应用密码学（第六讲）
— Hash函数

林东岱

信息安全国家重点实验室

2022年9月

Hash函数

Hash函数(也称散列函数或散列算法)的输入为任意长度的消息，而输出为某一固定长度的消息。即Hash函数是一种将任意长度的消息串M映射成一个较短的定长消息串的函数，记为H。称h=H(M)为消息M的Hash值或消息摘要（message digest），有时也称为消息的指纹。通常Hash函数应用于数字签名、消息完整性等方面。

设H是一个Hash函数，x是任意长度的二元串，相应的消息摘要为y=H(x)，通常消息摘要是一个相对较短的二元串（例如160比特）。假设我们已经计算出了y的值，那么如果有人改变x的值为x'，则通过计算消息摘要y'=hash(x')，验证y'与y不相等就可以知道原来的消息x已被改变。

通常，Hash函数可以分为两类：不带密钥的Hash函数和带密钥的Hash函数。不带密钥的Hash函数只需要有一个消息输入；带密钥的Hash函数规定要有两个不同的输入，即一个消息和一个秘密密钥。

6.1 Hash 函数的性质

6.1.1 Hash 函数的性质

Hash 函数的目的是为指定的消息产生一个消息“指纹”， Hash 函数通常具有以下这些性质：

（1）压缩性：Hash 函数将一个任意比特长度的输入 x 映射为固定长度为 n 的输出 H(x)。

（2）正向计算简单性：给定 Hash 函数 H 和任意的消息输入 x，计算 H(x)是简单的。

（3）逆向计算困难性：对所有预先给定的输出值，找到一个消息输入使得它的 Hash 值等于这个输出在计算上是不可行的。即对给定的任意值 y，求使得 H(x)=y 的 x 在计算上是不可行的。通常这一性质也称为 Hash 函数的单向性。

（4）弱无碰撞性：对于任何的输入，找到一个与它有相同输出的第二个输入,在计算上是不可行的。即给定一个输入 x，找到一个 x’，使得 H(x)=H(x’)成立是计算不可行的，如果单向 Hash 函数满足这一性质，则称其为弱单向 Hash 函数。

（5）强无碰撞性：找出任意两个不同的输入 x 与 x’，使得 H(x)=H(x’)在计算上是不可行的，如果单向 Hash 函数满足这一性质，则称其为强单向 Hash 函数。

攻击者可以对 Hash 函数发起两种攻击。第一种就是找出一个 x’，使得 H(x)=H(x’)。例如，在一个使用 Hash 函数的签名方案中，假设 s 是签名者对消息 x 的一个有效签名，s=sig(H(x))。攻击者

可能会寻找一个与 x 不同的消息 x'使得 H(x)=H(x')。如果能找到一个这样的 x'，则攻击者就可以伪造对消息 x'的签名，这是因为 s 也是对消息 x'的有效签名。Hash 函数的弱无碰撞性可以抵抗这种攻击。

攻击者可以发起另一种攻击。同样一个应用 Hash 函数的签名方案中，对手可能会寻找两个不同的消息 x 和 x'，使得 H(x)=H(x')。然后说服签名者对消息 x 签名，得到 s=sig(H(x))。由于 s=sig(H(x'))，所以攻击者得到了一个对消息 x'的有效签名。Hash 函数是强无碰撞性可以抵抗这种攻击。

6.1.2 生日攻击

第 1 类生日问题：假设已经知道 A 的生日为某一天，问至少有多少个人在一起时，至少有 1/2 的概率使有一个人和 A 的生日相同？在此，我们假定一年有 365 天，且所有人的生日均匀分布于 365 天中。下面我们求解所需的最少人数。

首先，有 1 人和 A 有相同生日的概率为 1/365,有不同生日的概率则为 1-1/365=364/365；K 个人与 A 生日不同的概率应为 $(364/365)^k$；K 个人至少有 1 个人与 A 的生日相同，且概率不小于 1/2 应为 $1-(364/365)^k \geq 1/2$，所以$(364/365)^k \leq 1/2$。即 $k \geq -\ln 2/\ln(364/365) \geq 0.6931471/0.027370 \geq 253$。

即至少为 253 人。若已知 A 的生日，则当至少有 253 个人时，才能保证有 1/2 的概率使有 1 人和 A 的生日相同。

那么有多少个人在一起，至少有 1/2 的概率存在两个人有相同的生日？这就是第 2 类生日问题。

第 2 类生日问题：假设一年有 365 天，每个人的生日均匀分布于 365 天，那么至少有多少个人在一起是，能保证至少有 1/2 的概率存在 2 个人有相同的生日。第 2 类生日问题也称生日悖论。

令 P_m 为 m 个人在一起，不存在相同生日的概率。根据假定，则 m-1 个人中无相同生日的概率为 P_{m-1}，m-1 个人共有生日 m-1 天。第 m 个人与前面 m-1 人无相同生日的概率为：

$$[365-(m-1)]/365=(366-m)/365$$

则可得递推关系：

$$P_m=(366-m)/365P_{m-1}$$

所以有：

$P_1=1$

$P_2=366-2/365P_1=364/365$

$P_3=363/365P_2=(364/365)\times(363/365)=(1/365^2)\times(364!/362!)$

$P_4=362/365P_3=(362/365)\times(1/365^2)\times(364!/362!)=(1/365^3)\times(364!/361!)$

……

$Pm=(1/365^{m-1})\times(364!/(365-m)!)$

可以验证当 $m\geqslant 23$ 时,$Pm<1/2$。即 23 个人在一起时，无相同生日的概率小于 1/2。反过来就是当 23 个让你在一起是，有两个人的生日相同的概率大于 1/2，结果有点出人意料。

那么基于这两类生日问题，下面我们介绍两类生日攻击：

（1）第 1 类生日攻击

如果已知一个 Hash 函数 H 有 n 个可能的输出，其中 H(x)是一个特定的输出。随机取 k 个输入，则至少有一个输入 y 使得 H(y)=H(x)的概率为 0.5 时，k 有多大？

第 1 类型生日攻击：与第 1 类生日问题相对比，称对 Hash 函数 h 寻找上述 y 的攻击为第 1 类型生日攻击。

因为 H 有 n 个可能的输出，所以输入 y 产生的输出 H(y)等于特定输出 H(x)的概率是 1/n，反过来说 H(y)≠H(x)的概率是 1-1/n。y 取 k 个随机值而函数的 k 个输出中没有一个等于 H(x),其概率等于每个输出都不等于 H(x)的概率之积，为$[1-1/n]^k$。所以 y 取 k 个随机值得到函数的 k 个输出中至少有一个等于 H(x)的概率为 $1-[1-1/n]^k$。

由$(1+x)^k \approx 1+kx$,其中$|x| \leq 1$,可得

$$1-[1-1/n]^k \approx 1-[1-1/n]=k/n$$

若使上述概率等于 0.5，则 k=n/2。特别地，如果 H 的输出为 m 比特，即可能的输出个数 $n=2^m$,则 $k=2^{m-1}$。假如 m=80，则 $k=2^{79}$，可见攻击付出的计算量是很大的。

（2）第 2 类生日攻击

第 2 类生日攻击：与第 2 类生日问题相类似，若一文件 m 的 Hash 值 H(m)为 32 比特，试问至少有多少的文件在一起，有两个文件的 Hash 值以至少 1/2 的概率相同。不同的是在这里将 365 改为 2^{32}。则有：

$Pm=((2^{32}-m+1)/2^{32})P_{m-1}$, $P_1=1$ 其中 Pm 表示 m 个长度为 32 比特的 0、1 符号串不存在两个相等的概率。类似可得

$$Pm=1/(2^{32})^{m-1}.(2^{32}-1)!/(2^{32}-m)!$$

验算可知 $m>2^{17}$(2^{17}=131072)时，Pm<1/2。即当有 2^{17} 个文件时，则存在两个文件有相同的 Hash 值输出的概率超过 1/2。对于现在的计算设备，2^{17} 的计算量不是困难的。

为防止攻击，通常的方法就是增加 Hash 值的比特长度，一般最小的可接受长度为 128 位（此时对于第二类生日攻击需要计算超过 2^{64} 个 Hash 值），例如 6.2 节将要介绍的 MD5 与 SHA 分别具有 128 比特和 160 比特的消息摘要。

6.1.3 迭代 Hash 函数的结构

目前使用的大多数Hash函数如MD5、SHA-1，其结构都是迭代型的，如图6.1所示。其中函数的输入M被分为L个分组$Y_0, Y_1, Y_2, \ldots\ldots, Y_{L-1}$，每一个分组的长度为b比特，如果最后一个分组的长度不够，需对其做填充。最后一个分组中还包括整个函数输入的长度值。这样，将使得攻击者的攻击更为困难，即攻击者若想成功地产生假冒的消息，就必需保证假冒消息的Hash值与原消息的Hash值相同，而且假冒消息的长度也要与原消息的长度相等。

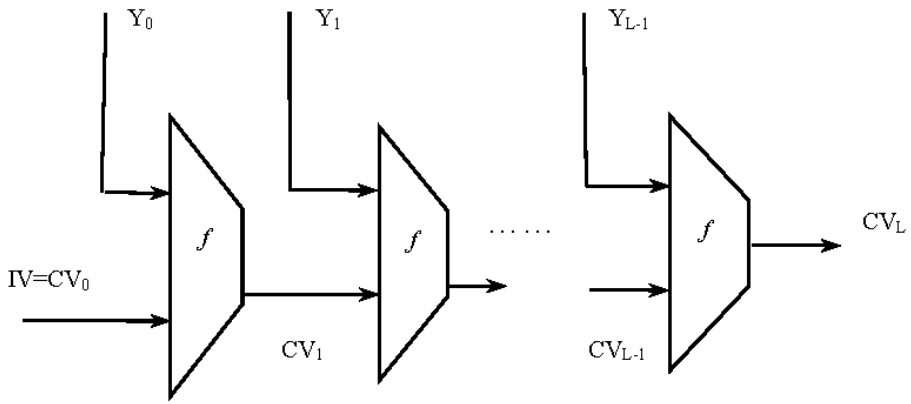

图 6.1 迭代型杂凑函数的一般结构

算法中重复使用函数f(称函数f为压缩函数)。f的输入有两项，一项是上一轮的输出n比特值CV_{i-1}，称为链接变量。另一项是算法在本轮的b比特输入分组Y_i。f 的输出为n比特值CV_i，CV_i又作为下一

轮的输入。算法开始时需对链接变量指定一个初始值IV，最后一轮输出的链接变量CV_L就是最终产生的Hash值。算法过程如下：

CV_0=IV =n比特长的初值；

$CV_i=f(CV_{i-1}, Y_{i-1})$,$1 \le i \le L$；

H(M)=CV_L，H(M)就是所产生的Hash值。

算法的核心技术是设计无碰撞的压缩函数f，而攻击者对算法的攻击重点是f 的内部结构。由于f 是压缩函数，其碰撞是不可避免的，因此在设计f 时就应保证找出其碰撞在计算上是不可行的。

6.2 Hash 函数实例

6.2.1 MD5 散列函数

MD5 是由 Ron Rivest 设计的单向散列函数。MD 表示消息摘要(Message Digest)，对输入的任意长度消息，算法产生 128 位的散列值（或消息摘要）。MD5 散列算法的前身是 MD4（ 1990 年由 Ron Rivest 提出），1992 年 4 月公布的改进后的 MD4 称为 MD5。

MD5 散列函数的处理过程分为如下几步：

（ 1 ）消息填充：对原始消息填充，使得其比特长在模 512 余 448，即填充后消息的长度为 512 的某一倍数减 64。这一步是必须的，即使原始消息的长度已经满足要求，仍需要填充。例如：消息的长度正好为 448 比特，则需要填充 512 比特，使其长度为 960 比特，因此填充的比特数在 1 到 512

之间。填充方式是固定的：第一位为 1，其它位为 0，例如需要填充 100 比特，则填充一个 1 和后面附上 99 个 0。

（2）添加消息长度：在第 1 步骤填充后，留有 64 个比特位，这 64 比特用来填充消息被填充前的长度。如果消息长度大于 2^{64}，则以 2^{64} 取模。

前两步完成以后，消息的长度为 512 的倍数（设倍数为 L），如图 6.2 所示填充后的消息。则可将消息表示成分组长为 512 的一系列分组 $Y_0, Y_1, \ldots\ldots, Y_{L-1}$。每一个 512 比特的分组是 16 个（32 比特的）字，因此消息中的总字数为 N=16 L。

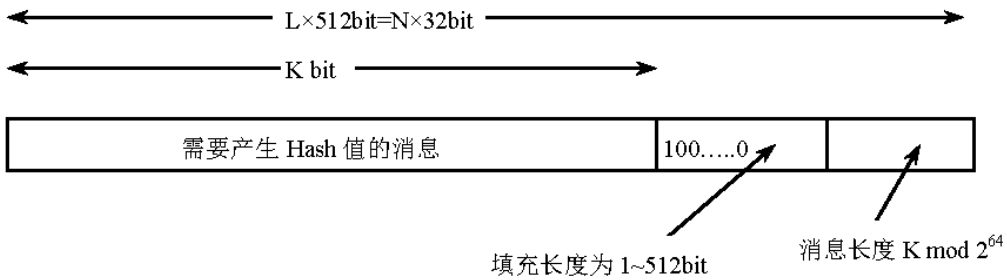

图 6.2 消息填充

（3）初始化 MD 缓冲区：MD5 算法使用 128 比特长的缓冲区以存储中间结果和最终 Hash 值。缓冲区可表示为4个32位长的寄存器(A，B，C，D)，将存储器初始化为以下的32位整数：A=67452301、B=EFCDAB、C=98BADCFE、D=10325476。而且每个寄存器都以 little-endian 方式存储数据，也就是最低有效字节存储在在低地址字节位置，4 个寄存器按如下存储：

A=01234567、B=89ABCDEF、C=FEDCBA98、D=7654321

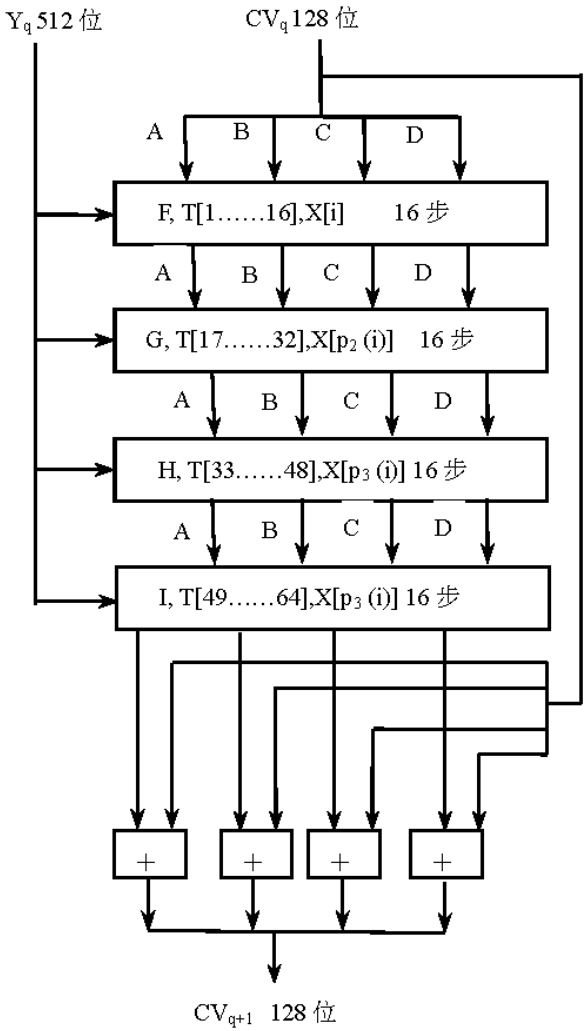
Y_q 512 位
CV_q 128 位
A B C D
F, T[1……16],X[i] 16 步
A B C D
G, T[17……32],X[p_2 (i)] 16 步
A B C D
H, T[33……48],X[p_3 (i)] 16 步
A B C D
I, T[49……64],X[p_3 (i)] 16 步
+ + + +
CV_{q+1} 128 位

图 6.3 MD5 的分组处理

（4）以分组为单位进行消息处理：每个分组 Y_q 都经过一个压缩函数 H_{MD5} 处理，包括 4 轮处理过程，如图 6.3 所示，MD5 算法是一种迭代型 Hash 函数，符合上节提到到的迭代型 Hash 函数的一般结构，压缩函数 H_{MD5} 是算法的核心。压缩函数 H_{MD5} 按如下方式工作：

①四个轮运算的结构相同，但各轮使用不同的基本逻辑函数，我们分别称之为 F、G、H 和 I。

②每轮的输入是当前要处理的 512 位的分组(Yq)和 128 位缓冲区的当前值 A、B、C、D 的内容，输出仍然放在缓冲区中以产生新的 A、B、C、D。

③每轮的处理过程还需要使用常数表 T 中元素的 1/4。第 4 轮的输出再与第 1 轮的输入 CV_q 相加，相加时将 CV_q 看作 4 个 32 比特的字，每个字与第 4 轮输出的对应的字按模 2^{32} 相加，相加的结果就是本轮压缩函数 H_{MD5} 的输出。

H_{MD5} 压缩函数要用到常数表 T，表 T 有 64 个元素，如表 6.1 所示，该表通过正弦函数构建。表中第 i 个元素 T[i]为 $2^{32}\times$abs(sin(i))的整数部分，其中 sin 为正弦函数，i 的单位为弧度。由于 abs(sin(i)) 大于 0 小于 1，所以 T[i]可由 32 比特的字来表示。这个表提供了一个随机化的 32bit 模式集，它将消除输入数据的任何规律性。

表 6.1 常数表 T

T[1]=D76AA478	T[2]=E8C7B756	T[3]=242070DB	T[4]=C1BDCEEE
T[5]=F57C0FAF	T[6]=4787C62A	T[7]=A8304613	T[8]=FD469501
T[9]=698098D8	T[10]=8B44F7AF	T[11]=FFFF5BB1	T[12]=895CD7BE
T[13]=6B901122	T[14]=FD987193	T[15]=A679438E	T[16]=49B40821

T[17]=F61E2562	T[18]=C040B340	T[19]=265E5A51	T[20]=E9B6C7AA
T[21]=D62F105D	T[22]=02441453	T[23]=D8A1E681	T[24]=E7D3FBC8
T[25]=21E1CDE6	T[26]=C33707D6	T[27]=F4D50D87	T[28]=455A14ED
T[29]=A9E3E905	T[30]=FCEFA3F8	T[31]=676F02D9	T[32]=8D2A4C8A
T[33]=FFFA3942	T[34]=8771F681	T[35]=699D6122	T[36]=FDE5380C
T[37]=A4BEEA44	T[38]=4BDECFA9	T[39]=F6BB4B60	T[40]=BEBFBC70
T[41]=289B7EC6	T[42]=EAA127FA	T[43]=D4EF3085	T[44]=04881D05
T[45]=D9D4D039	T[46]=E6DB99E5	T[47]=1FA27CF8	T[48]=C4AC5665
T[49]=F4292244	T[50]=432AFF97	T[51]=AB9423A7	T[52]=FC93A039
T[53]=655B59C3	T[54]=8F0CCC92	T[55]=FFEFF47D	T[56]=85845DD1
T[57]=6FA87E4F	T[58]=FE2CE6E0	T[59]=A3014314	T[60]=4E0811A1
T[61]=F7537E82	T[62]=BD3AF235	T[63]=2AD7D2BB	T[64]=EB86D391

（5）输出：消息的所有 L 个分组被处理完以后，最后一个 H_{MD5} 的输出即为产生的消息摘要（Hash 值）。图 6.4 为 MD5 的消息处理框图。

下面我们具体描述一下 MD5 的压缩函数 H_{MD5}：

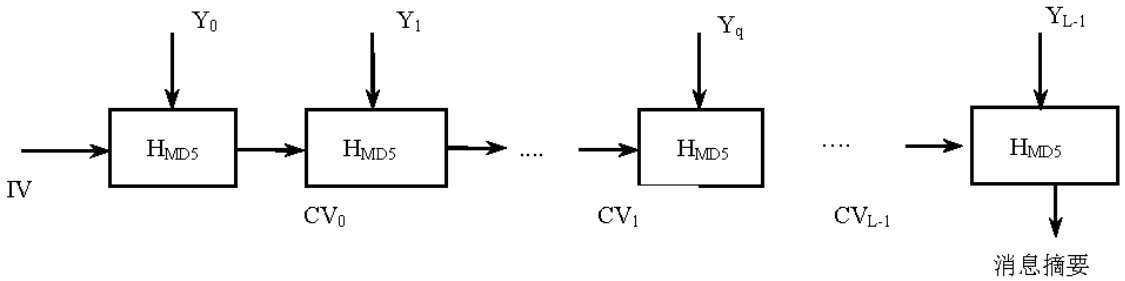

图 6.4 MD5 的消息处理框图

压缩函数 H_{MD5} 中有 4 轮处理过程，每轮又对缓冲区 A、B、C、D 进行 16 步迭代运算，每一步的运算形式为，如图 6.5 所示。

a<---b+(a+ g (b, c, d)+X[*k*]+T[*i*])<<<*s*

其中符号为：

a,b,c,d 表示缓存中四个字，在不同的步骤中有指名的顺序，在运算完成后再右循环一个字，即得这一步迭代的输出

g 表示函数 F,G,H,I 中的一个

<<<s 表示 32bit 的参数循环左移 s 个比特

X[*k*] 表示在第 q 个长度为 512 比特的分组中的第 *k* 个 32 比特的字

T[*i*] 表示常数表 T 中的第 i 个 32 比特的字

+ 表示模 2^{32} 的加法

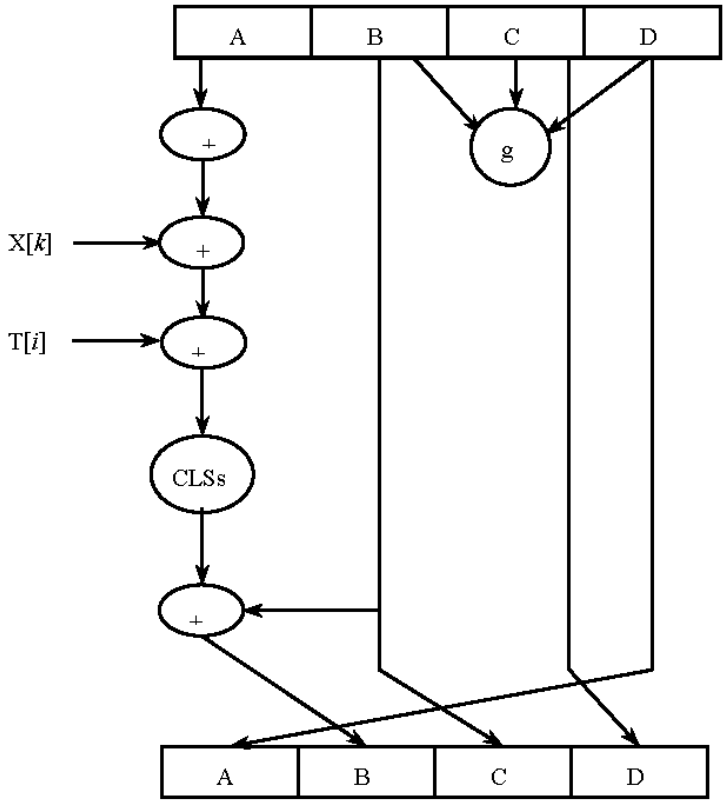

图 6.5

压缩函数中的一步迭代

MD5 压缩函数要经过 4 轮处理过程中，每轮又包括 16 步，且每轮以不同的次序使用 16 个字。其中在第一轮以字的初始次序使用，第二轮到第四轮，分别对字的次序 i 做置换后得到一个新次序，然后以新次序使用 16 个字。三个置换分别为：

$P_2(i)=(1+5i) \bmod 16$

$P_3(i)=(5+3i) \bmod 16$

$P_4(i)=7i \bmod 16$

4 轮处理过程中分别使用基本的逻辑函数 F、G、H、I。每个逻辑函数的输入为 3 个 32 比特的字，输出是一个 32 比特的字，其中的运算为逐比特的逻辑运算，四个逻辑函数的定义如下：

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

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

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

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

其中：$\wedge$是与，$\vee$是或，$\oplus$是异或，$\neg$是反

4 轮处理过程中，每一轮的 16 步都一个循环左移，移动的位数用 s 表示，如表 6.2 所示。

表 6.2 压缩函数每步左循环移位的位数

步数 / 轮数	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
一	7	12	17	22	7	12	17	22	7	12	17	22	7	12	17	22
二	5	9	14	20	5	9	14	20	5	9	14	20	5	9	14	20
三	4	11	16	23	4	11	16	23	4	11	16	23	4	11	16	23
四	6	10	15	21	6	10	15	21	6	10	15	21	6	10	15	21

如果设 M_j 表示消息的 512 比特的分组中的第 j 个 32 比特的字（$0 \leq j \leq 15$），<<<s 表示循环左移位 s 位，则对应于 4 轮中的 4 种操作：

FF(a, b, c, d, M_j, s, T[i])表示 a=b+((a+(F(b,c,d)+M_j+ T[i])<<<s)

GG(a, b, c, d, M_j, s, T[i])表示 a=b+((a+(G(b,c,d)+M_j+ T[i])<<<s)

HH(a, b, c, d, M_j, s, T[i])表示 a=b+((a+(H(b,c,d)+M_j+ T[i])<<<s)

I I(a, b, c, d, M_j, s, T[i]) 表示 a=b+((a+(I(b,c,d)+M_j+ T[i])<<<s)

则 MD5 压缩函数 H_{MD5} 的四轮可表示为:

第一轮的 16 步:

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)

第二轮的 16 步：

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)

第三轮的 16 步：

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)

第四轮的 16 步：

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)

6.2.2 安全 Hash 算法

安全 Hash 算法 (SHA)是美国国家标准技术研究所(NIST)设计，并于 1993 年作为联邦信息处理标准(FIPS 180)发布。安全 Hash 标于 1993 年 5 月 11 日正式公布后，NIST 又对其做了一点修改。1995 年 4 月 17 日，NIST 公布了修改后的 SHA 算法，通常称之为 SHA-1。SHA 算法基于 MD4，其结构与 MD4 非常相近。

SHA-1 算法的步骤：

SHA-1 算法的输入为小于 2^{64} 比特长的任意消息，输出为 160 比特长的消息摘要，而 MD5 的输入没有长度限制，且输出为 128 比特。与 MD5 算法处理消息的过程一样，SHA-1 算法也将消息按 512 位分组，但 hash 值的长度和链接变量的长度为 160 比特。图 6.6 为其消息处理框图。

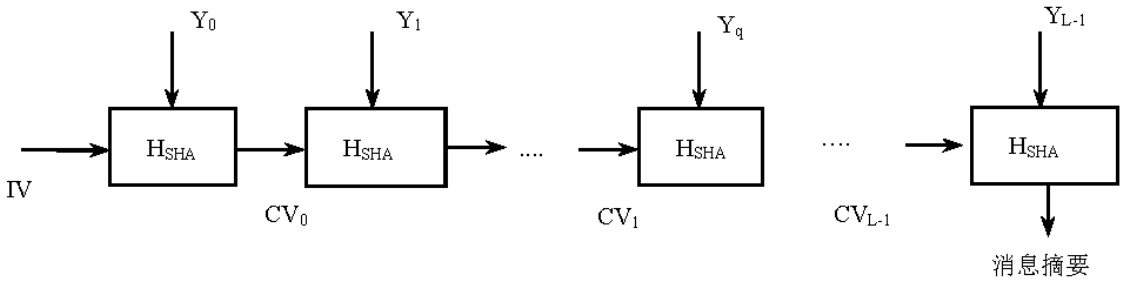

图 6.6 SHA-1 消息处理框图

SHA-1 的算法过程分为如下几步：

（1）消息填充：与 MD5 的消息填充相同。使得填充后消息的比特长为 512 的某个倍数减 64，

同样填充的比特数在 1 到 512 之间。填充的方式也是 100000....0000 的形式。

（2）附加消息的长度：与 MD5 的第 2 步类似，将填充前消息的长度的二进制表示，填充在第 1 步留出的 64 比特中。第 2 步完成后，消息的长度为 512 的倍数（设倍数为 L），同时消息可表示成分组长为 512 的一系列分组 $Y_0, Y_1, \ldots\ldots, Y_{L-1}$。其消息的填充图和 MD5 的填充图相同。

（3）初始化 MD 缓冲区：SHA-1 使用 160 比特长的缓冲区存储中间结果和最终 Hash 值，缓冲区可表示为五个 32 比特长的寄存器(A、B、C、D、E)，分别将其初始化为 A=67452301，B=EFCDAB89，C=98BADCFB，D=10325476，E=C3D2E1F0。其中，前四个值与 MD5 的值相同，但在 SHA-1 中这些值以 big-endian 的方式存储，也就是字的最高有效字节存于低地址字节位置，即按如下方式存储上述的值：

A=67452301，B=EFCDAB89，C=98BADCFB，D=10325476，E=C3D2E1F0

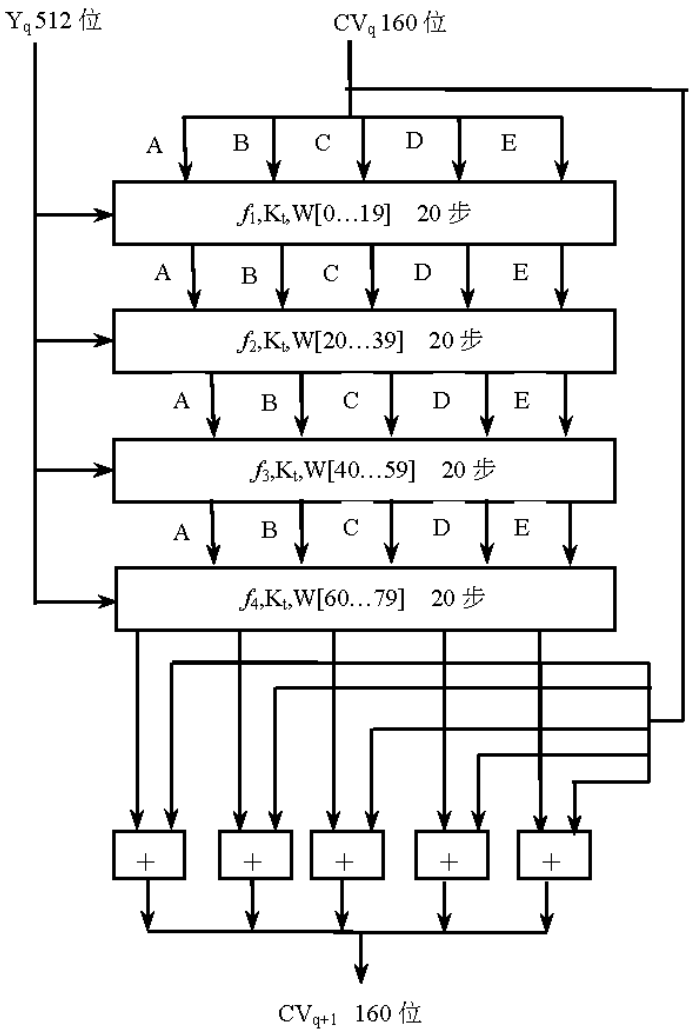
Y_q 512 位
CV_q 160 位
A
B
C
D
E
f_1,K_t,W[0…19] 20 步
f_2,K_t,W[20…39] 20 步
f_3,K_t,W[40…59] 20 步
f_4,K_t,W[60…79] 20 步
+
CV_{q+1} 160 位

图 6.7　SHA 的分组处理框图

（3）以分组为单位对消息进行处理：每一分组 Y_q 都经过压缩函数处理，压缩函数由 4 轮处理过程构成，如图 6.7，每一轮又由 20 步迭代组成。4 轮处理过程结构一样，但所用的基本逻辑函数不同，分别表示为 f_1、f_2、f_3、f_4。

SHA-1 每轮的输入为当前处理的消息分组 Y_q 和缓冲区 A、B、C、D、E 的当前值，输出仍放在缓冲区以替代 A、B、C、D、E 的旧值。每轮处理过程还需加上一个加法常量 K_t，其中 $0 \le t \le 79$，t 表示迭代的步数。

第 4 轮的输出（即第 80 步迭代的输出）再与第一轮的输入 CV_q 相加，以产生 CV_{q+1}，其中加法是缓冲区 5 个字中的每一个字与 CV_q 中相应的字模 2^{32} 相加。

（4）输出消息的 L 个分组都被处理完后，最后一个分组的输出就是 160 比特的消息摘要。

SHA-1 的压缩函数：

SHA 的压缩函数由 4 轮处理过程组成，每轮处理过程又由对缓冲区 ABCDE 的 20 步迭代运算组成，每一步迭代运算的形式如图 6.8：

$A,B,C,D,E \leftarrow (E+f_t(,B,C,D)+CLS(A),W_t,K_t),A,CLS_{30}(B),C,D$

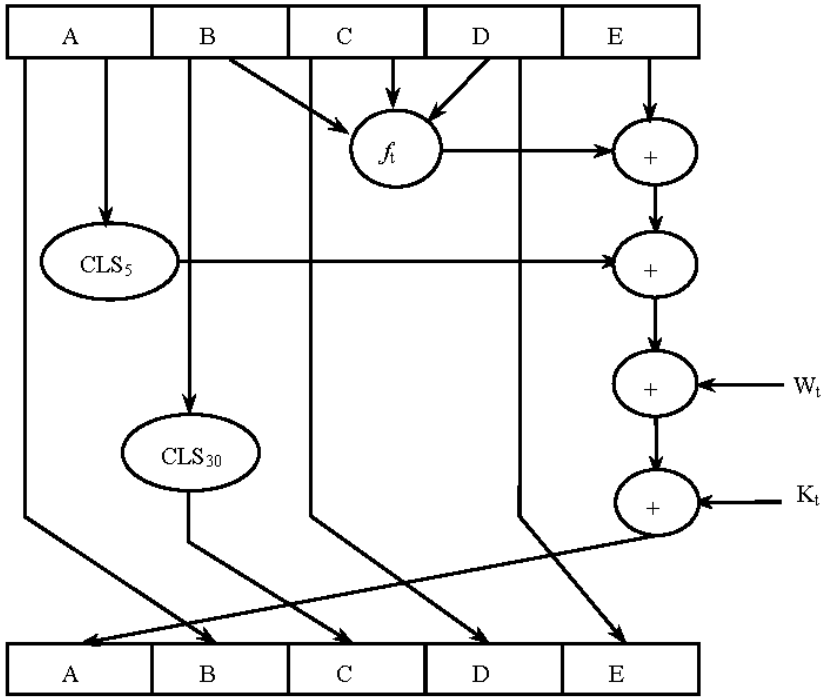

图 6.8 SHA 的压缩函数中一步迭代示意图

其中 A、B、C、D、E 为缓冲区的五个字，t 是迭代的步数（$0 \le t \le 79$），f_t 是第 t 步迭代使用的基本逻辑函数，CLS_s 为左循环移 s 位，Kt 是加法常量，+是模 2^{32} 加法，Wt 是由当前 512 比特长的分组导出的一个 32 比特长的字。

基本逻辑函数包括前面提到的函数 f_1, f_2, f_3, f_4 其定义分别为：

$f_1(X,Y,Z)=(X\wedge Y)\ \vee(\neg X\wedge Y)$

$f_2(X,Y,Z)=X\oplus Y\oplus Z$

$f_3(X,Y,Z)=(X\wedge Y)\ \vee(X\wedge Z)\ \vee(Y\wedge Z)$

$f_4(X,Y,Z)=f_2(X,Y,Z)=X\oplus Y\oplus Z$

其中：$\wedge$是与，$\vee$是或，$\oplus$是异或，$\neg$是反。四轮迭代共需要 80 个常量，实际上只有 4 个不同取值，如表 6.3 所示。

SHA-1 处理每个 512 的分组都需要 80 个字，前面的 16 个字 $W_0,W_1,W_2,\ldots\ldots,W_{15}$，直接从输入的分组中得到，其余的值由公式 $Wt=CLS_1(W[t-3]\oplus W[t-8]\oplus W[t-14]\oplus W[t-16])$扩展得到。相比 MD5，其直接使用一个消息分组的 16 个字作为每不迭代的输入，而 SHA-1 则将输入分组的 16 个字扩展成 80 个字以供函数使用，从而使寻找具有相同压缩值得不同消息分组更加困难。

表 6.3 SHA-1 的加法常量

K_1	0≤t≤19	5a827999
K_2	20≤t≤39	6ed9eba1
K_3	40≤t≤59	8f1bbcdc
K_4	60≤t≤79	ca62c1d6

综合上述内容可以将 SHA-1 的处理过程描述为：

第一轮：0≤t≤19，常数 K_1=5a827999

$TEMP=(A<<5)+f1(B,C,D)+E+W[t]+K_1$

$E=D,D=C,C=(B<<30),B=A,A=TEMP$

第二轮：20≤t≤39，常数 K_2=6ed9eba1

TEMP=(A<<5)+$f2$(B,C,D)+E+W[t]+K_2

E=D,D=C,C=(B<<30),B=A,A=TEMP

第三轮：40≤t≤59，常数 K_3=8f1bbcdc

TEMP=(A<<5)+$f3$(B,C,D)+E+W[t]+K_3

E=D,D=C,C=(B<<30),B=A,A=TEMP

第四轮：60≤t≤79，常数 K_4=ca62c1d6

TEMP=(A<<5)+$f4$(B,C,D)+E+W[t]+K_4

E=D,D=C,C=(B<<30),B=A,A=TEMP

6.3 Hash 函数的应用举例

Hash 函数可以用于认证，如图 6.9 所示的两种情况：

（a）使用分组密码算法仅对 Hash 值进行加密，即 E_K[H(M)]是变长消息 M 和密钥 K 的函数值，且它是一个定长的输出，对不知道该密钥的攻击者来说是安全的。

（b）该情况假定通信双方共享一个秘密值 S。发送方 A 对消息 M 和秘密值 S 算出 Hash 值，并将得出的 Hash 值附加在消息 M 后。因为秘密值本身并不被发送，攻击者无法更改中途截获的消息，也就无法产生假消息。

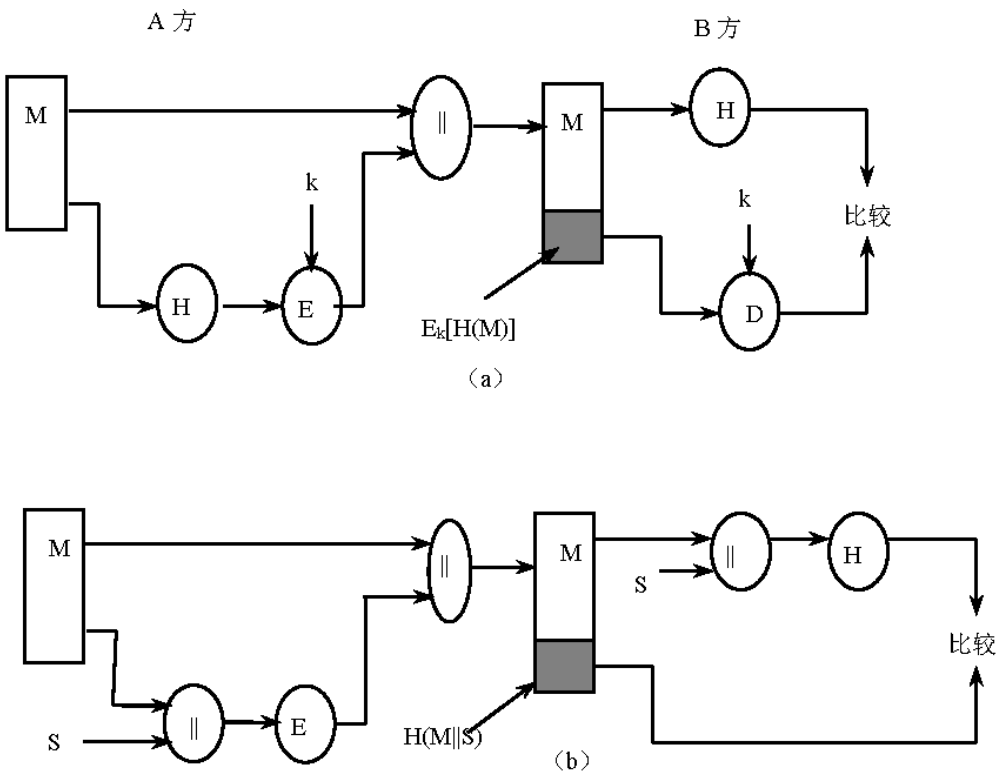

图 6.9 Hash 函数用于消息认证

应用密码学（第七讲）
— 消息认证码

林东岱

信息安全国家重点实验室

2021年9月

消息认证码

MAC 全称为Message Authentication Code（消息认证码），是用来保证数据完整性的一种工具。

数据完整性是信息安全的一项基本要求，它可以防止数据未经授权被篡改。随着网络技术的不断进步，尤其是电子商务的不断发展，保证信息的完整性变得越来越重要。特别是双方在一个不安全的信道上通讯的时候，就需要有一种方法能够保证一方所发送的数据能被另一方验证是正确的，即能够防止数据未经授权被篡改。 如果用数学语言来描述，MAC 实质上是一个双方共享的密钥k和消息m作为输入的函数，如将函数值记为MAC k (m)，这个函数值就是一个认证标记，这里用 δ 表示。攻击者发起攻击的时候，能得到的是消息和标记的序列对(m1, δ 1)，(m2, δ 2)，…， (mq, δ q)（其中 δ i=MACk(mi)）。如果攻击者可以找到一个消息m，m不在m1,…,mq之中，并且能够得到正确的证标记 δ =MAC(m)就说明攻击成功了。攻击者成功的概率就是其攻破 MAC 的概率。

7.1 消息认证码的构造

消息认证码的构造有多种方法，我们主要介绍两种构造方法：一种是基于分组密码的，另一种是基于带密钥的 Hash 函数的。

7.1.1 基于分组密码的 MAC

1、CBC-MAC

CBC-MAC 是最为广泛使用的消息认证算法之一同时它也是一个 ANSI 标准（X9.17）。CBC-MAC 实际上就相当于对消息使用 CBC 模式进行加密，取密文的最后一块作为认证码。

当取 DES 作为加密的分组密码时，称为基于 DES 的 CBC-MAC，若需要产生认证码的消息为 x，加密的 DES 密钥为 k，则生成 MAC 的过程如下（如图 7.1 所示）：

（1）填充和分组。对消息 x 进行填充，将填充得到的消息分成 t 个 n 比特（基于的 DES 的 CBC-MAC 通常 n=64）的分组，记为 $x_1,x_2,\ldots,x_t$。

（2）密码分组链接。令 E_k 表示以 k 为密钥的加密算法 DES，用以下方式计算 Hi:

$$H_1 \longleftarrow\!\!\!- E_k(x_1)$$

$$H_i \longleftarrow\!\!\!- E_k(H_{i-1} \oplus x_i) \quad , \ 2 \le i \le t$$

则 H_t 就是 x 的消息认证码，基于的 DES 的 CBC-MAC 的 MAC 为 64 比特。

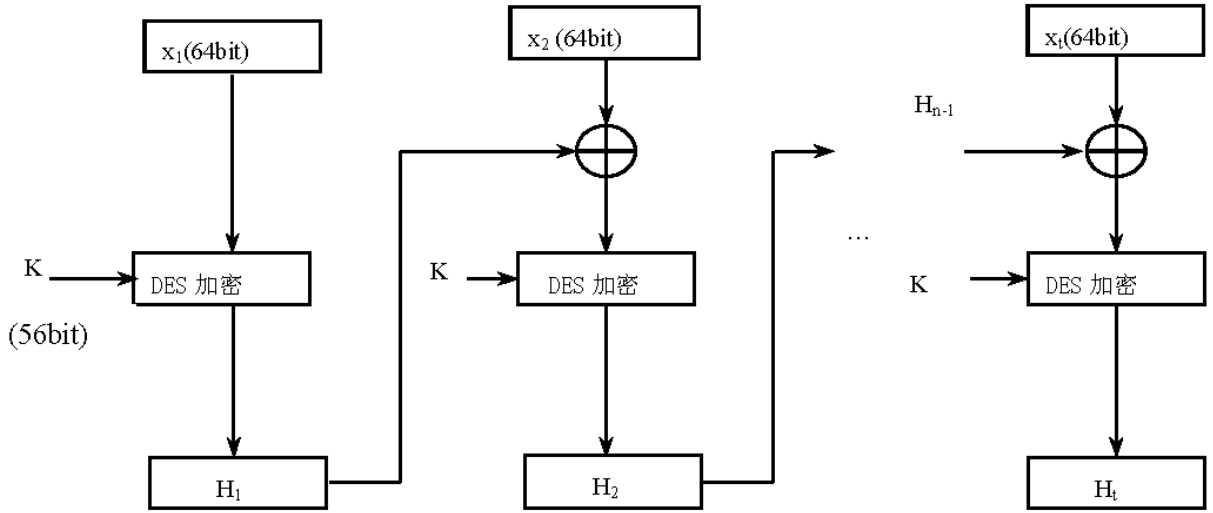

图 7.1 CBC-MAC 的算法过程

CBC-MAC 是一种经典的构造 MAC 的方法，构造方法简单，且底层的加密算法具有黑盒性质，可以方便的进行替换。

2、XOR-MAC

XOR-MAC 有两种方式：无状态（XMACR）和有状态（XMACC）。这种算法在计算过程中引入索引值使得分组密码每次加密的明文各不相同，最后再将所有的密文异或。具体的构造方法描述如下。

假定$|x|$代表消息x 的长度（即包含多少位），并且它是32 的倍数。$x=(x_1,x_2,\ldots,x_n)$ 其中$|x_i|=32$，i=1,..,n。假定n 小于2^{31}。<i>是数字i的长度为b的二进制表示，代表块的索引号。发送者维持一个长度为63 位的记数r，在XMACC 模式下它的初始值为0，每次增加1。在XMACR 模式下，r 是随机选取的一个长度为63 位的串。他们的具体构造方式如下：

$$
\begin{array}{ll}
function\ \ XMAR(x,k) & function\ XMACC(x,k) \\
pad(x)\ r \xleftarrow{R} \{0,1\}^{63} & pad(x)\ ctr \longleftarrow ctr+1 \\
y_0 = F_k(0\|r) & y_0 = F_k(0\|ctr) \\
partition\ \ x\ into\ x_1,x_2,...x_n & partition\ x\ into\ x_1,....x_n \\
for\ i=1\ to\ n & for\ i=1\ to\ n \\
y_i = y_i - 1 \oplus F_k(1\|<i>\|x_i) & y_i = yi-1 \oplus F_k(1\|<i>\|x_i) \\
return(r,y_n) & return\ (ctr,y_n)
\end{array}
$$

7.1.2 基于带密钥的 Hash 函数的 MAC

前面介绍的是两种基于分组密码的消息认证码算法。下面给出一个由 Hash 函数导出的 MAC。可以将 HMAC 表示如下：

HMACK=H[$(K^+ \oplus$opad)||H[$(K^+ \oplus$ipad)||M]]，HAMC 算法过程如下：如图 7.2。

（1）对密钥 K 的左端填充一些 0 生成一个 b 比特的串 K^+（例如，如果 K 的长度是 160 比特，

而 b=512，那么对 K 填充 44 个 0 字节 0x00）。

（2）将 K^+与 ipad 按比特异或产生一个 b 比特的分组 S_i

（3）将消息 M 附加到 S_i 后

（4）使用 H 计算第 3 步产生结果的散列值

（5）将 K^+与 opad 按比特异或产生一个 b 比特的分组 S_0

（6）将第 4 步产生的散列值附加到 S_0 后面

（7）使用 H 计算第 6 步产生结果的散列值，并输出这个结果

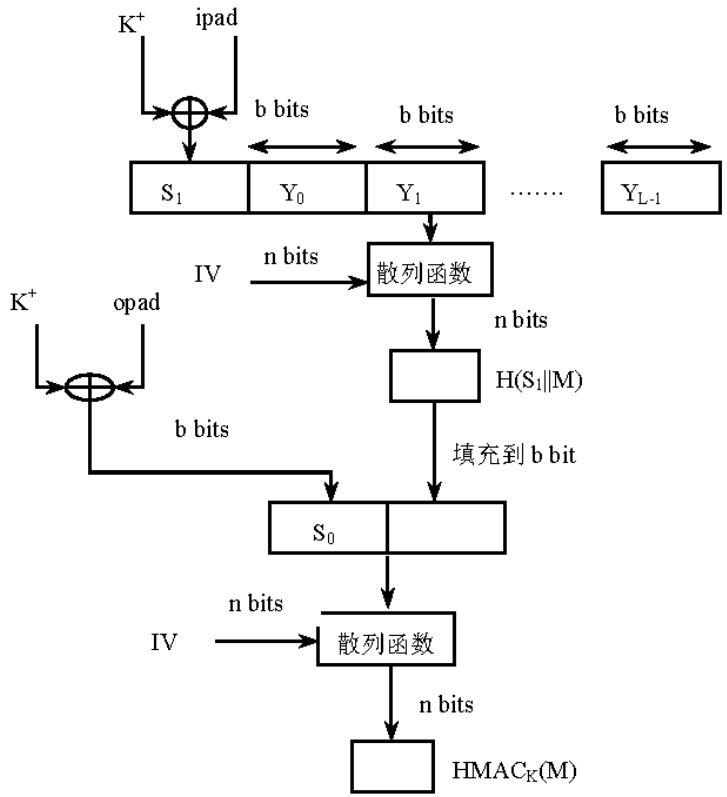

图 7.2 HAMC 的结构

其中：

H=嵌入的散列函数(如 MD5,SHA-1)

M=HMAC 的输入消息（包括嵌入散列函数所需的填充比特）

Y_i=M 中的第 i 个分组，$0 \le i \le L-1$

L=M 中的分组数

b= 一个分组的比特数

n=嵌入散列函数产生的散列码长度

K=密钥；如果密钥的长度大于 b，该密钥输入散列函数产生一个 n 比特的密钥；一般情况下密钥长度大于等于 n

K^+=在 K 的左边填充 0,使总长度等于 b

ipad=将 00110110 重复 b/8 次

opad=将 01011010 重复 b/8 次

7.2 MAC 函数的安全性

当使用加密算法加密消息时，无论采用对称加密或不对称加密，其安全性一般依赖于密钥的比特长度。排除加密算法可能有的弱点，攻击者可以使用穷举攻击法来尝试所有可能的密钥。一般情况下，对 k 比特长的密钥，穷绝攻击需要 $2^{(k-1)}$次尝试。特别是对于惟密文攻击(ciphertext-only attack)，攻击者需要通过获得的密文 C，尝试所有可能的密钥，直到产生适当的明文为止。

对于 MAC，情况则完全不同。MAC 函数一般是一个多对一的函数。攻击者如何用穷举攻击法获得密钥呢?MAC 函数域由任意长度的消息组成，其中域值由所有可能的 MAC 和所有可能的密钥

组成。如果使用一个长度为 n 比特的 MAC，那么将有 2^n 个可能的 MAC，而可能有 N 个消息，其中 $N>>2^n$。此外，对一个长度为 k 的密钥，还将有 2^k 个可能的密钥。

如果不考虑保密性，就是假设对手已获得消息的明文和相应的 MAC。假定 k>n，即密钥长度大于 MAC 长度。那么如果已知 M_1 和 MAC_1，$MAC_1=C_{k1}(M_1)$。密码分析者必须对所有可能的密钥值 K_i 执行 $MAC_i=C_{ki}(M_i)$。当 $MAC_i=MAC_1$ 时，则至少找到一个密钥。注意，由于总共将产生 2^k 个 MAC，但只有 2^n 个不同的 MAC 值。因此，许多密钥能产生同一个正确的 MAC，但攻击者却无法确认真正的正确密钥。平均说来，总共 $2^k/2^n=2(k-n)$个密钥将产生一个正确的 MAC。因此攻击者必须重复这样的攻击：

第 1 轮：已知 M_1, MAC_1,其中 $MAC_1=C_k(M_1)$，对所有可能的密钥计算 $MAC_i=C_{ki}(M_1)$，将得到 $2^{(k-n)}$个可能的密钥，

第 2 轮：已知 M_2, $MAC_2=C_k(M_2)$，对上一轮得到的 $2^{(k-n)}$个密钥计算 $MAC_i=C_{ki}(M_2)$，得到 $2^{(k-2n)}$ 个可能的密钥。

如此下去，如果 k=an，那么需要重复进行 a 轮。例如，如果使用 80bit 的密钥和产生 32bit 的 MAC，那么第 1 轮将产生 2^{48} 可能的密钥。第 2 轮将可能的密钥缩减为 2^{16}。第 3 轮仅产生惟一的一个密钥，一定是发送方使用的那个密钥。

如果密钥的长度小于或等于 MAC 的长度。那么在第 1 轮中就有可能找到正确的密钥，也有能找出多个可能的密钥，如果是后者，则需要执行第 2 轮的搜索。

所以对消息认证码的穷举搜索攻击比对使用相同长度密钥的加密算法的穷举搜速更困难，然而有些攻击方法却不需要寻找产生 MAC 所使用的密钥。

例如：令 $M=(X_1||X_2||.....||X_m)$为由 64bit 分组 Xi 串接而得到的消息。其消息认证码由以下方

式得到：

$\Delta(M)=X_1 \oplus X_2 \oplus \ldots\ldots \oplus X_m$

$C_k(M)=E_k[\Delta(M)]$

其中⊕是异或运算符。而加密算法采用电子密码本模式的 DES 算法。因此，密钥长度是 56 比特，MAC 长度是 64 比特。如果攻击者能窃取到$\{M||C_k(M)\}$，通过穷举攻击来确定 K 将至少需要 2^{56} 次加密。但攻击者也可以用以下方式攻击系统：通过用任意的值 Y_1 到 Y_{m-1} 来替代从 X_1 到 X_{m-1} 的值，并用 Y_m 来替代 X_m，其中 Y_m 计算如下：

$$Y_m=Y_1 \oplus Y_2 \oplus \ldots\ldots \oplus Y_{m-1} \oplus \triangle(M)$$

现在，攻击者可以成功伪造一个新的消息 $M'= Y_1 \oplus Y_2 \oplus \ldots\ldots \oplus Y_{m-1} \oplus Y_m$， M'的 MAC 与原消息的 MAC 相同。

考虑能遇到的攻击的类型，MAC 函数必须满足以下要求。其中，假定对手知道 MAC 函数 C 但不知道 K：

（1）如果攻击者得到 M 和 $C_k(M)$，构造一个消息 M'，使得 $C_k(M')=C_k(M)$，在计算上是不可行的；

（2）$C_k(M)$应该在以下意义下均匀分布。随机选择消息 M 和 M'，$C_k(M')=C_k(M)$的概率为 2^{-n}，其中 n 为 MAC 的比特长度。

（3）若 M'为 M 的某种已知变换，即 M'=f(M)，（例如，f 可能为将一个或多个特定的比特取反）那么，$p[C_k(M)=C_k(M')]=2^{-n}$。

第 1 个需求是针对上述的例子，即在攻击者不知道密钥的情况下，而伪造一个新消息与截获的 MAC 匹配在计算上是不可行的；

第 2 个需求是为了阻止基于选择明文穷举攻击的情况：假定攻击者不知道密钥 K，但能获得 MAC 函数，能对消息产生 MAC。那么攻击者可以对各种消息计算 MAC，知道找到与给定 NAC 相同的消息位置。如果 MAC 函数呈现均匀分布，那么用穷举攻击方法平均需要 $2^{(n-1)}$次尝试才能找到匹配给定 MAC 的消息。

第 3 个需求说明消息认证算法对消息的特定部分或比特不应该比别的更脆弱。否则攻击者获得 M 和 MAC 后又可能修改 M 中较弱的部分，从而伪造出一个与原 MAC 匹配的新消息。

7.3 消息认证码的应用

假定通信的双方为 A 和 B，有一个共享密钥 K。当 A 有要发往 B 的消息时，发送方 A 计算 MAC（MAC 是待发送的消息和密钥 K 的一个函数值，$MAC=C_k(M)$），然后将消息和 MAC 一起发往预定的接收者 B。在接收端 B，使用相同的密钥 K 对收到的消息执行相同的计算并得出新的 MAC，将收到的 MAC 与计算得出的 MAC 进行比较，如图 7.3 所示。如果假定只有接收方和发送方知道密钥 K，同时如果收到的 MAC 与计算得出的 MAC 匹配，那么：

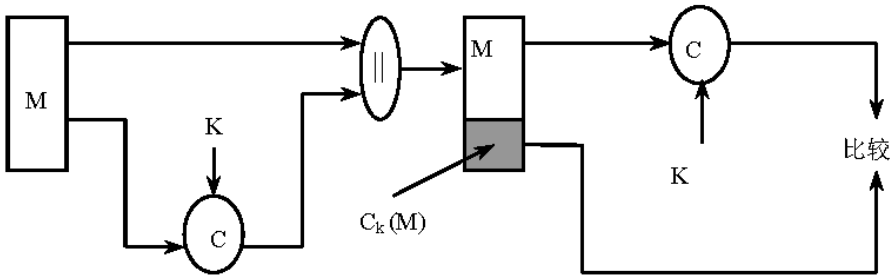

图 7.3 消息认证码实现认证

（1）接收者确信消息未被更改过。如果一个攻击者更改消息而未更改 MAC，那么接收者计算出的 MAC 将不同于接收到的 MAC。因为假定攻击者不知道该密钥，因此攻击者不可能更改 MAC 来对应更改后的消息。

（2）接收者确信消息来自所谓的发送者。因为没有其他人知道该密钥，因此没有人能够为一个消息准备合适的 MAC。

Thank you!

(To be continued....)