第 3 章　消息鉴别与数字签名

经典的密码学是关于加密和解密的理论，主要用于保密。目前，密码学已经得到了更加深入、广泛的发展和应用，不再局限于单一的加解密技术，而是被有效、系统地用于保证电子数据的机密性、完整性和真实性。这是因为，在公开的计算机网络环境中，传输中的数据可能遭到的威胁不仅仅局限于泄密，而是多种形式的攻击：

(1) **泄密**：消息的内容被泄漏给没有合法权限的任何人或过程。

(2) **通信业务量分析**：分析通信双方的通信模式。在面向连接的应用中，确定连接的频率和持续时间；在面向连接或无连接的环境中，确定双方的消息数量和长度。

(3) **伪造消息**：攻击者假冒真实发送方的身份，向网络中插入一条消息，或者假冒接收方发送一个消息确认。

(4) **篡改消息**：分成以下三种情形。

① **内容篡改**：对消息内容的修改，包括插入、删除、调换和修改。

② **序号篡改**：在依赖序号的通信协议(如 TCP)中，对通信双方消息序号进行修改，包括插入、删除和重新排序。

③ **时间篡改**：对消息进行延时和重放。在面向连接的应用中，整个消息序列可能是前面某合法消息序列的重放，也可能是消息序列中的一条消息被延时或重放；在面向无连接的应用中，可能是一条消息(如数据报)被延时或重放。

(5) **行为抵赖**：发送方否认发送过某消息，或者接收方否认接收到某消息。

对抗前两种攻击的方法属于消息保密性范畴，前面讲过的对称密码学和公钥密码学，都是围绕这个主题展开的；对付第(3)种和第(4)种攻击的方法一般称为消息鉴别；对付第(5)种攻击的方法属于数字签名。一般而言，数字签名方法也能够抗第(3)种和第(4)种中的某些或全部攻击。

3.1　消 息 鉴 别

完整性是安全的基本要求之一。篡改消息是对通信系统进行主动攻击的常见形式，被篡改的消息是不完整的；信道的偶发干扰和故障也破坏了消息的完整性。接收者应该能够检查所收到的消息是否完整。另外，攻击者还可以将一条声称来自合法授权用户的虚假消息插入网络，或者冒充消息的合法接收者发回假确认。因此，消息接收者还应该能够识别收到的消息是否确实来源于该消息所声称的主体，即验证消息来源的真实性。

保障消息完整性和真实性的重要手段是消息鉴别技术。

3.1.1　消息鉴别的概念

消息鉴别也称“报文鉴别”或“消息认证”，是一个对收到的消息进行验证的过程，验证的

内容包括两个方面:

(1) 真实性:信息的发送者是真正的而不是冒充的;

(2) 完整性:消息在传送和存储过程中未被篡改过。

从功能上看,一个消息鉴别系统可以分成两个层次,如图 3-1 所示。

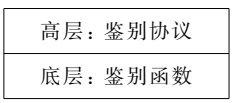

图 3-1 消息鉴别系统的功能分层结构

底层是一个鉴别函数,其功能是产生一个鉴别符,鉴别符是一个用来鉴别消息的值,即鉴别的依据。在此基础上,上层的鉴别协议调用该鉴别函数,实现对消息真实性和完整性的验证。鉴别函数是决定鉴别系统特性的主要因素。

根据鉴别符的生成方式,鉴别函数可以分为如下三类:

(1) **基于消息加密**:以整个消息的密文作为鉴别符。

(2) **基于消息鉴别码**:利用公开函数+密钥产生一个较短的定长值作为鉴别符,并与消息一同发送给接收方,实现对消息的验证。

(3) **基于散列函数**:利用公开函数将任意长的消息映射为定长的散列值,并以该散列值作为鉴别符。

目前,像对称加密、公钥加密等常规加密技术已经发展得非常成熟。但是,出于多种原因,常规加密技术并没有被简单地应用到消息鉴别符的生成中,实际应用中一般采用独立的消息鉴别符。用避免加密的方法提供消息鉴别符受到广泛的重视,而最近几年消息鉴别的热点转向由 Hash 函数导出 MAC 的方法。

3.1.2 基于 MAC 的鉴别

1. 消息鉴别码原理

消息鉴别码(Message Authentication Code,MAC)又称密码校验和(Cryptographic Checksum)。其实现鉴别的原理是:用公开函数和密钥生成一个固定大小的小数据块,即 MAC,并将其附加在消息之后传输。接收方利用与发送方共享的密钥进行鉴别。基于 MAC 提供消息完整性保护,MAC 可以在不安全的信道中传输,因为 MAC 的生成需要密钥。

基于 MAC 的鉴别原理见图 3-2。假定通信双方,比如 A 和 B,共享密钥 K。若 A 向 B 发送消息,则 A 计算 MAC,它是消息和密钥 K 的函数,即 $MAC=C(K,M)$,其中:

M——输入消息;

C——MAC 函数;

K——共享的密钥;

MAC——消息鉴别码。

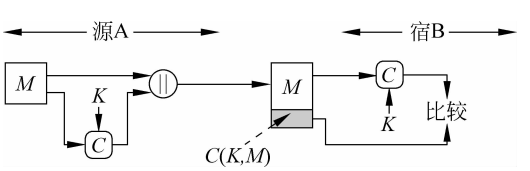

图 3-2 MAC 鉴别原理

消息和 MAC 一起将被发送给接收方 B。接收方 B 对收到的消息用相同的密钥 K 进行相同的计算,得出新的 MAC,并与接收到的 MAC 进行比较。如果假定只有收发双方知道该密钥,那么若接收到的 MAC 与计算得出的 MAC 相等,则:

(1) 接收方 B 可以相信消息在传送途中未被非法篡改。因为这里假定攻击者不知道密钥 K,攻击者可能修改消息,但不知道应如何改变 MAC 才能使其与修改后的消息相一致。这样,接收方计算出的 MAC 将不等于接收到的 MAC。

(2) 接收方 B 可以相信消息来自真正的发送方 A。因为其他各方均不知道密钥,因此他们不能产生具有正确 MAC 的消息。

(3) 如果消息中含有序列号(如 TCP 序列号),那么接收方可以相信消息顺序是正确的,因为攻击者无法成功地修改序列号并保持 MAC 与消息一致。

图 3-2 所示的过程仅仅提供鉴别而不能提供保密性,因为消息是以明文形式传送的。若将 MAC 附加在明文消息后对整个信息块加密,则可以同时提供保密和鉴别。这需要两个独立的密钥,并且收发双方共享这两个密钥。

MAC 函数与加密类似,但加密算法必须是可逆的,而 MAC 算法则不要求可逆性,在数学上比加密算法被攻击的弱点要少。与加密相比,MAC 算法更不易被攻破。

2. 基于 DES 的消息鉴别码

构造 MAC 的常用方法之一就是基于分组密码,并按密文块链接(Cipher Block Chaining,CBC)模式操作。在 CBC 模式中,每个明文分组在用密钥加密之前,都要先与前一个密文分组进行异或运算。用一个初始向量 IV 作为密文分组初始值。

数据鉴别算法也称 CBC-MAC(密文分组链接消息鉴别码),建立在 DES 之上,是使用最广泛的 MAC 算法之一,也是 ANSI 的一个标准,如图 3-3 所示。

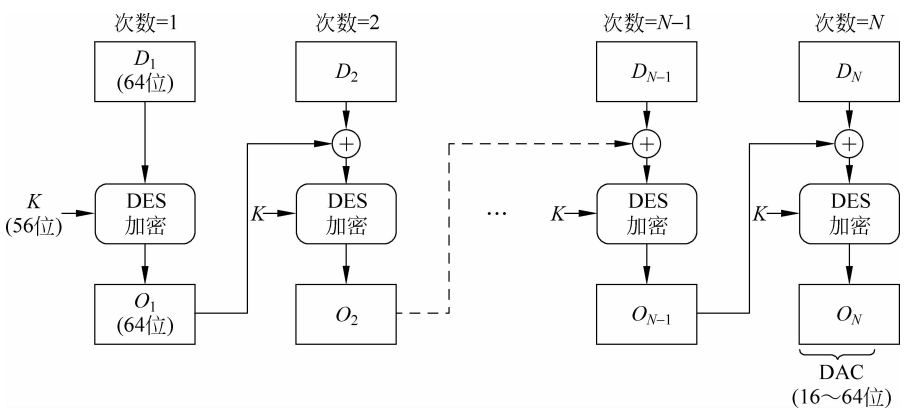

图 3-3　数据鉴别算法

数据鉴别算法采用 DES 运算的密文块链接方式,参见图 3-3。其初始向量 IV 为 0,需要鉴别的数据分成连续的 64 位的分组 $D_1, D_2, \cdots, D_N$,若最后分组不足 64 位,则在其后填 0 直至成为 64 位的分组。利用 DES 加密算法 E 和密钥 K,计算数据鉴别码(DAC)的过程如下:

$$
\begin{aligned}
O_0 &= IV \\
O_1 &= E_K(D_1 \oplus O_0) \\
O_2 &= E_K(D_2 \oplus O_1) \\
O_3 &= E_K(D_3 \oplus O_2) \\
&\vdots \\
O_N &= E_K(D_N \oplus O_{N-1})
\end{aligned}
$$

其中,DAC 可以取整个块 O_N,也可以取其最左边的 M 位,其中 $16 \leqslant M \leqslant 64$。

3.1.3 基于散列函数的鉴别

散列(Hash)函数是消息鉴别码的一种变形。与消息鉴别码一样,散列函数的输入是可变大小的消息 M,输出是固定大小的散列码 $H(M)$,也称为消息摘要,或散列值。与 MAC 不同的是,散列函数并不使用密钥,它仅是输入消息的函数。使用没有密钥的散列值作为消息鉴别码的机制是不安全的,因此实践中常将 Hash 函数和加密结合起来使用。

图 3-4 给出了将散列码用于消息鉴别的两种常用方法。

在图 3-4(a)中,消息发送方 A 首先计算明文消息 M 的散列值 $H(M)$,并将 $H(M)$串接在 M 后,然后用对称密码算法对消息及附加在其后的散列值加密,将密文发送给对方。在接收方 B,首先解密密文得到散列值 $H(M)$和明文消息 M,然后 B 自己亦根据同样的 Hash 算法计算散列值 $H'(M)$并验证 $H(M)=H'(M)$是否成立。如果成立,由于只有 A 和 B 共享密钥并且散列函数是一个单向函数,所以 B 可以确认消息一定是来自 A 且未被修改过的。散列码提供了鉴别所需的结构或冗余,并且由于该方法是对整个消息和散列码加密,所以也提供了保密性。

图 3-4(b)用对称密码仅对散列码加密。$E_K(H(M))$是变长消息 M 和密钥 K 的函数,它产生定长的输出值,若攻击者不知道密钥,则他无法得出这个值。这个方案只能提供鉴别,而无法提供保密。

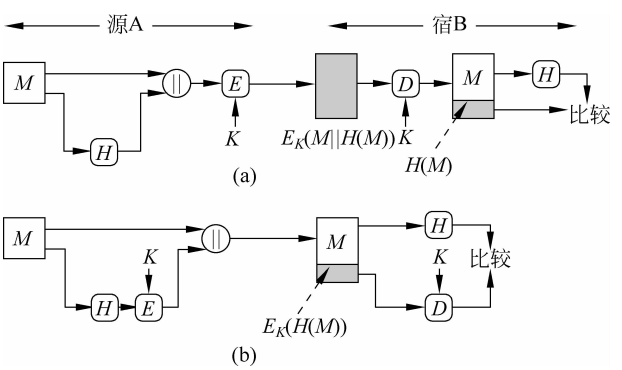

图 3-4 基于散列函数的消息鉴别

近年来,人们越来越感兴趣于利用散列函数来设计 MAC。这是因为利用对称加密算法产生 MAC 要对全部消息进行加密,运算速度较慢,而散列函数的执行速度比对称分组密码要快。

散列函数并不是专为 MAC 而设计的,不依赖于密钥,所以它不能直接用于 MAC。目前,已经提出了许多方案将密钥加到现有的散列函数中。HMAC 是最受支持的方案,它是一种依赖于密钥的单向散列函数,同时提供对数据的完整性和真实性的验证。HMAC 是 IP 安全里必须实现的 MAC 方案,并且其他 Internet 协议中(如 SSL)也使用了 HMAC。

RFC 2104 给出了 HMAC 的设计目标:

- 不必修改而直接使用现有的散列函数,即将散列函数看做是“黑盒”,可以使用多种散列函数。
- 如果找到或需要更快或更安全的散列函数,应能很容易地替代原来嵌入的散列

函数。

- 应保持散列函数的原有性能，不能过分降低其性能。
- 对密钥的使用和处理应较简单。
- 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。

图 3-5 给出了 HMAC 的总体结构。

定义下列符号：

H——嵌入的散列函数（如 MD5，SHA-1，RIPEMD-160）；

IV——作为散列函数输入的初始值；

M——HMAC 的消息输入（包括由嵌入散列函数定义的填充位）；

Y_i——M 的第 i 个分组，$0 \leqslant i \leqslant (L-1)$；

L——M 中的分组数；

b——每一分组所含的位数；

n——嵌入的散列函数所产生的散列码长；

K——密钥；建议密钥长度$\geqslant n$。若密钥长度大于 b，则将密钥作为散列函数的输入，来产生一个 n 位的密钥；

K^+——为使 K 为 b 位长而在 K 左边填充 0 后所得的结果；

ipad——内层填充，00110110（十六进制数 36）重复 $b/8$ 次的结果；

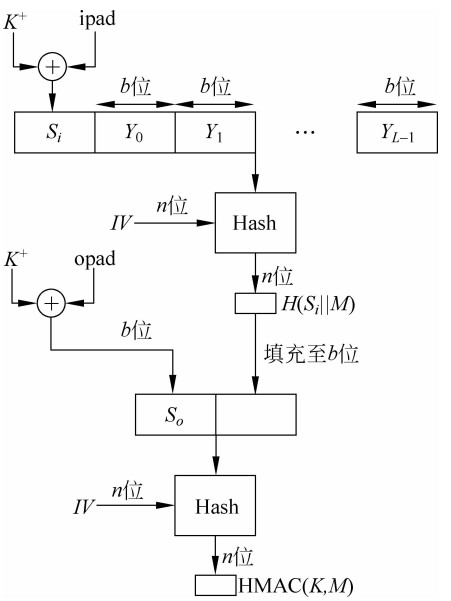

图 3-5　HMAC 的总体结构

opad——外层填充，01011100（十六进制数 5C）重复 $b/8$ 次的结果。

HMAC 可描述如下：

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

也就是说：

（1）在 k 左边填充 0，得到 b 位的 K^+（例如，若 K 是 160 位，$b=512$，则在 K 中加入 44 个 0 字节 0×00）。

（2）K^+ 与 ipad 执行异或运算（逐位异或）产生 b 位的分组 S_i。

（3）将 M 附于 S_i 后。

（4）将 H 作用于步骤（3）所得出的结果。

（5）K^+ 与 opad 执行异或运算（位异或）产生 b 位的分组 S_o。

（6）将步骤（4）中的散列码附于 S_o 后。

（7）将 H 作用于步骤（6）所得出的结果，并输出该函数值。

注意，K 与 ipad 异或后，其信息位有一半发生了变化；同样，K 与 opad 异或后，其信息位的另一半也发生了变化，这样，通过将 S_i 与 S_o 传给散列算法中的压缩函数，可以从 K 伪随机地产生出两个密钥。

HMAC 多执行了三次散列压缩函数（对 S_i，S_o 和内部的散列产生的分组），但是对于长消息，HMAC 和嵌入的散列函数的执行时间应该大致相同。

3.1.4 散列函数

散列函数又叫做散列算法,是一种将任意长度的消息映射到某一固定长度消息摘要(散列值,或哈希值)的函数。消息摘要相当于消息的"指纹",用来防止对消息的非法篡改。如果消息被篡改,则"指纹"就不正确了。即使消息不具有保密性,也可以通过消息摘要来验证其完整性。

令 h 代表一个散列函数,M 代表一个任意长度的消息,则 M 的散列值 h 表示为:

$$h = H(M)$$

且 $H(M)$长度固定。假设 h 安全,发送方将散列值 h 附于消息 M 后发送;接收方通过重新计算散列值 h'并比较 $h=h'$是否成立,可以验证该消息的完整性。

1. 散列函数安全性

对散列函数最直接的攻击就是攻击者得到消息 M 的散列值 $h(M)$后,试图伪造消息 M',使得 $h(M')=h(M)$。因此,密码学中的散列函数必须满足一定的安全特征,主要包括三个方面:单向性、强对抗碰撞性和弱对抗碰撞性。

单向性是指对任意给定的散列码 h,找到满足 $H(x)=h$ 的 x 在计算上是不可行的,即给定散列函数 h,由消息 M 计算散列值 $H(M)$是容易的,但是由散列值 $H(M)$计算 M 是不可行的。

强抗碰撞性是指散列函数满足下列四个条件:

(1) 散列函数 h 的输入是任意长度的消息 M;

(2) 散列函数 h 的输出是固定长度的数值;

(3) 给定 h 和 M,计算 $h(M)$是容易的;

(4) 给定散列函数 h,寻找两个不同的消息 M_1 和 M_2,使得 $h(M_1)=h(M_2)$,在计算上是不可行的。如果有两个消息 M_1 和 M_2,$M_1 \neq M_2$ 但是 $h(M_1)=h(M_2)$,则称 M_1 和 M_2 是碰撞的。

弱抗碰撞性的散列函数满足强抗碰撞散列函数的前三个条件,但具有一个不同的条件:给定 h 和一个随机选择的消息 M,寻找消息 M',使得 $h(M)=h(M')$在计算上是不可行的,即不能找到与给定消息具有相同散列值的另一消息。

2. SHA-1 算法

目前人们已经设计出了大量的散列算法,其中,SHA (Secure Hash Algorithm,安全散列算法)和 MD5 是最著名的两个。

SHA 是由美国国家安全局(NSA)设计,经美国国家标准与技术研究所 (NIST) 发布的一系列密码散列函数。1993 年发布了 SHA,后来人们给它取了一个非正式的名称 SHA-0,以避免与它的后继者混淆。1995 年发布了 SHA-1,该算法产生 160 比特的散列值。另外还有三种变体:SHA-256、SHA-384 和 SHA-512,其散列值长度分别为 224、256、384、512 比特。

SHA-1 算法的输出是 160 比特的消息摘要,输入消息以 512 比特分组为单位进行处理。处理消息和输出摘要的过程包含下列步骤。

步骤 1:附加填充位。

填充消息使其长度模 512 与 448 同余,即长度在对 512 取模以后的余数是 448。即使

消息已经满足上述长度要求，仍然需要进行填充。填充是这样进行的：先补一个 1，然后再补 0，直到长度满足对 512 取模后余数是 448。因此，填充时至少填充一位，最多填充 512 位。

步骤 2：附加长度。

将原始数据的长度补到已经进行了填充操作的消息后面。通常用一个 64 位的数据来表示原始消息的长度。

前两步的结果是产生了一个长度为 512 整数倍的扩展消息。然后，扩展的消息被分成长度为 512 比特的消息块 $M_1, M_2, \cdots, M_N$，因此扩展消息的长度为 $N \times 512$ 比特。

步骤 3：初始化散列缓冲区。

SHA-1 算法的计算过程中需要两个缓冲区，每个缓冲区都由 5 个 32 位的字组成。第一个 5 个字的缓冲区被标识为 A, B, C, D, E，第二个 5 个字的缓冲区被标识为 H0，H1，H2，H3，H4，并将第一个 5 个字的缓冲初始化为下列 32 比特的整数(十六进制值)：

$$A = 0\text{x}67452301$$
$$B = 0\text{x}\text{EFCDAB89}$$
$$C = 0\text{x}98\text{BADCFE}$$
$$D = 0\text{x}10325476$$
$$E = 0\text{x}\text{C3D2E1F0}$$

还需要一个 80 个 32 位字的缓冲区，标识为 W_0 到 W_{79}，以及一个字的 TEMP 缓冲区。

步骤 4：计算消息摘要。

步骤 2 中得到长度 512 的消息块 $M_1, M_2, \cdots, M_N$ 会依次进行处理，处理每个消息块 M_i 都要运行一个具有 80 轮运算的函数，每一轮都把 160 比特缓冲区的值 $ABCDE$ 作为输入，并更新缓冲区的值。

每一轮运算将使用附加的常数 K_t，其中 $0 \leqslant t \leqslant 79$($t$ 代表运算的轮数)，这些常数如下：

$$K_t = 0\text{x}5\text{A}827999 \quad (0 \leqslant t \leqslant 19)$$
$$K_t = 0\text{x}6\text{ED9EBA1} \quad (20 \leqslant t \leqslant 39)$$
$$K_t = 0\text{x}8\text{F1BBCDC} \quad (40 \leqslant t \leqslant 59)$$
$$K_t = 0\text{x}\text{CA62C1D6} \quad (60 \leqslant t \leqslant 79)$$

每一轮还将使用一个非线性函数 f_t：

$$f_t(X,Y,Z) = (X \wedge Y) \vee (\overline{X} \wedge Z), \quad 0 \leqslant t \leqslant 19$$
$$f_t(X,Y,Z) = X \oplus Y \oplus Z, \quad 20 \leqslant t \leqslant 39$$
$$f_t(X,Y,Z) = (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z), \quad 40 \leqslant t \leqslant 59$$
$$f_t(X,Y,Z) = X \oplus Y \oplus Z, \quad 60 \leqslant t \leqslant 79$$

其中，$\wedge$ 表示逐位“与”，$\vee$ 表示逐位“或”，$\oplus$ 表示逐位“异或”，$\overline{X}$ 则表示 X 的逐位取反。

对一个消息块 M_i，首先用下面的算法将消息块(16 个 32 比特，共 512 比特)变成 80 个 32 比特子块(W_0 到 W_{79})：

$$W_j = M_j, \quad 0 \leqslant j \leqslant 15$$
$$W_j = (W_{j-3} \oplus W_{j-8} \oplus W_{j-14} \oplus W_{j-16}) \ll 1, \quad 16 \leqslant j \leqslant 79$$

其中，$\ll$ 表示循环左移位。

将第一个 5 个字的缓冲区内容复制到第二个 5 个字的缓冲区：

$$H0=A,H1=B,H2=C,H3=D,H4=E$$

对每一个 $W_j(1\leqslant j\leqslant 79)$:

$$\begin{aligned}&\text{TEMP}=(A\ll 5)+f_j(B,C,D)+E+W_j+K_j;\\&E=D;\\&D=C;\\&C=B\ll 30;\\&B=A;\\&A=\text{temp}\end{aligned}$$

其中,+表示模 2^{32} 的加法运算。

最后,执行:

$$A=H0+A,B=H1+B,C=H2+C,D=H3+D,E=H4+E$$

步骤 5:输出。

所有的 N 个 512 比特分组都处理完以后,从第 N 阶段输出的是 $ABCDE$,长度为 160 比特的消息摘要。

3. MD5

MD5 即 Message-Digest Algorithm 5(消息摘要算法 5),是广泛使用的散列算法(又称为哈希算法)之一。MD5 的设计者是麻省理工学院的 Ronald L. Rivest,经 MD2、MD3 和 MD4 发展而来。MD4 算法发布于 1990 年,该算法没有基于任何假设和密码体制,运行速度快,实用性强,受到了广泛的关注。但后来人们发现 MD4 存在安全缺陷,于是 Ronald L. Rivest 于 1991 年对 MD4 做了几点改进,改进后的算法就是 MD5。虽然 MD5 比 MD4 稍微慢一些,但更为安全。

MD5 输入任意长度的消息,生成 128 位的散列值。输入消息被以 512 位的长度来分组,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,算法的输出由四个 32 位分组组成,将这四个 32 位分组级联后生成一个 128 位散列值。

MD5 计算消息摘要时,执行下述步骤。

(1) 消息填充

在 MD5 算法中,首先需要对消息进行填充,使其长度(以二进制位为单位)对 512 求余的结果等于 448。因此,信息的位长(Bits Length)将被扩展至 $N\times 512+448$,即 $N\times 64+56$ 个字节(Bytes),N 为一个正整数。填充的方法如下,在信息的后面填充一个 1 和多个 0,直到满足长度对 512 求余的结果等于 448 才停止填充。

(2) 添加长度

消息填充的结果后面附加一个以 64 位二进制表示的填充前信息长度。经过这两步的处理,现在的信息的位长 $=N\times 512+448+64=(N+1)\times 512$,即长度恰好是 512 的整数倍。换句话说,消息长度现在是 16 个 32 位字的整数倍。这样做的原因是满足后面处理中对信息长度的要求。

(3) 初始化缓冲区

MD5 中用一个四字缓冲区表示四个 32 位寄存器,也被称做链接变量(Chaining Variable)。这个 128 为缓冲区用于计算消息摘要。这四个寄存器被初始化为:

a＝0x01234567

b＝0x89abcdef

c＝0xfedcba98

d＝0x76543210

将上面四个链接变量复制到另外四个变量中：a 到 AA，b 到 BB，c 到 CC，d 到 DD。

当设置好这四个链接变量后，就开始进入算法的主循环，循环的次数是信息中 512 位信息分组的数目。

(4) 定义辅助函数

MD5 算法要用到四个辅助函数。这四个非线性函数(每轮一个)都以三个 32 位字为输入，生成一个 32 位字输出。它们被表示为：

$$F(X,Y,Z) = (X \wedge Y) \vee (\overline{X} \wedge Z)$$

$$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \overline{Z})$$

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

$$I(X,Y,Z) = Y \oplus (X \vee \overline{Z})$$

这里，$\wedge$ 表示逐位“与”，$\vee$ 表示逐位“或”，$\oplus$ 表示逐位“异或”，$\overline{X}$ 则表示 X 的逐位取反。

此外，MD5 还使用了四种操作。假设 M_j 表示消息的第 j 个子分组(从 0 到 15)，则四种操作：

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

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

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

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

其中，$\ll s$ 表示循环左移 s 位，$+$ 表示整数模 2^{32} 加法运算。

(5) 四轮计算

主循环有四轮(MD4 只有三轮)，每轮循环都很相似。每一轮进行 16 次操作。每次操作对 a、b、c 和 d 中的三个做一次非线性函数运算，然后将所得结果加上第四个变量、一个子分组和一个常数，再将所得结果循环左移，并加上 a、b、c 或 d 其中之一。最后用该结果取代 a、b、c 或 d 其中之一。

这四轮是：

① 第一轮

$FF(a,b,c,d,M_0,7,0xd76aa478)$

$FF(d,a,b,c,M_1,12,0xe8c7b756)$

$FF(c,d,a,b,M_2,17,0x242070db)$

$FF(b,c,d,a,M_3,22,0xc1bdceee)$

$FF(a,b,c,d,M_4,7,0xf57c0faf)$

$FF(d,a,b,c,M_5,12,0x4787c62a)$

$FF(c,d,a,b,M_6,17,0xa8304613)$

$FF(b,c,d,a,M_7,22,0xfd469501)$

$FF(a,b,c,d,M_8,7,0x698098d8)$

$FF(d,a,b,c,M_9,12,0x8b44f7af)$

$FF(c,d,a,b,M_{10},17,0xffff5bb1)$

$FF(b,c,d,a,M_{11},22,0x895cd7be)$

$FF(a,b,c,d,M_{12},7,0x6b901122)$

$FF(d,a,b,c,M_{13},12,0xfd987193)$

$FF(c,d,a,b,M_{14},17,0xa679438e)$

$FF(b,c,d,a,M_{15},22,0x49b40821)$

② 第二轮

$GG(a,b,c,d,M_{1},5,0xf61e2562)$

$GG(d,a,b,c,M_{6},9,0xc040b340)$

$GG(c,d,a,b,M_{11},14,0x265e5a51)$

$GG(b,c,d,a,M_{0},20,0xe9b6c7aa)$

$GG(a,b,c,d,M_{5},5,0xd62f105d)$

$GG(d,a,b,c,M_{10},9,0x02441453)$

$GG(c,d,a,b,M_{15},14,0xd8a1e681)$

$GG(b,c,d,a,M_{4},20,0xe7d3fbc8)$

$GG(a,b,c,d,M_{9},5,0x21e1cde6)$

$GG(d,a,b,c,M_{14},9,0xc33707d6)$

$GG(c,d,a,b,M_{3},14,0xf4d50d87)$

$GG(b,c,d,a,M_{8},20,0x455a14ed)$

$GG(a,b,c,d,M_{13},5,0xa9e3e905)$

$GG(d,a,b,c,M_{2},9,0xfcefa3f8)$

$GG(c,d,a,b,M_{7},14,0x676f02d9)$

$GG(b,c,d,a,M_{12},20,0x8d2a4c8a)$

③ 第三轮

$HH(a,b,c,d,M_{5},4,0xfffa3942)$

$HH(d,a,b,c,M_{8},11,0x8771f681)$

$HH(c,d,a,b,M_{11},16,0x6d9d6122)$

$HH(b,c,d,a,M_{14},23,0xfde5380c)$

$HH(a,b,c,d,M_{1},4,0xa4beea44)$

$HH(d,a,b,c,M_{4},11,0x4bdecfa9)$

$HH(c,d,a,b,M_{7},16,0xf6bb4b60)$

$HH(b,c,d,a,M_{10},23,0xbebfbc70)$

$HH(a,b,c,d,M_{13},4,0x289b7ec6)$

$HH(d,a,b,c,M_{0},11,0xeaa127fa)$

$HH(c,d,a,b,M_{3},16,0xd4ef3085)$

$HH(b,c,d,a,M_{6},23,0x04881d05)$

$HH(a,b,c,d,M_{9},4,0xd9d4d039)$

$HH(d,a,b,c,M_{12},11,0xe6db99e5)$

$HH(c,d,a,b,M_{15},16,0x1fa27cf8)$