间戳以建立知识优先级,但还不透露内容。在这种情况下,我可以发布文档的哈希值,或者将其发送到商业时间戳服务,或者将其挖掘到比特币区块链中。后来,当我展示文件时,其散列上的时间戳确定我当时已经写好了。

同样,生成碰撞对的算法不会破坏这一点,因为您在执行时间戳时必须手头有一对。

Merkle 树将大量输入散列为单个散列输出。输入被散列为形成树叶的值;每个非叶节点都包含其子节点上所有哈希值的哈希值,因此根节点上的哈希值是叶子上所有值的哈希值。这是散列大型数据结构的快速方法;它用于代码签名,在这种情况下,您可能不想等待应用程序的所有文件在打开之前都经过签名检查。

它还广泛用于区块链应用程序;事实上,区块链就是一棵 Merkle 树。它是由 Ralph Merkle 发明的,他首先提出用它来计算大型公钥文件的短散列 [1296],特别是对于公钥只使用一次的系统。例如,Lamport 数字签名可以从哈希函数构建:您创建一个包含 512 个随机 256 位值ki的私钥,并将验证密钥 V 作为其 Merkle 树哈希发布。

然后签署 h = SHA256(M)如果 h 的第 i 位为零,您将揭示k2i ,否则揭示k2i+1。如果散列函数是安全的,这是安全的,但缺点是每个密钥只能使用一次。 Merkle 发现您可以通过使用主密钥加密计数器来生成一系列私钥,然后使用树对生成的公钥进行哈希处理。然而,对于大多数目的,人们使用基于数论的签名算法,我将在下一节中对此进行描述。

散列函数的一个安全协议用途值得一提:密钥更新和自动密钥。密钥更新是指两个或多个共享密钥的委托人在约定的时间通过单向哈希函数传递密钥： Ki = h(Ki1)。

关键是,如果攻击者破坏了他们的其中一个系统并窃取了密钥,他只能获得当前密钥而无法解密回溯。散列函数的单向性打破了妥协链。此属性也称为向后安全性。一种变体是自动键控,其中委托人通过将密钥与自上次密钥更改以来交换的消息进行散列来更新密钥： Ki+1 = h(Ki, Mi1, Mi2,...)。如果攻击者现在破坏了他们的一个系统并窃取了密钥,那么一旦他们交换了他无法观察或猜测的消息,安全性就会恢复;妥协链再次被打破。此属性称为前向安全性。它首先在澳大利亚的 EFT 支付终端中用于银行业务 [207, 209]。非对称密码学的使用允许一种稍微更强的前向安全形式,即一旦受损终端与对手无法控制的未受损终端交换消息,即使消息是显而易见的,安全性也可以恢复.接下来我将描述它是如何工作的。

5.7 非对称密码原语

非对称密码学、公钥加密和数字签名中常用的构建模块均基于数论。我将在此处进行简要概述,并在讨论应用程序时在第 II 部分中更详细地介绍一些机制。

基本思想是使密码的安全性取决于解决已知困难的数学问题的难度,从某种意义上说,很多人都试图解决它但都失败了。两个问题用的差不多

所有真实系统都是因式分解和离散对数。

5.7.1 基于分解的密码学

质数是没有适当约数的正整数:能整除质数的只有 1 和这个数本身。

根据定义,1 不是质数;所以质数是 {2, 3, 5, 7, 11, ...}。算术基本定理指出,每个大于 1 的自然数都以在因子阶数内唯一的方式分解为素数。找到质数并将它们相乘得到一个合数很容易,但将一个合数分解成它的因数就难多了。自从我们开始使用基于因式分解的密码学以来,许多聪明人都非常努力地尝试过。 2020 年分解的两个大随机素数的最大复合乘积是 RSA-250,它是一个 829 位数字（250 位十进制数字）。这相当于在单个 2.2GHz 内核上花费了 2700 年的时间;之前的记录是 2019 年的 RSA-240,耗时相当于 900 年 [301]。有可能秘密地进行因式分解,也许使用僵尸网络； 2001 年,当最先进的技术是分解 512 位数字时,Simon Singh 的“Code Book”中提出了这样的挑战,五名瑞典学生使用他们可以访问的数百台计算机 [43] 解决了这个问题。至于 1024 位数字,我希望 NSA 已经可以对它们进行因式分解,我在第二版中指出“对因式分解记录历史的推断表明第一个因式分解将在 2018 年发布。”摩尔定律正在放缓,我们迟到了一年。无论如何，希望密钥在多年内保持安全的组织至少已经在使用 2048 位数字。

通常用于进行公钥加密和基于因式分解的数字签名的算法是 RSA,以其发明者 Ron Rivest、Adi Shamir 和 Len Adleman 的名字命名。它使用费马小定理,该定理指出对于所有不除以 a 的素数 p， ap1 ⌘ 1 (mod p)（证明:取集合 {1, 2, ..., p 1} 并将它们中的每一个乘以模 p a, 然后取消 (p 1)! 每边）。对于一般整数 n， a(n) ⌘ 1 (mod p) 其中欧拉函数 (n) 是小于 n 且没有共同约数的正整数的数量（证明类似）。因此,如果 n 是两个素数 pq 的乘积,则 (n)=(p 1)(q 1)。

在 RSA 中,加密密钥是一个难以因式分解的模数 N（对于两个随机选择的大素数 p 和 q,取 N = pq,比如每个素数 1024 位)加上一个与 p 1 没有公因数的公共指数 e或 1。

私钥是因子 p 和 q,它们是保密的。其中 M 是消息,C 是密文,加密定义为

$$C \equiv M^{e} \pmod{N}$$

解密是逆向操作：

$$M \equiv C^{d} \pmod{N}$$

谁知道私钥__N 的因子 p 和 q 就可以轻松计算出pe C (mod N)。由于 (N)=(p 1)(q 1) 和 e 没有共同点

因素与 (N),钥匙的所有者可以找到一个数字 d 使 de ⌘ 1 (mod (N)) – 她分别找到 d 的值对 p 1 和 q 1 求模,然后组合答案。 pe C (mod N) 现在计算为Cd (mod N),并且由于费马定理,解密有效:

$$Cd \cmd \{Me\}d \cmd Med \cmd M1+k(N) \cmd M.Mk(N) \cmd M.1 \cmd M \pmod N$$

同样,私钥的所有者可以用它对消息进行操作以产生签名

$$Sigd(M) \cmd Md \pmod N$$

并且可以通过将其提高到 e mod N 的幂来验证此签名（因此,使用 e 和 N 作为公共签名验证密钥)并检查消息 M 是否已恢复:

$$M \cmd (Sigd(M))e \pmod N$$

单独使用 RSA 加密和签名都不安全。原因是,由于加密是一个代数过程,它保留了某些代数性质。例如,如果明文之间存在 M1M2 = M3 等关系,那么密文C1C2 = C3和签名Sig1Sig2 = Sig3之间也将存在相同的关系。此属性称为乘法同态;同态是保留某些数学结构的函数。原始 RSA 的同态性质意味着它不符合公钥加密或签名的随机 oracle 模型定义。

公钥加密的另一个普遍问题是,如果明文是从一个小集合中提取的,例如“攻击”或“撤退”,并且加密过程是确定性的（如 RSA）,那么攻击者可能只是预先计算可能的密文并在它们出现时识别它们。对于 RSA,使用小指数 e 将同一消息加密给多个收件人也是危险的,因为这可能导致代数攻击。为了阻止猜测攻击、低指数攻击和基于同态的攻击,明智的做法是在加密前向明文块中添加一些随机性和一些冗余。每次我们加密同一条短消息,比如“攻击”,我们都希望得到一个完全不同的密文,并且这些密文彼此之间以及与“撤退”的密文无法区分。

这样做有好的方法也有坏的方法。

几十年来,密码理论家们一直在努力分析非对称密码术可能出现的所有问题,并找到解决问题的方法。

Shafi Goldwasser 和 Silvio Micali 提出了正式的概率加密模型,我们在其中为加密过程和语义安全添加了随机性,我们已经提到过;在这种情况下,这意味着攻击者根本无法获得有关加密为密文 C 的明文 M 的任何信息,即使允许他请求解密不等于 C的任何其他密文C0 [777]。换句话说,我们希望加密既能抵抗选择密文攻击,又能抵抗选择明文攻击。有一个

提供语义安全的结构的数量,但它们往往过于笨拙而无法实际使用。

通常的现实世界解决方案是最佳非对称加密填充
(OAEP),我们将消息 M 与随机数 N 连接起来,并使用哈希函数 h 将它们组合起来:

$$C1 = M \cdot h(N)$$

$$C2 = N \cdot h(C1)$$

实际上,这是一个使用 h 作为其轮函数的两轮 Feistel 密码。
结果,即组合C1、 C2,然后使用 RSA 加密并发送。然后接收者将 N 计算为C2 h(C1)并将 M 恢复为C1 h(N)
[212]。
这最终被证明是安全的。有许多公钥加密标准； PKCS #1 描述了 OAEP [993]。这些阻止了 20 世纪发现的大量攻击,但人们大多忘记了这些攻击,例如如果您使用两个不同的 RSA 密钥加密同一消息,对手可以检测到这一事实。事实上,我们在 1990 年代学到的一件事是,随机化有助于使加密协议更稳健地抵御各种攻击,而不仅仅是数学攻击。

侧信道攻击甚至设备的物理探测需要更多的工作。

使用签名,事情会稍微简单一些。通常,在应用私钥之前对消息进行散列通常就足够了： Sigd = [h(M)]d
(mod N); PKCS #7 描述了用于签署消息摘要的简单机制 [1008]。

然而,在某些应用程序中,人们可能希望在签名块中包含更多数据,例如时间戳或一些随机性以使侧信道攻击更加困难。

实际实现中出现的许多问题都与侧通道和错误处理有关。 One spectacular example was when Daniel Bleichenbacher found a way to break the RSA implementation in SSL v 3.0 by sending suitably chosen ciphertexts to the victim and observing any resulting error messages.如果他可以从目标中了解到给定的 c 在解密为cd (mod n) 时是否对应于 PKCS #1 消息,那么他可以使用它来解密或签署消息 [264]。对常见公钥实现的边信道攻击还有很多,通常是通过测量解密所用的精确时间。 RSA 在数学上也很脆弱;你可以使用同态来破解它,或者如果你有相同的密文在太多不同的小密钥下加密,或者如果消息太短,或者如果两条消息通过已知的多项式相关,或者在其他几个边缘情况下。计算中的错误也可能给出正确的模数模数的一个因子和错误的模数另一个的结果,从而使模数能够被因式分解;错误可以通过干扰加密设备在战术上插入,或者在战略上插入,例如芯片制造商安排 64 位乘法的一个特定值被错误计算。然而,其他攻击涉及堆栈溢出,无论是通过将攻击代码作为密钥发送,还是作为实施不当的标准中的填充。

5.7.2 基于离散对数的密码学

虽然 RSA 是第一个部署在 SSL 和 SSH 协议中的公钥加密算法,但现在最流行的公钥算法是基于离散对数的。有多种风格,一些使用普通的模运算,而另一些使用椭圆曲线。我将首先解释正常情况。

原根模 p 是一个数,其幂生成所有非零数模 p;例如,当对 7 求模时,我们发现52 = 25 减少为 4（模 7）,那么我们可以将53计算为52 5 或 4 5 即 20,减少为 6（模 7）,等等如图 5.17 所示： 51 = 5 (mod 7) 52 = 25 ⌘ 4 (mod 7) 53 ⌘ 4x5 ⌘ 6 (mod 7) 54 ⌘ 6x5 ⌘ 2 (mod 7) 55 ⌘ 2x5 ⌘ 3 (mod 7) 56 ⌘ 3x5 ⌘ 1（模 7）

图 5.17 – 离散对数计算示例

因此 5 是一个原根模 7。这意味着给定任何 y,我们总是可以求解方程 y = 5x (mod 7)； x 然后称为 y 模 7 的离散对数。像这样的小例子可以通过检查解决,但是对于一个大的随机素数 p,我们不知道如何有效地做到这一点。所以映射 f : x ! gx (mod p) 是单向函数,具有 f(x + y) = f(x)f(y) 和 f(nx) = f(x)n 的附加属性。换句话说,它是一种单向同态。因此,它可用于构建数字签名和公钥加密算法。

5.7.2.1 单向交换加密

想象一下我们回到了古罗马,安东尼想要向布鲁图斯发送一个秘密,而唯一可用的通信渠道是一个不可信的信使（比如,一个属于凯撒的奴隶）。 Anthony 可以把信息放在一个盒子里，上锁,然后让快递员把它带给 Brutus。布鲁图斯也可以在上面挂上他自己的挂锁,并将它带回给安东尼。他反过来会取下他的挂锁,并将它带回给布鲁图斯,布鲁图斯现在终于打开了它。

使用合适的通勤加密函数可以完成完全相同的操作,即具有{{M}KA}KB = {{M}KB} KA 的属性。 Alice 可以获取消息 M 并使用她的密钥 KA 对其进行加密以获得 {M}KA，并将其发送给 Bob。 Bob 使用他的密钥 KB 再次加密它,得到{{M}KA}KB。

但是交换性意味着这只是{{M}KB}KA,所以 Alice 可以使用她的密钥 KA 得到{M}KB 来解密它。她把这个发给 Bob,他可以用 KB 解密,最后恢复消息 M。

如何实现合适的交换加密?一次性纸确实通勤,但不适合这里。假设 Alice 选择一个随机密钥 xA 并向 Bob 发送 M xA,而 Bob 返回 M xB,Alice 最后向他发送 M xA xB,那么攻击者可以简单地排他或这些

三个消息在一起;由于对于所有 X XX = 0,xA 和 xB 的两个值都抵消,留下明文 M。

离散对数问题就派上用场了。如果基于原根模 p 的离散对数问题很难,那么我们可以使用离散指数作为我们的加密函数。例如,Alice 将她的消息编码为原根 g,选择一个随机数 xA,计算gxA模 p 并将其与 p 一起发送给 Bob。 Bob 同样选择一个随机数 xB 并形成gxAxB modulo p,他将其传回给 Alice。 Alice 现在可以删除她的求幂:使用费马定理,她计算gxB = (gxAxB)(pxA) (mod p) 并将其发送给 Bob。 Bob 现在也可以去掉他的求幂,所以最终得到了 g。该方案的安全性取决于离散对数问题的难度。实际上,将消息编码为原始根可能很棘手;但是有一种更简单的方法可以达到同样的效果 ect。

5.7.2.2 Die-Hellman 密钥建立

Whitfield Die 和 Martin Hellman 于 1976 年发布的第一个公钥加密方案具有固定的原根 g 并使用gxAxB modulo p 作为共享密钥加密系统的密钥。值 xA 和 xB 可以是双方的私钥。

让我们来看看这个。素数 p 和生成元 g 对所有用户都是公共的。爱丽丝选择一个秘密的随机数 xA,计算出 yA = gxA并将其公布在公司电话簿中与她名字相对的位置。 Bob 也这样做,选择一个随机数xB并发布 yB = gxB。为了与 Bob 通信,Alice 从电话簿中获取 yB,形成yBxA ,也就是gxAxB,并使用它来加密发送给 Bob 的消息。收到它后,Bob 查找 Alice 的公钥yA并形成yAxB ,它也等于gxAxB,因此他可以解密她的消息。

或者,Alice 和 Bob 可以使用临时密钥,并获得提供前向安全性的机制。和以前一样,让素数 p 和生成器 g 为所有用户所共有。 Alice选择一个随机数RA,计算gRA并发送给Bob； Bob 也这样做,选择一个随机数 RB并将gRB发送给 Alice;然后它们都形成gRARB ,它们将其用作会话密钥（参见图 5.19）。

一个！ B : gRA (mod p)
乙！ A : gRB (mod p)
一个！ B： {M}gRARB

图 5.18 – Die-Hellman 密钥交换协议

Alice 和 Bob 现在可以使用会话密钥gRARB来加密对话。
如果他们使用临时密钥而不是长期密钥,他们就成功地“无中生有”地创建了一个共享秘密。即使对手在该协议启动之前检查了他们的两台机器,并且知道他们所有存储的私钥,然后满足一些基本条件（例如,他们的随机数生成器不可预测并且没有留下恶意软件）

对手仍然无法窃听他们的踪迹。这是我在 5.6.2 节中提到的前向安全属性的增强版本。对手无法根据他可能获得的先前密钥的知识继续前进。如果 Alice 和 Bob 在使用后都销毁了共享的秘密,那么他们也将拥有后向安全性:随后获得他们设备访问权的对手不能向后工作以打破他们的旧轨道。

在下文中,当我们不必明确说明我们在哪个组中工作并且不需要明确写出时,我们可以将从RA和RB派生的 Die-Hellman 密钥写为 DH(RA, RB)哪个是私钥RA哪个是公钥gRA 。

要提供完整的解决方案,还需要做更多的工作。选择参数 p 和 g 时需要小心;例如,我们可以从斯诺登的披露中推断,NSA 可以解决常用的 1024 位素数的离散对数问题6。还有其他一些细节取决于我们是否需要前向安全性等属性。

但是这个协议有一个小问题:虽然 Alice 和 Bob 最终得到
会话密钥,他们都不知道与谁共享。

假设在我们的挂锁协议中,Caesar 刚刚命令他的奴隶将盒子带给他,并将他自己的挂锁放在 Anthony 的旁边。奴隶把盒子带回给安东尼,安东尼取下了他的挂锁,然后把盒子带回给凯撒,凯撒打开了它。 Caesar 甚至可以在协议中运行两种姿态,向 Anthony 假装他是 Brutus,向 Brutus 假装他是 Anthony。一种解决方法是安东尼和布鲁图斯将他们的印章贴在他们的锁上。

使用 Die-Hellman 协议,同样的想法会导致中间人攻击。查理拦截爱丽丝发给鲍勃的消息并回复;同时,他冒充 Alice 发起与 Bob 的密钥交换。他最终得到了一个与爱丽丝共享的密钥 DH(RA, RC) 和另一个与鲍勃共享的密钥 DH(RB, RC)。只要他继续坐在网络中间并在他们之间翻译消息,他们可能很难检测到他们的通信受到威胁。通常的解决方案是验证临时密钥,并且有多种可能性。

在现在已经过时但您可以在米德堡的 NSA 博物馆中看到的 STU-2 电话中,两位负责人会读出他们生成的密钥的八位哈希值,并检查它们之前是否具有相同的值开始讨论机密问题。蓝牙版本 4 及更高版本中实现了类似的东西,但由于协议已经发展到支持具有不同用户界面的设备的许多版本而变得复杂。该协议遭受了多次攻击,最近一次是蓝牙（KNOB)攻击的密钥协商,它允许中间人强制使用很容易被暴力破解的单字节密钥； 2018 年之前生产的所有设备都存在漏洞 [123]。该标准允许密钥长度在 1 到 16 个字节之间;

6 可能的离散对数算法 NFS 涉及对每个质数进行大量计算,然后对每个离散对数以该质数为模进行较小的计算。开放记录为 795 位,在 2019 年[?] 花费了 3,100 个核心年,使用的 NFS 版本比十年前效率高出三倍。一直有关于 NSA 进一步改进的谣言,无论如何,该机构可以在重要的计算中投入更多的力量。

由于密钥长度协商是明文执行的,因此攻击者可以将长度强制为下限。所有符合标准的芯片都容易受到攻击;这可能是我在第 26.2.7 节中讨论的加密战争中更多的有毒废物。蓝牙的早期版本更像是 14.3.3.3 节中描述的 HomePlug 协议的“正常工作”模式,因为它们主要旨在帮助您在良性环境中与正确的设备设置配对密钥,而不是防御敌对攻击中的复杂攻击。更现代的似乎更好,但它实际上只是剧院。

很多事情都出错了:协议将生成或接受非常弱的密钥,因此仅提供保护的外观;通过侧通道泄露密钥的程序,例如解密所需的时间长度;以及导致堆栈溢出和其他黑客攻击的软件漏洞。如果你正在实施公钥加密,你需要参考最新的标准,使用经过适当认可的工具包,并让知识渊博的人来评估你所做的事情。并且请不要自己编写实际的加密代码 正确地编写它需要很多不同的技能,从计算数论到边信道分析和形式化方法。即使使用好的加密库也会给你很多机会让你大吃一惊 。

5.7.2.3 ElGamal 数字签名和 DSA

假设基数 p 和生成器 g 是以某种合适的方式选择的公共值,并且每个希望签署消息的用户都有一个私人签名密钥 X 和一个公共签名验证密钥 Y = gX。 ElGamal 签名方案的工作原理如下。随机选择一个消息密钥k,形成r = gk (mod p)。现在使用 k、r、消息 M 和私钥 X 中的线性方程式来形成签名 s。有许多方程式都可以;恰好在 ElGamal 签名中使用的那个是

$$rX + sk = M$$

所以 s 的计算公式为 s = (M rX)/k;这是以模 (p) 完成的。当两边都通过我们的单向同态 f(x) = gx mod p 时,我们得到:

$$grXgsk \equiv gM$$

或者

$$Y\ rrs \equiv gM$$

消息 M 上的 ElGamal 签名由值 r 和 s 组成,并且
收件人可以使用上面的等式来验证它。

需要修复更多细节以获得功能性数字签名方案。和以前一样,p 和 g 的错误选择会削弱算法。我们还想使用哈希函数对消息 M 进行哈希处理,以便我们可以对任意长度的消息进行签名,这样对手就无法使用算法的代数结构在从未签名的消息上伪造签名。关注这些细节并应用一两个优化后,我们得到

数字签名算法 (DSA) 是美国标准,广泛用于政府应用程序。

DSA 假设素数 p 通常为 2048 位 7,素数 q 为 256 位除法 (p 1),整数模 p 中的 q 阶元素 g,秘密签名密钥 x 和公共验证密钥 y = gx。消息 M 的签名Sigx(M) 是 (r, s) 其中

$$r \equiv (g^k \pmod p) \pmod q$$

$$s \equiv (h(M) + xr)/k \pmod q$$

默认使用的哈希函数是 SHA2568。

DSA 是没有消息恢复的随机数字签名方案的经典例子。现在最常用的版本是 ECDSA,它是一种基于椭圆曲线的变体,我们现在将讨论它 例如,这是加密货币的标准,并且越来越多地用于银行智能卡中的证书。

5.7.3 椭圆曲线密码学

离散对数及其类似物存在于许多其他数学结构中。椭圆曲线密码术在椭圆曲线上使用离散对数 一条由方程y2 = x3 + ax + b 给出的曲线。这些曲线具有您可以在其上定义加法运算的属性,生成的莫德尔群可用于密码学。代数变得有点复杂,这本书不是列出它的地方 9。然而,椭圆曲线密码系统之所以有趣,至少有两个原因。

首先是性能;他们提供了熟悉的原语版本,例如 Die-Hellmann 密钥交换和数字签名算法,这些算法使用更少的计算,也有更短的变量;两者在受限环境中都很受欢迎。椭圆曲线密码学用于从最新版本的 EMV 支付卡到比特币的应用程序中。

其次,一些椭圆曲线有一个双线性对,Dan Boneh 和 Matt Franklin 用它来构建密码系统,其中你的公钥就是你的名字 [286]。回想一下,在 RSA 和 Die-Hellmann 中,用户选择他的私钥,然后计算出相应的公钥。在所谓的基于身份的密码系统中,您选择自己的身份,然后前往中央机构,该机构会向您颁发与该身份相对应的私钥。有一个全球公众

7 在 1990 年代,p 可能在 512–1024 位范围内,q 可能在 160 位范围内;这在 2001 年 [1402] 中更改为 1023-1024 位,在 2009 年更改为 1024-3072 位,其中 q 在 160-256 位范围内 [1403]。

8p 的默认大小选择为 2048 位和 q 为 256 位,以均衡两种最著名的密码分析攻击的工作因数,即运行速度取决于 p 大小的数域筛选和取决于 pollard 的 rho q的大小。如果您对摩尔定律或算法的进步感到焦虑,可以选择更大的尺寸。

9 有关介绍,请参阅 Katz 和 Lindell [1022]。

密钥,任何人都可以使用它来加密消息到您的身份;您可以使用您的私钥对其进行解密。早些时候,Adi Shamir 发现了基于身份的签名方案,允许您使用私钥对消息进行签名,这样任何人都可以根据您的名字验证签名 [1704]。在这两种情况下,您的私钥都是由中央机构使用只有自己知道的系统范围内的私钥计算出来的。基于身份的原语已在一些专业系统中使用:Zcash 中用于支付隐私机制,以及英国政府密钥管理协议 Mikey-Sakke。从人们的电子邮件地址或其他标识符计算他们的私钥似乎是一种巧妙的技巧,但当政府部门重组或更名时,这可能会很昂贵 [115]。大多数组织和应用程序使用带有公钥认证的普通公钥系统,我将在下面讨论。

5.7.4 认证机构

现在我们可以进行公钥加密和数字签名,我们需要某种机制将用户绑定到密钥。 Die 和 Hellman 在发明数字签名时提出的方法是拥有一个系统授权用户的公钥目录,就像电话簿一样。由 Loren Kohnfelder 提出的一种更常见的解决方案是证书颁发机构 (CA) 签署用户的公共加密密钥或他们的签名验证密钥,提供包含用户名、一个或多个公共密钥以及诸如此类的属性的证书作为授权。 CA 可能由本地系统管理员运行;但最常见的是第三方服务,例如 Verisign,其业务是在对公共密钥是否由其中指定的委托人进行尽职调查后签署公共密钥。

证书可以象征性地描述为

$$CA = SigKS\ （TS、\ L、A、\ KA、\ VA）\tag{5.1}$$

其中TS是证书的起始日期和时间,L 是它的有效时间长度,A 是用户名， KA是她的公共加密密钥， VA 是她的公共签名验证密钥。这样,只需要将管理员的公开签名验证密钥以可信任的方式传递给所有委托人。

认证很难,原因有很多。对于初学者来说,命名很难;我们将在下一章分布式系统中讨论这个问题。但通常名称并不是协议必须建立的真正内容,因为在现实世界中它通常是关于授权而不是身份验证。政府系统通常不仅要建立用户的姓名或角色,还要建立他们的安全许可级别。在银行系统中,它与您的余额、您的可用信用和您的消费权限有关。在商业系统中,通常是将远程用户链接到基于角色的访问控制。在面向用户的系统中,倾向于将尽可能多的合规成本转嫁给客户 [524]。

系统工程级别的认证还有许多其他问题。在政治层面,一个典型的浏览器中有数百个证书颁发机构,它们或多或少都受到同等信任,并且

许多民族国家至少可以胁迫其中一个 10。坏证书的撤销通常是不稳定的,如果它有效的话。稍后将有更多关于这些主题的内容。有了这些警告,是时候看看最常用的公钥协议 TLS 了。

5.7.5 TLS

我在上面说过,服务器可以发布公钥 KS,然后任何 Web 浏览器都可以向其发送包含信用卡号的消息 M,并使用 KS 加密： {M}KS。这本质上就是 TLS 协议（当时称为 SSL)在电子商务开始时设计的目的。它由 Paul Kocher 和 Taher ElGamal 于 1995 年开发,支持双向加密和身份验证,因此可以保护 http 请求和响应免受窃听和操纵。这是当您在浏览器工具栏上看到挂锁时激活的协议。

以下是 TLS v1 协议基本版本的简化描述：

1. 客户端向服务器发送一个客户端问候消息,其中包含其名称 C、事务序列号 C# 和随机数NC ；

2. 服务器回复一条服务器问候消息,其中包含其名称 S、事务序列号 S#、随机数NS和包含其公钥 KS 的证书 CS。客户端现在检查证书 CS,如果需要,检查在另一个证书中签署它的密钥,依此类推回到由 Verisign 等公司颁发并存储在浏览器中的根证书；

3. 客户端发送包含预主密钥K0的密钥交换消息,该密钥在服务器公钥 KS 下加密。它还会发送一条已完成的消息,其中包含根据迄今为止的所有消息计算出的消息身份验证代码 (MAC)。此 MAC 的密钥是主密钥K1。该密钥是通过使用客户端和服务器发送的随机数对预主密钥进行哈希计算得出的： K1 = h(K0, NC , NS)。从这一点开始，所有的trac都被加密了;我们将在客户端-服务器方向将其写为 {...}KCS,从服务器到客户端将其写为 {...}KSC。这些密钥是通过使用K1对随机数据进行哈希处理而依次生成的。

4.服务器还发送一个完成的消息,其中包含在所有迄今为止的消息。然后它终于开始发送数据。

10少数做不到的人,试图作弊。 2011 年,伊朗入侵了 CA Diginotar,2019 年,哈萨克斯坦强迫其公民在其浏览器中添加当地警方证明。在这两种情况下,浏览器供应商都快速而艰难地进行了反击:Diginotar 在被列入黑名单后失败了,而哈萨克斯坦证书即使其公民手动安装也被阻止。这当然会引发主权问题。

C！ S:C,C#,数控S！ C：
S,S#， NS,CS C！ S： {K0}
KS C！ S : {f 完成,MAC(K1,
everythingtodate)}KCS S！ C : {f 完成, MAC(K1,
everythingtodate)}KSC , {data}KSC

一旦客户端和服务器建立了一个预主密钥,就不需要更多的公钥操作,因为可以通过使用新的随机数对其进行散列来获得更多的主密钥。

5.7.5.1 TLS 使用

完整的协议比这更复杂,并且经历了多个版本。它支持许多不同的密码套件,最初是为了将软件的出口版本限制为 40 位密钥 这是美国政府多年来强制实施的出口许可条件。

这导致了降级攻击,中间人可以强制使用弱密钥。其他密码套件支持签署 Die-Hellman 密钥交换临时密钥,以提供前向和后向保密。 TLS 还具有双向身份验证选项,因此如果客户端也有证书,则可以由服务器进行检查。此外,工作密钥 KCS 和 KSC 可以包含用于加密和身份验证的单独子密钥,这是 CBC 加 CBC MAC 等传统操作模式所需要的。

除了用于加密网络流量外,从 Windows 2000 开始,TLS 也可用作 Windows 中的身份验证选项;您可以使用它代替 Kerberos 在公司网络上进行身份验证。我将在网络攻防章节中更详细地描述它的使用。

5.7.5.2 TLS 安全

尽管 SSL 的早期版本有许多错误 [1973],但 SSL 3.0 及更高版本似乎是可靠的； SSL 3.0之后的版本更名为 TLS 1.0。它在 1998 年由 Larry Paulson 正式验证,因此我们知道该协议的理想化版本没有任何错误 [1502]。

然而,在这之后的二十多年里,却发生了十几起严重的袭击事件。甚至在 1998 年,Daniel Bleichenbacher 就基于测量服务器解密所花费的时间,或服务器为响应精心设计的协议响应而返回的错误消息,提出了众多攻击中的第一个 [264]。 TLS 1.1 出现于 2006 年,可以防止 CBC 加密和填充错误的利用;两年后TLS 1.2紧随其后,将哈希函数升级为SHA256,支持鉴权加密;与此同时,出现了许多针对各种攻击的补丁。由于很难更改广泛使用的协议,因此其中许多补丁都相当不雅观;很难同时更改服务器端和客户端,因为任何客户端仍然需要与数百万台服务器进行交互,其中许多运行着过时的软件,而且大多数网站都希望能够处理所有年龄段和各种设备上的浏览器.大型服务公司已经解决了这个问题,他们改变了浏览器以拒绝过时的

密码套件,并添加严格传输安全 (STS) 等功能,网站可以指示浏览器将来仅使用 https 与其交互（以防止降级攻击）。浏览器公司还强制执行了许多其他支持措施,从缩短证书生命周期到证书透明度,我们将在网络攻击和防御一章中讨论这些措施。

5.7.5.3 TLS 1.3

经过两年的讨论,IETF 于 2019 年 1 月批准了核心协议 TLS 1.3 的最新重大升级。它放弃了向后兼容性以结束对许多旧密码的支持,并强制在每个会话开始时通过 Die Hellman 密钥交换建立端到端的前向保密。这引起了银行业的争议,银行业经常拦截加密会话以进行合规性监控。这将不再可能,因此银行将不得不承担使用过时加密的法律不适或重新开发系统以监控端点合规性的财务成本 11。

5.7.6 其他公钥协议

许多其他公钥协议已得到广泛使用,包括以下协议,我们稍后将详细讨论其中的大部分协议。这里我简单提一下代码签名、PGP和QUIC。

5.7.6.1 代码签名

代码签名是在 1990 年代引入的,当时人们开始下载软件而不是将其放在软盘上。它现在被广泛用于确保软件的来源。您可能认为在您的软件中有一个公共签名验证密钥以便版本 N 可以验证对版本 N + 1 的更新将是公共密钥密码术的简单应用,但事实并非如此。许多平台签署他们的操作系统代码,包括更新,以防止持续存在的恶意软件;这些机制通常涉及可信硬件,例如 TPM,我将在下一章的 6.2.5 节中讨论它们。某些平台,例如 iPhone,将只运行签名代码;这不仅可以确保软件的来源,还可以让平台所有者将应用程序货币化,正如我将在第 22.3.2 节中讨论的那样;游戏机是相似的。

由于一些用户竭尽全力对他们的设备进行越狱,因此此类平台通常具有可信赖的硬件来存储验证密钥。在不可用的情况下,可以使用经过混淆的代码进行验证,使恶意软件（或客户)更难篡改它;这是一场持续不断的军备竞赛,我将在第 24.3.3 节中对此进行讨论。至于签名密钥,开发人员可以将其保存在硬件安全模块中,这很昂贵并且会以 20.5 节中讨论的微妙方式破坏;可能有一条信任链返回商业 CA,但随后不得不担心政府的法律强制

11COVID-19 大流行给了一些喘息的机会:微软原定于 2020 年春季取消对旧版 TLS 的支持,但现在推迟了。

机构,我在第 26.2.7 节中讨论;为了安心,您甚至可以实施自己的 CA。简而言之,代码签名并不像看起来那么容易,尤其是当用户是敌人时。

5.7.6.2 PGP/GPG

在 1990 年代的“加密战争”期间,网络活动家与政府争夺加密电子邮件的权利,而政府则推动制定限制加密的法律;我将在 26.2.7 节讨论历史和政治。加密活动家 Phil Zimmermann 编写了一个开源加密产品 Pretty Good Privacy (PGP),并通过在纸质书中发布源代码来规避美国的出口管制,该书可以发布、扫描和编译。随着后来的兼容产品如 GPG,它在极客中得到了相当广泛的使用。例如,系统管理员、计算机紧急响应小组 (CERT) 和恶意软件研究人员使用它来共享有关攻击和漏洞的信息。它还被内置到出售给犯罪团伙的定制手机中以支持消息传递;我将在 25.4.1 节稍后讨论这个。

PGP 具有许多功能,但在其最基本的形式中,每个用户手动生成私钥/公钥对并与联系人共享公钥。有命令行选项可以使用您的签名密钥对消息进行签名和/或使用每个预期收件人的公钥对其进行加密。手动密钥管理避免了对可能被破解或强制的 CA 的需要。从 1990 年代 PGP 的部署和使用中学到了很多东西。正如我在第 3.2.1 节中所述,Alma Whitten 和 Doug Tygar 通过评估有动机但不熟悉密码的用户是否能够很好地理解它以安全地驱动程序,从而撰写了关于安全可用性的开创性论文。十二名受试者中只有四名能够正确地将加密电子邮件发送给其他受试者,并且每个受试者都至少犯了一个重大错误。

5.7.6.3 快速

QUIC 是一种新的基于 UDP 的协议,由 Google 设计并作为 TLS 的替代品进行推广,它允许更快地建立会话,从而减少页面加载时发生的广告拍卖中的延迟;当人们在接入点之间移动时,会话可以持续存在。这是通过保存客户端最后一个 IP 地址的 cookie 实现的,该地址由服务器加密。它于 2013 年出现在 Chrome 中,目前拥有约 7% 的互联网流量;它获得了一个充满活力的标准化社区。谷歌声称它减少了 8% 的搜索延迟和 18% 的 YouTube 缓冲时间。独立评估表明,好处主要体现在桌面上而不是移动设备上 [1007],并且存在隐私问题,因为服务器可以为每个客户端使用单独的公钥,并将其用于跟踪。

作为一般原则,人们应该警惕公司试图用专有标准取代开放标准,无论是 IBM 1950 年代的 EBCDIC 编码标准和 1970 年代的 SNA,还是 Microsoft 试图“拥抱和扩展”邮件标准和安全协议自 1990 年代以来,或 Facebook 在非洲推广互联网接入,使用户主要留在其围墙花园内。我将在第 8 章详细讨论我们行业的垄断倾向。

5.7.7 专用原语

研究人员发明了大量具有特殊属性的公钥和签名原语。目前已经出现在实际产品中的两种是门限密码和盲签名。

阈值加密是一种机制,签名密钥或解密密钥可以在 n 个委托人之间拆分,以便 n 中的任何 k 都可以签署消息（或解密）。对于 k = n,构造很容易。例如,对于 RSA,您可以将私钥 d 拆分为 d = d1 + d2 + ... + dn。对于 k<n,它稍微复杂一些（但不多 您使用拉格朗日插值公式)[554]。门限签名首先用于多个服务器独立处理事务并对结果进行独立投票的系统;它们最近被用于实施加密货币钱包的业务规则,例如“支付必须由七名公司董事中的任何两名授权”。

盲签名是一种在不知道消息是什么的情况下对消息进行签名的方式。例如,如果我们使用 RSA,我可以取一个随机数 R,形成ReM (mod n),并将它提供给计算(ReM)d = R.Md (mod n) 的签名者。当他把这个还给我时,我可以除以 R 得到签名Md。现在你可能会问为什么有人要在不知道文件内容的情况下签署文件,但是有一些应用程序。

第一个是数字现金;您可能希望能够向客户发行匿名支付令牌,而最早的想法,由于 David Chaum,是一种在不知道其序列号的情况下签署“数字硬币”的方法 [411]。一家银行可能同意以 10 美元的价格承兑任何具有唯一序列号和指定冗余形式的字符串 M,并带有使用公钥 (e, n) 验证为正确的签名。盲签名协议确保客户可以在银行不知道其序列号的情况下让银行签署硬币,并且它被用于原型道路收费系统。效果是数字现金对于消费者来说可以是匿名的。数字现金的主要问题是检测两次使用同一枚硬币的人,正如我在第 20.7 节中讨论的那样,最终使用区块链或其他分类账机制解决了这个问题。

数字现金未能取得成功 ,因为银行和政府都不希望匿名支付:自 9/11 以来的反洗钱法规限制匿名支付服务只能提供小额金额,而银行和比特币矿工都喜欢收取交易费用。

匿名数字凭证现在用于证明:您 PC 主板上的 TPM 芯片可能会在不识别您身份的情况下证明您机器上运行的软件的某些信息。不幸的是,这导致了 SGX（及其 AMD 等价物)的认证设计,这意味着单个受损设备会破坏整个生态系统。匿名签名也存在于用于进行电子选举的原型系统中,我将在第 25.5 节中返回。

5.7.8 非对称密码primi有多强蒂夫?

为了提供与对称块密码相同级别的保护,非对称加密原语通常需要至少两倍的块长度。椭圆曲线系统似乎达到了这个界限； 256 位椭圆方案可能与使用 128 位密钥的 128 位分组密码一样难以破解； NSA 的军事算法套件 B 中唯一使用的公钥加密方案是 384 位椭圆曲线系统。传统的基于因式分解和离散对数的方案现在需要 3072 位密钥来保护 Top Secret 的材料,因为存在数字字段筛选等快捷攻击算法。因此,椭圆曲线密码系统更快。

当我在 2000 年写这本书的第一版时,数字字段筛选已经被用来攻击高达 512 位的密钥,这个任务的难度与 56 位 DES 密钥的密钥搜索相当;当我在 2007 年为第二版重写这一章时,64 位对称密钥已经被暴力破解,663 位挑战号码 RSA-200 已经被分解。到 2019 年的第三版,比特币矿工每十分钟就会发现 68 位哈希冲突,RSA-768 已经被分解,Ed Snowden 告诉我们 NSA 可以为 1024 位质数模数做离散日志。

对量子计算机进行了大量研究 使用叠加的量子态同时执行大量计算的设备。 Peter Shor 已经表明,如果可以建造一台足够大的量子计算机,那么因式分解和离散对数计算都将变得容易 [1725]。到目前为止,只建造了非常小的量子设备;尽管偶尔有人声称“量子至上” 量子计算机执行任务的速度比传统计算机快得多,以说服我们量子叠加或纠缠正在做任何真正的工作 但它们似乎无处可去。我怀疑（和许多物理学家一样)这项技术是否会威胁到真实的系统。我更怀疑量子密码学的价值;它可能能够重新加密使用 AES 进行批量加密的线路加密设备,但我们已经知道如何做到这一点。

更重要的是,我发现为基于纠缠的量子密码学提供的安全证明并不令人信服 [311]。自 20 世纪 70 年代初杰拉德·特霍夫特 (Gerard t Hooft) 通过证明杨-米尔斯 (Yang-Mills) 的可重整性来完成标准模型以来,理论物理学一直停滞不前。从那时起,一系列的想法来来去去,例如弦理论 [2032]。量子信息论是最新的热潮。它的支持者大谈贝尔测试的奥秘,它应该证明物理学不能同时是局部的和因果的。但是其他解释,如 t Hooft 的元胞自动机模型 [916] 和 Grisha Volovik 的超流体模型 [1967] 表明,贝尔测试仅仅证明了量子真空中长程有序的存在,就像超流体的序参数一样。现在有经典的力学实验可以证明与贝尔测试相关的量子力学特性 [1557]。当我指出量子密码系统所声称的证明中的此类漏洞时,支持者的唯一回答似乎是人身攻击。

我个人认为更有可能是对公钥密码的重大挑战

地理学将以更好的算法形式出现,用于计算椭圆曲线上的离散对数。这些曲线有很多结构;一些世界上最聪明的纯数学家对它们进行了深入研究； 2013 年发现了更好的用于小特征曲线的离散对数算法 [168];美国国家安全局显然正在放弃使用椭圆曲线加密。

如果量子计算机能够工作,我们还有其他“后量子”算法可供使用,而量子计算机并没有提供明显的优势。 2020 年,NIST 开始对后量子密码学标准化进程提交的第三轮公开审查。通过两轮审查12,最初提交的 65 份已减少到 15 份。现在将选择一种或多种算法并将其标准化,因此使用它们的密码套件可以作为升级放入 TLS 等协议中。许多正在使用的协议甚至可以重新设计以在 Kerberos 上使用变体。如果椭圆对数变得容易,我们就有了这些资源,也可以回退到素数域中的离散对数,或者 RSA。但是,如果椭圆日志变得容易,比特币将变得微不足道,加密货币生态系统可能会崩溃,结束我在第 20.7 节中描述的极其浪费的挖矿操作。因此,关心地球未来的数学家可能比考虑椭圆对数问题做得更糟。

5.7.9 还有什么问题

现在很少有系统攻击涉及对加密算法或密钥进行数学攻击的密码分析。确实存在针对 20 世纪设计的系统的攻击,主要涉及因出口管制规则、无能的设计或两者而保持太短的密钥。我已经在第 4.3.1 节中讨论了弱密码如何助长了一波汽车盗窃浪潮,因为所有用于远程钥匙输入的设备在 2005-15 年相继被攻破。在后面的章节中,我将举例说明加密战争及其出口管制规则如何导致对门锁（第 13.2.5 节）、手机（第 22.2.1 节)和版权执法（第 24.2.5 节)的攻击。

现在大多数攻击都利用了这个实现。在第 2 章中,我提到了 NIST 标准化基于椭圆曲线的复杂随机数生成器的丑闻,结果发现它包含 NSA 后门;参见第 2.2.1.5 节。糟糕的随机数生成器导致了许多其他故障:具有公因数的 RSA 密钥 [1140]、离散日志的可预测种子 [1676] 等。这些漏洞仍在继续;多亏了物联网,人们可以在互联网上找到的与其他 RSA 密钥具有共同因素的 RSA 证书的比例在 2012 年至 2020 年间实际上有所上升； 172 个 IoT 证书中有 1 个很容易受到攻击 [1046]。

在过去 20 年中迫使重大变化的密码实现的许多实际攻击都利用了时序和功率分析等辅助通道;我将第 19 章专门介绍这些内容。

12 其中之一,即 McEliece 密码系统,自 1978 年以来一直存在;我们拥有基于散列函数的数字签名的时间差不多一样长,我们中的一些人在 1990 年代使用它们来避免支付 RSA 的专利使用费。

在第 20 章中,我将讨论一些系统,它们以复杂的方式使用公钥机制来获得有趣的涌现属性,包括 Signal 消息传递协议、TOR 匿名系统和加密货币。

我还将研究 SGX 飞地的加密方面。这些也有有趣的故障模式,其中一些但不是全部都与侧信道有关。

在第 21 章中,我将讨论网络基础设施中使用的协议,例如
DKIM、DNSSec 与 DNS over HTTP 和 SSH。

5.8 总结

许多密码因为使用不当而失败,因此安全工程师需要清楚地了解不同类型的密码的作用。这可以在不同的层面上解决;一个是在密码学层面,可以讲随机预言机模型、具体模型和语义安全模型,希望避免使用弱操作模式等构造。下一个层次是个体密码的设计,例如 AES,或作为公钥密码系统和数字签名机制基础的数论机制。这些也有自己的专业数学领域,即块密码密码分析和计算数论。下一个级别涉及实施不良,这更加棘手和混乱。

这涉及处理时间、错误处理、功耗和各种其他肮脏的细节,并且是现代密码系统在实践中往往会被破解的地方。

深入了解真实系统,我们已经讨论了如何通过仔细组合替换和排列来构建用于对称密钥应用程序的分组密码;对于非对称应用,例如公钥加密和数字签名,可以使用数论。在这两种情况下,都有大量的数学知识。其他类型的密码 流密码和散列函数 可以通过在合适的操作模式下使用分组密码来构建。它们具有不同的错误传播、模式隐藏和完整性保护特性。许多系统失败是因为流行的加密库鼓励程序员通过公开不安全的默认值来使用不适当的操作模式。除非您真正了解自己在做什么,否则切勿使用 ECB 模式。

还有许多其他可能出错的地方,从侧信道攻击到糟糕的随机数生成器。特别是,即使在组件发生故障(或被鼓励发生故障)并且加密机制与访问控制和物理安全等其他措施很好地集成的情况下,构建仍然健壮的系统也非常困难。我将在后面的章节中反复讨论这一点。

寓意是:不要自己动手!不要设计自己的协议或密码;除非绝对必要,否则不要编写自己的加密代码。如果你这样做了,那么你不仅需要读这本书(然后再仔细读一遍);您需要阅读相关的专业材料,与专家交谈,并让有能力的积极分子尝试打破它。至少,您需要对您的工作进行同行评审。设计加密货币有点像杂耍

链锯;犯致命错误太容易了。

研究问题

密码学研究中有许多活跃的线索。其中许多是加密与特定数学分支（数论、代数几何、复杂性理论、组合学、图论和信息论)相遇的地方。业务的实证端与设计用于加密、签名和复合操作的原语有关,这些原语在可用平台上表现得相当好。两人在从密码分析到寻找将可证明的安全属性与良好性能相结合的原语等主题的研究中相遇。

了解事物在理论上发生了什么的最好方法是阅读过去几年的研究会议论文集,例如 Crypto、Eurocrypt 和 Asiacrypt;密码设计工作出现在快速软件加密;对实现的攻击经常出现在 CHES 上;而在顶级系统安全会议（如 IEEE 安全和隐私、CCS 和 Usenix)中可以找到关于密码在系统中的使用方式的攻击。

进一步阅读

Whit Die 和 Martin Hellman [556] 以及 Ron Rivest、Adi Shamir 和 Len Adleman [1607] 的经典论文最接近该主题的必读内容。 Bruce Schneier 的 Applied Cryptography [1667] 在非数学家可以理解的水平上涵盖了很多领域,并且尽管有美国出口管制法律,但在 1990 年代就已经有了密码代码,但现在有点过时了。 Alfred Menezes、Paul van Oorshot 和 Scott Vanstone 的 Handbook of Applied Cryptography [1289] 是一本关于数学细节的参考书。卡茨和林德尔是我们让学生阅读数学的书。它介绍了标准密码理论以及公钥密码所需的数论（包括椭圆曲线和指数演算） ,但也过时了:它们没有提到 GCM,例如 [1022]。

还有更多的专业书籍。关于差分密码分析的圣经是由它的发明者 Eli Biham 和 Adi Shamir [245] 编写的,而关于线性和差分密码分析的一个很好的简短教程是由 Howard Heys [895] 编写的。
Doug Stinson 的教科书对线性密码分析 [1829] 有另一种详细解释;分组密码的现代理论可以通过快速软件加密会议系列中的论文来追溯。关于操作模式的原著由 Carl Meyer 和 Steve Matyas [1301] 合着。Neal Koblitz 对公钥密码学背后的数学有很好的基本介绍 [1060]； Arjen 和 Henrik Lenstra [1141] 描述了数域筛。关于过去 20 年对 TLS 的实际攻击,请参阅 Christopher Meyer 和 Joerg Schwenk 的调查论文 [1302] 以及本书后面有关侧通道的章节。

如果您想了解理论密码学的数学细节,可以阅读 Dan Boneh 和 Victor Shoup 合着的最新研究生教科书 [287]。

在 [835] 中对随机性和算法进行了不太彻底但更具可读性的介绍。在 FOCS、STOC、Crypto、Eurocrypt 和 Asiacrypt 会议上可以找到密码学理论端的研究。

密码学的历史引人入胜，许多老问题不断出现。标准工作是 Kahn [1001]；还有来自 Cryptologia [531、529、530] 的历史文章汇编，以及 Kahn、Marks、Welchman 等人撰写的关于二战密码学历史的几本书 [438、1002、1224、2007]。位于马里兰州乔治米德堡的美国国家安全局博物馆也值得一游，但最好的也许是位于英格兰布莱切利公园的博物馆。

最后，任何介绍公钥加密的章节都必须提到，在“非秘密加密”的名义下，它是 James Ellis 于 1969 年左右首次发现的。然而，由于 Ellis 在 GCHQ 工作，他的工作仍然存在分类。 RSA 算法随后由 Cli ord Cocks 发明，并且也保密。这个故事在[625]中讲述。保密的一个影响是他们的工作没有被使用：虽然它的动机是陆军密钥分发的费用，但英国国防部直到 1992 年才开始为其主要网络构建电子密钥分发系统。

而且机密社区并没有预先发明数字签名；它们仍然是 Whit Die 和 Martin Hellman 的成就。