# 第11章 SHA-1算法

## 11.1 算法原理

在安全应用中使用的Hash函数称为密码学Hash函数。常用的几种密码学Hash函数有MD系列、SHA系列等。安全Hash算法SHA（Security Hash Algorithm）是使用最为广泛的Hash函数，由美国标准与技术研究所（NIST）设计，并于1993年作为联邦信息处理标准（FIPS 180）发布，该版本称为SHA-0，修订版于1995年发布（FIPS 180），通常称之为SHA-1，即安全Hash标准。

### 11.1.1 SHA-1总体结构

SHA-1 算法要求输入消息长度小于位，将输入消息按512位分组进行处理，输出长度为160位。图11-1显示了处理消息、输出摘要的总体过程。算法过程如下：

第一步：位填充。填充消息使其比特长度满足，填充由一个1和若干个0组成；

第二步： 长度填充。用64位表示消息位填充前的长度，将其附加在位填充的消息后面；

第三步：初始化缓冲区；

第四步：以512位分组为单位处理消息；

第五步：输出结果。

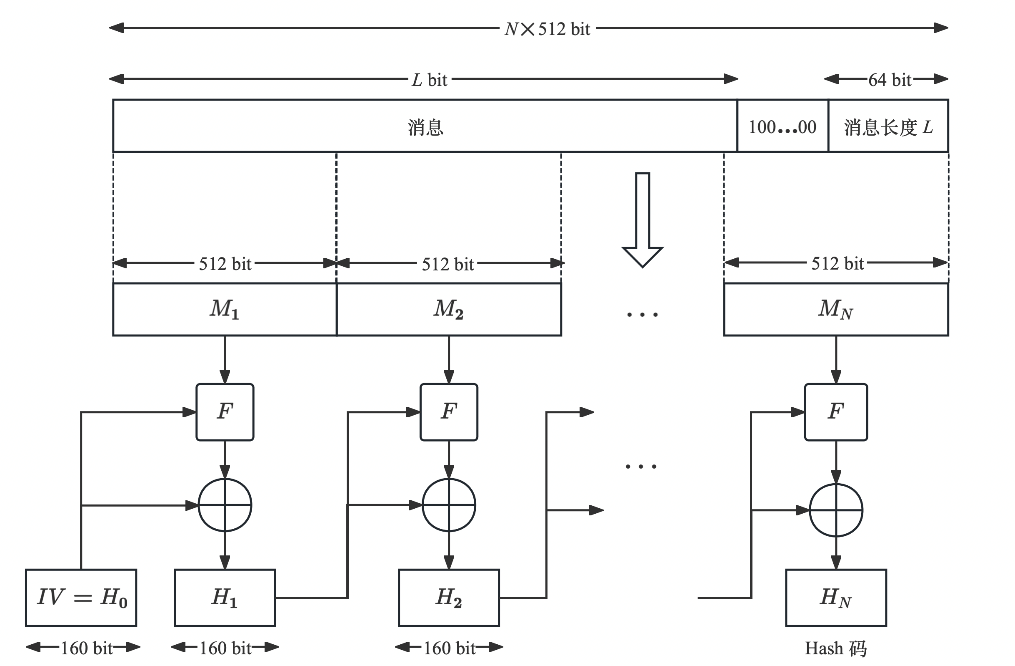


图11-1 SHA-1生成消息摘要

### 11.1.2 SHA-1详细结构

#### SHA-1的分组

对于任意长度的消息，首先需要对消息添加位数，使消息总长度模512与448同余。在消息后添加位数的方法是第一位是l，其余都是0。之后将原始的长度（没有添加位数以前的消息比特长度）以64位表示，附加于消息后，此时的消息长度正好是512位的倍数。SHA-1的原始消息长度不能超过。另外，SHA-1的消息长度从低位开始填充。对填充后的消息按512位的长度进行分组，表示为。

对于512位的消息分组，SHA-1将其再分成16个子消息分组，每个子消息分组为32位，使用表示。之后将16个子消息分组扩充到80个子消息分组进行后续计算，记为，扩充方法如下。

，当。

, 当。

#### SHA-1的4轮运算

SHA-1有4轮运算，每一轮包括20个步骤（共80步），最后产生160位摘要，如图11-2所示。

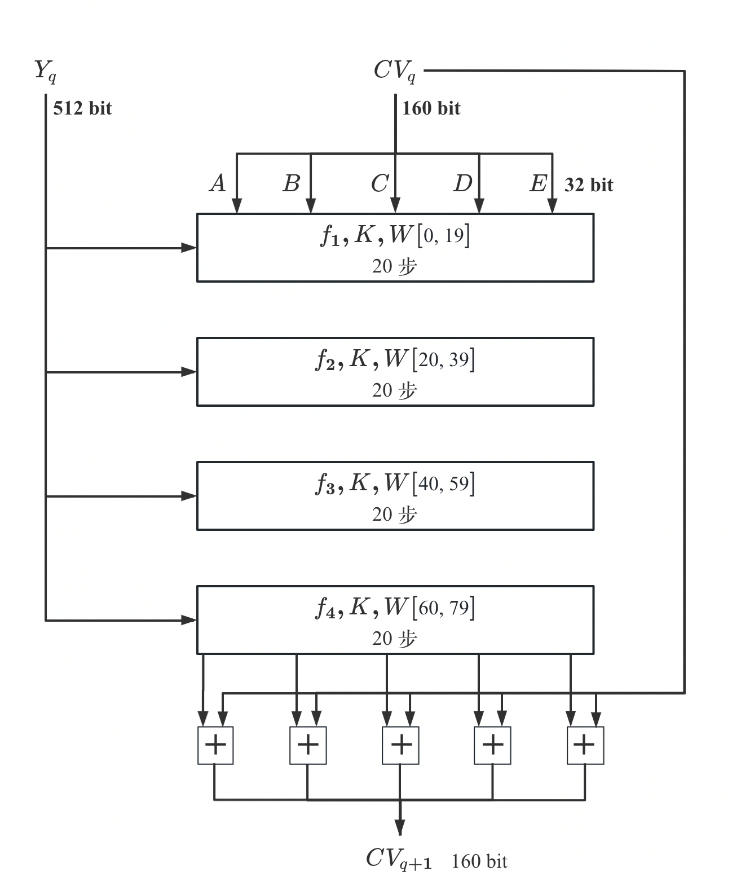


图11-2 SHA-1对单个512分组的处理

160位摘要存放在5个32位的链接变量中，分别标记为*，*初始值以16进制表示如下：

当第1轮运算中的第1步开始处理时，五个链接变量中的值先赋值到另外5个记录单元中。这5个值将保留，用于在第4轮的最后一个步骤完成之后与链接变量进行求和操作。SHA-1的4轮运算，共80步使用同一个函数，该函数的内部结构如图11-3所示，表示为如下形式：

其中 为逻辑函数，为子明文分组，为固定常数。

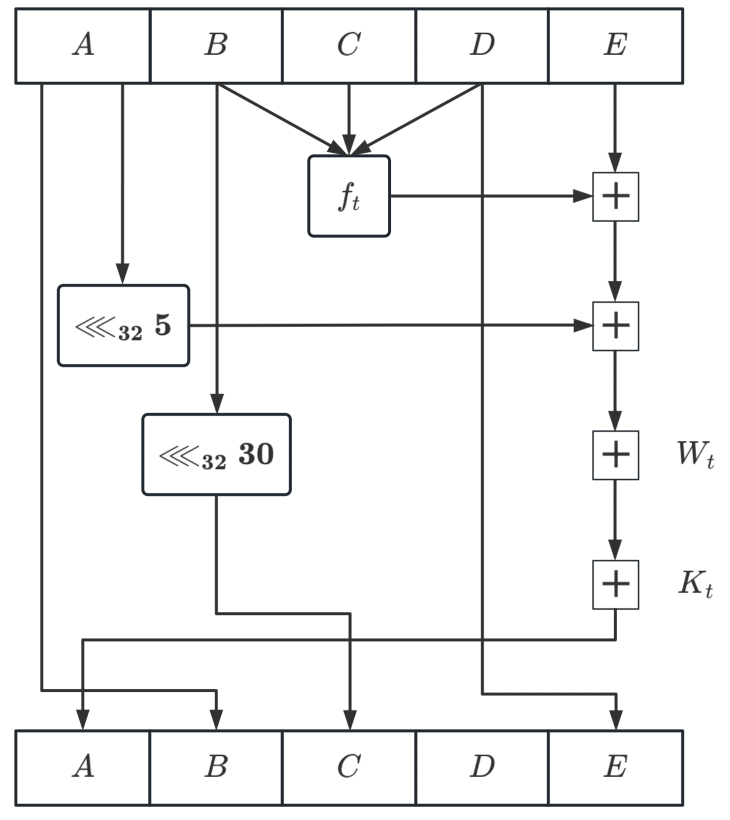


图11-3 SHA-1的单轮操作

具体这个函数的意义为：

（1）将的结果赋值给链接变量；

（2）将链接变量初始值赋值给链接变量；

（3）将链接变量初始值循环左移30位赋值给链接变量；

（4）将链接变量初始值赋值给链接变量；

（5）将链接变量初始值赋值给链接变量*。*

#### SHA-1的逻辑函数

SHA-1的逻辑函数如表11-1所示。

表11-1 SHA-1的逻辑函数

| **轮** | **步骤** | **函数定义** |
| --- | --- | --- |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |

在操作程序中需要使用固定常数，取值如表11-2所示。

表11-2 固定常数的取值

| **轮** | **步骤** | **函数定义** |
| --- | --- | --- |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |

## 11.2 算法伪代码

### 11.2.1 摘要生成函数

摘要生成函数输入明文消息的编码，输出产生的摘要。伪代码如下所示：

**算法11.2.1**

// 输入：消息的编码

// 输出：消息的哈希值

将按照512 bit一组分为组

**for** **to** **do**

中第个分组

**return**

### 11.2.2 消息填充函数

消息填充函数负责把消息填充到需要的长度，并且写入消息的长度信息，附加完的消息二进制长度应该为512的整数倍。消息填充函数输入原始消息的编码，输出16进制经填充的消息。伪代码如下所示：

**算法11.2.2**

// 输入：需要填充的消息

// 输出：填充后的

将消息转换成二进制串

的比特长度

// 填充

**while** **do**

将用64位的格式补充在最后

将转为十六进制串

**return**

### 11.2.3 字扩展函数

字扩展函数输入，输出经扩展的字节分组。伪代码如下所示：

**算法11.2.3**

// 输入：需要扩充的一组消息

// 输出：扩展后的80个子分组

**for** **to** 79 **do**

**return**

### 11.2.4 轮函数

轮函数输入经扩展的字节分组、数组和参数，输出迭代更新后的数组。伪代码如下所示：

**算法11.2.4**

// 输入：子分组W、数组、参数

// 输出：迭代更新后的数组

**for** **to** 79 **do**

**return**

### 11.2.5 逻辑函数

逻辑函数在80轮中各不相同，函数具体如下：

**算法11.2.5**

// 输入：逻辑函数输入数据，当前轮次

// 输出：逻辑函数输出

**if** **then**

**else if** **then**

**else if** **then**

**else**

**return**

# 第15章 SM3算法

SM3密码杂凑算法由中国国家密码管理局在2010年发布，于2012年发布为密码行业标准(GM/T 0004-2012)，2016年发布为国家密码杂凑算法标准(GB/T 32905-2016)。SM3适用于密码应用中的数字签名和验证、消息认证码的生成与验证以及随机数的生成，可满足多种密码应用的安全需求。

## 15.1 算法原理

### 15.1.1 SM3总体结构

SM3算法和MD5的迭代过程类似，采用Merkle-Damgard结构，是在SHA-256基础上改进实现的一种算法，其安全性和SHA-256相当。SM3算法的消息分组长度为512位，摘要值长度为256位，对长度为比特的消息，SM3杂凑算法经过填充和迭代压缩，生成杂凑值，杂凑值长度为256比特。SM3算法总体流程如图15-1所示。

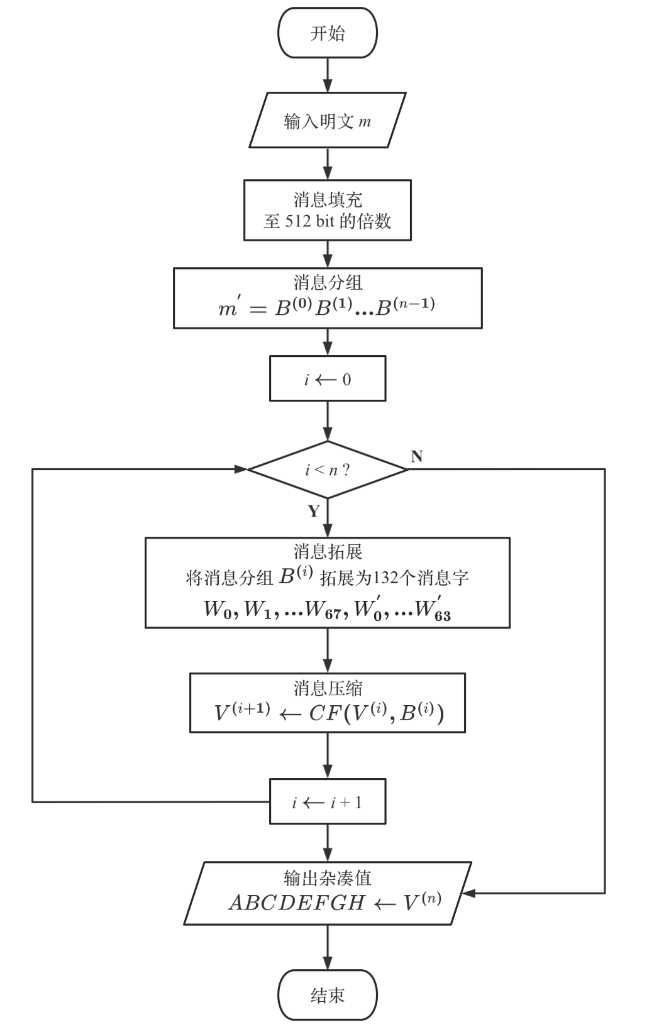


图15-1 SM3总体流程图

### 15.1.2 SM3详细结构

#### 1. 常量与函数

（1）初始值

初始值由8个32比特字构成，其16进制表示如下所示。

1. 常量
2. 布尔函数

式中、、是32比特字，表示取反运算。

1. 置换函数

式中是32比特字。

#### 2. 消息填充

假设消息的长度为比特，则首先将比特“1”添加到消息的末尾，再添加个“0”，是满足的最小的非负整数。之后再添加一64位比特串，该比特串是长度的二进制表示。填充后的消息的比特长度为512的倍数。

例如：对消息：01100001 01100010 01100011，其长度，经填充得到比特串：

#### 3. 迭代压缩

（1）迭代过程

将填充后的消息按512比特进行分组：

其中。

对按下列方式迭代：

**for** **to** **do**

其中是压缩函数，为256比特初始值，为填充后的消息分组，迭代压缩的结果为。

（2）消息扩展

将消息分组按以下方法扩展生成132个消息字，用于压缩函数：

第一步，将消息分组划分为16个字。

第二步，

**for** **to** **do**

第三步，

**for** *j* = 0 **to** 63 **do**

（3）压缩函数

令为字寄存器，为中间变量，压缩函数。计算过程描述如下：

**for** *j* = 0 **to** 63 **do**

其中，字的存储为大端（big-endian），左边为高有效位，右边为低有效位。

#### 4. 输出杂凑值

输出256比特的杂凑值。

## 15.2 算法伪代码

15.2.1 摘要生成算法

摘要生成算法输入消息，输出摘要值。其中状态数据的初始值为8个32比特字，的取值见15.1.2章节常量与函数中。

**算法15.2.1**

// 输入：消息

// 输出：摘要值

初始值

的字节长度

**for** **to** **do**

**return**

### 15.2.2 消息填充算法

消息填充算法负责把消息填充到需要的长度，填充后的消息二进制长度应该为512的整数倍。算法输入消息，输出填充后的消息。算法伪代码如下：

**算法15.2.2**

// 输入：消息

// 输出：填充后的消息

字节长度(使用64比特长存储)

长度的全0字节串

**return**

### 15.2.3 消息扩展算法

消息扩展算法函数将当前组的消息进行扩展，输出当前组的消息的扩展结果和，其中的置换函数见15.1.2章节常量与函数中。算法伪代码如下：

**算法15.2.3**

// 输入：64字节长度的消息分组

// 输出：字扩展结果和

**for** **to** **do**

**for** **to** **do**

**return**

### 15.2.4 压缩轮函数

迭代压缩函数输入消息扩展结果和及上一轮的状态数据，输出本轮压缩后的状态数据，其中的常量、布尔函数和、置换函数见15.1.2章节中的常量与函数部分。算法伪代码如下：

**算法15.2.4**

// 输入：消息扩展结果和，状态数据

// 输出：更新后的状态输出

**for** **to** **do**

**return**