第 3 章　公钥密码和消息认证

学习目标

学习完这一章后，你应该能够：

- 定义术语消息认证码。
- 列出并解释消息认证码的基本要求。
- 解释为什么用在消息认证码中的散列函数必须是安全的。
- 理解原像攻击、第二原像攻击和碰撞攻击之间的区别。
- 理解 SHA-512 的操作。
- 给出 HMAC 的概述。
- 给出公钥密码系统基本原则的概述。
- 解释公钥密码系统两种不同的用途。
- 给出 RSA 算法的概述。
- 定义 Diffie-Hellman 密钥交换协议。
- 理解中间人攻击方法。

除消息机密性外，消息认证也是一个重要的网络安全功能。本章分析消息认证的三个方面。首先，研究使用**消息认证码**（MAC）和散列函数进行消息认证。然后分析公钥密码原理和两个具体的公钥算法。这些算法在交换常规公钥时非常有用。最后分析使用公钥密码生成数字签名，从而提供了加强版的消息认证。

3.1　消息认证方法

加密可以防止被动攻击（窃听）。加密的另一个要求是可以防止主动攻击（伪造数据和业务）。防止这些攻击的办法被称为消息认证。

当消息、文件、文档或其他数据集合是真实的且来自合法来源，则称其为可信的。消

息认证是一种允许通信者验证所收消息是否可信的措施。认证包括两个重要方面：验证消息的内容有没有被篡改和验证来源是否可信[1]。还可能希望验证消息的时效性（消息有没有被人为地延迟或重放）以及两实体之间消息流的相对顺序。这些问题属于第 1 章介绍的数据完整性范畴。

3.1.1 利用常规加密的消息认证

仅简单地使用常规加密似乎也可以进行消息认证。假设只有发送者和接收者共享一个密钥（这是合理的），那么假定接收者能够识别有效消息时，只有真正的发送者才能够成功地为对方加密消息。 此外，如果消息里带有错误检测码和序列号，则接收者能够确认消息是否被篡改过和序列号是否正常。如果消息里还包含时间戳，则接收者能够确认消息没有超出网络传输的正常延时。

事实上，对数据认证而言只使用对称加密的方法不是一个合适的工具。一个简单的例子是，在分组加密算法的 ECB 工作模式下，攻击者重排密文分组次序后每一个分组仍然能被成功地解密。虽然在某些层级（例如各 IP 包）可以使用序列号，但通常情况下一个单独的序列号不一定与明文中的每个 b 比特分组发生联系。因此，分组的重排是一种威胁。

3.1.2 非加密的消息认证

本节分析几种不依赖于加密的消息认证方法。 所有这些方法都会生成认证标签，并且附在每一条消息上用于传输。消息本身并不会被加密，所以它在目的地可读而与目的地的认证功能无关。

由于本节讨论的方法并不加密消息，所以不提供消息的保密性。正如上面所指出的，通过对消息自身的加密并不能提供一种安全的认证形式。然而，在一个单一算法中通过加密消息并附上认证标签的方法把消息的认证和保密结合起来是可能的。通常，消息认证与消息加密是两个独立的功能。[DAVI89]给出了三种无须保密的消息认证情况：

（1）许多应用需要把相同的消息广播到多个目的地。其中两个例子就是：通知当前的用户网络不可使用和控制中心的警报信号。只由一端负责监控的认证既经济又安全。因此，消息必须以带有相关消息认证标签的明文形式进行发送。负责监控的系统执行认证。一旦出现冲突，则普通的警报信号就会警告其他目的端系统。

（2）在信息交换中，另一种可能的情况是通信某一端的负载太大，没时间解密所有传入的消息。这时就会有选择的随机抽取消息进行认证。

（3）对明文形式的计算机程序进行认证是很有意义的工作。这些程序不用每次都进行解密就可以运行，从而节省了处理器资源。然而，如果程序含有消息认证标签，则当消息完整性需要确认的时候要进行认证。

可见，在满足安全需求上认证和加密都有其应用场所。

消息认证码。一种认证技术利用私钥产生一小块数据，称之为**消息认证码**，将其附到

1 为简单起见，本章余下部分统称消息认证，它既表示被传输消息的认证，也表示存储数据的认证（数据认证）。

消息上。该技术假设两个通信实体（如 A 和 B）共享一个公共密钥 K_{AB}。当 A 有消息要发送给 B 时，A 计算消息认证码（MAC），作为消息和密钥的一个函数：$\mathrm{MAC}_M = F(K_{AB}, M)$。消息连同 MAC 被一起传送给预定接收者。接收者对接收到的消息使用相同的密钥做相同运算，生成新的 MAC。比较收到的 MAC 和计算得到的 MAC（见图 3.1）。假设只有接收者和发送者知道密钥，若收到的认证码与计算得到的认证码相吻合，则可得出下列结论：

（1）接收者能够确认消息没有被篡改。如果攻击者篡改消息却没有改变相应的 MAC，则接收者计算的 MAC 不同于收到的 MAC。因为假设攻击者不知道密钥，他不能修改 MAC 使其与改动后的消息保持一致。

（2）接收者能够确保消息来自合法的发送者。因为没有其他人知道密钥，所以他们就不能生成具有正确 MAC 的消息。

（3）如果消息中包含序列号（如 X.25、HDLC 和 TCP 使用的序列号），而攻击者不能成功地修改序列号，那么接收者就可以确认消息的正确序列。

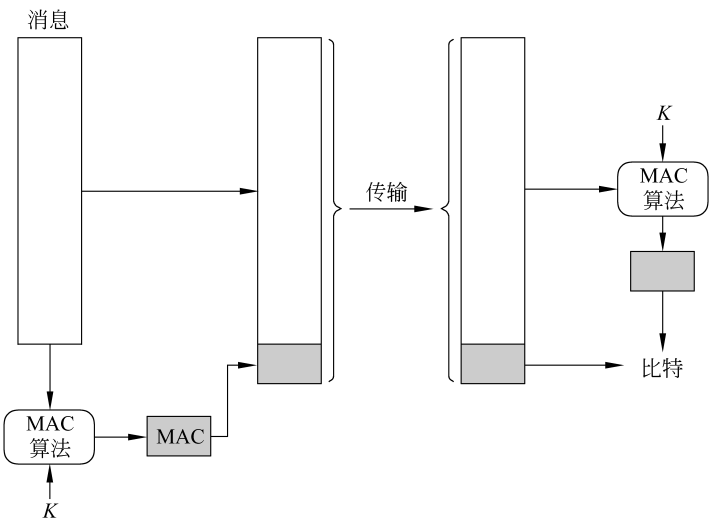


图 3.1　使用消息认证码（MAC）的消息认证

许多算法都可以生成 MAC，NIST 标准 FIPS PUB 113 推荐使用 DES。DES 可以生成加密形式的消息，密文的最后几个比特用作 MAC。典型的有 16 或 32 比特的 MAC。

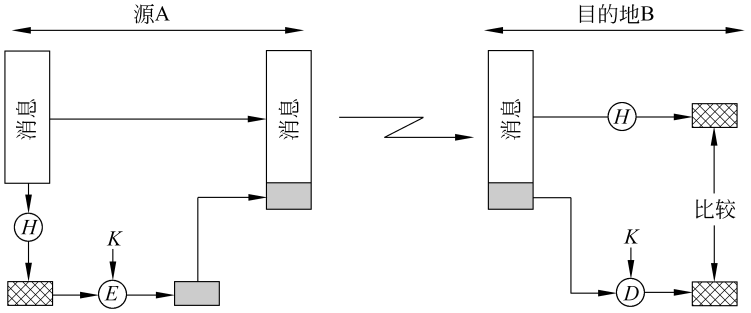
前面描述的过程类似于加密。不同点是认证算法不需要可逆，而可逆对于解密则是必需的。可以看出，由于认证函数的数学特性，与加密相比它更不容易被攻破。

单向散列函数。MAC 的一种替代方法是使用**单向散列函数**。如同 MAC，散列函数接收变长的消息 M 作为输入，生成定长的消息摘要 $H(M)$作为输出。与 MAC 不同的是散列函数不需要密钥输入。为了认证消息，消息摘要随消息一起以可信的形式传送。

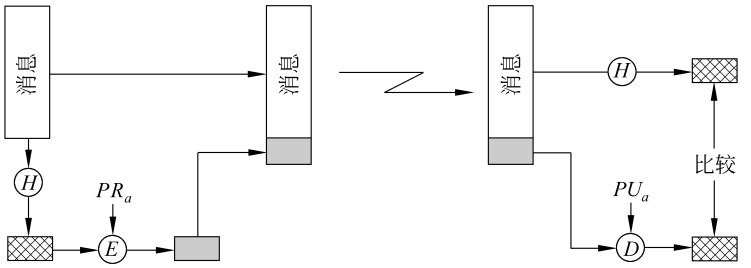
图 3.2 显示了消息认证的三种方式。消息摘要可用传统密码（见图 3.2（a））；如果只有发送者和接收者共享密钥，则保密性是能保证的。消息也可以用公钥方式加密（见图 3.2（b））；这将在 3.5 节进行讲述。公钥方法有两个优势：

（1）既能提供数字签名又能提供消息认证；

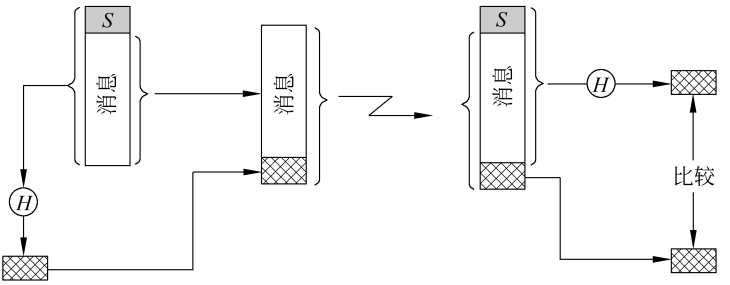
（2）不需要在通信各方之间分发密钥。



(a) 使用传统加密



(b) 使用公钥加密



(c) 使用秘密值

图 3.2 使用单向散列函数的消息认证

这两种方法与那些加密整个消息的方法相比具有一定的优势，因为它们只需很少的计算量。所以开发一种无须整体加密的技术很有吸引力，其原因在[TSUD92]中给予了说明：

- 软件加密速度非常低。虽然每条消息需加密的数据量很少，但是也可能会形成一个稳定的消息流输入和输出系统。
- 硬件加密成本不容忽视。用廉价芯片实现 DES 是可行的，但如果网络中的所有节点都必须有该硬件，则总成本太大。
- 硬件加密对大批量数据来说具有优势。但对于小块数据，将会有大量时间花费在前期的初始化/调用上面。
- 加密算法可能受专利保护。

图 3.2（c）是一种使用了散列函数但是没有使用加密的消息认证技术。这一技术假设通信双方（例如 A 和 B）共享秘密值 S_{AB}。当 A 要给 B 传送消息时，A 计算秘密值与消息

串接后的散列函数：$MD_M = H(S_{AB} \| M)$[2]。然后发送[$M \| MD_M$]给 B。由于 B 拥有 S_{AB}，它能重新计算 $H(S_{AB} \| M)$，验证 MD_M。因为秘密值本身并不会发送，所以攻击者不可能修改所截获的消息。只要不泄露秘密值，攻击者就不可能生成伪消息。

第三种技术的一种变体称作 HMAC，在 IP 安全中使用了该认证方法（在第 9 章中描述）；对 SNMPv3 也有描述（见第 13 章）。

3.2　安全散列函数

单向散列函数或安全散列函数之所以重要，不仅在于消息认证，还有数字签名。本节从安全散列函数的要求开始讨论，然后研究最重要的散列函数：SHA。

3.2.1　散列函数的要求

散列函数的目的是为文件、消息或其他数据块产生“指纹”。为满足在消息认证中的应用，散列函数 H 必须具有下列性质：

（1）H 可适用于任意长度的数据块。

（2）H 能生成固定长度的输出。

（3）对于任意给定的 x，计算 $H(x)$相对容易，并且可以用软/硬件方式实现。

（4）对于任意给定值 h，找到满足 $H(x)=h$ 的 x 在计算上不可行。满足这一特性的散列函数称为具有**单向性**，或具有**抗原像攻击性**[3]。

（5）对于任意给定的数据块 x，找到满足 $H(y) = H(x)$的 $y \neq x$ 在计算上是不可行的。满足这一特性的散列函数被称为具有**抗第二原像攻击性**，有时也称为具有**抗弱碰撞攻击性**。

（6）找到满足 $H(x) = H(y)$ 的任意一对(x, y)在计算上是不可行的。满足这一特性的散列函数被称为**抗碰撞性**，有时也被称为**抗强碰撞性**。

前三个性质是使用散列函数进行消息认证的实际可行要求。第四个属性，抗原像攻击，是单向性：给定消息容易产生它的散列码，但是给定散列码几乎不可能恢复出消息。如果认证技术中使用了秘密值（见图 3.2（c）），则单向性很重要。秘密值本身并不会传输；然而，假如散列函数不是单向的，而攻击者能够分析或截获消息 M 和散列码 $C = H(S_{AB} \| M)$，则攻击者很容易发现秘密值。攻击者对散列函数取逆得到 $S_{AB} \| M = H^{-1}(C)$。因为这时攻击者拥有 M 和 $S_{AB} \| M$，所以他们可以轻而易举地恢复 S_{AB}。

抗第二原像攻击性质保证了对于给定的消息，不可能找到具有相同散列值的可替换消息。利用加密的散列码可防止消息被伪造（见图 3.2（a）和图 3.2（b））。如果该性质非真，则攻击者可以进行如下操作：第一，分析或截获消息及其加密的散列码；第二，根据消息产生没有加密的散列码；第三，生成具有相同散列码的可替换消息。

满足上面前 5 个性质的散列函数称为弱散列函数。如果还满足第 6 个性质则称其为强

2　‖ 表示串接。

3　对 $f(x) = y$，称 x 为 y 的一个原像。除非 f 是一一映射，否则对给定的 y，可能会有多个原像存在。

散列函数。第 6 个性质可以防止像生日攻击这种类型的复杂攻击。生日攻击的细节超出了本书的范围。这种攻击把 m 比特的散列函数的强度从 2^m 简化到 $2^{m/2}$。详细内容可参考[STAL11]。

除提供认证之外，消息摘要还能验证数据的完整性。它与帧检测序列具有相同的功能：如果在传输过程中，意外地篡改任意比特，消息摘要则会出错。

3.2.2 散列函数的安全性

正如对称加密，也有两种方法可以攻击一个安全散列函数：密码分析法和蛮力攻击法。对于对称加密算法，密码分析法利用了该算法在逻辑上的缺陷。

散列函数抵抗蛮力攻击的强度完全依赖于算法生成的散列码长度。攻击一个长度为 n 的散列码所需要付出的代价如下：

抗原像	2^n
抗第二原像	2^n
抗碰撞	$2^{n/2}$

如果需要抗碰撞（即一般安全散列码所需要的），则数值 $2^{n/2}$ 是抗蛮力攻击能力的一个度量。Van Oorschot 和 Wiener[VANO94]曾经提出，花费 1000 万美元设计一个被专门用来搜索 MD5 算法（散列长度为 128 比特）碰撞的机器，则平均在 24 天内就可以找到一个碰撞（这一结果是 2004 年之前的情况。2004 年 8 月中国密码学家王小云教授等首次公布了提出一种寻找 MD5 碰撞的新方法。目前利用该方法用普通微机数分钟内即可找到 MD5 的碰撞。MD5 已被彻底攻破。有关这方面情况，读者可参考其他资料——译者注）。因而，128 比特的散列长度仍然不够。今后，一个散列码可能会是 32 比特的序列，它的散列长度为 160 比特。对于 160 比特的散列长度，同样的搜索机器要花费超过 4000 年的时间才能找到一个冲突。在当今的技术下，搜索时间可能会缩短许多，所以 160 比特的散列长度现在看来也有些不可信。

3.2.3 简单散列函数

所有散列函数都按照下面基本原理操作。把输入（消息、文件等）看成 n 比特块的序列。对输入用迭代方式每次处理一块，生成 n 比特的散列函数。

一种最简单散列函数的每一个数据块都按比特异或。这可以用下式表示：

$$C_i = b_{i1} \oplus b_{i2} \oplus \cdots \oplus b_{im}$$

其中：

C_i 为散列码的第 i 比特，$1 \leqslant i \leqslant n$；

m 为输入中 n 比特数据块的数目；

b_{ij} 为第 j 块的第 i 比特；

$\oplus$ 为异或操作。

图 3.3 说明了这种操作；它为每一比特位置产生简单的奇偶校验，这称为纵向冗余校

验。这种校验对于随机数据的完整性检验相当有效。因为每个 n 比特的散列值都有相同的可能性，所以数据出错却不改变散列值的概率是 2^{-n}。随着可预测的格式化数据增多，函数的有效性越来越差。例如，在大多数标准的文本文件中，每个 8 位字节的高阶比特总是零。因此，如果使用 128 比特的散列值，则作用于该类型数据块的散列函数的有效概率为 2^{-112}，而不是 2^{-128}。

	第1比特	第2比特		第n比特
第1块	b_{11}	b_{21}		b_{n1}
第2块	b_{12}	b_{22}		b_{n2}
	⋮	⋮	⋮	⋮
第m块	b_{1m}	b_{2m}		b_{nm}
散列码	C_1	C_2		C_n

图 3.3　使用按比特异或的简单散列函数

对上面的方案进行改进的一种简单方法是在每个数据块处理后，对散列值循环移动或旋转 1 比特。其步骤归纳如下：

（1）最初将 n 比特散列值设置为零。

（2）如下连续处理每个 n 比特的数据块：

a. 将当前的散列值向左旋转 1 比特。

b. 异或数据块生成散列值。

这些操作会使输入数据随机化得更加彻底，消除了输入中出现的任何规则性。

虽然步骤（2）提供了很好的数据完整性方法，但如图 3.2（a）和图 3.2（b）所示，当加密散列码和明文消息一起使用时，它对于数据安全性几乎不起作用。给定一个消息，很容易生成新的具有相同散列码的消息：简单地准备希望替换的消息，然后附加上 n 比特的数据块，迫使新消息连同该数据块生成期望的散列码。

如果仅仅对散列码加密，简单异或或者旋转异或（RXOR）不够安全。但是如果连同散列码一起加密消息，可能会觉得这个简单函数还是有用的。但是必须非常小心。最初由美国国家标准局提出的一项技术就是对 64 比特消息块进行简单异或操作，然后用密码分组链接（CBC）模式加密整条消息。可以如下定义该方案：给定由 64 比特分组 $X_1, X_2, \cdots, X_N$ 序列组成的消息，定义散列码 C 为逐块异或或者所有分组的异或，再把得到的散列码附加上作为最后一个模块：

$$C = X_{N+1} = X_1 \oplus X_2 \oplus \cdots \oplus X_N$$

然后，用 CBC 模式把整条消息和散列码一起加密，生成加密后的消息 $Y_1, Y_2, \cdots, Y_{N+1}$。[JUEN85]指出了操作消息密文而不会被散列码检测到的几种方法。例如，根据 CBC（见图 2.9）定义，有：

$$X_1 = \mathrm{IV} \oplus D(K, Y_1)$$
$$X_i = Y_{i-1} \oplus D(K, Y_i)$$
$$X_{N+1} = Y_N \oplus D(K, Y_{N+1})$$

但是散列码 X_{N+1}：

$$X_{N+1} = X_1 \oplus X_2 \oplus \cdots \oplus X_N$$
$$= [\mathrm{IV} \oplus \mathrm{D}(K, Y_1)] \oplus [Y_1 \oplus \mathrm{D}(K, Y_2)] \oplus \cdots \oplus [Y_{N-1} \oplus \mathrm{D}(K, Y_N)]$$

因为上述等式中的项可以按任意顺序进行异或，所以如果对密文分组进行置换则散列码不变。

3.2.4 SHA 安全散列函数

近些年，应用最为广泛的散列函数是安全散列算法（SHA）。由于其他每一种被广泛应用的散列函数都已被证实存在着密码分析学中的缺陷，截止到 2005 年，SHA 或许是最后仅存的标准散列算法。SHA 由美国国家标准与技术研究所（NIST）开发，并在 1993 年公布成为美国联邦信息处理标准（FIPS 180）。当人们发现 SHA 中也存在缺陷之后（目前已知在 SHA-0 中存在），FIPS 180 的修订版本 FIPS 180-1 于 1995 年公布出来，通常称为 SHA-1。现行的标准文献被命名为“安全散列标准”。SHA 是基于散列函数 MD4，并且其构架跟 MD4 高度相仿。RFC 3174 中也列出了 SHA-1，但它实质上是 FIPS 180-1 的复制品，只是增加了 C 语言代码实现。

SHA-1 生成的 160 比特的散列值。2002 年，NIST 制定了修订版本的标准：FIPS-2。它定义了三种新版本的 SHA，散列长度分别为 256、384、512 比特，分别称为 SHA-256、SHA-384、SHA-512，三者并称为 SHA-2。这些新版本使用了与 SHA-1 相同的底层结构和相同类型的模运算以及相同的二元逻辑运算。2008 年发布出来的修订文献 FIP PUB 180-3 增加了 224 比特的版本（如表 3.1 所示）。RFC 4634 中也列出了 SHA-2，但它实质上是 FIPS 180-3 的复制品，只是增加了 C 语言代码实现。

表 3.1 SHA 参数的比较

	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
消息摘要大小	160	224	256	384	512
消息大小	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
块大小	512	512	512	1024	1024
字大小	32	32	32	64	64
步骤数	80	64	64	80	80

注： 所有的大小以比特衡量。

2005 年，NIST 宣布计划到 2010 年不再认可 SHA-1，转为信任 SHA-2。此后不久，有研究团队描述了一种攻击方法，该方法可以找到产生相同 SHA-1 的两条独立的消息，它们只用 2^{69} 次操作，远少于以前认为找到 SHA-1 碰撞所需的 2^{80} 次操作[WANG05]。这个结果将加快 SHA-1 过渡到 SHA-2（该段描述的情况有误。2005 年，王小云教授等提出了对 SHA-1 的攻击，使得找到碰撞的复杂度降到 2^{69} 次操作，之后又降到 2^{63} 次操作。这些结果迫使 NIST 在 2006 年宣布 2010 年后不再推荐使用 SHA-1——译者注）。

本节对 SHA-512 做一介绍，其他 SHA 算法与之很相似。

该算法以最大长度不超过 2^{128} 比特的消息作为输入，生成 512 比特的消息摘要输出。输入以 1024 比特的数据块进行处理。图 3.4 描述了处理消息生成摘要的全过程。处理过程

包括以下步骤：

- **第 1 步：追加填充比特**。填充消息使其长度模 1024 同余 896[长度≡896(模 1024)]。即使消息已经是期望的长度，也总是要添加填充。因此，填充比特的范围是 1～1024。填充部分是由单个比特 1 后接所需个数的比特 0 构成。
- **第 2 步：追加长度**。将 128 比特的数据块追加在消息上。该数据块被看作 128 比特的无符号整数（高位字节在前），它还含有原始消息（未填充前）的长度。

  前两步生成了长度为 1024 比特整数倍的消息。在图 3.4 中，被延展的消息表示为 1024 比特的数据块序列 $M_1, M_2, \cdots, M_N$，所以延展后消息总长度为 $N\times1024$ 比特。

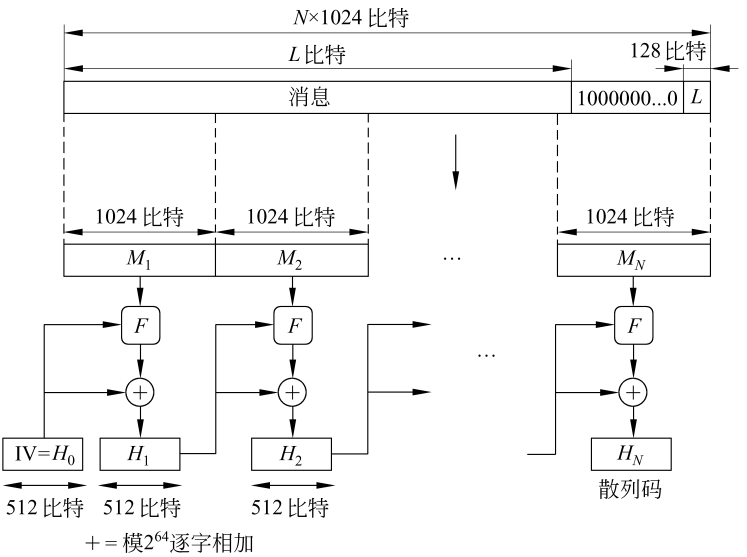


图 3.4　用 SHA-512 生成消息摘要

- **第 3 步：初始化散列缓冲区**。用 512 比特的缓冲区保存散列函数中间和最终结果。缓冲区可以是 8 个 64 比特的寄存器（a、b、c、d、e、f、g、h）。这些寄存器初始化为 64 比特的整数（十六进制值）：

  a=6A09E667F3BCC908　　e=510E527FADE682D1
  b=BB67AE8584CAA73B　　f=9B05688C2B3E6C1F
  c=3C6EF372FE94F82B　　g=1F83D9ABFB41BD6B
  d=A54FF53A5F1D36F1　　h=5BE0CDI9137E2179

  这些值以逆序的形式存储，即字的最高字节存在最低地址（最左边）字节位置。这些字的获取方式如下：前 8 个素数取平方根，取小数部分的前 64 位。
- **第 4 步：处理 1024 比特（128 字）的数据块消息**。算法的核心是 80 轮迭代构成的模块；该模块在图 3.4 中标记为 F，图 3.5 说明其逻辑关系。

  每一轮都以 512 比特的缓冲区值 abcdefgh 作为输入，并且更新缓冲区内容。在第一轮时，缓冲区里的值是中间值 H_{i-1}。在任意第 t 轮，使用从当前正在处理的 1024 比特的数据块(M_i)导出的 64 比特值 W_t。每一轮还使用附加常数 K_t，其中 $0\leqslant t\leqslant 79$ 表示 80 轮中的某一轮。这些常数的获取方式如下：前 8 个素数取立方根，取小数部分的前 64 位。这些常数提供了 64 位随机串集合，可以初步消除输入数据中的

任何规则性。第 80 轮输出加到第 1 轮输入（H_{i-1}）生成 H_i。缓冲区里的 8 个字与 H_{i-1} 中相应的字进行模 2^{64} 加法运算。

- **第 5 步：输出**。当所有 N 个 1024 比特的数据块都处理完毕后，从第 N 阶段输出的便是 512 比特的消息摘要。

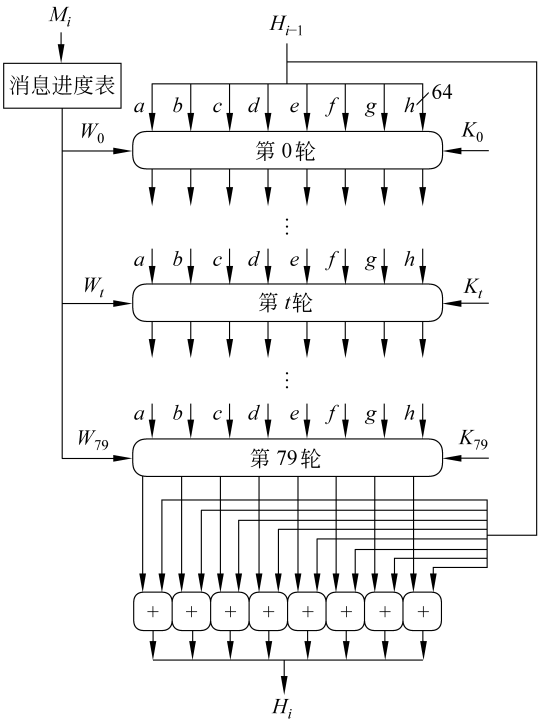


图 3.5　SHA-512 处理单个 1024 比特的数据块

SHA-512 算法使得散列码的任意比特都是输入端每 1 比特的函数。基本函数 F 的复杂迭代产生很好的混淆效果；即随机选取两组即使有很相似的规则性的消息也不可能生成相同的散列码。除非 SHA-512 隐含一些直到现在还没有公布的弱点，构造具有相同消息摘要的两条消息的难度的数量级为 2^{256} 步操作，而找出给定摘要的消息的难度为 2^{512}。

3.3　消息认证码

3.3.1　HMAC

近年来，人们对从加密散列码（如 SHA-1）中开发 MAC 越来越感兴趣。这种兴趣的动机是：

- 采用软件实现时，加密散列函数执行速度比传统密码算法（如 DES）快。
- 有许多共享的密码学 Hash 函数代码库。

诸如 SHA-1 这样的散列函数并不是专为 MAC 而设计的，因为散列函数不依赖于密钥，所以它不能直接用于此目的。已经有很多方案可以把密钥合并到现有的散列算法中。最被

广泛接受的方案就是 HMAC [BELL96a、BELL96b]。HMAC 已经发布为 RFC 2104 标准，它是 IP 安全里必须实现的 MAC 方案。HMAC 也被用于其他 Internet 协议，如传输层安全（TLS，即 Transport Layer Security，它将很快取代安全套节字层）和安全电子交易（SET，即 Secure Electronic Transaction）。

HMAC 的设计目标。RFC 2104 为 HMAC 列出了下列设计目标。

- 不必修改而直接使用现有的散列函数。特别是很容易免费得到软件上执行速度较快的散列函数及其代码。
- 嵌入式散列函数要有很好的可移植性，以便开发更快或更安全的散列函数。
- 保持散列函数的原有性能，不发生显著退化。
- 使用和处理密钥简单。
- 如果已知嵌入的散列函数的强度，则完全可以知道认证机制抗密码分析的强度。

前两个目标是 HMAC 为人们所接受的重要原因。HMAC 把散列函数当成一个“黑盒”。这有两个优点：第一，实现 HMAC 时现有的散列函数可以用作一个模块。这样，已经预先封装的大量 HMAC 代码可以不加修改地使用。第二，如果想替换一个 HMAC 实现中的特定散列函数，所需做的只是除去现有的散列函数模块，放入新模块。如果需要更快的散列函数时就可以这样操作。更为重要的是，如果嵌入式散列函数的安全性受到威胁，可通过简单地用更加安全的模块取代嵌入式散列函数，从而保证了 HMAC 的安全。

实际上，上面列表中的最后一个目标是 HMAC 相对其他散列方案最主要的优点。如果能提供有一定合理性的抗密码分析强度的嵌入式散列函数，就能够证明 HMAC 是安全的。在本节后面再对此进行阐述，但首先分析 HMAC 的结构。

HMAC 算法。图 3.6 描述了 HMAC 的整个操作。定义下列术语：

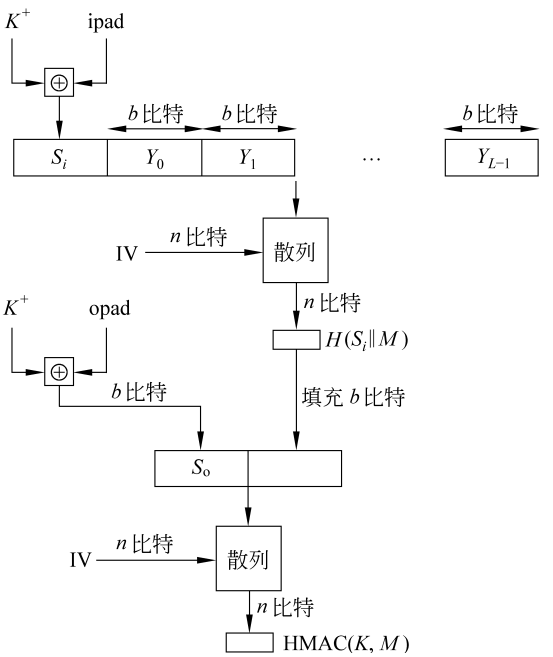


图 3.6　HMAC 结构

H = 嵌入的散列函数（如 SHA-1）。

M = 输入 HMAC 的消息（包括嵌入式散列函数中特定的填充部分）。

Y_i = M 的第 i 个分组（$0 \leqslant i \leqslant (L-1)$）。

L = M 中的分组数。

b = 分组中的比特数。

n = 嵌入式散列函数产生的散列码长度。

K = 密钥；如果密钥长度大于 b，则将其输入给散列函数生成 n 比特的密钥；建议长度大于或等于 n。

K^+ = 为使 K 为 b 位长而在 K 左边填充 0 后所得的结果。

ipad = 00110110（十六进制数为 36）重复 b/8 次。

opad = 01011100（十六进制数为 5C）重复 b/8 次。

那么 HMAC 可用下式表示：

$$\text{HMAC}(K, M) = H[(K^+ \oplus \text{opad}) \| H[(K^+ \oplus \text{ipad}) \| M]]$$

该算法描述如下：

（1）在 K 的左端追加 0 构成 b 比特的字符串 K^+（如 K 的长度为 160 比特，b=512，K 将被追加 44 个 0 字节 0x00）。

（2）ipad 与 K^+ 进行 XOR（按比特异或）生成 b 比特的分组 S_i。

（3）将 M 追加在 S_i 上。

（4）将 H 应用于步骤（3）所产生的数据流。

（5）opad 与 K^+ 进行 XOR 生成 b 比特的分组 S_o。

（6）将步骤（4）产生的散列结果追加在 S_o 上。

（7）将 H 应用于步骤 6 产生的数据流，输出结果。

注意，与 ipad 进行异或将导致 K 一半的比特翻转。类似地，与 opad 进行异或也导致 K 一半的比特翻转，但翻转的比特却不同。实际上，用散列算法处理 S_i 和 S_o，已经从 K 伪随机地生成了两个密钥。

HMAC 的执行时间应该与嵌入式散列函数处理长消息所用的时间近似相等。HMAC 多执行了三次基本的散列函数(S_i、S_o 和内部散列生成的数据块)。

3.3.2 基于分组密码的 MAC

在这一章中将讨论几种基于分组密码的 MAC。

基于密文的消息认证码（CMAC）。基于密文的消息认证码的操作模式适用于 AES 和 3DES，它在美国国家标准及技术研究所特刊（*NIST Special Publication*）800-38B 中被明确规定了。

首先，当这个信息为一个长度为 b 的密码块的整数倍数 n 时，让我们考虑一个基于密文的消息认证码的操作方式。在 AES 中，b=128，在 3DES 中，b=64，并且这个信息被分为 n 块（$M_1, M_2, \cdots, M_n$）。这个运算方式利用了一个 k 比特的加密密钥 K 和 n 比特的密钥，K_1。在 AES 中，密钥大小 k 为 128、192 或者 256 比特；在 3DES 中，密钥大小为 112 或者 168 比特。基于密文的消息认证码按照下面的步骤被计算出（见图 3.7）。

$$C_1=E\ (K,\ M_1)$$
$$C_2=E\ (K,\ [M_2 \oplus C_1])$$
$$C_3=E\ (K,\ [M_3 \oplus C_2])$$
$$\vdots$$
$$C_n=E\ (K,[M_n \oplus C_n-1 \oplus K_1])$$
$$T=\mathrm{MSB}_{Tlen}\ (C_n)$$

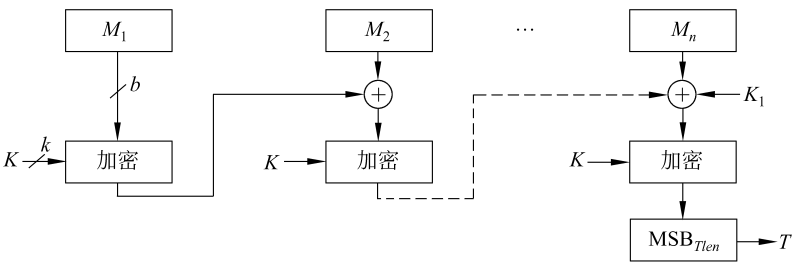
其中：

T =信息认证码，或者称为标签；

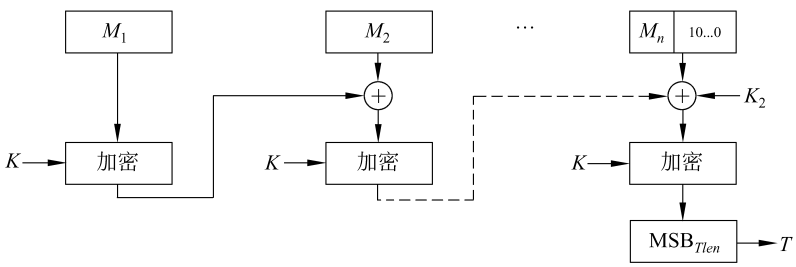
Tlen =T 的位长度；

MSBs(X)=位串 X 的最左边的 s 比特二进制数。

如果这个信息不是一个密码块长度的整数倍数，那么将在最后一个块（最低有效位）的右边填充由一位 1 和若干位 0 组成的位串，以使得最后一个块的长度也为 b 比特。CMAC 操作和以前一样，只是它使用一个不同的 n 比特密钥 K_2 取代 K_1。



(a) 信息长度是块大小的整数倍



(b) 信息长度不是块大小的整数倍

图 3.7　基于密文的消息认证码（CMAC）

为了生成两个 n 比特的密钥，分组密码应用于一个全部由 0 组成的块。通过对密文块左移一比特并且根据块大小异或一个常数，可以获得一个子密钥。第二个子密钥的产生方式与第一个子密钥的产生方式相同。

具有密码块链式信息认证码的计数器。CCM 的操作模式在 NIST SP 800-38C 中定义，又被称为认证加密模式。认证加密是一个被用来描述加密系统的专业术语，这个加密系统同时确保了信息的机密性和可靠性（完整性）。许多应用和协议需要这两种形式的安全，但是直到近期仍然单独设计这两种服务。

CCM 的核心算法是 AES 加密算法（见 2.2 节），CTR 操作模式（见 2.5 节）和 CMAC 认证算法。一个单独的密钥 K 同时被加密和 MAC 算法使用。CCM 加密数据处理的输入端

包含了 3 个基本部分。

（1）能被认证和加密的数据。这是数据块的明文 P。

（2）能被认证但是不能被加密的关联数据 A。例如，为了确保协议操作正确，一个协议头只能未加密进行传输，但是需要被认证。

（3）指定了负荷量和相关联的数据的随机数 N。这个特殊的值在每个例子中都不同，并且被用来阻止重放攻击和其他类型的攻击。

图 3.8 说明了 CCM 的操作过程。在认证方面，输入包括了随机数、相关联的数据和明文。这个输入数据按照 B_0 到 B_r 的序列格式。第一个块包含了随机数以及格式化的比特，这些比特给出了 N、A 和 P 元素的长度。第一块后面紧跟着 0 块或者更多的含有 A 的块，之后是 0 块或者更多的含有 P 的块。这些块的序列作为 CMAC 算法的输入，能产生一个长度为 *Tlen* 的 MAC 值，*Tlen* 小于或者等于块的长度（如图 3.8（a）所示）。

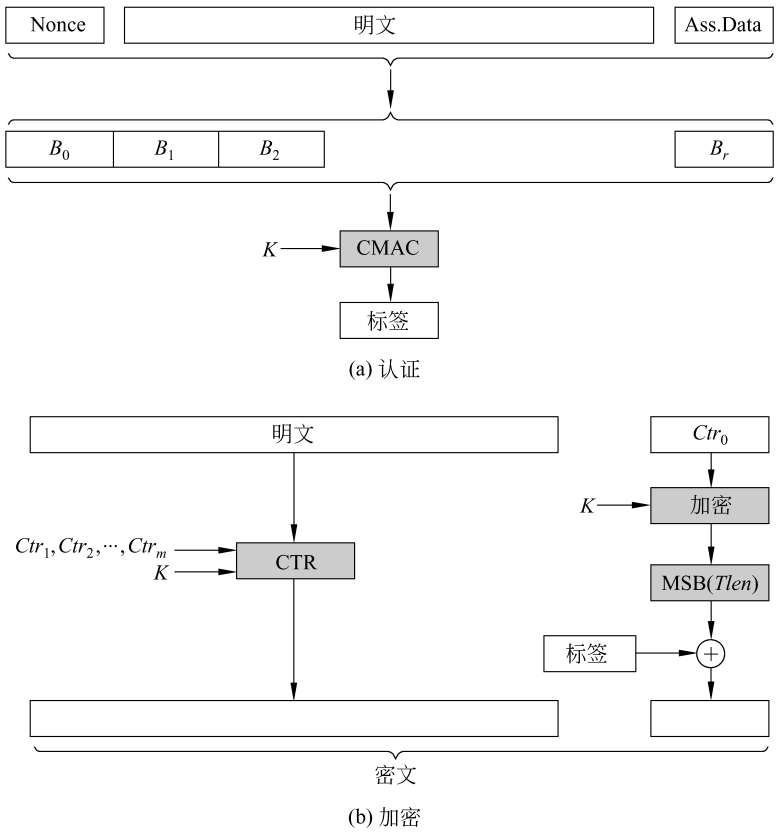


图 3.8　具有密码块链式信息认证码的计数器（CCM）

在加密中，计数器的值必须独立于随机数而产生。认证标签使用只有一个计数器 Ctr_0 的 CTR 模式加密。输出的 *Tlen* 位最高有效位跟认证标签异或，从而产生加密标签。剩下的计数器被用来在 CTR 模式中加密明文（如图 2.11 所示）。加密的明文和加密标签连接起来以获得最后的密文输出。

3.4　公钥密码原理

公钥密码与传统密码同等重要，它还可以用来进行消息认证和密钥分发。本节首先考虑公钥密码的基本概念，然后初步研究密钥的分发问题。3.5 节介绍两种最重要的公钥算法：RSA 和 Diffie-Hellman。3.6 节介绍数字签名。

3.4.1　公钥密码思想

Diffie 和 Hellman 在 1976 年首次公开提出了公钥密码思想[DIFF76]，这是有文字记载的几千年来密码领域第一次真正革命性的进步。公钥算法基于数学函数，而不像对称加密算法那样是基于比特模式的简单操作。更为重要的是公钥密码系统是非对称的，它使用两个单独的密钥。与此相比，对称的传统密码只使用一个密钥。使用两个密钥对于保密性、密钥分发和认证都产生了意义深远的影响。

在继续进行前，首先看看关于公钥密码的几种常见误解。第一种误解是，公钥密码比传统密码更抗密码分析。实际上，任何加密方案的安全性取决于：（1）密钥长度；（2）攻破密码所需的计算量。从抗密码分析的角度来说，原则上不能说传统密码优于公钥密码，也不能说公钥密码优于传统密码。第二种误解认为公钥密码是一种通用技术，传统密码则过时了。相反，由于当前公钥密码方案的计算开销太大，传统密码被淘汰似乎不太可能。最后一种误解认为，传统密码中与密钥分配中心的会话是很麻烦的事情，而用公钥密码实现的密钥分配非常简单。事实上，公钥密码也需要某种形式的协议，该协议包含一个中心代理。所以与传统密码相比，公钥密码所需的操作并不简单或者高效。

公钥密码方案由 6 个部分组成（见图 3.9（a））：

- **明文**：算法的输入，它是可读的消息或数据。
- **加密算法**：加密算法对明文进行各种形式的变换。
- **公钥和私钥**：算法的输入，这对密钥如果一个密钥用于加密，则另一个密钥就用于解密。加密算法所执行的具体变换取决于输入端提供的公钥或私钥。
- **密文**：算法的输出，取决于明文和密钥。对于给定的消息，两个不同的密钥将产生两个不同的密文。
- **解密算法**：该算法接收密文和匹配的密钥，生成原始的明文。

顾名思义，密钥对中的公钥是公开供其他人使用的，而只有自己知道私钥。通常的公钥密码算法根据一个密钥进行加密，根据另一个不同但相关的密钥进行解密。

基本步骤如下：

（1）每个用户都生成一对密钥用来对消息进行加密和解密。

（2）每个用户把两个密钥中的一个放在公共寄存器或其他可访问的文件里，这个密钥便是公钥，另一个密钥自己保存。如图 3.7（a）所示，每个用户都收藏别人的公钥。

（3）如果 Bob 希望给 Alice 发送私人消息，则他用 Alice 的公钥加密消息。

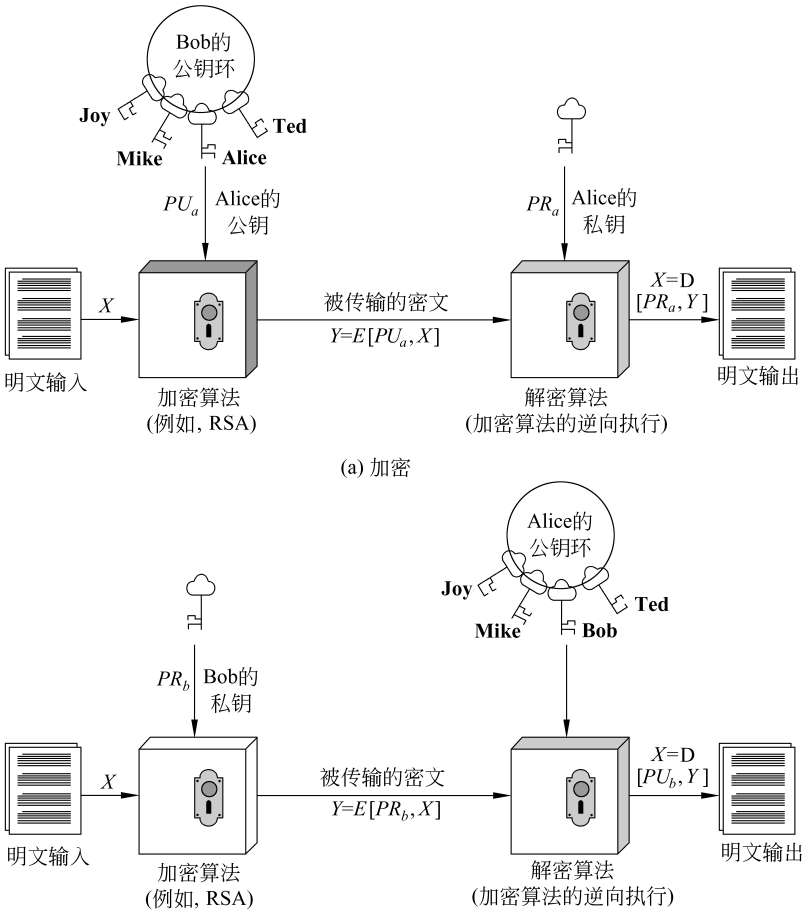


图 3.9　公钥密码

（4）当 Alice 收到这条消息，她用私钥进行解密。因为只有 Alice 知道她自己的私钥，其他收到消息的人无法解密消息。

用这种方法，任何参与者都可以获得公钥。由于私钥由每一个参与者在本地产生，故不需要分发。只要能够保护好他或她的私钥，以后的通信就会安全。在任何时候，用户都能够改变私钥且发布相应的公钥代替旧公钥。

传统密码算法中使用的密钥被特别地称为**密钥**。用于公钥密码的两个密钥被称为**公钥**和**私钥**。私钥总是保密的，但仍然被称为私钥而不是密钥，这是为了避免与传统密码混淆。

3.4.2　公钥密码系统的应用

在进一步介绍公钥密码系统的应用前，需要对公钥密码系统有个清楚的理解，否则容易引起混淆。公钥系统的特征就是使用具有两个密钥的加密算法，其中一个密钥为私人所有，另一个密钥是公共可用的。根据应用，发送者要么使用发送者的私钥，要么使用接收者的公钥，或者两个都使用，从而实现某种类型的加密函数。在广义上可以把公钥密码系统分为如下三类。

- **加密/解密**：发送者用接收者的公钥加密消息。
- **数字签名**：发送者用自己的私钥“签名”消息。签名可以通过对整条消息加密或者对消息的一个小的数据块加密来产生，其中该小数据块是整条消息的函数。
- **密钥交换**：通信双方交换会话密钥。这可以使用几种不同的方法，且需要用到通信一方或双方的私钥。

有些算法可用于上述三种应用，而其他一些算法仅适用于这些应用中的一种或两种。表 3.2 列出了本章讨论的算法（RSA 和 Diffie-Hellman）所支持的应用。该表还包括了本章后面将会提到的数字签名标准（DSS）和椭圆曲线密码。

表 3.2　公钥密码系统的应用

算　法	加密/解密	数字签名	密钥交换
RSA	是	是	是
Diffie-Hellman	否	否	是
DSS	否	是	否
椭圆曲线	是	是	是

3.4.3　公钥密码的要求

图 3.9 所示的密码系统建立在两个相关联密钥的密码算法之上。Diffie 和 Hellman 假设了这个系统是存在的，但是并没有证明这种算法的存在性。然而，他们给出了这些算法必须满足的条件[DIFF76]：

（1）接收方 B 计算生成密钥对（公钥 PU_b、私钥 PR_b）是容易的。

（2）已知公钥和需要加密的消息 M 时，发送方 A 容易计算生成相应的密文：

$$C = E(PU_b, M)$$

（3）接收方 B 用私钥解密密文时，比较容易通过计算恢复原始消息：

$$M = D(PR_b, C) = D[PR_b, E(PU_b, M)]$$

（4）当攻击者已知公钥 PU_b 时，不可能通过计算推算出私钥 PR_b。

（5）攻击者在已知公钥 PU_b 和密文 C 的情况下，通过计算不可能恢复原始消息 M。

还可以加上第 6 个要求。尽管有用，但是这对于所有的公钥应用并不是必需的。

（6）两个相关密钥中的任何一个都可以用于加密，另一个密钥用于解密。

$$M = D[PU_b, E(PR_b, M)] = D[PR_b, E(PU_b, M)]$$

3.5　公钥密码算法

RSA 和 Diffie-Hellman 是使用最广泛的两种公钥算法。本节分析这两种算法，然后简要介绍另外两种算法[4]。

4　本节使用一些数论的基本概念。请参考附录 A。

3.5.1 RSA 公钥密码算法

最初的公钥方案是在 1977 年由 Ron Rivest、Adi Shamir 和 Len Adleman 在 MIT 提出的，并且于 1978 年首次发表[RIVE78]。RSA 方案从那时起便占据了绝对的统治地位，成为最广泛接受和实现的通用公钥加密方法。**RSA** 是分组密码，对于某个 n，它的明文和密文是 0～n–1 之间的整数。

对于某一明文块 M 和密文块 C，加密和解密有如下的形式：

$$C = M^e \bmod n$$

$$M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n$$

发送方和接收方都必须知道 n 和 e 的值，并且只有接收者知道 d 的值。RSA 公钥密码算法的公钥 $KU = \{e,n\}$，私钥 $KR = \{d,n\}$。为使该算法能够用于公钥加密，它必须满足下列要求：

（1）可以找到 e、d、n 的值，使得对所有的 $M<n$，$M^{ed} \bmod n = M$ 成立。

（2）对所有满足 $M<n$ 的值，计算 M^e 和 C^d 相对容易。

（3）给定 e 和 n，不可能推出 d。

前两个要求很容易得到满足。当 e 和 n 取很大的值时，第三个要求也能够得到满足。

图 3.10 总结了 RSA 算法。开始时选择两个素数 p 和 q，计算它们的积 n 作为加密和解密时的模。接着需要计算 n 的欧拉函数值 $\phi(n)$。$\phi(n)$ 表示小于 n 且与 n 互素的正整数的个数。然后选择与 $\phi(n)$ 互素的整数 e（即 e 和 $\phi(n)$ 的最大公约数为 1）。最后，计算 e 关于模 $\phi(n)$ 的乘法逆元 d。d 和 e 具有所期望的属性。

生 成 密 钥	
选择 p、q	p 和 q 都是素数，且 $p \neq q$
计算 $\phi(n) = (p-1)(q-1)$	
选择整数 e	$\gcd(\phi(n),e) = 1$；$1 < e < \phi(n)$
计算 d	$de \bmod \phi(n) = 1$
公钥	$KU = \{e, n\}$
私钥	$KR = \{d, n\}$
加　　密	
明文	$M<n$
密文	$C = M^e \pmod n$
解　　密	
密文	C
明文	$M = C^d \pmod n$

图 3.10　RSA 算法

假设用户 A 已经公布了他的公钥，且用户 B 希望给 A 发送消息 M。那么 B 计算 $C = M^e \pmod n$ 并且发送 C。当接收到密文时，用户 A 通过计算 $M = C^d \pmod n$ 解密密文。

图 3.11 显示了[SING99]中的一个例子。对于这个例子，按下列步骤生成密钥：

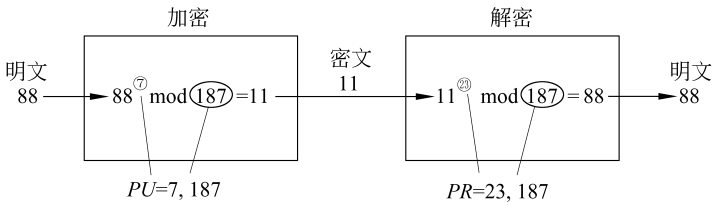


图 3.11　RSA 算法的例子

（1）选择两个素数：p=17 和 q=11。

（2）计算 $n = pq = 17 \times 11 = 187$ 。

（3）计算 $\phi(n) = (p-1)(q-1) = 16 \times 10 = 160$ 。

（4）选择 e，使得 e 与 $\phi(n)$=160 互素且小于 $\phi(n)$；选择 e=7。

（5）计算 d，使得 $de \bmod 160 = 1$ 且 d<160。正确的值是 d=23，这是因为 $23 \times 7 = 161 = 10 \times 16 + 1$ 。

这样就得到公钥 $PU = \{7, 187\}$，私钥 PR={23, 187}。下面的例子说明输入明文 $M = 88$ 时密钥的使用情况。

对于加密，需要计算 $C = 88^7 \bmod 187$ 。利用模运算的性质，计算如下：

$$88^7 \bmod 187 = [(88^4 \bmod 187) \times (88^2 \bmod 187) \times (88^1 \bmod 187)] \bmod 187$$

$$88^1 \bmod 187 = 88$$

$$88^2 \bmod 187 = 7744 \bmod 187 = 77$$

$$88^4 \bmod 187 = 59\ 969\ 536 \bmod 187 = 132$$

$$88^7 \bmod 187 = (88 \times 77 \times 132) \bmod 187 = 894\ 432 \bmod 187 = 11$$

对于解密，计算 $M = 11^{23} \bmod 187$：

$$11^{23} \bmod 187 = [(11^1 \bmod 187) \times (11^2 \bmod 187) \times (11^4 \bmod 187) \times (11^8 \bmod 187) \times (11^8 \bmod 187)] \bmod 187$$

$$11^1 \bmod 187 = 11$$

$$11^2 \bmod 187 = 121$$

$$11^4 \bmod 187 = 14\ 641 \bmod 187 = 55$$

$$11^8 \bmod 187 = 214\ 358\ 881 \bmod 187 = 33$$

$$11^{23} \bmod 187 = (11 \times 121 \times 55 \times 33 \times 33) \bmod 187 = 79\ 720\ 245 \bmod 187 = 88$$

有两种可能的方法可以用来攻击 RSA 算法。第一种方法是蛮力攻击：试遍所有可能的私钥。所以 e 和 d 的比特数越大，算法越安全。然而，因为密钥的生成和加密/解密所需的运算都比较复杂，所以密钥越大，系统运行得越慢。

关于分析破译 RSA 的大部分讨论都集中于如何分解 n 为两个素数。由于大数 n 具有很大的素因子，因式分解问题非常困难，但是它已经不如以前那么困难了。发生在 1977 年的著名事件恰好反映了这种情况。RSA 的三名创始人挑战“Scientific American”的读者，让读者去破译他们发表在 *Martin Gardner* 的“Mathematical Games”专栏中的密文[GARD77]。

每翻译出来一句明文，他们就提供 100 美元的奖励。他们预计在大约 4×10^{16} 年之内都不可能有人破译出明文。1994 年 4 月，Internet 上的一个工作组使用了 1600 多台计算机仅仅工作了 8 个月便获得了该奖项[LEUT94]。该挑战使用的公钥长度（n 的大小）为 129 比特的十进制数字或者大约 428 比特的二进制数字。这样的结果并没有使 RSA 失效；它只是意味着必须使用更大长度的密钥。目前 1024 比特（大约 300 比特的十进制数）的密钥对于几乎所有的应用都可以认为强度已经足够了。

3.5.2 Diffie-Hellman 密钥交换

第一个发表的公钥算法出现在 Diffie 和 Hellman 的原创性论文中，该算法定义了公钥密码学[DIFF76]，通常称该算法为 **Diffie-Hellman 密钥交换**。许多商业产品都采用了这种密钥交换技术。

Diffie-Hellman 算法的目的就是使得两个用户能够安全地交换密钥，供以后加密消息时使用。该算法本身局限于密钥交换。

Diffie-Hellman 算法的有效性是建立在计算离散对数是很困难这一基础之上。可以用如下方式简要地定义离散对数。首先，定义素数 p 的本原根，它是一个整数，且它的幂能够生成 1～p−1 的所有整数的数。即如果 a 是素数 p 的一个本原根，则下列各个数：

$$a \bmod p, a^2 \bmod p, \cdots, a^{p-1} \bmod p$$

各不相同，但它们组成了 1～p−1 之间整数的一个置换。

对于任意小于 p 的整数 b 和 p 的本原根 a，能够找到唯一的指数 i，使得

$$b = a^i \bmod p \quad \text{其中} \quad 0 \leqslant i \leqslant (p-1)$$

称指数 i 是 b 的基为 a 模为 p 的离散对数。把这个值记作 $d\log_{a,p}(b)$[5]。

算法。在这个背景下，定义 Diffie-Hellman 密钥交换，如图 3.12 所示。这个方案有两个公开的数值：素数 q 和 q 的本原根 α。假设用户 A 和 B 希望交换密钥。用户 A 选择一个随机整数 $X_A < q$，计算 $Y_A = \alpha^{X_A} \bmod q$。类似地，用户 B 独立地选择一个随机整数 $X_B < q$，计算 $Y_B = \alpha^{X_B} \bmod q$。双方都保持 X 值作为私有，向对方公开 Y 值。用户 A 计算密钥 $K = (Y_B)^{X_A} \bmod q$，用户 B 也计算密钥 $K = (Y_A)^{X_B} \bmod q$。这两个计算产生相同的结果：

$$\begin{aligned}
K &= (Y_B)^{X_A} \bmod q \\
&= (\alpha^{X_B} \bmod q)^{X_A} \bmod q \\
&= (\alpha^{X_B})^{X_A} \bmod q \\
&= \alpha^{X_B X_A} \bmod q \\
&= (\alpha^{X_A})^{X_B} \bmod q \\
&= (\alpha^{X_A} \bmod q)^{X_B} \bmod q \\
&= (Y_A)^{X_B} \bmod q
\end{aligned}$$

这样，双方都交换了密钥。此外，因为 X_A 和 X_B 为私有，所以攻击者只能利用如下元

5 许多文章称离散对数为指数。这个概念还没有得到多数人的赞同，距离统一命名还很遥远。