第 5 章

散列函数、消息摘要和数字签名

本章学习目标

- 掌握散列函数的相关知识
- 掌握消息摘要的相关知识
- 掌握数字签名的相关知识
- 掌握 PGP 安全电子邮件的配置过程

5.1 散列函数及消息摘要

5.1.1 散列函数

散列函数又称哈希函数(Hash)或杂凑函数,通常用字母 H 来表示。它将可变的输入长度串 M(可以足够长)转换成固定长度输出值 h(又称散列值(Hash value))的一种函数,这种转换是一种压缩映射,是将任意长度的信息压缩到某一固定长度的消息摘要的函数。即一个从明文到密文的不可逆映射,该过程只有加密过程,没有解密过程,可表示为:

$$h = H(M)$$

其中,h 为输出固定长度的字符串;M 为输入可变长的字符串;H 为散列函数。h 被称为 M 的散列值。

为了防止传输和存储的信息被有意或无意地篡改,采用散列函数对信息进行运算,生成信息摘要,将生成的摘要附在信息之后发出或与信息一起存储,可有效保证数据的完整性。

提取数据特征的算法称为单向散列函数,单向散列函数就是一种采集信息“指纹”的技术,单向散列函数所生成的散列值就相当于信息的“指纹”。

单向散列函数可以根据消息的内容计算出散列值,而散列值就可以被用来检查消息的完整性,消息可以是文字,也可以是图像文件或声音文件。单向散列函数不需要知道消息实际代表的含义。对于任何信息,单向散列函数都会将它作为单纯的比特序列来处理,即根据比特序列计算出散列值。

散列值的长度和信息的长度无关,无论信息是 1b,还是 100MB,甚至是 100GB,单向散列函数都会计算出固定长度(一般为 128 位或 160 位)的散列值。

单向散列函数具有以下几个性质。

(1) 确定性。散列函数能处理任意大小的信息,生成的消息摘要数据块的长度总是

具有固定的大小，对同一个源数据，反复执行该函数得到的输出一定相同；对于通过同一散列函数得出的两个不同的散列值，其原始输入也一定是不相同的。

(2) 易计算。对给定的信息，很容易计算出消息摘要，计算散列值所花费的时间短。

(3) 单向性。给定消息摘要和公开的散列函数算法，要推导出信息是极其困难的，也称不可逆性。

(4) 一一对应。不同消息的散列值是不相同的，要想伪造另一个信息，使它的消息摘要和原信息的消息摘要一样，是极其困难的。

散列函数的输出值有固定的长度，任何一位或多位的变化都将导致散列值的变化。通过散列值不可能推导出消息 M，也很难通过伪造消息 M 来生成相同的散列值。

对散列函数有两种穷举攻击。一是给定消息的散列函数 $H(x)$，破译者逐个生成其他消息 y，以使 $H(y)=H(x)$。二是攻击者寻找两个随机的消息 x、y，并使 $H(x)=H(y)$。这就是所谓的冲突攻击。

单向散列函数通常用于提供消息或文件的指纹。与人类的指纹类似，由于散列指纹是唯一的，因而能够提供对消息的完整性认证。

5.1.2　消息摘要

消息摘要(message digest)又称数字摘要(digital digest)、报文摘要或数字指纹，对要发送的信息进行单向 Hash 变换运算，得到的固定长度的密文(也称为提取信息的特征)就是消息摘要。

发送方向接收方发送消息并验证信息完整性的基本过程是：发送方先提取发送信息的消息摘要，并在传输信息时将消息摘要加入文件一同发送给接收方；接收方收到文件后，用相同的方法对接收到的信息进行变换运算得到另一个摘要；然后将自己运算得到的摘要与发送过来的摘要进行比较，从而验证数据的完整性。

不同的明文消息经过相同的哈希函数变换后得到的消息摘要是不同的，而同样的明文信息通过相同的哈希函数变换后得到的消息摘要或称数字指纹一定是一致的。

哈希函数的抗冲突性使得如果明文信息稍微有一点点变化，哪怕只是更改一个字母或者一个标点符号，通过哈希函数作用后，得到的信息摘要将会是天壤之别。哈希函数的单向性，使得要找到哈希值相同的两个不同的输入消息在计算上是不可行的。因此数据的哈希值，即消息摘要，可以检验数据的完整性。

下面介绍常见的两种消息摘要算法即散列函数，它们是 MD5 以及 SHA-1。

5.1.3　常见的消息摘要算法(即散列函数)

1. MD5

MD5(message digest algorithm 5)是一种广泛使用的散列函数，是一种消息摘要算法，该算法可以产生 128 位(16B)的散列值(Hash value)，用于确保信息传输的完整性。MD5 算法是美国密码学家李维斯特设计的，用以取代 MD4 算法。1996 年后，该算法被证实存在弱点，无法防止碰撞。

MD5 以 512 位分组来处理输入的信息，且每一分组又被划分为 16 个 32 位子分组，

接着对它们进行一系列处理。该算法的输出由 4 个 32 位分组组成，将这 4 个 32 位分组级联后生成一个 128 位散列值。MD5 的加密过程如图 5.1 所示。

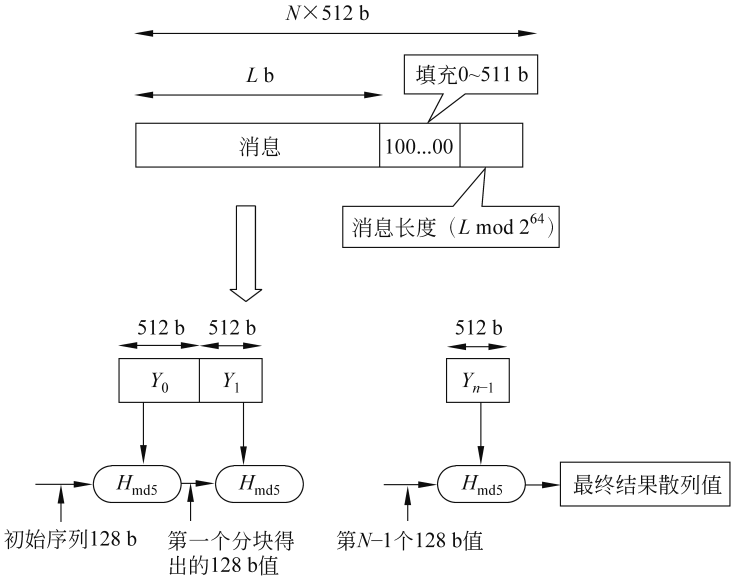

图 5.1　MD5 的加密过程

具体过程说明如下。

第一步，需要对信息进行填充，存在下面两种情况。

(1) 通过在信息的后面填充一个 1 和无数个 0，使填充后的信息的位长对 512 求余的结果等于 448，因此信息的位长将被扩展至 $N\times512+448$。

(2) 在结果的后面附加没有填充时的 64 位二进制信息，如果二进制表示的填充前的信息长度超过 64 位，则取低 64 位。

经过这两步的处理，信息的位长 $=N\times512+448+64=(N+1)\times512$，即长度恰好是 512 的整数倍。这样操作的原因是为了满足接下来处理中对信息长度的要求。

第二步，初始化 MD5 参数(参数值一般不变)。

MD5 中有 4 个 32 位被称作链接变量(chaining variable)的整数参数(A、B、C、D)，用来计算信息摘要，每个变量被初始化成低位字节在前的十六进制表示的数值，其分别为 A=0x01234567、B=0x89abcdef、C=0xfedcba98、D=0x76543210。在程序中要写成 A=0x67452301、B=0xefcdab89、C=0x98badcfe、D=0x10325476。当设置好这 4 个链接变量后，就开始进入算法的 4 轮循环运算，每轮循环的次数是信息中 512 位信息分组的数目。将上面 4 个变量复制到另外 4 个变量中：A 到 a，B 到 b，C 到 c，D 到 d。

第三步，定义 4 个 MD5 基本的按位操作函数，分别为 F、G、H、I，下列函数表达式中的 X、Y、Z 为 32 位整数。

$$F(X,Y,Z)=(X \text{ and } Y) \text{ or } (\text{not}(X) \text{and } Z)$$

$$G(X,Y,Z)=(X \text{ and } Z) \text{ or } (Y \text{ and not}(Z))$$

$$H(X,Y,Z)=X \text{ xor } Y \text{ xor } Z$$

$$I(X,Y,Z)=Y \text{ xor } (X \text{ or not}(Z))$$

再定义4个分别用于4轮变换的函数，设Mj表示消息的第j个子分组(从0到15)，<<<s表示循环左移s位，则4种操作如下。

FF(a,b,c,d,Mj,s,ti)表示 $a=b+((a+(F(b,c,d)+\text{Mj}+\text{ti}))<<<s)$；

GG(a,b,c,d,Mj,s,ti)表示 $a=b+((a+(G(b,c,d)+\text{Mj}+\text{ti}))<<<s)$；

HH(a,b,c,d,Mj,s,ti)表示 $a=b+((a+(H(b,c,d)+\text{Mj}+\text{ti}))<<<s)$；

II(a,b,c,d,Mj,s,ti)表示 $a=b+((a+(I(b,c,d)+\text{Mj}+\text{ti}))<<<s)$。

第四步，对输入数据作变换。

处理数据N是总的字节数，以64个字节为一组，每组作一次循环，每次循环进行4轮操作，要变换的64个字节用16个32位的整数数组[0..15]表示，而数组T[1..64]表示一组常数，T[i]为4294968296×abs(sin(i))的32位整数部分，其中4294968296为2^{32}，i的单位是弧度，i的取值是1～64。

这4轮(共64步)具体步骤如下。

第一轮：

FF(a, b, c, d, M0, 7, 0xd76aa478)

FF(d, a, b, c, M1, 12, 0xe8c7b756)

FF(c, d, a, b, M2, 17, 0x242070db)

FF(b, c, d, a, M3, 22, 0xc1bdceee)

FF(a, b, c, d, M4, 7, 0xf57c0faf)

FF(d, a, b, c, M5, 12, 0x4787c62a)

FF(c, d, a, b, M6, 17, 0xa8304613)

FF(b, c, d, a, M7, 22, 0xfd469501)

FF(a, b, c, d, M8, 7, 0x698098d8)

FF(d, a, b, c, M9, 12, 0x8b44f7af)

FF(c, d, a, b, M10, 17, 0xffff5bb1)

FF(b, c, d, a, M11, 22, 0x895cd7be)

FF(a, b, c, d, M12, 7, 0x6b901122)

FF(d, a, b, c, M13, 12, 0xfd987193)

FF(c, d, a, b, M14, 17, 0xa679438e)

FF(b, c, d, a, M15, 22, 0x49b40821)

第二轮：

GG(a, b, c, d, M1, 5, 0xf61e2562)

GG(d, a, b, c, M6, 9, 0xc040b340)

GG(c, d, a, b, M11, 14, 0x265e5a51)

GG(b, c, d, a, M0, 20, 0xe9b6c7aa)

GG(a, b, c, d, M5, 5, 0xd62f105d)

GG(d, a, b, c, M10, 9, 0x02441453)

GG(c, d, a, b, M15, 14, 0xd8a1e681)

GG(*b*, *c*, *d*, *a*, M4, 20, 0xe7d3fbc8)
GG(*a*, *b*, *c*, *d*, M9, 5, 0x21e1cde6)
GG(*d*, *a*, *b*, *c*, M14, 9, 0xc33707d6)
GG(*c*, *d*, *a*, *b*, M3, 14, 0xf4d50d87)
GG(*b*, *c*, *d*, *a*, M8, 20, 0x455a14ed)
GG(*a*, *b*, *c*, *d*, M13, 5, 0xa9e3e905)
GG(*d*, *a*, *b*, *c*, M2, 9, 0xfcefa3f8)
GG(*c*, *d*, *a*, *b*, M7, 14, 0x676f02d9)
GG(*b*, *c*, *d*, *a*, M12, 20, 0x8d2a4c8a)

第三轮：

HH(*a*, *b*, *c*, *d*, M5, 4, 0xfffa3942)
HH(*d*, *a*, *b*, *c*, M8, 11, 0x8771f681)
HH(*c*, *d*, *a*, *b*, M11, 16, 0x6d9d6122)
HH(*b*, *c*, *d*, *a*, M14, 23, 0xfde5380c)
HH(*a*, *b*, *c*, *d*, M1, 4, 0xa4beea44)
HH(*d*, *a*, *b*, *c*, M4, 11, 0x4bdecfa9)
HH(*c*, *d*, *a*, *b*, M7, 16, 0xf6bb4b60)
HH(*b*, *c*, *d*, *a*, M10, 23, 0xbebfbc70)
HH(*a*, *b*, *c*, *d*, M13, 4, 0x289b7ec6)
HH(*d*, *a*, *b*, *c*, M0, 11, 0xeaa127fa)
HH(*c*, *d*, *a*, *b*, M3, 16, 0xd4ef3085)
HH(*b*, *c*, *d*, *a*, M6, 23, 0x04881d05)
HH(*a*, *b*, *c*, *d*, M9, 4, 0xd9d4d039)
HH(*d*, *a*, *b*, *c*, M12, 11, 0xe6db99e5)
HH(*c*, *d*, *a*, *b*, M15, 16, 0x1fa27cf8)
HH(*b*, *c*, *d*, *a*, M2, 23, 0xc4ac5665)

第四轮：

II(*a*, *b*, *c*, *d*, M0, 6, 0xf4292244)
II(*d*, *a*, *b*, *c*, M7, 10, 0x432aff97)
II(*c*, *d*, *a*, *b*, M14, 15, 0xab9423a7)
II(*b*, *c*, *d*, *a*, M5, 21, 0xfc93a039)
II(*a*, *b*, *c*, *d*, M12, 6, 0x655b59c3)
II(*d*, *a*, *b*, *c*, M3, 10, 0x8f0ccc92)
II(*c*, *d*, *a*, *b*, M10, 15, 0xffeff47d)
II(*b*, *c*, *d*, *a*, M1, 21, 0x85845dd1)
II(*a*, *b*, *c*, *d*, M8, 6, 0x6fa87e4f)
II(*d*, *a*, *b*, *c*, M15, 10, 0xfe2ce6e0)
II(*c*, *d*, *a*, *b*, M6, 15, 0xa3014314)

II(b，c，d，a，M13，21，0x4e0811a1)

II(a，b，c，d，M4，6，0xf7537e82)

II(d，a，b，c，M11，10，0xbd3af235)

II(c，d，a，b，M2，15，0x2ad7d2bb)

II(b，c，d，a，M9，21，0xeb86d391)

所有这些完成之后，将 a、b、c、d 分别在原来基础上再加上 A、B、C、D。即 $A=a+A$，$B=b+B$，$C=c+C$，$D=d+D$，然后用下一分组数据继续运行以上算法。

第五步，输出结果。

最后输出的是 A、B、C、D 的级联，就是输出的结果，A 是低位，D 为高位，$DCBA$ 组成 128 位输出结果。

随着计算机能力的不断提升以及 MD5 的弱点被不断发现，通过碰撞的方法有可能构造出两个具有相同 MD5 值的信息，从实践的角度看，不同信息具有相同 MD5 的可能性很低。

2. SHA-1

安全散列算法(secure hash algorithm，SHA)是一种密码散列函数，由美国国家安全局设计，并由美国国家标准与技术研究院发布。SHA 家族有五个算法，分别是 SHA-1、SHA-224、SHA-256、SHA-384、SHA-512，后四者有时并称为 SHA-2。SHA-1 在许多安全协议中被广泛使用，包括 TSL、SSL、PGP、SSH、S/MIME 以及 IPSec，曾被视为 MD5 的后继者。在安全性方面，SHA-1 的安全性遭受严重挑战。目前虽然没有出现对 SHA-2 的有效攻击，但是由于它的算法和 SHA-1 基本上相似，因此科学家在积极寻找其他可以替代的杂凑算法。

SHA-1 可将一个最大 2^{64} b 的消息转换成一串 160 位的消息摘要；其设计原理类似于李维斯特所设计的密码学散列算法 MD4 和 MD5。与 MD5 相似，SHA 算法也是首先填充消息数据，使其长度为 512 位的倍数。不同的是，其杂凑运算总共进行 80 轮迭代运算，每轮运算都要改变 5 个 32 位寄存器的内容，且 SHA-1 产生的是一个 160 位(20B)的杂凑值，也称散列值，该值通常是以 40 个十六进制数的形式呈现。

2005 年，密码分析人员发现对 SHA-1 的有效攻击方法，表明该算法已经不够安全，不能继续使用，2005 年 2 月，王小云、殷益群以及于红波发表了对完整版 SHA-1 的攻击方法，只需少于 2^{69} 的计算复杂度就能找到一组碰撞。

2005 年 8 月 17 日的 CRYPTO 会议接近尾声中，王小云、姚期智、姚储枫再度发表更有效率的 SHA-1 攻击方法，能在 2^{63} 的计算复杂度内找到碰撞。

在 2006 年的 CRYPTO 会议上，Christian Rechberger 和 Christophe De Cannière 宣布他们能在容许攻击者决定部分原信息的条件之下找到 SHA-1 的一个碰撞。

自 2010 年以来，许多组织建议使用 SHA-2 或 SHA-3 来代替 SHA-1。2017 年 2 月 23 日，荷兰国家数学和计算机科学研究中心与谷歌宣布了一个成功的 SHA-1 碰撞攻击，发布了两份内容不同但 SHA-1 散列值相同的 PDF 文件作为概念证明。

5.2 数字签名

数字签名是现代密码学的一个重要发明，其初衷就是实现在网络环境中对手工签名和印章的模拟。在实际应用中，数字签名比手工签名具有更多的优势，因此在电子政务、电子银行、电子证券等领域有着重要的应用。在实际的通信过程中，往往出现下面的状况。

用户 A 与用户 B 之间要进行通信，双方拥有共享的会话密钥 K，在通信过程中可能会遇到如下问题。

(1) A 伪造一条消息，并称该消息来自 B。A 只需要产生一条伪造的消息，用 A 和 B 的共享密钥通过哈希函数产生认证码，并将认证码附于消息之后。由于哈希函数的单向性和密钥 K 是共享的，因此无法证明该消息是 A 伪造的。

(2) B 可以否认曾经发送过某条消息。因为任何人都有办法伪造消息，所以无法证明 B 是否发送过该消息。

哈希函数可以进行报文鉴别，但无法阻止通信用户的欺骗和抵赖行为。当通信双方不能互相信任，需要用除了报文鉴别技术以外的其他方法来防止类似的抵赖和欺骗行为。

数字签名可以避免以上情况的出现。数字签名能够在数据通信过程中识别通信双方的真实身份，保证通信的真实性以及不可抵赖性，起到与手写签名或者盖章的同等作用。1999 年，美国参议院已通过立法，规定数字签名与手写签名的文件、邮件在美国具有同等的法律效力。

5.2.1 数字签名的概念

数字签名在 ISO 7498-2 标准中定义为：附加在数据单元上的一些数据，或是对数据单元所做的密码变换，这种数据和变换允许数据单元的接收者确认数据单元来源和数据单元的完整性，并保护数据，以防止数据被人(例如接收者)伪造。

数字签名标准(digital signature standard，DSS)FIPS186-2 对数字签名作了如下解释：利用一套规则和一个参数对数据计算得到结果，用此结果能够确认签名者的身份和数据的完整性。

联合国国际贸易法委员会《电子签名示范法》定义数字签名为："在数据电文中以电子形式所含、所附或在逻辑上与数据电文有联系的数据，它可用于鉴别与数据电文相关的签名人和表明签名人认可数据电文所含信息。"

数字签名所要解决的主要问题是验证发送方的身份，以免发送方在信息发送完以后产生类似抵赖的行为。做法是要求发送方拿私钥来生成一个签名块，接收方只能拿发送方的公钥来验证。因为私钥只有发送方持有，别人是无法伪造的。如果直接拿私钥对原文进行加密，其加密速度是相当慢的，所以在这个过程中引入了具有完整性验证功能的 Hash 函数。首先对原文进行 Hash 运算，获得一个摘要，最后使用非对称加密算法直接对摘要加密，生成一个签名块，将签名块附在消息的后面，来确认信息的来源和数据信息的完整性，并保护数据，防止伪造。当通信双方发生争议时，仲裁机构就能够根据信息上

的数字签名来进行正确的裁定，从而实现防抵赖的安全服务，过程如图 5.2 所示。

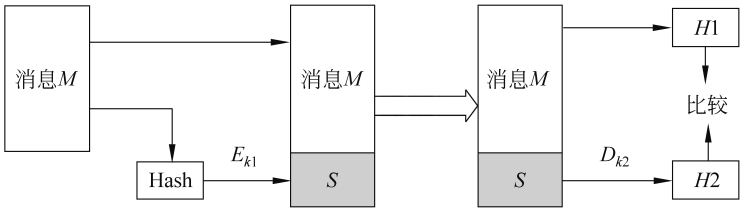

图 5.2　数字签名过程

数字签名的具体过程描述如下。

（1）信息发送者采用散列函数对消息生成数字摘要。

（2）将生成的数字摘要用发送者的私钥进行加密，生成数字签名。

（3）将数字签名与原消息结合在一起，发送给信息接收者。

（4）信息的接收者接收到信息后，将消息与数字签名分离开来，发送者的公钥解密签名得到数字摘要，同时对原消息经过相同的散列算法生成新的数字摘要。

（5）最后比较两个数字摘要，如果相等，则证明消息没有被篡改。如果不相等，则证明消息被篡改。同时也能证明信息来源的可靠性。

数字签名具有以下特征。

（1）签名是可信的：因为消息的接收者是用消息的发送者的公钥解开加密文件的，这说明原文件只能被消息发送者的私钥加密，然而只有消息的发送者才知道自己的私钥。

（2）无法被伪造：只有消息的发送者知道自己的私钥。因此只有消息的发送者能用自己的私钥加密一个文件，因此加密文件无法被伪造。

（3）无法重复使用：签名在这里就是一个加密过程，无法重复使用。

（4）文件被签名以后无法被篡改：因为加密后的文件被改动后是无法被消息发送者的公钥解开的。

（5）签名具有不可否认性：因为除消息发送者以外，无人能用消息发送者的私钥加密一个文件。

5.2.2　数字签名的类型

常见的数字签名的类型包括：使用对称加密和仲裁者实现数字签名、使用公钥体制实现数字签名以及使用公钥体制和单向散列函数实现数字签名 3 种。

1. 使用对称加密和仲裁者实现数字签名

用户 A 与 B 要进行通信，每个从 A 发往 B 的签名报文首先发送给仲裁者 C，C 检验该报文及其签名的出处和内容，然后对报文注明日期，同时指明该报文已通过仲裁者的检验。仲裁者的引入解决了直接签名方案中所面临的问题以及发送方的否认行为。A 与 B 进行通信时，A 要对自己发送给 B 的文件进行数字签名，以向 B 证明是自己发送的，并防止其他人伪造。利用对称加密系统和一个双方都信赖的第三方（仲裁者）可以实现。假设 A 与仲裁者共享一个秘密密钥 K_{AC}，B 与仲裁者共享一个秘密密钥 K_{BC}，实现的过程如下。

(1) A 用 K_{AC} 加密准备发给 B 的消息 M，并将之发给仲裁者。

(2) 仲裁者用 K_{AC} 解密消息。

(3) 仲裁者把这个解密的消息及自己的证明 S(证明消息来源于 A)用 K_{BC} 加密。

(4) 仲裁者把加密的信息发给 B。

(5) B 用与仲裁者共享的秘密密钥 K_{BC} 解密接收到的消息，看到来自 A 的消息和仲裁者的证明 S。

2. 使用公钥体制实现数字签名

公钥体制的发明使得数字签名变得更加简单，它不再需要第三方去签名和验证。签名的实现过程如下。

(1) A 用他的私钥加密消息，从而对文件进行签名。

(2) A 将签名的消息发给 B。

(3) B 用 A 的公钥解密消息，从而验证签名。

由于 A 的私人密钥只有他一个人知道，因而用私人密钥加密形成签名，别人无法伪造；B 只有使用了 A 的公钥才能解密，即可以确信消息的来源为 A，且 A 无法否认自己的签名。

3. 使用公钥体制和单向散列函数实现数字签名

利用单向散列函数产生消息的指纹，用公钥算法对指纹加密，形成数字签名，过程如下。

(1) A 使消息 M 通过单向散列函数 H 产生散列值，即消息指纹或消息认证码。

(2) A 使用私钥加密散列值，形成数字签名。

(3) A 把消息和数字签名一起发给 B。

(4) B 收到消息和数字签名后，用 A 的公钥解密数字签名，再用同样的算法对消息产生散列值。

(5) B 将自己产生的散列值和解密的数字签名相比较，看是否匹配，从而验证信息的完整性，以及发送方的不可否认性。

整个过程如图 5.3 所示。

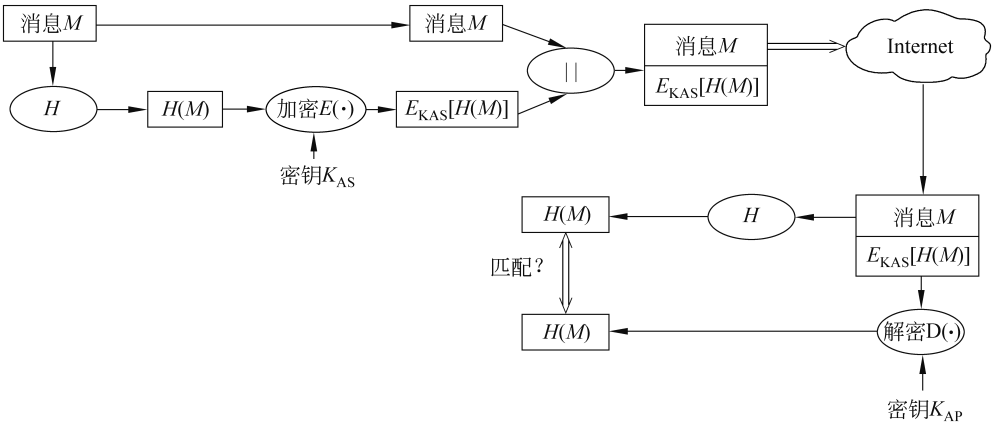

图 5.3 使用公钥体制和单向散列函数的数字签名方法

5.2.3　数字签名的算法

常见的数字签名算法有RSA数字签名算法、DSA数字签名算法以及椭圆曲线数字签名算法。

RSA是目前计算机密码学中最经典的算法，也是目前为止使用最广泛的数字签名算法，用RSA或其他公钥密码算法的最大方便是没有密钥分配问题，网络越复杂、网络用户越多，其优点越明显。公钥加密使用两个不同的密钥，其中一个是公开的，另一个是保密的。公钥可以保存在以下地方：①系统目录内；②未加密的电子邮件信息中；③电话黄页上；④公告牌中。网上的任何用户都可获得公开密钥。而私有密钥是用户专用的，由用户本人持有，它可以对由公开密钥加密的信息进行解密。

由于RSA数字签名算法存在计算时间长的弱点，因此实际对文件签名前，需要对消息进行Hash变换。

RSA数字签名算法主要可分为MD系列和SHA系列。MD系列主要包括MD2withRSA和MD5withRSA。SHA系列主要包括SHA1withRSA、SHA224withRSA、SHA384withRSA以及SHA512withRSA。

散列函数对发送数据的双方都是公开的，只有加入数字签名及验证，才能真正实现在公开网络上的安全传输。加入数字签名和验证的文件传输过程如下。

(1) 发送方首先用散列函数从原文得到数字指纹，然后采用公钥体系，用发送方的私有密钥对数字指纹进行加密，得到数字签名，并把数字签名附加在要发送的原文后面。

(2) 发送方选择对称加密算法，利用一个秘密密钥对文件进行加密，并把加密后的文件通过网络传输到接收方。(这样做是为了实现信息的保密传输，为什么不直接采用更加安全的公钥加密算法？是因为公钥加密算法速度慢。)

(3) 发送方用接收方的公开密钥对秘密密钥进行加密，并通过网络把加密后的秘密密钥传输到接收方。(利用公钥加密的安全性，实现对对称加密密钥的安全传输。)

(4) 接收方使用自己的私有密钥对密钥信息进行解密，得到秘密密钥的明文。

(5) 接收方用秘密密钥对文件进行解密，得到经过加密的信息的明文。

(6) 接收方用发送方的公开密钥对数字签名进行解密，得到数字指纹。

(7) 接收方用得到的明文和散列函数重新进行Hash运算，得到新的数字指纹，将新的数字指纹与解密后的数字指纹进行比较。如果两个Hash值相同，说明文件在传输过程中没有被破坏。

如果第三方冒充发送方发出了一个文件，因为接收方在对数字签名进行解密时使用的是发送方的公开密钥，只要第三方不知道发送方的私有密钥，解密出来的数字签名和经过计算的数字签名必然是不相同的。这就提供了一个安全地确认发送方身份的方法。

5.3　实验：PGP安全电子邮件

PGP(pretty good privacy)是一个基于RSA公钥体制的邮件或文件加密软件。其可对用户邮件或文件内容进行保密，以防止非法授权者阅读。它具有对用户的电子邮件或