第3章　信息认证技术

3.1　概　　述


视频讲解

在当前开放式的网络环境中，任何在网络上的通信都可能遭到黑客的攻击，窃听机密消息，伪造、复制、删除和修改消息等攻击越来越多。所有的攻击都可能对正常通信造成破坏性的影响。因此，一个真实可靠的通信环境成为能够有效进行网络通信的基本前提。作为信息安全中的一个重要组成部分，认证技术也显得尤为重要。

为了防止通信中的消息被非授权使用者攻击，有效的方法就是要对发送或接收到的消息具有鉴别能力，能鉴别消息的真伪和通信对方的真实身份。实现这样功能的过程称为认证。

一个安全的认证系统应满足以下条件。

(1) 合法的接收者能够检验所接收消息的合法性和真实性。

(2) 合法的发送方对所发送的消息无法进行否认。

(3) 除了合法的发送方之外，任何人都无法伪造、篡改消息。

通常情况下，一个完整的身份认证系统中除了有消息发送方和接收方外，还要有一个可信任的第三方，负责密钥分发、证书的颁发、管理某些机密信息等工作。当通信双方遇到争执、纠纷时，第三方还需要充当仲裁者的角色。

通信双方进行认证的目的是进行真实而安全的通信。所谓认证就是在通信过程中，通信一方验证另一方所声称的某种属性。信息安全中的认证技术主要有两种：消息认证和身份认证。

如果验证的是消息的某种属性，则该认证方式称为消息认证。消息认证用于保证信息的完整性与不可抵赖性，验证消息在传送和存储过程中是否遭到篡改、重放等攻击。若认证的属性是关于通信中的某一方或双方身份，则该认证过程称为身份认证。身份认证主要用于鉴别用户身份，是用户向对方出示自己身份的证明过程，通常是确认通信的对方是否拥有进入某个系统或使用系统中某项服务的合法权利的第一道关卡，并确认消息的发送者和接收者是否合法。

3.2　哈 希 函 数

哈希函数也称单向散列函数，是信息安全领域广泛使用的一种密码技术，它主要用于提供消息的完整性验证。哈希函数以任一长度的消息 M 为输入，产生固定长度的数据输出。这个定长输出称为消息 M 的散列值或消息摘要。由于哈希函数具有单向的特性，因此该散列值也称数据的“指纹”。

3.2.1　哈希函数概述

由于哈希(Hash)函数通过产生定长的散列值作为数据的特征“指纹”,因此用于消息认证的哈希函数必须具有以下性质。

(1) 哈希函数的输入可以是任意长度的数据块 M,产生固定长度的散列值 h。

(2) 给定消息 M,很容易计算散列值 h。

(3) 给定散列值 h,根据 $H(M)=h$ 推导出 M 很难,这条性质被称为单向性。单向性要求根据报文计算散列值很简单,但反过来根据散列值计算出原始报文十分困难。

(4) 已知消息 M,通过同一个 $H(\cdot)$计算出不同的 h 是很困难的。

(5) 给定消息 M,要找到另一消息 M'满足 $H(M)=H(M')$,这在计算上是不可行的,这条性质被称为弱抗碰撞性。该性质是保证无法找到一个替代报文,否则就可能破坏使用哈希函数进行封装或签名的各种协议的安全性。哈希函数的重要之处就是赋予 M 唯一的“指纹”。

(6) 对于任意两个不同的消息 $M\neq M'$,它们的散列值不可能相同,这条性质被称为强抗碰撞性。强抗碰撞性对于消息的哈希函数安全性要求更高,该性质保证了对生日攻击的防御能力。

碰撞性是指对两个不同的消息 M 和 M',如果它们的散列值相同,则发生了碰撞。实际中需要处理的消息是无限的,但可能的散列值却是有限的。不同的消息可能会产生同一散列值,因此碰撞是存在的。但是,哈希函数要求用户不能按既定的需要找到一个碰撞,意外的碰撞更是不太可能的。显然,从安全性的角度来看,哈希函数输出的位越长,抗碰撞的安全强度越大。

在信息认证技术中,哈希函数扮演着非常重要的角色,在通信安全中起着重要的作用,同时也是许多密码协议的基本模块。哈希函数的散列值也被称为哈希值、消息摘要、数字指纹、密码校验和、信息完整性检验码、操作检验码等。即使对明文进行轻微的改动,哪怕只是一个字母或一个标点符号,其对应的散列值也会有很大的不同。

哈希函数的设计建立在压缩函数的思想上。压缩函数的输入是消息分组和文本前一分组的输出,即分组 M_j 的散列值为:

$$h_j=f(M_j,h_{j-1})$$

该散列值和下一轮的消息分组一起作为压缩函数下一轮的输入。最后一个分组的散列值就成为整个消息的散列值。

目前常用的哈希函数有 MD5、SHA-1 和 RIPEMD-160 等,本章重点介绍 MD5 和 SHA-1 算法的基本原理。

3.2.2　MD5

MD5 算法是由麻省理工学院的 Ron Rivest 提出的。MD5 是一种常用的哈希函数,它可以将任意长度的消息经过变换得到一个 128 位的散列值,目前被广泛用于各种软件的密码认证和密钥识别上。

对 MD5 算法简要的叙述可以概括为: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,经过了一系列的处理后,算法的输出由 4 个 32 位分组组成,将这 4 个 32 位分组级联后将生成一个 128 位散列值。

MD5 是经 MD2、MD3 和 MD4 发展而来的,它的作用是让大容量信息在数字签名前

被“压缩”成一种保密的格式(将一个任意长度的字串变换成一定长的大整数)。无论是MD2、MD4,还是MD5,它们都需要获得一个随机长度的信息并产生一个128位的信息摘要。虽然这些算法的结构或多或少有些相似,但MD2的设计与MD4和MD5完全不同,这是因为MD2是为8位计算机做设计优化的,而MD4和MD5是面向32位计算机的。

MD5的算法步骤如下。

1) 数据填充与分组

数据填充与分组的具体步骤如下。

(1) 将输入信息M按顺序进行分组,每512位为一组,即$M=M_1,M_2,\cdots,M_{n-1},M_n$。

(2) 将M_n的长度填充为448位。当M_n的长度L小于448位时,在信息M_n后加一个“1”,然后再填充若干“0”,使得最后M_n的长度为448位。当M_n的长度大于448位时,在信息M_n后添加一个“1”,然后再填充$512-L+448$个“0”,使最后信息M_n的长度为512位,M_{n+1}的长度为448位。此时填充后的信息M的长度恰好是一个比512的倍数少64位的数,也就是信息长度等于$512\times(n-1)+448$,其中n为某个整数。

(3) 将填充前的信息M的长度L转换为64位二进制数,如果原信息长度L超过64位所能表示的范围,则只保留最后64位。

(4) 再将该64位二进制数增加到填充后的信息M的M_n后面,使得最后一块的长度为512位。

经过这些处理,现在信息的位长为$(n-1)\times512+448+64=n\times512$,即长度恰好是512的整数倍。这样做的原因是为了满足后面处理中对信息长度的要求。

2) 初始化散列值

MD5算法的中间结果和最终结果保存在128位的缓冲区中,缓冲区用4个32位的变量表示,这些变量被称为链接变量(chaining variable)。对链接变量进行初始化:

$A=$0x01234567

$B=$0x89abcdef

$C=$0xfedcba98

$D=$0x76543210

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。循环的总次数是信息中512位信息分组的数目。

3) 计算散列值

将上面4个链接变量复制到另外4个变量中,即A到a,B到b,C到c,D到d。

主循环有4轮(MD4只有3轮),每轮循环都很相似,主循环如图3.1所示。第一轮进行16次操作,每次操作对a、b、c和d中的3个做一次非线性函数运算,然后将所得结果加上第4个变量(文本的一个子分组和一个常数)。将函数运算后的所得结果循环左移某个位数,并加上a、b、c或d,以该结果取代a、b、c或d。

每轮运算使用的非线性函数都是一个基本的逻辑函数,其输入是3个32位的信息,输出是一个32位的信息,按位进行逻辑运算。4轮运算所使用的函数分别为:

$$F(X,Y,Z)=(X\wedge Y)\vee((\neg X)\wedge Z)$$

$$G(X,Y,Z)=(X\wedge Z)\vee(Y\wedge(\neg Z))$$

$$H(X,Y,Z)=X\oplus Y\oplus Z$$

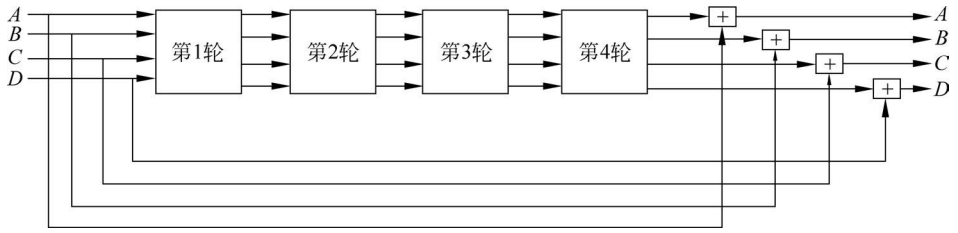

图 3.1　MD5 主循环

$$I(X,Y,Z)=Y\oplus(X\vee(\neg Z))$$

其中，∧是与，∨是或，¬是非，⊕是异或。如果 X、Y 和 Z 的对应位是独立且均匀的，那么这 4 个函数结果的每一位也应是独立且均匀的。

MD5 的基本操作过程如图 3.2 所示。

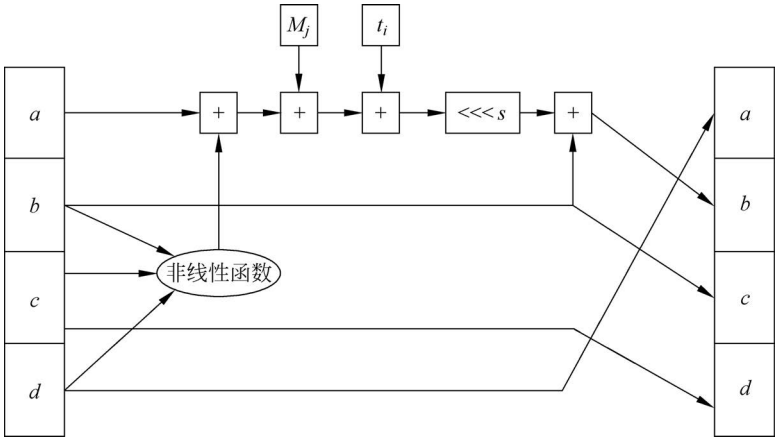

图 3.2　MD5 的基本操作过程

4 轮的迭代操作分别为：

FF(a,b,c,d,M_j,s,t_i) 表示 $a=b+((a+F(b,c,d)+M_j+t_i)\lll s)$

GG(a,b,c,d,M_j,s,t_i) 表示 $a=b+((a+G(b,c,d)+M_j+t_i)\lll s)$

HH(a,b,c,d,M_j,s,t_i) 表示 $a=b+((a+H(b,c,d)+M_j+t_i)\lll s)$

II(a,b,c,d,M_j,s,t_i) 表示 $a=b+((a+I(b,c,d)+M_j+t_i)\lll s)$

其中，M_j 表示消息的第 j 个子分组(从 0 到 15)；常数 $t_i=2^{32}\times\text{abs}\sin(i)$ 的整数部分；$i=1,2,\cdots,64$，i 的单位是弧度；＋为模 2^{32} 加法；$\lll s$ 表示循环左移 s 位。

4 轮迭代共 64 步如下。

第 1 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 2 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 3 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 4 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)
```

4 轮循环操作完成之后，将 A、B、C、D 分别加上 a、b、c、d，即

$$A = A + a$$
$$B = B + b$$
$$C = C + c$$
$$D = D + d$$

这里的加法是模 2^{32} 加法。然后用下一分组数据继续运行算法。

4）输出

对每个分组都作相应的处理后，最后的输出就是 A、B、C 和 D 的级联，即 A 作为低位，D 作为高位，共计 128 位输出。

MD5 被广泛用于加密和解密技术中，在很多操作系统（如 Linux 等）中，用户的密码是以 MD5 值的方式保存的。用户在登录验证时，系统首先把用户输入的密码计算成 MD5 值，然后再与系统中保存的密码 MD5 值进行比较。在该操作过程中，系统在不知道用户密码的情况下就可以确定用户登录系统的合法性。这不但可以避免用户的密码被具有系统管理员权限的用户知道，而且还在一定程度上增加了密码被破解的难度。

3.2.3　SHA-1

安全散列算法（Secure Hash Algorithm，SHA）是由美国国家标准与技术研究院提出的，并于 1993 年作为联邦信息处理标准（FIPS 180）发布，1995 年又发布了其修订版（FIPS 180-1），通常称为 SHA-1。SHA 算法也是建立在 MD4 算法之上的，其设计是在 MD4 的基础上改进而成的。

SHA-1 算法的输入是长度小于 2^{64} 位的消息，输出是 160 位的散列值，输入消息以 512 位的分组为单位进行处理。

与 MD5 算法相同，SHA 算法首先也需要对消息进行填充补位。补位是这样进行的：先添加一个 1，然后再添加若干 0，使得消息长度满足对 512 取模后余数是 448。以“abc”为例显示补位的过程。

原始信息:

01100001 01100010 01100011

补位第一步:

01100001 01100010 01100011 1

首先补一个"1",然后补423个"0"。

补位第二步:

01100001 01100010 01100011 10…0

可以将最后补位完成后的数据用十六进制写为:

61626380 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000

现在,数据的长度是448了,可以进行下一步操作。

填充完信息后,需要将原始信息的长度补到已经进行了补位操作的信息后面。通常用一个64位的数据来表示原始信息的长度。如果信息长度不大于2^{64},那么第一个字就是0。在进行了补长度的操作后,整个信息就变为(十六进制格式):

61626380 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000018

如果原始的信息长度超过了512,需要将它补成512的倍数。然后将整个信息分成几个512位的数据块,分别处理每一个数据块,从而得到散列值。

与MD5算法不同,SHA的中间结果和最终结果保存在160位的缓冲区中,缓冲区用5个32位的变量表示,这些变量初始化为:

$$A=0\text{x}67452301$$
$$B=0\text{xefcdab89}$$
$$C=0\text{x}98\text{badcfe}$$
$$D=0\text{x}10325476$$
$$E=0\text{xc}3\text{d}2\text{e}1\text{f}0$$

在进入主循环函数处理前,将上面5个变量复制到5个变量中:A到a,B到b,C到c,D到d,E到e。

当设置好这5个变量后,就开始进入每轮20步的4轮循环运算,循环的总次数是信息中512位信息分组的数目,主循环结构如图3.3所示。每一步操作都使用一个非线性的逻辑函数对a、b、c、d、e中的3个变量进行一次按位的逻辑运算。

这几个非线性函数定义为:

$$f_t(X,Y,Z)=\begin{cases}(X\wedge Y)\vee((\neg X)\wedge Z) & 0\leqslant t\leqslant 19\\ X\oplus Y\oplus Z & 20\leqslant t\leqslant 39\\ (X\wedge Y)\vee(X\wedge Z)\vee(Y\wedge Z) & 30\leqslant t\leqslant 59\\ X\oplus Y\oplus Z & 40\leqslant t\leqslant 79\end{cases}$$

其中,$\wedge$、$\vee$、$\neg$和$\oplus$分别是与、或、非和异或运算。

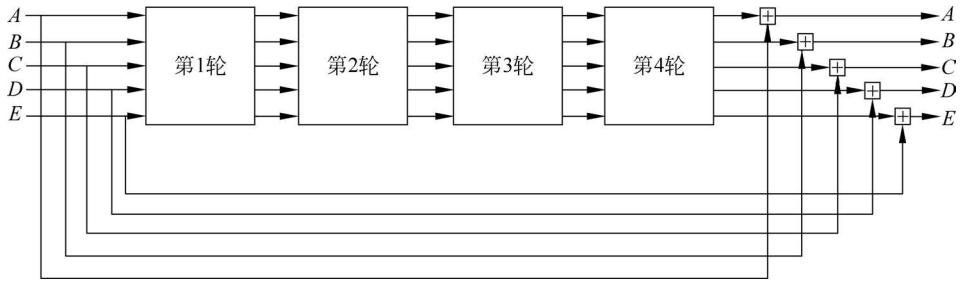

图 3.3　SHA-1 的主循环

与此同时，每一步操作也都使用一个加法常量 K_t。K_t 定义如下：

$$K_t=\begin{cases}0\text{x}5\text{a}827999 & 0\leqslant t\leqslant 19\\ 0\text{x}6\text{ed}9\text{eba}1 & 20\leqslant t\leqslant 39\\ 0\text{x}8\text{f}1\text{bbcdc} & 40\leqslant t\leqslant 59\\ 0\text{xca}62\text{c}1\text{d}6 & 60\leqslant t\leqslant 79\end{cases}$$

这些数的取值来自 $0\text{x}5\text{a}827999=2^{30}\times\sqrt{2}$，$0\text{x}6\text{ed}9\text{eba}1=2^{30}\times\sqrt{3}$，$0\text{x}8\text{f}1\text{bbcdc}=2^{30}\times\sqrt{5}$，$0\text{xca}62\text{c}1\text{d}6=2^{30}\times\sqrt{10}$。

接着对 512 位的信息进行处理，将其从 16 个 32 位的信息分组($M_0,\cdots,M_{15}$)变成 80 个 32 位的信息分组($W_0,\cdots,W_{79}$)。W_t 定义如下：

$$W_t=\begin{cases}M_t & t=0,1,\cdots,15\\ (M_{t-3}\oplus M_{t-8}\oplus M_{t-14}\oplus M_{t-16})\lll 1 & t=16,17,\cdots,79\end{cases}$$

设 t 是操作序号($t=0,\cdots,79$)，$\lll s$ 表示循环左移 s 位，则 SHA-1 中的每一步操作可表示为：

$\text{TEMP}=(a\lll 5)+f_t(b,c,d)+e+W_t+K_t$

$e=d$

$d=c$

$c=b\lll 30$

$b=a$

$a=\text{TEMP}$

图 3.4 所示为 SHA-1 的一次基本操作的运算过程。在 4 轮循环结束后，将进行：

$A=A+a$

$B=B+b$

$C=C+c$

$D=D+d$

$E=E+e$

这里的加法是模 2^{32} 加法。

然后用同样的方法对下一个分组进行运算，直到所有分组都处理完毕，最后将 A、B、C、D、E 输出，就得到 SHA 的散列值。

从上面的原理可以看出，通过使用不同移位、不同逻辑函数和不同初始变量，SHA 和 MD5 实现了同样的功能。

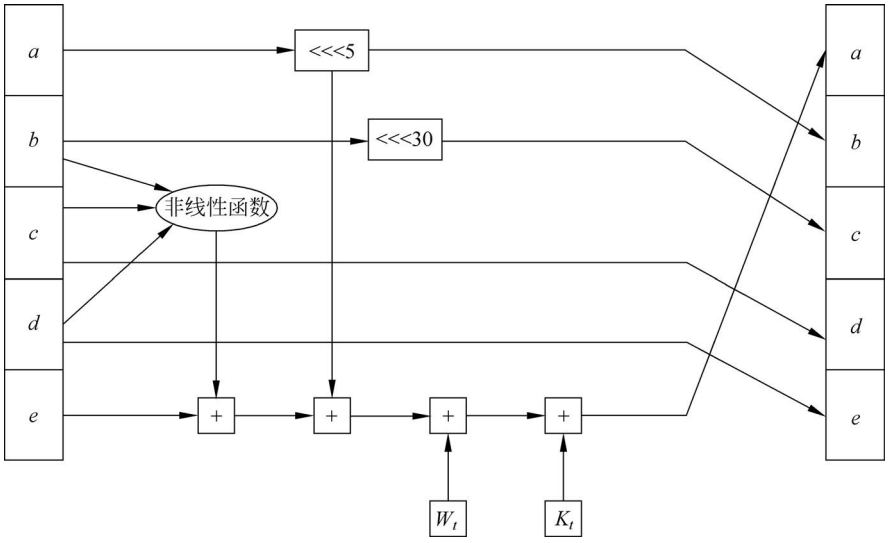

图 3.4　SHA-1 的基本操作过程

2001 年,NIST 发布 FIPS 180-2,新增了 3 个哈希函数,分别为 SHA-256、SHA-384 和 SHA-512,其散列值的长度分别为 256、384 和 512。同时,NIST 指出 FIPS 180-2 的目的是要与使用 AES 而增加的安全性相适应。SHA 性质对比如表 3.1 所示。

表 3.1　SHA 性质对比

性　　质	SHA-1	SHA-256	SHA-384	SHA-512
散列值长度	160	256	384	512
信息长度	$<2^{64}$	$<2^{64}$	$<2^{128}$	$<2^{128}$
分组大小	512	512	1024	1024
字长	32	32	64	64
步数	80	80	80	80

3.3　消息认证技术

消息认证是指使合法的接收方能够检验消息是否真实的过程。检验内容包括验证通信的双方和验证消息内容是否伪造或遭到篡改。消息认证技术主要通过密码学的方法来实现,对通信双方的验证可采用数字签名和身份认证技术,对消息内容是否伪造或遭到篡改的验证可采用比较认证码的方式。

本节首先对消息认证技术进行概述,然后介绍基于密码学的各种认证方法。

3.3.1　概述

随着 Internet 技术的发展,对网络传输过程中信息的保密性提出了更高的要求,主要包括以下内容:

(1) 对敏感的信息进行加密,即使别人截取信息也无法得到其内容。

(2) 保证数据的完整性,防止截获人在信息中加入其他信息。

(3) 对数据和信息的来源进行验证,以确保发信人的身份。

现在业界普遍采用加密技术来实现以上要求,以实现消息的安全认证。消息认证就是验证所收到的消息确实是来自真正的发送方且未被修改的消息,也可以验证消息的顺序和及时性。

消息认证实际上是对消息产生一个指纹信息——消息认证码(Message Authentication Code,MAC)。消息认证码是利用密钥对待认证消息产生的新数据块,是对该数据块加密得到的。它对待保护的信息来说是唯一的,因此可以有效地保证消息的完整性,以及实现发送消息方的不可抵赖和不能伪造性。

消息认证技术可以防止数据伪造和篡改,以及证实消息来源的有效性,已广泛应用于当今的信息网络环境中。随着密码技术与计算机计算能力的提高,消息认证码的实现方法也在不断地改进和更新,实现方式的多样化为安全的消息认证提供了保障。

3.3.2　消息认证方法

消息认证主要使用密码技术来实现。在实际使用中,通过消息认证函数 f 产生用于鉴别的消息认证码,将其用于某个身份认证协议,发送方和接收方通过消息认证码对其进行相应的认证。

由此可见,在消息认证中,认证函数 f 是认证系统的一个重要组成部分。常见的认证函数主要有以下 3 种。

(1) 消息加密:将整个消息的密文作为认证码。

(2) 哈希函数:通过哈希函数产生定长的散列值作为认证码。

(3) 消息认证码:将消息与密钥一起产生定长值作为认证码。

1. 基于加密方法的消息认证

就加密的方式而言,消息认证可以分为基于对称加密的消息认证和基于非对称加密的消息认证。

基于对称加密方式的消息认证简单明了,如图 3.5 所示。假设 K 是通信双方共同拥有的会话密钥,发送方 A 只需使用 K 对消息 M 进行加密,将密文 C 发送给接收方 B 即可。由于密钥 K 只有 A 和 B 共同拥有,因此能够保证消息的机密性。此外,由于 A 是除 B 外唯一拥有密钥和产生正确密文 M 的一方,若 B 使用 K 对密文 C 进行解密还原出正确的消息 M,就可以知道消息 M 的内容没有遭到篡改,同时也保证消息来自 A。

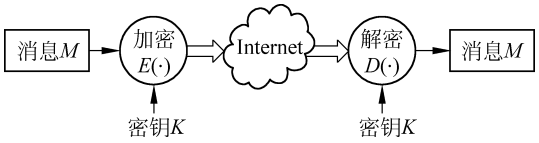

图 3.5　基于对称加密方式的消息认证过程

然而,在实际使用中,简单的加密并不能达到消息认证的真正目的。消息 M 对接收方 B 来说是未知的,因此当 B 对密文进行解密后,需要判断 M 的合法性。如果 M 本身具有某种结构,如文本文章,那么 B 只需对解密后的消息进行结构上的分析即可判断 M 的合法性。

但是,在实际通信中,消息 M 可能是随机的二进制位序列,如可执行代码、声音文件等,即使解密后仍无法判断消息 M 是否合法。

解决这一问题的方法是发送方在对消息 M 进行加密前,首先对消息通过校验函数 $F(\cdot)$产生一个校验码,将校验码附加在消息 M 之上,再进行加密,整个过程如图 3.6 所示。

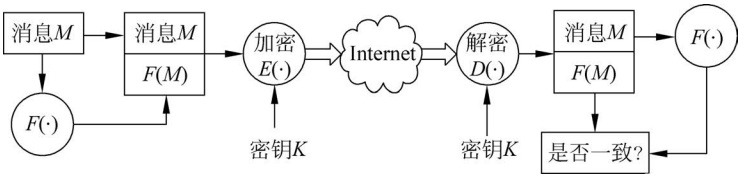

图 3.6　添加校验码的消息认证过程

在公开密钥加密体制中,如图 3.7 所示,发送方 A 可以使用自己的私钥 K_{AS} 对消息 M 进行加密,由于只有对应 A 的公钥 K_{AP} 才能正确解密出消息 M,因此采用该方法可以对消息 M 的来源进行认证。同时,该方法与前面所讲的对称加密方法一样,在实际应用中需要在消息 M 加密之前附加一定的校验码来提高认证的能力。

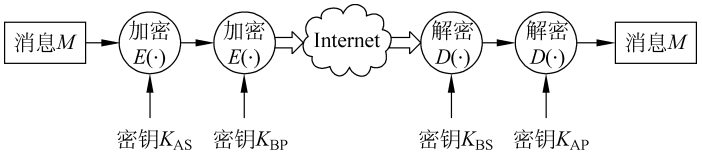

图 3.7　基于公钥加密的消息认证过程

由于解密时使用的是 A 的公钥 K_{AP},因此该方法不能保证消息的机密性,要保证消息的机密性,必须使用接收方 B 的公钥 K_{BP}。A 可以先使用自己的私钥 K_{AS} 对消息进行加密,然后再使用接收方 B 的公钥 K_{BP} 进行加密,则同时既保证了机密性,又提供了消息认证的能力。

2. 基于哈希函数的消息认证

哈希函数由于其单向性和抗碰撞性,因此常用来做消息认证。哈希函数以一个变长的消息 M 作为输入,产生一个定长的散列值 $H(M)$,即消息摘要。散列值是原始消息的函数,原始信息任何内容的变化都将导致散列值的改变,因此散列值可用于检测信息的完整性。

简单的消息认证方法可以用通信双方的共享密钥 K 对散列值 $H(M)$进行加密,将加密后的结果 $C=E_K(H(M))$以附件的方式附着在消息 M 上进行传输,接收方收到消息后,只需对 C 进行解密,即可获得散列值 $H(M)$,然后使用哈希函数对消息 M 计算另一个散列值 $H'(M)$,通过比较 $H(M)$与 $H'(M)$二者是否匹配,即可完成对消息进行的认证,如图 3.8 所示。

若需要保证消息的机密性,可将散列值附加在消息上,并使用双方的会话密钥 K 对其进行加密,得到加密后的密文 $C=E_K(M \| H(M))$,并对其进行传输,如图 3.9 所示。由于哈希函数的散列值具有对原始消息进行差错检测的能力,因此接收方可以通过这种方式来验证消息是否遭到篡改。因为只有使用通信双方所拥有的会话密钥 K 才能对密文进行解密,因此只要密钥不泄露就可验证消息来自正确的发送方,同时也保证了消息的机密性。

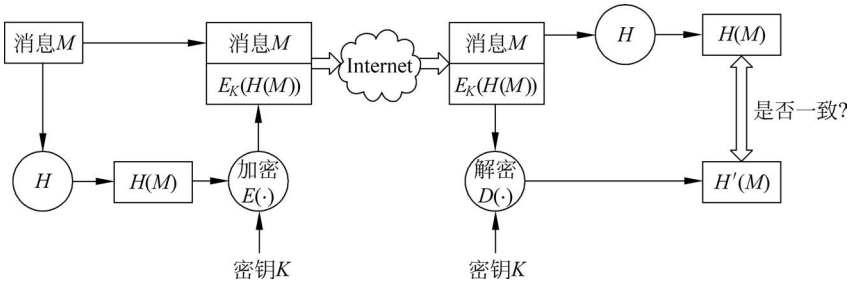

图 3.8　使用哈希函数的消息认证过程

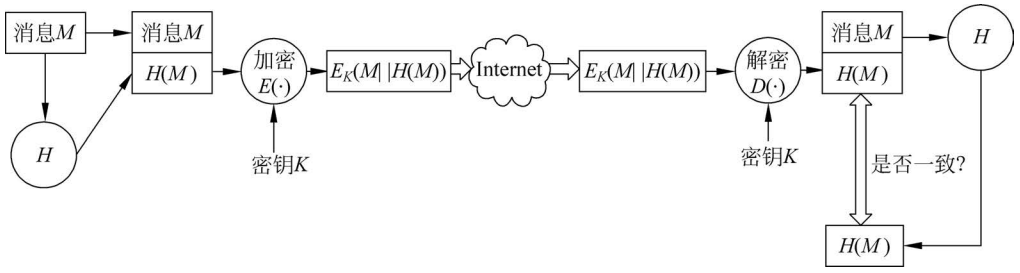

图 3.9　保证机密性的哈希函数消息认证过程

采用公钥加密的方法同样可以用于消息认证。该方法是发送方 A 使用自己的私钥 K_{AS} 对散列值 $H(M)$进行加密，将加密后的密文 $C=E_{KAS}(H(M))$附着在原始消息 M 上进行传输。接收方 B 只需使用 A 的公钥对密文进行解密，得到散列值 $H(M)$后就能对消息进行认证。

同样，如果需要保证消息的机密性，则可使用接收方 B 的公钥 K_{BP} 对消息 M 和加密后的密文 $C=E_{KAS}(H(M))$进行加密，得到新的密文 $X=E_{KBP}(M \| E_{KAS}(H(M)))$。由于使用了接收方 B 的公钥进行加密，因此只有正确的接收方 B 才能对密文进行正确解密，从而保证消息的机密性并提供认证的能力。

采用公钥进行非对称加密能提供很好的机密性，而且与对称加密相比，密钥的管理相对容易。但由于非对称加密算法产生的密文不紧凑，加密速度慢，不适合加密数据量较大的消息，因此在实际使用中，通常将对称加密与公钥加密相结合使用。具体方法是使用一个对称密钥 K 对消息 M 和加密后的密文 $C=E_{KAS}(H(M))$进行加密，再使用接收方 B 的公钥 K_{BP} 对密钥 K 进行加密，将两个加密结果进行传输，如图 3.10 所示。由于使用密钥 K 对消息进行了加密，同时使用了接收方的公钥 K_{BP} 对密钥 K 进行加密，因此只有正确的接收方 B 才能获得对称密钥 K，保证了消息的机密性和认证功能。同时，由于对称加密的速度较快，因此在保证了安全性的基础上提高了运算的速度。

3. 基于消息认证码(MAC)的消息认证

使用消息认证码进行消息认证，其基本思想与使用哈希函数类似，同样都是对消息产生一个定长的输出，用于鉴别消息的完整性。使用哈希函数时往往需要对散列值进行加密，在不需要保证消息机密性的条件下，使用加密反而会影响速度。消息认证码在进行定长输出时，使用了一个密钥来与消息一起产生定长的输出，这个定长的输出就是消息认证码。

消息认证码的使用过程如图 3.11 所示，假设通信双方 A、B 拥有会话密钥 K，用于产生

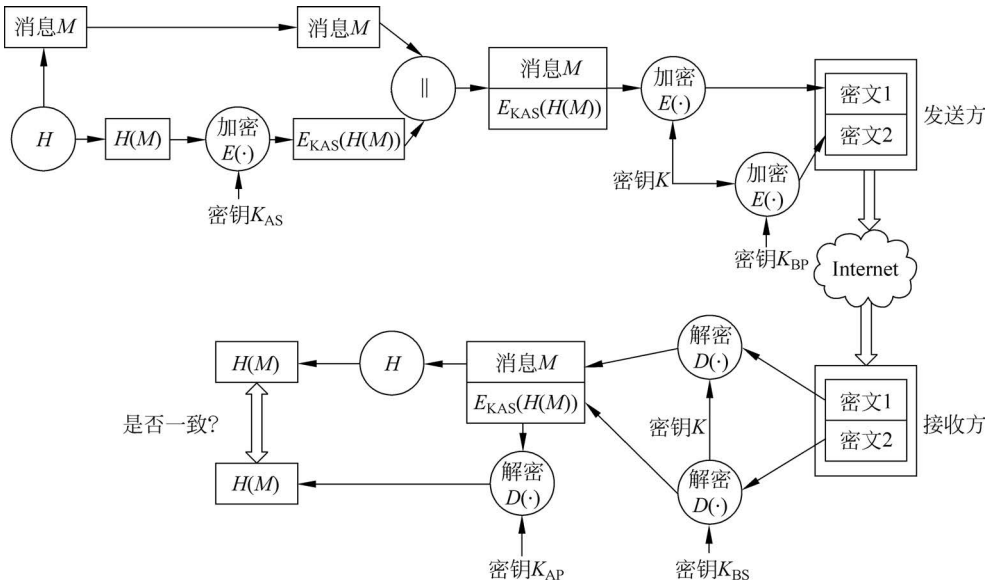

图 3.10　混合加密认证

MAC 的函数为 C。当发送方 A 要向接收方 B 发送消息 M 时，先计算出消息 M 的 MAC 值，即 $\text{MAC}=C_K(M)$，然后将 MAC 值附加在消息 M 上一起发送给 B。接收方 B 收到消息后，使用与发送方相同的会话密钥 K 计算出消息 M 的 MAC 值，然后与发送方 A 发送过来的 MAC 值进行比较，若二者匹配，则消息合法。由于共享密钥 K 只有 A 和 B 共享，攻击者想篡改消息 M，但没有密钥 K，那么计算出来的 MAC 值将与原先的 MAC 值不同，因此接收方 B 就能通过比较 MAC 值来判断消息的合法性。

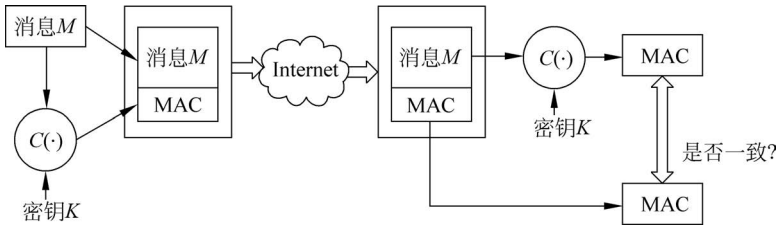

图 3.11　基于消息认证码的认证过程

MAC 函数与加密函数的相似之处在于使用了密钥，但差别在于加密函数是可逆的，而 MAC 函数是单向的，它无须可逆，因此比加密更不容易破解。当然，使用 MAC 函数只能保证信息的完整性，若要保证信息的机密性，仍然要使用加密的方法，具体方法与使用哈希函数的方法类似。

创建 MAC 函数的一种常见方法是采用分组算法的 CBC 模式来产生消息认证码。基于 DES 的 MAC 算法是一种常见的 MAC 算法，该算法采用 DES 运算的密码分组连接(CBC)方式，其初始向量为 0，将需要认证的数据分成连续的 64 位分组 $D_1, D_2, \cdots, D_N$，如果最后一个分组不足 64 位，则在其后用 0 填充成 64 位的分组数据块。具体计算过程如下：

$$C_1 = E_K(D_1)$$

$$C_2 = E_K(D_2 \oplus C_1)$$

$$C_3 = E_K(D_3 \oplus C_2)$$
$$\vdots$$
$$C_N = E_K(D_N \oplus C_{N-1})$$

其中，$E(\cdot)$表示 DES 的加密算法，K 表示加密密钥。最后的消息认证码由 C_N 最左边的 M 位表示($16 \leqslant M \leqslant 64$)。

哈希函数同样也可以用来产生消息认证码。假设 K 是通信双方 A 和 B 共同拥有的密钥，A 要发送消息 M 给 B 时，在不需要进行加密的条件下，A 只需将 M 和 K 共同通过哈希函数计算出其散列值，即 $H(M \| K)$，该散列值就是 M 的消息认证码。由于密钥 K 只有 A 和 B 才共享，因此攻击者能够获得消息 M，但没有密钥 K 也无法计算出正确的散列值，从而保证了消息 M 的完整性。

3.4　数 字 签 名

数字签名是采用密码学的方法对传输中的明文信息进行加密，以保证信息发送方的合法性，同时防止发送方的欺骗和抵赖。可以说，数字签名在网络信息安全中起到与现实生活中签名相同的功能。本节首先介绍数字签名的原理，接着介绍两类数字签名方法，即直接签名和仲裁签名，最后介绍数字签名标准(Digital Signature Standard，DSS)的原理。

3.4.1　数字签名概述

在实际网络通信中，用户可能受到来自多方面的攻击。在现实环境中，可以通过当面交易的方式或通过手写签名盖章的方式来解决通信双方的欺骗和抵赖行为。但在网络环境中，每个人都是虚拟的，如何能够实现同现实中手写签名类似的功能，这就是数字签名要解决的问题。在了解数字签名概念之前，先看下面的例子。

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

(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。

上述例子说明使用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。

因此，当通信双方不能互相信任的情况下，需要用除了报文鉴别以外的技术来防止类似的欺骗和抵赖行为。

数字签名也称电子签名。1999 年通过的欧盟《电子签名共同框架指令》对其定义为："以电子形式所附或逻辑上与其他电子数据相关的数据，作为一种判别的方法。"

2001 年审议通过的联合国贸法会《电子签名示范法》对其定义为："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。”

由此可见,数字签名应该能够在数据通信过程中识别通信双方的真实身份,保证通信的真实性及不可抵赖性,起到与手写签名或盖章同等作用。

数字签名的基本原理可以描述如下。

假设A要发送一个电子文件给B,A、B双方只需经过下面3个步骤即可。

(1) A用其私钥加密文件,这便是签名过程。

(2) A将加密的文件送到B。

(3) B用A的公钥解开A送来的文件。

以上方法符合Schneier总结的5个签名特征。

(1) 签名是可信的。因为B是用A的公钥解开加密文件的,这说明原文件只能被A的私钥加密,而只有A才知道自己的私钥。

(2) 签名是无法被伪造的。因为只有A知道自己的私钥,因此只有A能用自己的私钥加密一个文件。

(3) 签名是无法重复使用的。签名在这里就是一个加密过程,自己无法重复使用。

(4) 文件被签名以后是无法被篡改的。因为加密的文件被改动后是无法被A的公钥解开的。

(5) 签名具有不可否认性。因为除A以外无人能用A的私钥加密一个文件。

3.4.2 数字签名的实现

实现数字签名的方法有多种,这些方法可分为两类:直接数字签名和仲裁数字签名。

1. 直接数字签名

直接数字签名实现比较简单,只涉及通信双方。在直接数字签名中,接收方需要知道发送方的公钥,按照上面提到的数字签名的3个步骤直接实现即可。但在数字签名的具体应用中,还有一些问题需要解决。

签名后的文件可能被B重复使用。例如,如果签名后的文件是一张支票,B很容易多次用该电子支票兑换现金,为此A需要在文件中加上一些该支票的特有凭证,如时间戳(timestamp)等,以防止上述情况发生。

另外,公钥算法的效率是相当低的,不宜用于长文件信息的加密,为此可以采用哈希函数,将原文件信息M通过一个单向的哈希函数作用,生成相当短的输出H。首先得到$\mathrm{Hash}(M)=H$,然后将公钥算法作用在H上生成签名S,记作$E_{k_1}(H)=S$(k_1为A的私钥),A将$M \| S$传给B,B收到$M \| S$后,需要验证S是否为A的签名。

此时需要验证$H_1=H_2$,即$D_{k_2}(S)=\mathrm{Hash}(M)$,如果两者相等,则认为$S$就是A的签名。

以上方法实际上是将签名过程从原文件转移到一个很短的散列值上,从而大大地提高了效率,并得以被广泛采用。直接数字签名如图3.12所示,该过程可以总结为以下步骤。

(1) 发送方首先对被发送文件采用哈希函数进行运算,得到一个固定长度的数字串,称为报文摘要。

(2) 发送方生成发送文件的报文摘要,用自己的私钥对摘要进行加密,形成发送方的数

字签名 S。

(3) 这个数字签名将作为报文的附件和报文 M 一起发送给接收方。

(4) 接收方接收到报文后,用同样的哈希算法计算出新的报文摘要,再用发送方的公钥对报文附件的数字签名进行解密。此时比较两个报文摘要,如果值相同,则接收方可以确认该数字签名是发送方的。

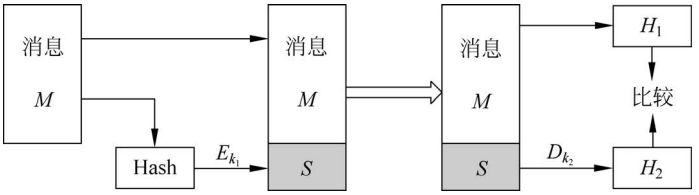

图 3.12　直接数字签名

以上数字签名只涉及通信双方,并且假定接收方知道发送方的公开密钥,称为直接数字签名。

直接数字签名的弱点是:签名的有效性依赖于发送方私人密钥的安全性,如果发送方的私人密钥丢失或被盗用,则攻击者可以伪造签名。这个弱点可以通过仲裁的方式进行解决。

2. 仲裁数字签名

由于直接数字签名存在安全缺陷,在实际应用中多采用仲裁数字签名,通过引入仲裁者来解决直接数字签名中的问题。

在仲裁数字签名中,假设用户 A 与 B 要进行通信,每个从 A 发往 B 的签名报文首先都需要发送给仲裁者 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,如图 3.13 所示。仲裁者的引入解决了直接签名方案中所面临的问题,即发送方的否认行为。在这种方案中,仲裁者的地位十分关键和敏感,它必须是一个所有通信方都能充分信任的仲裁机构,也就是说仲裁者 C 必须是一个可信的系统。

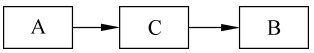

图 3.13　仲裁签名

下面讨论仲裁数字签名的实现方案。

方案 1:采用对称加密算法的数字签名。

设 C 是可信第三方,它能同时与 A、B 通信。它与 A 有共享密钥 K_A,与 B 有共享密钥 K_B。

(1) A 产生报文 M 并计算其散列值 $H(M)$,然后将附加了数字签名的报文发送给仲裁者 C,并用 K_A 加密,数字签名由 A 的标识符 ID_A 和报文的散列值 $H(M)$构成。

(2) 仲裁者 C 对数字签名进行解密,验证其散列值是有效散列值。

(3) 经过验证后,C 向 B 发送一个用 K_B 加密的报文,该报文包括 A 的标识符 ID_A、A 发出的原始报文 M、A 的数字签名和时间戳 T。

(4) B 解密恢复出报文和签名。

时间戳 T 的作用是让 B 能够判断 M 是否为过时的报文。

如果用符号

$$P \rightarrow Q: M$$

来表示“P 向 Q 发送一个报文 M”,那么上述方案就可以表述为:

(1) A→C: $M \| E_{KA}(ID_A \| H(M))$

(2) C→B: $E_{KB}(ID_A \| M \| E_{KA}(ID_A \| H(M)) \| T)$

B可以存储报文 M 及签名,当发生争执时可以将下列消息发送给C,以证明曾收到过来自A的报文:

$$E_{KB}(ID_A \| M \| E_{KA}(ID_A \| H(M)))$$

仲裁者C先用 K_B 恢复出 ID_A、M 和签名,然后用 K_A 解密该签名并验证其散列值,这样可断定报文 M 是否来自A。

在这种方案中,B不能直接验证A的签名,签名是用来解决争端的。B可以认定报文 M 来自A是因为 M 经过了C的验证,这种方案中通信双方A、B对C是高度信任的,即A可以相信C不会泄露 K_A,因此不会产生伪造的签名。B也相信C发送的报文 M 是经过验证的,确实来自A。此外,A、B还必须相信C能公平地解决争端。

这种方案的缺陷在于报文 M 的内容是以明文的形式传送给仲裁者C,任何攻击者都能获取该消息。

方案2:使用对称密码算法,密文传输。

方案2在方案1的基础上加强了数据的机密性。在此方案中,通信双方A、B使用共享密钥 K_S 来加密所要传送的报文 M。A向C传送的报文中包含A的标识符 ID_A、使用 K_S 加密原始报文 M 后的密文及数字签名,其中数字签名是由 ID_A 和加密报文的散列值构成的。仲裁者C经过检验,将收到的报文添加时间戳后,加密发送给接收方B。整个交互过程可以表述为:

(1) A→C: $ID_A \| E_{KS}(M) \| E_{KA}(ID_A \| H(E_{KS}(M)))$

(2) C→B: $E_{KB}(ID_A \| E_{KS}(M) \| E_{KA}(ID_A \| H(E_{KS}(M))) \| T)$

在这种方案中,尽管仲裁者C无法读取消息报文 M 中的内容,但他仍能防止A或B中任何一方的欺诈。两种方案都存在的问题是:仲裁者C可能与发送方勾结来否认签名报文,或者与接收方共同伪造发送方的签名。

方案3:使用公开密钥算法,密文传输。

针对上述两种方案的缺陷,采用公开密钥方案就能够迎刃而解。使用公开密钥进行数字签名时,A对报文 M 进行两次加密:先用其私钥 K_{AS} 对消息 M 进行加密,再用B的公钥 K_{BP} 加密,得到加密后的签名;A再用 K_{AS} 对其标识符 ID_A 和上述加密后的签名进行加密,然后连同 ID_A 一起发送给C。经过双重加密后,报文 M 只有B才能阅读,对C来说是安全的,但C能通过外层的解密,从而证实报文确实是来自A的(因为只有A有私钥 K_{AS})。C通过验证A的公/私钥对(K_{AP} 和 K_{AS})的有效性完成对报文的验证,然后再用自己的私钥 K_{CS} 对A的标识符 ID_A、双重加密后的 M 及时间戳进行加密后发送给B。整个交互过程可以表述为:

(1) A→C: $ID_A \| E_{KAS}(ID_A \| E_{KBP}(E_{KAS}(M)))$

(2) C→B: $E_{KCS}(ID_A \| E_{KBP}(E_{KAS}(M)) \| T)$

采用公开密钥的数字签名方案具有以下优点。

(1) 通信前,通信各方没有任何共享信息,从而避免了联合欺诈。

(2) A发给B的消息对其他人是保密的,包括C。

(3) 即使 A 的私钥 K_{AS} 已泄密或被盗，但 C 的私钥 K_{CS} 没有泄密，那么时间戳不正确的消息仍然是不能被发送的。

3.4.3　数字签名标准

数字签名标准(Digital Signature Standard，DSS)由美国国家标准与技术研究院在1991 年提出作为美国联邦信息处理标准(FIPS)。DSS 采用了美国国家安全局主持开发的数字签名算法(Digital Signature Algorithm，DSA)，它使用安全散列算法(Secure Hash Algorithm，SHA)给出了一种新的数字签名方法。DSS 分别于 1993 年和 1996 年做了修改。2000 年发布该标准的扩充版，即 FIPS 186-2，其中包括基于 RSA 和椭圆曲线密码的数字签名算法。本节只介绍最初修订的 DSS。

DSS 数字签名方法与 RSA 数字签名方法不同。如图 3.14 所示，在 RSA 方法中，散列函数的输入是要签名的消息，输出是定长的散列值，用发送方的私钥对该散列值进行加密而形成签名。接收方接收到消息及其签名后用发送方的公钥对收到的签名进行解密，同时采用相同的散列函数计算收到的消息的散列值，如果两个散列值相同，则认为签名是有效的。

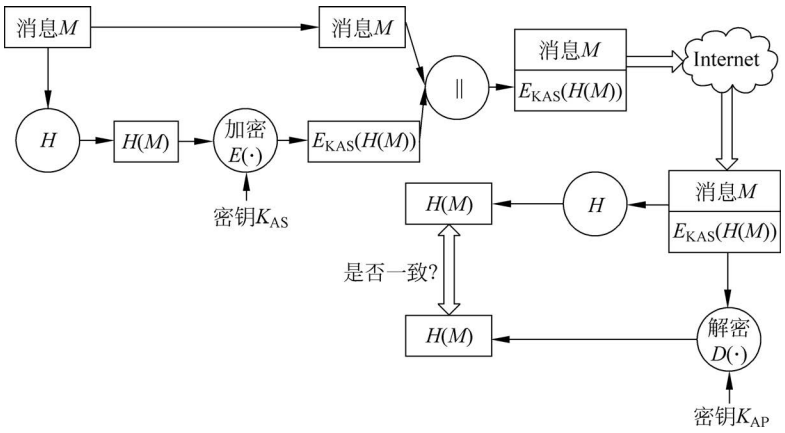

图 3.14　RSA 数字签名方法

DSS 方法也需要使用散列函数，它产生的散列值及其为此次签名产生的随机数 k 作为签名函数的输入。此外，签名函数还需要使用发送方的私钥和一组参数，这组参数被一组通信伙伴所共享，通常认为这组参数构成全局公钥。签名的结果由两部分组成，记作 s 和 r。接收方对接收到的消息产生散列值，并和签名一起作为验证函数的输入，若签名有效，则验证函数的输出会等于签名分量 r。签名函数保证只有发送方的私钥才能产生有效的签名。DSS 数字签名方法如图 3.15 所示。

DSS 的核心是其定义的数字签名算法，该算法的提出基于离散对数分解十分困难这个数学难题。该算法将 3 个主要参数 p、q、g 作为全局的公开密钥，它们被一组用户所共享。

(1) p 是一个大素数，长度为 512～1024，即 $2^{L-1}<p<2^{L}$，$512\leqslant L\leqslant 1024$，且 L 是 64 的倍数。

(2) q 是一个长度为 160 位的素数，q 能整除 $p-1$，即 $2^{159}<q<2^{160}$，且 $(p-1) \bmod q=0$。

(3) $g=h^{(p-1)/q} \bmod p$，其中 $1<h<(p-1)$，且 $h^{(p-1)/q} \bmod p>1$ 的任何整数。

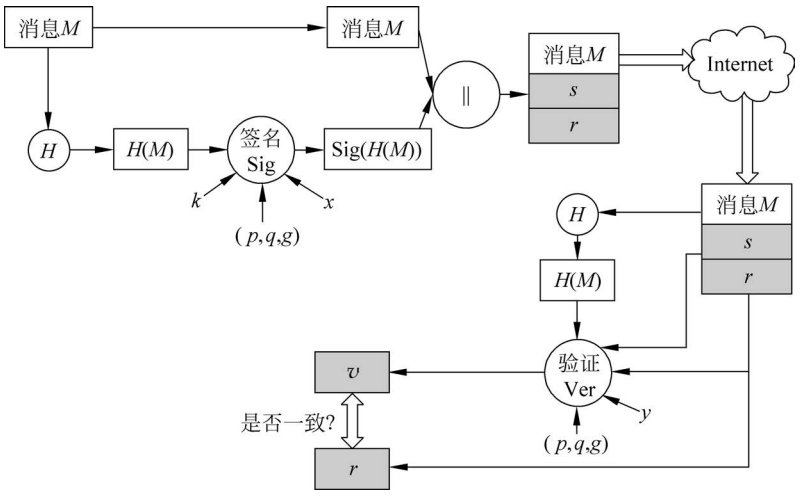

图 3.15 DSS 数字签名方法

确定 p、q、g 以后,每个用户就可以确定其私钥并产生公钥。

私钥 x 必须是一个 $1\sim q-1$ 的随机数或伪随机数。

公钥 y 可以由私钥 x 计算得出:$y=g^x \bmod p$。由给定的 x 计算 y 比较简单,但由给定的 y 确定 x 在计算上是不可行的,因为这相当于求 y 的以 g 为底的模 p 的离散对数,其计算复杂度非常高。

签名是通过签名函数计算产生的,该函数包含两个分量 r 和 s。r 和 s 是公钥(p、q、g)、用户私钥 x、消息的散列值 $H(M)$和随机数 k 的函数,k 在每次签名中都是不同的。

签名函数(Sig)的定义如下。

$$r=f_1(k,p,q,g)=(g^k \bmod p) \bmod q$$

$$s=f_2(H(M),k,x,r,q)=(k^{-1}(H(M)+xr)) \bmod q$$

其中,k^{-1} 表示 k 模 q 的乘法逆元,签名$=(r,s)$。

DSS 签名函数如图 3.16 所示。

验证函数(Ver)的定义如下。

$$w=f_3(s',q)=(s')^{-1} \bmod q$$

$$u_1=[H(M)w] \bmod q$$

$$u_2=(r')w \bmod q$$

$$v=f_4(y,q,g,H(M),w,r')=[g^{u_1}y^{u_2} \bmod p] \bmod q$$

其中,需要检验 $v=r'$。

DSS 验证函数如图 3.17 所示。

在图 3.16 和图 3.17 中,该算法有这样一个特点:接收方的验证依赖于 r,但 r 却不依赖于原始消息,它是 k 和全局公钥的函数。k 模 p 的乘法逆元传给函数 f_2,f_2 的输入还包含消息的散列值和用户私钥。这种函数结构使接收方可以利用其收到的消息和签名、他的公钥和全局公钥来恢复 r。由于离散对数的求解困难性,攻击者想从 r 恢复出 k 或从 s 恢复出 x 都是不可行的。

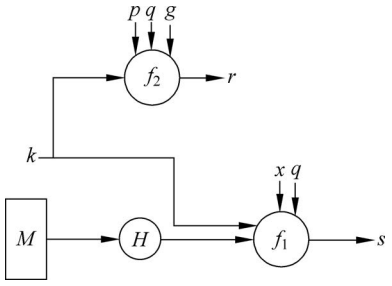

图 3.16　DSS 签名函数

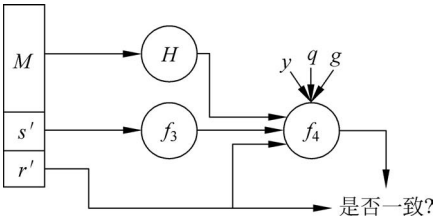

图 3.17　DSS 验证函数

从计算的复杂度来看，DSS 数字签名的计算量主要是 $g^k \bmod p$。由于它不依赖于被签名的消息，因此可以预先计算。另一项计算量较大的工作是计算 k 的乘法逆元 k^{-1}，当然也可以采用预先计算的方法。

3.5　身 份 认 证

身份认证是建立安全通信环境的前提条件，只有通信双方相互确认对方身份后才能通过加密等手段建立安全信道，同时它也是授权访问（基于身份的访问控制）和审计记录等服务的基础，因此身份认证在网络信息安全中占据着十分重要的位置。身份认证协议在解决分布式，尤其是开放环境中的信息安全问题时起到非常重要的作用。

3.5.1　概述

身份认证的目的在于对通信中某一方的身份进行标识和验证，其主要方法是验证用户所拥有的可被识别的特征。一个身份认证系统一般由以下几部分组成：一方是提出某种申请要求，需要被验证身份的人；另一方是验证者，验证申请者身份的人；第三方是攻击者，对消息进行窃取等攻击的人，可以伪装成通信中的任何一方。与此同时，在某些认证系统中需要引入第四方，即作为仲裁或调解机构的可信任机构。

现实世界中的身份认证可以通过出示带相片的身份证件来完成，某些特殊的区域可能还使用指纹或虹膜等生物特征对进出人员的身份进行确认。不管用什么方法，身份认证机制就是将每个人的身份标识出来，并确认其身份的合法性。

在计算机系统中，传统的物理身份认证机制并不适用，其身份认证主要通过口令和身份认证协议来完成。在计算机网络通信中，身份认证就是用某种方法来证明正在被鉴别的用户身份是合法的授权者。

口令技术由于其简单易用，因此成为目前一种常用的身份认证技术。使用口令技术存在的最大隐患就是口令的泄露问题。口令泄露可以有多种途径，如用户登录时被他人窥视、攻击者从计算机存放口令的文件中获取、口令被在线攻击者或离线攻击者破解。

由于基于口令的认证方法存在较大的问题，因此在网络环境中，通常使用身份认证协议来鉴别通信中的对方是否合法，以及是否与他所声称的身份一致。身份认证协议是一种特殊的通信协议，它定义了参与认证服务的所有通信方在身份认证过程中需要交换的消息格

式、消息发生的次序及消息的语义。在通信过程中,通常采用加密算法、哈希函数来保证消息的完整性、保密性。

使用密码学方法的身份认证协议比传统的基于口令的认证更安全,并能提供更多的安全服务。通过使用各种加密算法,可以对通信过程中的密钥进行很好的保护。在通信过程中,当需要传输用户提供的口令时,可以将用户口令先进行加密处理,对加密后的口令进行传输,在接收端再进行相应的解密处理,从而对用户口令或密钥进行很好的保护。

身份认证协议一般有两个通信方,可能还会有一个双方都信任的第三方参与。其中一个通信方按照协议的规定向另一方或第三方发出认证请求,对方按照协议的规定作出响应,当协议顺利执行完毕时双方应该确信对方的身份。

从使用加密的方法来看,身份认证可分为基于对称密钥的身份认证和基于公钥加密的身份认证。

基于对称密钥的身份认证思想是从口令认证的方法发展而来的。传统检验对方传递来的口令是否合法的做法很简单,口令容易在传递过程中被窃听而泄露。因此在实际网络环境中,必须采用既能够验证对方拥有共同的秘密,又不会在通信过程中泄露该秘密的方法。与此同时,在实际通信过程中,一台计算机可能需要与多台计算机进行身份认证,如果全部采用共享密钥的方式,那么就需要与众多的计算机都建立共享密钥。这样做在大型网络环境中既不经济也不安全,同时大量共享密钥的建立、维护和更新将是非常复杂的。这时需要一个可信赖的第三方负责完成密钥的分配工作,称为密钥分发中心(Key Distribution Center,KDC)。在通信开始阶段,通信中的每一方都只与 KDC 有共享密钥,通信双方之间的认证借助 KDC 才能完成。KDC 负责给通信双方创建并分发共享密钥,通信双方获得共享密钥后再使用对称加密算法的协议进行相互之间的身份认证。

基于公钥加密的身份认证协议比基于对称密钥的身份认证能提供更强有力的安全保障,公钥加密算法可以让通信中的各方通过加密、解密运算来验证对方的身份。在使用公钥方式进行身份认证时需要事先知道对方的公钥,因此同样需要一个可信第三方负责分发公钥。在实际应用中,公钥的分发是采用证书的形式来实现的。证书中含有证书所有人的名字、身份信息、公钥,以及签发机构、签发日期、序列号、有效期等相关数据,并用证书权威机构自己的私钥进行签名。证书被设计存放在目录服务系统中,通信中的每一方都拥有证书权威机构的公钥,可以从目录服务中获得通信对方的证书,通过验证证书权威机构签名可以确认对方证书中公钥的合法性。

与此同时,从认证的方向性来看,可分为相互认证和单向认证。

相互认证用于通信双方的互相确认,同时可进行密钥交换。在认证过程中,密钥分配是重点。保密性和时效性是密钥交换中的两个重要问题。从机密性的角度来看,为防止假冒和会话密钥的泄露,用户的身份信息和会话密钥等重要信息必须以密文的形式传送。另一方面,攻击者可以利用重放攻击对会话密钥进行攻击或假冒通信双方中的某一方,密钥的时效性可防止重放攻击的威胁。

常见的重放攻击如下。

(1) 简单重放。攻击者简单地复制消息并在此之后重放这条消息。

(2) 可检测的重放。攻击者在有效的时限内重放有时间戳的消息。

(3) 不可检测的重放。由于原始消息可能被禁止而不能到达接收方,只有通过重放消

息才能发送给接收方，此时可能出现这种攻击。

(4) 不加修改的逆向重放。如果使用对称密码，并且发送方不能根据内容来区分发出的消息和接收的消息，那么可能出现这种攻击。

对于重放攻击，一般可使用以下方式来预防。

(1) 序列号。这种方法是为每个需要认证的消息添加一个序列号，新的消息到达后先对序列号进行检查，只有满足正确次序的序列号的消息才能被接收。这种方法存在的一个问题就是通信各方都必须记录最近处理的序列号，而且还必须保持序列号的同步。

(2) 时间戳。这种方法是为传送的报文添加时间戳，当接收到新的消息时，首先对时间戳进行检查，只有在消息的时间戳与本地时钟足够接近时才认为该消息是一个新的消息。时间戳要求通信各方必须保持时钟的同步。使用时间戳方法存在 3 个问题：第一，通信各方的时间同步需要由某种协议来维持，同时为了能够应对网络的故障和恶意攻击，该协议还必须具有容错性和安全性；第二，如果由于通信一方时钟机制而出错，那么攻击者的成功率将大大增加；第三，网络延时的可变性和不可预知性使得各分布时钟无法保持精确同步，因此需要申请足够大的时间窗口以适应网络延时，这与小时间窗口的要求是矛盾的。

(3) 随机数/响应。这种方法是在接收消息前先发送一个临时的交互号(随机数)，并且所发送的消息要包含该临时交互号。随机数/响应不适合于无连接的应用，因为它要求在任何无连接传输之前必须先握手，这与无连接的特征相违背。

单向认证主要用于电子邮件等应用中，其主要特点在于发送方和接收方不需要同时在线。以电子邮件为例，邮件消息发送到接收方的电子邮箱中，并一直保存在邮箱中，等待接收方阅读邮件。电子邮件的存储-转发一般是由 SMTP(简单邮件传输协议)或 X.400 来处理的，因此邮件报头必须是明文形式。但是，用户都希望邮件以密文的形式传输或转发，邮件要能够加密，而且邮件处理系统无法对其进行解密。此外，电子邮件的认证还包括邮件的接收方必须能够确认邮件消息是来自真正的发送方。

3.5.2　基于口令的身份认证

基于口令的认证方法是传统的认证机制，主要用于用户对远程计算机系统的访问，确定用户是否拥有使用该系统或系统中的服务的合法权限。由于使用口令的方法简单，容易记忆，因此成为比较广泛采用的一种认证技术。基于口令的身份认证一般是单向认证。

常见的使用口令的方法是采用哈希函数对口令进行验证。假设用户 A 想要登录服务器系统 S，这时用户 A 只需向服务器发送服务器分配给他的 ID_A 号和口令 PW_A，即

$$A \rightarrow S: ID_A \parallel PW_A$$

服务器在收到用户发送过来的信息后，首先将收到的 PW_A 通过哈希函数 $H(\cdot)$产生散列值，然后在自己的口令文档或数据库中查找与(ID_A，$H(PW_A)$)相匹配的记录，如果找到，则认证成功，允许用户使用自己的服务。在这种方法中，为了确定用户是否有合法的权限使用该系统，服务器只要能够区分输入的口令是有效的还是无效的即可，并不需要知道口令本身的内容。因此，即使攻击者通过窃听双方通信或窃取了服务器中的口令列表，得到了 $H(PW_A)$，也无法假冒用户 A 来进行攻击。

在上述方法中，服务器保存了用户的口令列表，虽然该列表是口令的散列值，但存在着一定的不安全因素。由于用户的口令通常都比较短，因此当攻击者 C 已经获得服务器的口

令列表时,可采用以下的方法进行攻击。攻击者C可以在本地搜集很多个常用的口令(如100万个),用哈希函数对这些口令进行计算并得到相应的散列值,将这些结果存储起来。此时将服务器的口令列表与自己存储的文件相比较,得到匹配的数据,这样攻击者C就获得了某个或某些用户的口令,这种攻击方式称为字典攻击(Dictionary Attack,DA)。

为了消除字典攻击,服务器中建立的口令列表记录可以修改成(ID,salt,H(PW,salt))的形式。ID表示用户的身份,salt表示一个随机数,H(PW,salt)表示用户口令和随机数合起来的散列值。

在这种方式中,用户的口令在发送给服务器之前,首先和随机数一起进行散列,产生散列值H(PW,salt),即

$$A \rightarrow S: ID_A \parallel salt \parallel PW_A$$

服务器在收到用户的消息后,在自己的口令列表中查找与(ID_A,salt,H(PW_A,salt))匹配的记录,如果找到,则允许A访问自己的服务。

虽然添加salt的方法能抵抗字典攻击,但也有一定的安全隐患,即不能抵抗口令窃听的攻击。攻击者可以使用各种方法获得用户口令的明文,从而进行相应的攻击。

口令窃听攻击之所以成功的原因,很大一部分在于用户每次登录时总是使用同一个口令。如果用户每次登录都使用不同的"口令",那么攻击者进行口令窃听攻击成功的概率将大大降低。

此外,还可以使用哈希链方法。在该方法中,服务器首先对用户进行初始化,保存用户最初的口令记录(ID,n,H_n(PW)),其中ID是用户的身份标识,n是一个整数,$H(\cdot)$是哈希函数,H_n(PW)定义为$H_n(PW)=H(H_{n-1}(PW))$,$n=1,2,\cdots$,即对用户口令PW通过哈希函数产生散列值,并将该散列值再通过哈希函数产生新的散列值,以此类推,一共进行n次哈希运算。用户在登录时只需要记住自己的口令PW,当用户登录到服务器时,服务器会更新所保存的用户记录。

当客户机进行首次口令认证时,客户机对口令PW重复计算哈希函数$n-1$次,得到H_{n-1}(PW)。客户机将计算结果发送给服务器,服务器收到H_{n-1}(PW)后,再进行一次哈希函数的运算,得到H_n(PW),并检查新的散列值是否与自己保存的用户记录所匹配。如果匹配,则表示认证通过,服务器确定对方就是合法授权的用户。接着,服务器更新所保存的口令记录,用(ID,$n-1$,H_{n-1}(PW))更新(ID,n,H_n(PW))。

在这种方法中,由于用户发给服务器的口令PW通过哈希函数计算后得到H_n(PW)的次数是不同的,而且哈希函数是单向的,因此攻击者无法从H_n(PW)中得到有用的信息。即使攻击者通过某种手段获得了服务器所保存的口令列表,也无法得到用户的口令PW。

在基于哈希链的认证方法中,作为计数器的n值是变化的,依次递减到1,当n最终减为1时,客户机和服务器端需要重新初始化以设置口令。

3.5.3　基于对称密钥的身份认证

1. 基于对称密钥的双向身份认证

在基于对称密钥的双向身份认证方法中,可以通过使用两层传统的加密密钥结构来保证网络环境中通信的保密性。为此需要使用一个可信赖的密钥分配中心(KDC)。通信各方与KDC都有一个共享的密钥,称为主密钥,KDC负责产生通信各方通信时短期使用的密

钥，称为会话密钥，主密钥负责保护会话密钥的分发。

1) Needham-Schroeder 协议

Needham-Schroeder 协议利用 KDC 进行密钥分配，同时具备了身份认证的功能。假设通信双方 A、B 与 KDC 分别共享密钥 K_A 和 K_B。

(1) A→KDC：$ID_A \| ID_B \| N_1$

(2) KDC→A：$E_{KA}(K_S \| ID_B \| N_1 \| E_{KB}(K_S \| ID_A))$

(3) A→B：$E_{KB}(K_S \| ID_A)$

(4) B→A：$E_{KS}(N_2)$

(5) A→B：$E_{KS}(f(N_2))$

该协议的目的是保证将会话密钥 K_S 安全地分配给 A 和 B。

第 1 步，A 将他的身份信息 ID_A、B 的身份信息 ID_B 及一个作为临时交互值的随机数 N_1 组成的消息发给 KDC，表明 A 要与 B 认证并通信。

第 2 步，KDC 产生 A、B 之间的会话密钥 K_S，用 KDC 与 B 的共享密钥 K_B 对会话密钥 K_S 和 A 的身份信息 ID_A 进行加密，然后用它和 A 的共享密钥 K_A 对随机数 N_1、B 的身份信息 ID_B、会话密钥 K_S 和已加密的信息进行加密，然后将它发送给 A。

第 3 步，A 将消息解密并获得 K_S，比较 N_1 和第一步所发送的 N_1 是否一致，然后将 KDC 发来的用 K_B 加密的消息发送给 B。

第 4 步，B 对消息进行解密并获得 K_S，然后产生另一随机数 N_2，用 K_S 加密并发送给 A。

第 5 步，A 对消息解密，用函数 f 产生新的结果，并用 K_S 加密，然后发给 B。

第 6 步，B 对消息解密，并验证它是否为 f 产生的结果。

在这个过程中，第 4、5 步可以防止某些重放攻击。例如，若攻击者窃听到第 3 步中的报文并进行重放，重放报文中的 K_S 是一个过期的会话密钥，若没有第 4、5 步的交互过程，B 将试图使用这个过期密钥，从而产生混乱。

尽管如此，该协议仍然存在漏洞，容易受到重放攻击。例如，攻击者 X 可能从某些途径获得一个过期的会话密钥。X 就可以冒充 A 重放第 3 步的报文，欺骗 B 使用过期的会话密钥，除非 B 明确记得以前与 A 通信所使用的所有会话密钥，否则 B 无法确定该消息是否为重放的消息。

2) Denning 协议

Denning 协议对 Needham-Schroeder 协议进行了修改，引入了时间戳机制，整个过程如下。

(1) A→KDC：$ID_A \| ID_B$

(2) KDC→A：$E_{KA}(K_S \| ID_B \| T \| E_{KB}(K_S \| ID_A \| T))$

(3) A→B：$E_{KB}(K_S \| ID_A \| T)$

(4) B→A：$E_{KS}(N_1)$

(5) A→B：$E_{KS}(f(N_1))$

时间戳 T 使 A 和 B 确信会话密钥 K_S 是最新产生的，这样 A 和 B 都知道此次交换的是一个新的会话密钥。A 和 B 通过验证下列式子来验证密钥的及时性：

$$|c - T| < \Delta t_1 + \Delta t_2$$

其中,c 是本地时钟的时间值,T 是报文携带的时间戳,Δt_1 是 KDC 时钟与本地时钟的正常偏差,Δt_2 是网络的正常时延值,满足该公式的时间戳被认为是合法的。由于是使用与 KDC 的共享密钥对时间戳进行加密,因此即使攻击者知道旧的会话密钥,也不能成功地重放消息,因为 B 可以根据消息的及时性进行检测。

与 Needham-Schroeder 协议相比,Denning 协议的安全性更高,但同时也带来了新的问题,即如何安全、准确地通过网络进行时钟同步。因此,该协议也存在着一定的危险,由于时钟同步机制的出错或受到破坏,通信各方的时钟不同步,协议将容易遭到重放攻击。例如,发送方的时钟快于接收方的时钟,攻击者可以窃听到发送端的报文,由于报文中的时间戳快于接收方的本地时间,攻击者可以等到接收方时钟等于报文时间戳时重放该报文,这种重放可能导致不可预知的结果,这样的攻击称为抑制-重放攻击。

解决抑制-重放攻击的一种方法是要求通信各方必须根据 KDC 的时钟周期性地校验时钟。另一种方法是基于随机数的临时交互值的认证协议,它不要求时钟同步,并且接收的临时交互值对发送方而言是不可预知的,从而不易受到抑制-重放攻击。

3) Neuman-Stubblebine 协议

Neuman-Stubblebine 协议提出的目的是为了试图解决抑制-重放攻击,同时解决 Needham-Schroeder 协议中出现的问题,整个过程如下。

(1) A→B: $ID_A \| N_1$

(2) B→KDC: $ID_B \| N_2 \| E_{KB}(ID_A \| N_1 \| T)$

(3) KDC→A: $E_{KA}(ID_B \| N_1 \| K_S \| T) \| E_{KB}(ID_A \| K_S \| T) \| N_2$

(4) A→B: $E_{KB}(ID_A \| K_S \| T) \| E_{KS}(N_2)$

第 1 步,A 发起认证。A 产生临时交互值 N_1,连同自己的身份信息 ID_A 以明文的形式发送给 B,N_1 的作用是在进行密钥分发时将返回给 A,A 通过验证 N_1 的值来确认消息的时效性。

第 2 步,B 向 KDC 申请会话密钥。B 将 A 的身份信息 ID_A、临时交互值 N_1 及时间戳 T 用他和 KDC 的共享密钥 K_B 加密,将加密结果、自己的身份信息 ID_B 和新的临时交互值 N_2 一起发送给 KDC。其中用 K_B 加密的数据 $E_{KB}(ID_A \| N_1 \| T)$ 的作用是请求 KDC 向 A 发布一个可信的“票据”,指定了“票据”的接收者、有效期,以及 A 发送的临时交互值 N_1。

第 3 步,KDC 产生会话密钥 K_S,然后产生两个消息。第一个消息是由 B 的身份信息 ID_B、A 的临时交互值 N_1、会话密钥 K_S 和时间戳组成,并用他与 A 的共享密钥 K_A 加密;第二个消息是由 A 的身份信息 ID_A、会话密钥 K_S 和时间戳组成,并用他与 B 的共享密钥 K_B 加密。此时,将这两个消息连同 B 的临时交互值 N_2 一起发送给 A。时间戳 T 给出了会话密钥的使用时限,ID_B 用于证实 B 已经收到初始报文,N_1 能够检测重放攻击。

第 4 步,A 用 KDC 与 B 的共享密钥 K_B 加密的消息和加密后的 N_2 发送给 B。B 从加密消息中得到共享密钥并解密出 N_2,通过比较 N_2 来鉴别消息是来自 A 还是一次重放攻击。

这个协议为 A、B 双方建立会话提供了一种安全有效的会话密钥交换方式。在协议中,时间戳 T 只是相对 B 的本地时钟,也只有 B 对其进行校验,因此不需要时钟的同步。同时,A 可以保存用于鉴别 B 的消息,从而减少与 KDC 的多次交互。假设 A、B 完成了上面的协议和通信,然后终止连接,当 A 要与 B 再次建立新的会话时,只要 A 保存了原有的消息且在

密钥的有效期限内，则不必依赖 KDC 就能够在 3 步之内重新进行身份认证。

(1) A→B：$E_{KB}[ID_A \| K_S \| T] \| N_1'$

(2) B→A：$N_2' \| E_{KS}(N_1')$

(3) A→B：$E_{KS}(N_2')$

B 在第 1 步收到消息后可以验证密钥有没有过期，新产生的 N_1'、N_2'用来检测是否有重放攻击。

2. 基于对称密钥的单向身份认证

基于对称密钥的单向认证一般也采用以 KDC 为基础的方法。但是在电子邮件的应用中，无法要求发送方和接收方同时在线，因此在协议过程中不存在双方的交互。该协议的具体过程如下。

(1) A→KDC：$ID_A \| ID_B \| N_1$

(2) KDC→A：$E_{KA}[K_S \| ID_B \| N_1 \| E_{KB}(K_S \| ID_A)]$

(3) A→B：$E_{KB}(ID_A \| K_S) \| E_{KS}(M)$

可以看出，该协议比较简洁，可以保证只有真正的接收方才能读取消息，同时也可以保证发送方的确是 A，但同样无法抵抗重放攻击。由于电子邮件的转发和处理过程中存在时延较大，因此通过添加时间戳的方式来抵抗重放攻击的可能性不大。

3.5.4　基于公钥的身份认证

1. 基于公钥的双向身份认证

1) Denning-Sacco 协议

在公开密钥加密的身份认证中，也需要有一个类似的中心系统来分发通信各方的公开密钥证书。因为在没有认证中心或密钥分配中心的情况下，要使通信各方都能拥有对方的当前公钥是不切实际的。

Denning-Sacco 协议是一种使用时间戳机制的公钥分配和认证方法。假设通信双方分别为 A 和 B，AS 为认证服务器，整个过程如下。

(1) A→AS：$ID_A \| ID_B$

(2) AS→A：$E_{KSAS}(ID_A \| K_{PA} \| T) \| E_{KSAS}(ID_B \| K_{PB} \| T)$

(3) A→B：$E_{KSAS}(ID_A \| K_{PA} \| T) \| E_{KSAS}(ID_B \| K_{PB} \| T) \| E_{KPB}(E_{KSA}(K_S \| T))$

其中，K_{PA} 和 K_{PB} 分别为 A 和 B 的公钥。K_{PAS} 和 K_{SAS} 分别为 AS 的公钥和私钥。在这个协议中，认证中心系统不负责密钥的分配，而只提供公钥证书，所以称为认证服务器(AS)。会话密钥 K_S 的选择和加密完全由 A 来完成，因此不存在被 AS 泄露的危险。同时使用了时间戳机制，可以防止重放攻击对密钥安全性的威胁。

这个协议简洁明了，但不足之处仍然是需要严格的时钟同步才能保证协议的安全。

2) Woo-Lam 协议

Woo-Lam 协议使用随机数作为临时交互值来代替时间戳，它是一种以 KDC 为中心的认证协议。该协议的具体过程如下。

(1) A→KDC：$ID_A \| ID_B$

(2) KDC→A：$E_{KSK}(ID_B \| K_{PB})$

(3) A→B: $E_{KPB}(N_1 \| ID_A)$

(4) B→KDC: $ID_B \| ID_A \| E_{KPK}(N_1)$

(5) KDC→B: $E_{KSK}(ID_A \| K_{PA}) \| E_{KPB}(E_{KSK}(N_1 \| K_S \| ID_B))$

(6) B→A: $E_{KPA}(E_{KSK}(N_1 \| K_S \| ID_B) \| N_2)$

(7) A→B: $E_{KS}(N_2)$

在协议刚开始,A向KDC发送一个要与B建立安全连接的请求,KDC将B的公钥证书副本返回给A,A通过B的公钥告诉B想与他通信,同时将临时交互值N_1发给B。然后,B向KDC请求A的公钥证书和会话密钥,由于B发送消息中包含A的临时交互值,因此KDC可以用临时交互值对会话密钥加戳,其中临时交互值受KDC的公钥保护。接着,KDC将A的公钥证书的副本和消息$\{N_1, K_S, ID_B\}$一起返回给B。这条消息说明,K_S是KDC为B产生的且与N_1有关的密钥。N_1使A确信K_S是新会话密钥。用KDC的私钥对三元组$\{N_1, K_S, ID_B\}$加密,使得B可以验证该三元组确实来自KDC。由于是用B的公钥对该三元组加密,因此其他各方均不能利用该三元组与A建立假冒连接。在第(6)步中,B用A的公钥对$E_{KSK}(N_1 \| K_S \| ID_B)$和B产生的随机数$N_2$加密后发送给A,A先解密得出会话密钥$K$,然后用$K_S$对$N_2$加密发送给B,这样可以使B确信A已经获得正确的会话密钥。

相比Denning-Sacco协议,这个协议对抵抗攻击的能力更强,但也存在着某些安全隐患。改进的方法是在第5步和第6步中加入A的身份信息ID_A,将会话密钥与双方的身份信息绑定在一起。将ID_A和N_1绑定在一起唯一标识了A的连接请求,具体改进过程如下。

(1) A→KDC: $ID_A \| ID_B$

(2) KDC→A: $E_{KSK}(ID_B \| K_{PB})$

(3) A→B: $E_{KPB}(N_1 \| ID_A)$

(4) B→KDC: $ID_B \| ID_A \| E_{KPK}(N_1)$

(5) KDC→B: $E_{KSK}(ID_A \| K_{PA}) \| E_{KPB}(E_{KSK}(N_1 \| K_S \| ID_A \| ID_B))$

(6) B→A: $E_{KPA}(E_{KSA}(N_1 \| K_S \| ID_A \| ID_B) \| N_2)$

(7) A→B: $E_{KS}(N_2)$

2. 基于公钥的单向身份认证

公开密钥由于其自身的特性,比对称密钥更适合用于单向认证。一般情况下,发送方需要掌握接收方的公钥,而接收方也需要拥有发送方的公钥,这样才能对消息进行加密,同时也能对消息的签名进行解密。

使用公钥进行验证的步骤相对简洁,主要有以下几种使用方法。

(1) A→B: $E_{KPB}(K_S) \| E_{KS}(M)$

(2) A→B: $E_{KSA}(H(M)) \| M$

(3) A→B: $E_{KPB}(K_S) \| E_{KS}(M \| E_{KSA}(H(M)))$

(4) A→B: $E_{KPB}(K_S) \| E_{KS}(M \| E_{KSA}(H(M))) \| E_{KSCA}(T \| ID_A \| K_{PA})$

方法(1)主要适用于强调机密性,不需要数字签名的应用环境。该方法首先使用会话密钥K_S对消息进行加密,接着再使用接收方B的公钥对会话密钥进行加密,再将结果发送给

B。由于使用 B 的私钥进行加密，因此只有 B 才能恢复出会话密钥 K_S，并解密出消息 M。使用会话密钥 K_S 对消息 M 进行对称加密的原因是对称加密的效率比非对称加密的效率高得多，因此对消息 M 进行加密采用会话密钥 K_S 进行加密，而不直接使用 B 的公钥 K_{PB} 对消息进行加密。方法(2)主要强调的应用是使用数字签名。首先使用哈希函数对消息 M 产生散列值，然后使用 A 的私钥对消息进行签名，接收方 B 收到消息后使用 A 的公钥进行解密，并验证散列值即可知道消息是否来自 A，或者是否被篡改过。

方法(1)主要是保证消息的机密性，但没有保证消息的不可否认性和完整性。方法(2)保证了消息的不可否认性和完整性，但消息是以明文的形式传输，因此内容容易遭到窃取。方法(3)是对方法(1)和方法(2)的综合，将消息 M 和 A 的签名 $E_{KSA}(H(M))$ 放在一起使用会话密钥 K_S 进行加密，解决了方法(1)和方法(2)各自的不足。

在实际使用中，公钥通常以数字证书的形式发布，证书由证书权威机构(Certificate Authority，CA)颁发，证书中包含有公钥及有效期等数据。因此在实际使用中，通信各方需要获取的是对方当前尚未过期的公钥。$E_{KSCA}(T \| ID_A \| K_{PA})$ 是证书权威机构对 A 的公钥的签名，以确保 K_{PA} 是 A 当前有效的公钥。

习　题　3

在线测试

一、选择题

1. 身份认证是安全服务中的重要一环，以下关于身份认证的说法，错误的是(　　)。
   A. 身份认证是授权控制的基础
   B. 身份认证一般不用提供双向的认证
   C. 目前一般采用基于对称密钥加密或公开密钥加密的方法
   D. 数字签名机制是实现身份认证的重要机制
2. 数据完整性可以防止(　　)。
   A. 假冒源地址或用户的地址欺骗攻击
   B. 抵赖做过信息的递交行为
   C. 数据中途被攻击者窃听获取
   D. 数据中途被攻击者篡改或破坏
3. 数字签名要预先使用单向哈希函数进行处理的原因是(　　)。
   A. 多一道加密工序使密文更难破译
   B. 提高密文的计算速度
   C. 缩小签名密文的长度，加快数字签名和验证签名的运算速度
   D. 保证密文能正确地还原成明文
4. MD5 没有使用到(　　)运算。
   A. 幂　　B. 逻辑与或非　　C. 异或　　D. 移位
5. 以下关于安全散列算法的说法，错误的是(　　)。
   A. 它是一系列散列函数的统称　　B. SHA-1 生成的特征值长度为 160 位
   C. 生成的特征值通常称为摘要　　D. SHA-512 处理的分组长度为 512 位

二、填空题

1. MD5 和 SHA-1 产生的散列值分别是_____位和_____位。

2. 基于哈希链的口令认证,用户登录后将口令表中的(ID,$k-1$,H_{k-1}(PW))替换为_____。

3. Denning-Sacco 协议中使用时间戳 T 的目的是_____。

4. 在本章 3.5.4 节介绍的 Woo-Lam 协议中,第(6)、(7)步使用随机数 N_2 的作用是_____。

5. 在消息认证技术中,MD 算法可以用于为消息计算_____。

三、简答题

1. 弱抗碰撞性和强抗碰撞性有什么区别?

2. 什么是消息认证码?

3. 比较 MD5 和 SHA-1 的抗穷举攻击能力和运算速度。

4. MD5 和 SHA-1 的基本逻辑函数是什么?

5. Woo-Lam 协议一共 7 步,可以简化为以下 5 步。

(1) A→B:

(2) B→KDC:

(3) KDC→B:

(4) B→A:

(5) A→B:

请给出每步中传输的信息。

6. Needham-Schroeder 协议存在的一个致命漏洞是旧的会话密钥仍有价值,假设黑客 H 通过某种途径获得旧的密钥 K_S,H 就可以假装成 A 发起一次攻击。请说明 H 是在协议的哪一步开始发动攻击的,详细说明其过程(假设 H 能获得协议中每次传输的内容)。