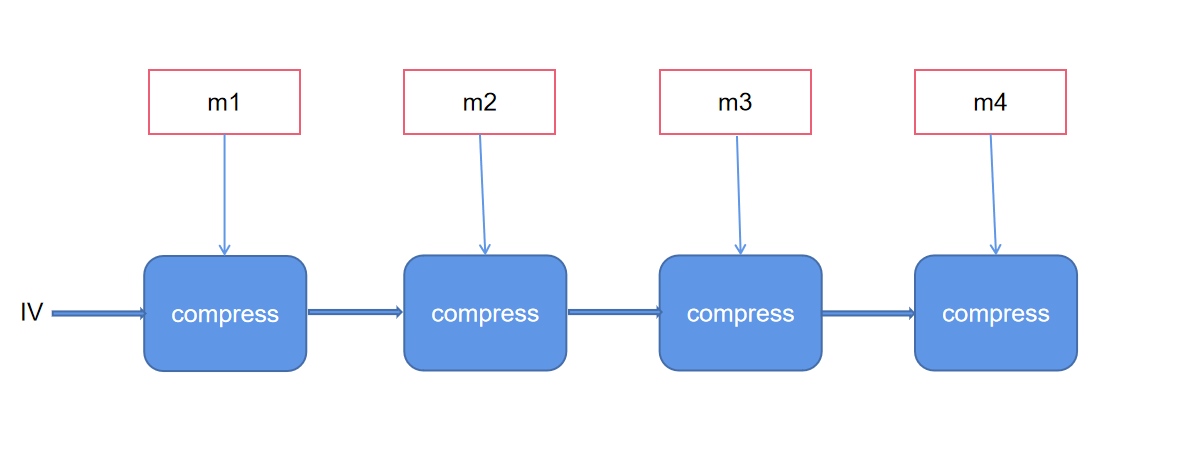
# SM3实现与长度扩展攻击

1. **SM3原理**

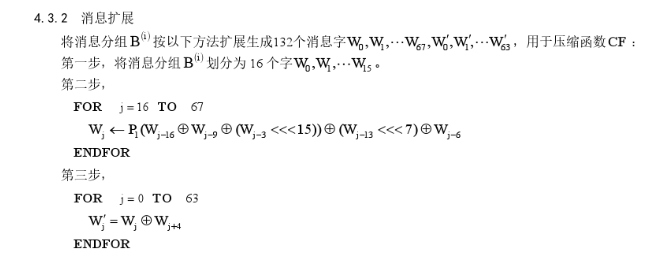
SM3是典型的MD结构，MD结构如下图，其中compress是具体的压缩函数。



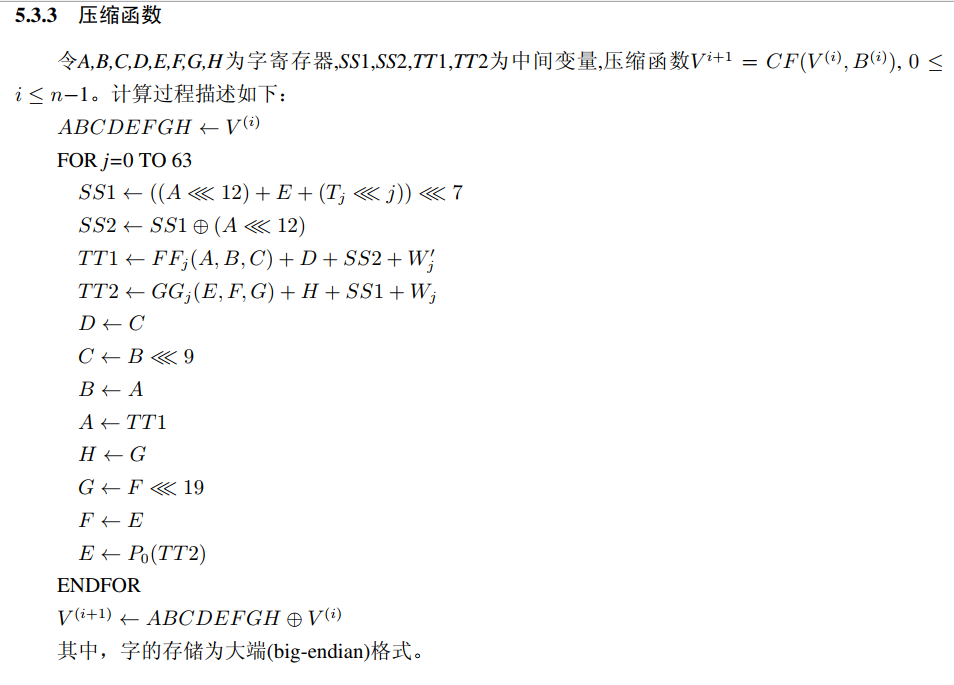
对于SM3而言，compress函数由消息扩展和迭代压缩两个部分组成，在执行compress函数之前，还要经过一个消息填充的步骤，将不满512bit的数据填充到512bit。

消息填充的具体步骤为：1.将512bit的后64位存放数据的长度；2.在数据的后一位加一个1；3.将剩下的所有空闲位都填为0。

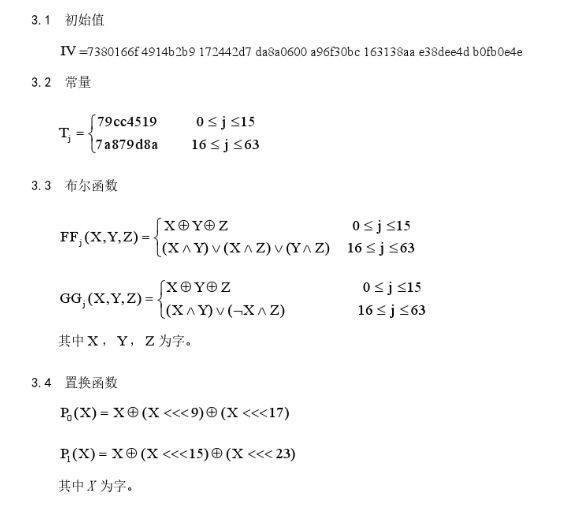
消息扩展将512bit（即16字节）的数据扩展为132字节，具体算法见下图



迭代压缩要进行64轮迭代，算法如下图

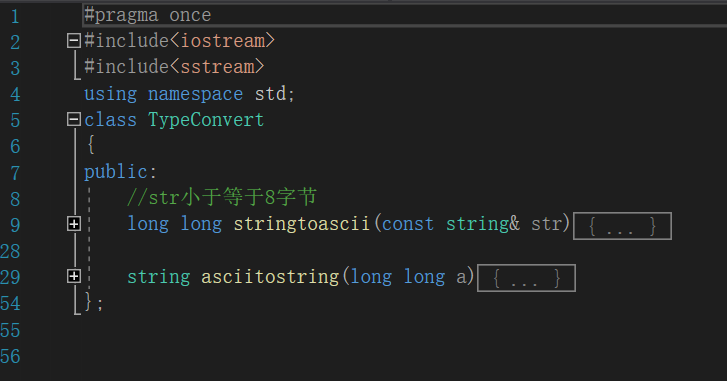


对于SM3函数中涉及的一些置换和布尔函数，见下图

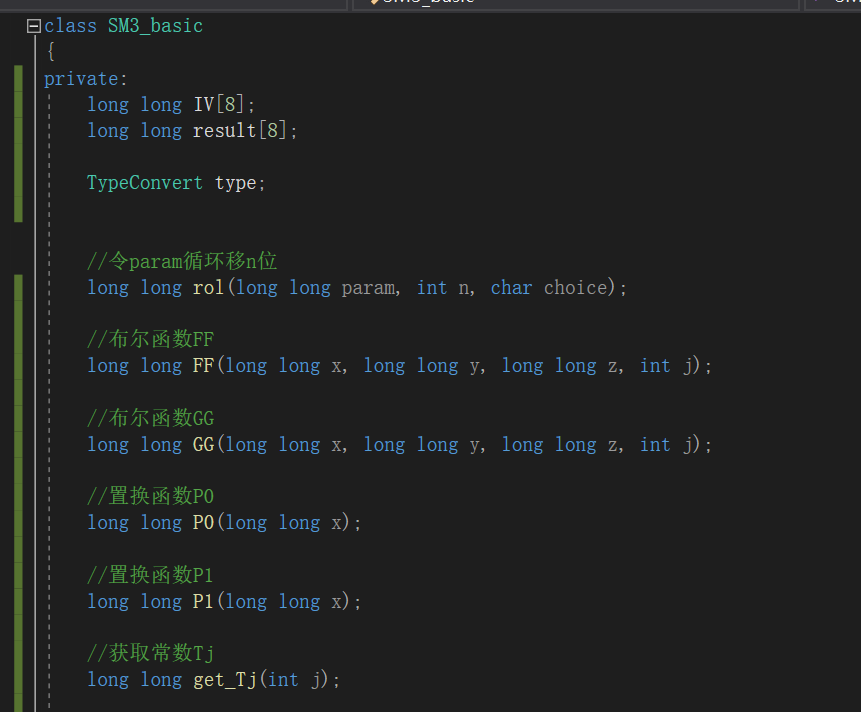


#### 二.SM3实现

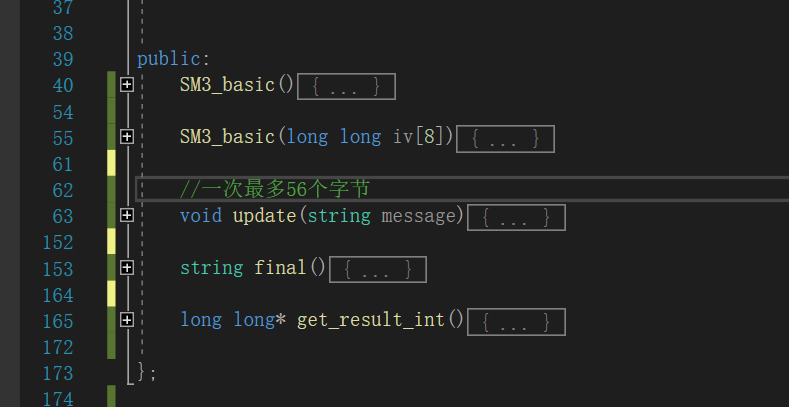
首先定义一个类型转换类，含有有两个方法。其中stringtoascii方法可以把长度小于等于8个字节的string类型字符串转为长整型的ascii码串。asciitostring是stringtoascii的逆过程，即把长整型的ascii码串转为相应的字符串。



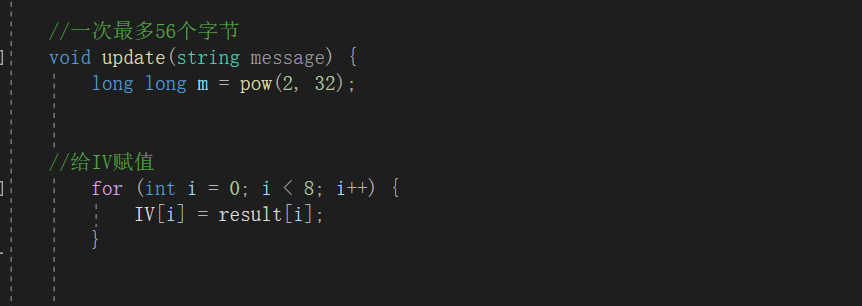
然后定义SM3\_basic类，执行SM3算法。类中有两个属性，result和IV，result是哈希的结果，IV是初始向量。两个属性都是长整型，都使用ascii码串进行运算。在private域中，还定义了一些其它函数，这些函数都是执行SM3算法所必须的函数，其作用见下图中注释。



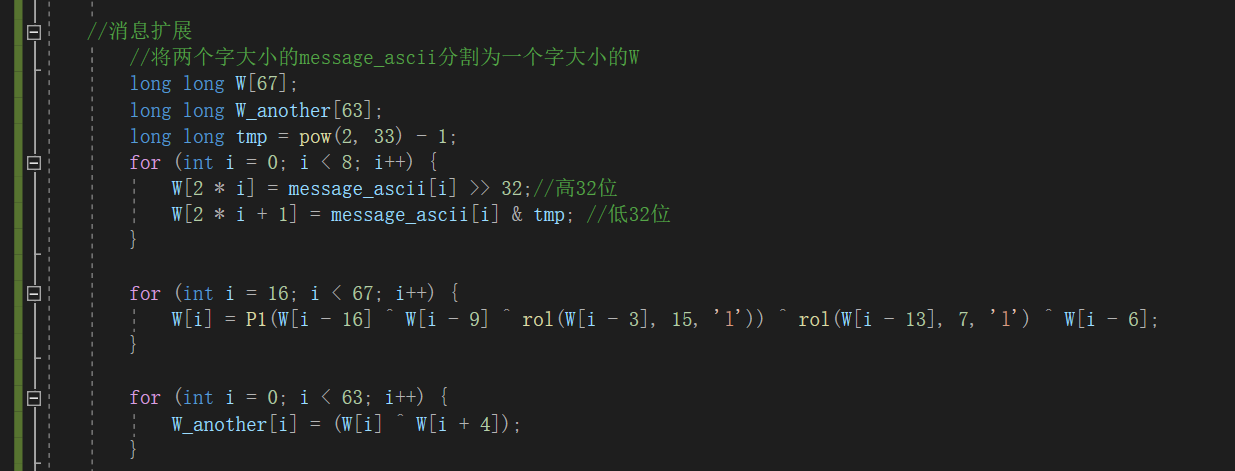
在SM3\_basic类中，有五个方法可以访问。首先是两个构造函数，当构造函数里面没有任何参数时，IV使用SM3指定的IV，即0x738166f 4914b2b9 172442d7 da8a0600 a96f30bc 163138aa e38dee4d b0fb0e4e。这个IV以32bit为一组，分别存储在IV数组的8个元素中。当构造函数中包含参数时，可以指定IV，这个函数主要作用是为了下面的长度扩展攻击。而后是update方法，这个方法是SM3\_basic类的核心方法，每一次update可以对小于等于56个字节的字符串进行哈希。如果一次要哈希的数据很多，那么可以把这些数据按照56个字节的大小进行分割，并调用n次update方法。对于final方法，返回值是一个string类型的字符串，这个字符串就是哈希函数的结果。最后还定义了一个get\_result\_int方法，返回result属性，也是为了下面的长度扩展攻击。

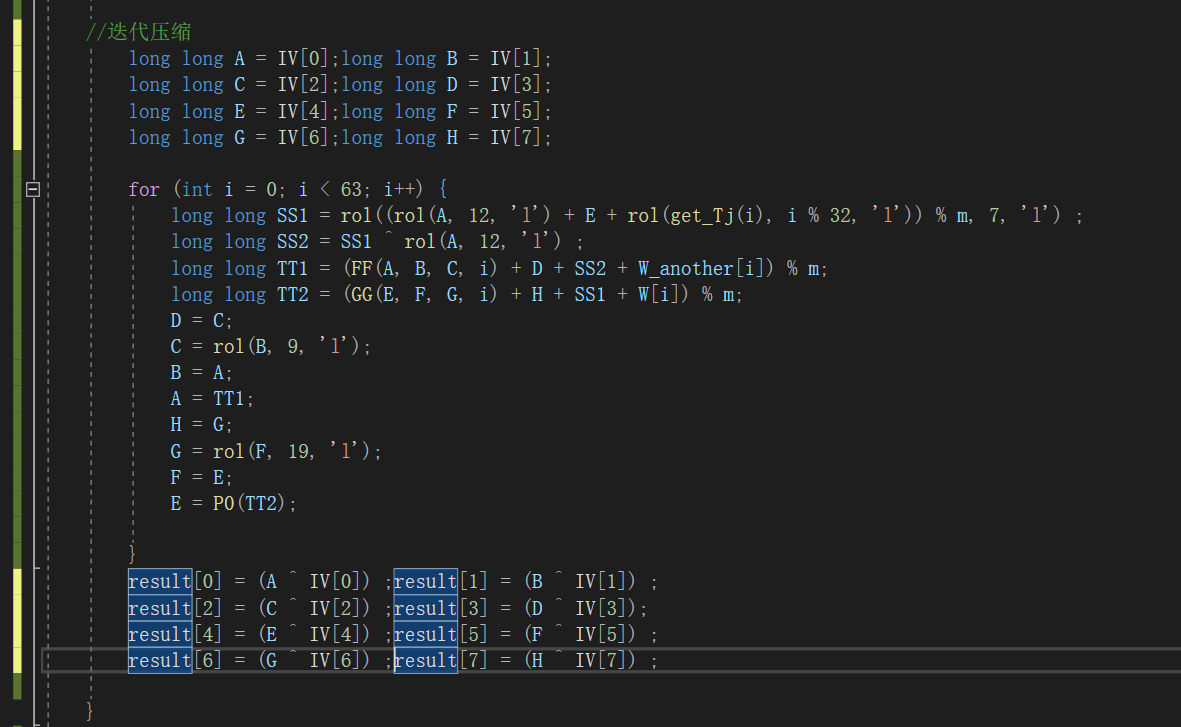


对于update算法，大概包括消息填充，消息扩展，迭代压缩三个部分。在算法开始之前，需要用上次update的结果作为下次update的IV，以此保证:一个实例不管执行多少次update操作，最后的结果都是256bit。其他的三个步骤按照SM3算法的定义即可一步步的实现，如下图

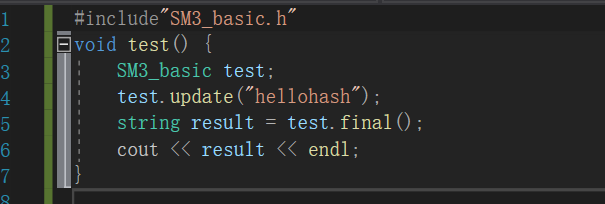




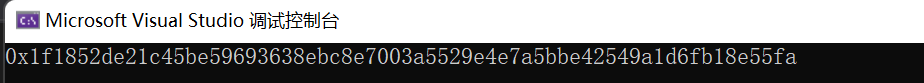




如下图是SM3\_basic的一个使用示例，对”hellohash”进行哈希，并把结果打印。



运行程序得到哈希结果



#### 三.SM3的长度扩展攻击

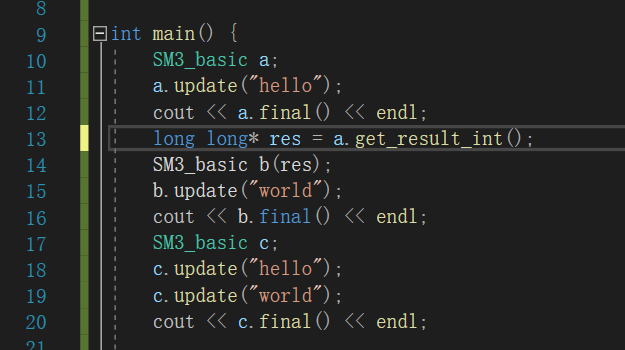
###### 长度扩展攻击的定义

长度扩展攻击的一个例子为hash(m1||m2) = hash(hash(m1)||m2)。即使用m1的哈希结果加上m2，再进行哈希之后，其结果与m1加m2直接哈希的结果相等。对于多个消息同理。这个漏洞是由MD结构的定义决定的，所有使用MD结构的哈希函数算法都有这个缺点。

要想修补这个漏洞也很容易，只需要在最后一个compress函数中略作修改，使之与前面的compress函数不同即可。

###### 长度扩展攻击的实现

长度扩展攻击的实现非常简单，直接调用SM3算法即可。在示例程序中，m1等于”hello”,m2等于”world”。程序中定义了三个SM3\_basic类的实例a,b,c。其中a对”hello”做哈希，b以a的结果为初始向量，对”world”做哈希。故b的结果为hash(hash(m1)||m2)。c直接对”hello”和”world”做哈希，故c的结果为hash(m1||m2)。



分别打印a、b、c的结果。下图中a、b、c的结果分别在一、二、三行。可以看到：b,c的结果相等，故长度扩展攻击成立。

