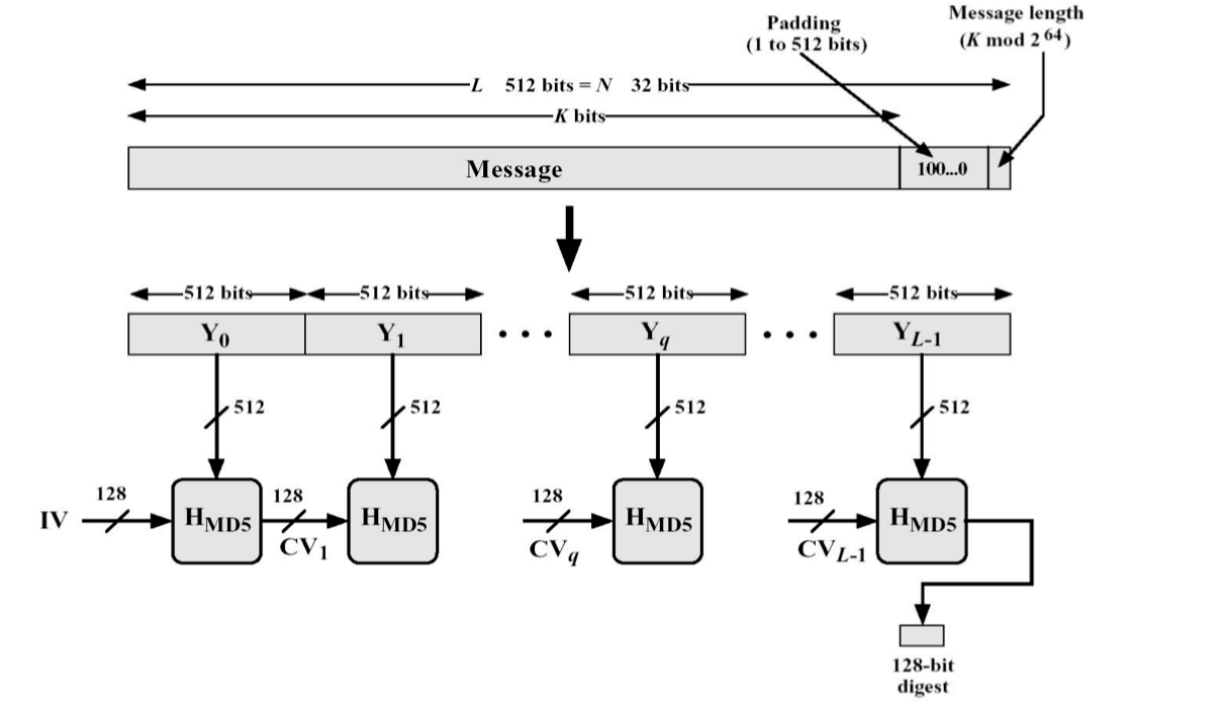
MD5

1.算法原理概述

- MD5使用little-endian(小端模式)，输入任意不定长度信息，以512-bit进行分组，生成四个32-bit数据，最后联合输出固定128-bit的信息摘要。
- MD5算法的基本过程为：填充、分块、缓冲区初始化、循环压缩、得出结果。 基本流程图如下



2.算法步骤及其实现

1. 填充 因为对于我们所需要加密的信息而言，一般来讲都是字符串，所以首先进行对字符串的处理，就是所谓的填充，每一个自负的长度是8位，填充的规则如下 （1） 在长度为 K bits 的原始消息数据尾部填充长度为 P bits 的标识 100…0，1□P□ 512 (即至少要填充1个bit)，使得填充后的消息位 数为：K + P = 448 (mod 512) 注意到当 K = 448 (mod 512) 时，需要 P= 512 （2） 再向上述填充好的消息尾部附加 K 值的低64位 (即 K mod 2^{64})， 最后得到一个长度位数为 K + P + 64 = 0 (mod 512) 的消息 具体实现如下

```
    cin>>str;
    int len=str.size();
    int bit_len = len * 8;
    int left = bit_len%512;
    int fill;
    if(left < 448){
        fill=448 - left;
    }
    else{
        fill=448 - left + 512;
    }
```

- len:字符串长度
- bit_len:字符串的位长度(字符串长度位多少个位)
- left:bit_len对512取余的结果
- fill:要填充的位数 接下来我们还需要64位来存储bit_len 在这里64位分为两个32位，分别将它们命名为AB，按照小端规则排列就是BA，若是长度位40，用十六进制表示就是0X00000028 那么这里A=0X00000000，B=0X00000028 首先我们要将长度转换位二进制，这里我使用的是vector

```
vector<int>v;
    int temp=length;
    while(temp>0){
        int add=temp%2;
        temp/=2;
        v.push_back(add);
    }
```

当然这里得到的二进制数字是反的，所以我们还需要做一下逆变换

```
for(int i=0;i<len;i++){
        tail[i+front]=v[len-i-1];
    }
```

tail是一个64位的数组，用来存储bit_len的长度，front是在二进制数字前面补得位数 例如上面的例子，我们转换之后得到的十六进制是28，但是我们填入的时候是00000028，这时候前面就要填上6个0 front就是表示我们前面所需要填的0的数量 求front操作如下

```
if(len%8==0){
        front=0;
    }
    else{
        front=8-(len%8);
    }
```

然后进行分组，每512个位是一组，我们在填充的时候最后一组的后64位记录长度，前面的448位在表示完原文信息之后进行填充的时候第一位填1，剩下的填0

```
while(start < 512 && pos<len){
            c = str[pos];
            char_to_bin(c, Y, start);
            start += 8;
            pos++;
        }
        if(start != 512 && start != 0){
            Y[start] = 1;
        }
        if(start == 0 && fill == 448){
            Y[start] = 1;
        }
```

- start:因为每个模块要512位，所以start用来表示当前所填充的内容应该在多少位上，所以在while里面它每次加8
- pos:表示当前要写入的是字符串中对应自负的索引 第一个if语句表示字符串写入完毕了，在后面补一个1 第二个if语句表示字符串写完的时候，对应模块正好落在512位的位置上，意味着要开始下个模块了，而此时的fill应该等于448，所以在后面补1

```
if(i == group - 1) {
            for(int k = 0; k < 64; k++) {
                Y[512 - 64 + k] = tail[k];
            }
        }
```

上面这段代码表示到了最后一组的时候需要补充上tail的内容

2. 分组 512个字节需要分成16组，每组32个字节 这16组我们可以用一个长度为16的数组来表示，每个数组对应一个无符号整数来表示32个字节 在这里还可以通过位运算来简化过程

```
  for(int i=0;i<16;i++){
      unsigned int temp[4]={0};
      for(int j=0;j<4;j++){
          for(int k=0;k<8;k++){
              temp[j]+=a[i*32+8*j+k]*pow(2,7-k);
          }
          b[i]+=temp[j]<<(8*j);
      }
  }
```

   这里的b是用来存储结果的数组，这里面最后通过循环与位运算实现了把每个512位的模块分成了16组

3. 缓冲区初始化 所谓的缓冲区就是4个32位的数

```
unsigned int MD[4]={0x67452301,0xEFCDAB89,0x98BADCFE,0x10325476};
```

4. 循环压缩 先来认识几个函数

```
unsigned int F(unsigned int b,unsigned int c,unsigned int d){
 return (b & c)|(~b & d);
```

```
}
unsigned int G(unsigned int b,unsigned int c,unsigned int d){
 return (b & d)|(c & ~d);
}
unsigned int H(unsigned int b,unsigned int c,unsigned int d){
 return (b ^ c ^ d);
}
unsigned int I(unsigned int b,unsigned int c,unsigned int d){
 return (c ^ (b | ~d));
}
unsigned int CLS(unsigned int a, int s){
 unsigned int temp1=a << s;
 unsigned int temp2=a >> (32-s);
 return ((a << s) | (a >> (32-s)));
}
```

其中FGHI都是一些逻辑运算，CLS是循环移位，这里使用了一个非常巧妙的方法，就是分别向左向右进行移位，然后再对其进行或运算 还有4个复杂一点的函数

```
   FF(a,b,c,d,Mj,s,ti)//a=b+((a+F(b,c,d)+Mj+ti)<<<s)
   GG(a,b,c,d,Mj,s,ti)//a=b+((a+G(b,c,d)+Mj+ti)<<<s)
   HH(a,b,c,d,Mj,s,ti)//a=b+((a+H(b,c,d)+Mj+ti)<<<s)
   II(a,b,c,d,Mj,s,ti)//a=b+((a+I(b,c,d)+Mj+ti)<<<s)
```

具体实现如下

```
void FF(unsigned int &a,unsigned int b,unsigned int c,unsigned int d,unsigned
int Mj,unsigned int s,unsigned int ti){
    unsigned int temp=a+F(b,c,d)+Mj+ti;
    a=b+CLS(temp,s);

}
void GG(unsigned int &a,unsigned int b,unsigned int c,unsigned int d,unsigned
int Mj,unsigned int s,unsigned int ti){
    unsigned int temp=a+G(b,c,d)+Mj+ti;
    a=b+CLS(temp,s);

}
void HH(unsigned int &a,unsigned int b,unsigned int c,unsigned int d,unsigned
int Mj,unsigned int s,unsigned int ti){
    unsigned int temp=a+H(b,c,d)+Mj+ti;
    a=b+CLS(temp,s);

}
void II(unsigned int &a,unsigned int b,unsigned int c,unsigned int d,unsigned
int Mj,unsigned int s,unsigned int ti){
    unsigned int temp=a+I(b,c,d)+Mj+ti;
    a=b+CLS(temp,s);

}
```

这里要注意的是第一个参数a需要使用引用，因为我们需要在函数里面改变它的值 然后就需要进行冗长的4*16次迭代运算

```
unsigned int A=MD[0];
    unsigned int B=MD[1];
    unsigned int C=MD[2];
    unsigned int D=MD[3];
    unsigned int a,b,c,d;
    a=A;
    b=B;
    c=C;
    d=D;
    FF(a,b,c,d,X[0],7,0xd76aa478);
    FF(d,a,b,c,X[1],12,0xe8c7b756);
    FF(c,d,a,b,X[2],17,0x242070db);
    FF(b,c,d,a,X[3],22,0xc1bdceee);
    FF(a,b,c,d,X[4],7,0xf57c0faf);
    FF(d,a,b,c,X[5],12,0x4787c62a);
    FF(c,d,a,b,X[6],17,0xa8304613);
    FF(b,c,d,a,X[7],22,0xfd469501);
    FF(a,b,c,d,X[8],7,0x698098d8);
    FF(d,a,b,c,X[9],12,0x8b44f7af);
    FF(c,d,a,b,X[10],17,0xffff5bb1);
    FF(b,c,d,a,X[11],22,0x895cd7be);
    FF(a,b,c,d,X[12],7,0x6b901122);
    FF(d,a,b,c,X[13],12,0xfd987193);
    FF(c,d,a,b,X[14],17,0xa679438e);
    FF(b,c,d,a,X[15],22,0x49b40821);
    GG(a,b,c,d,X[1],5,0xf61e2562);
    GG(d,a,b,c,X[6],9,0xc040b340);
    GG(c,d,a,b,X[11],14,0x265e5a51);
    GG(b,c,d,a,X[0],20,0xe9b6c7aa);
    GG(a,b,c,d,X[5],5,0xd62f105d);
    GG(d,a,b,c,X[10],9,0x02441453);
    GG(c,d,a,b,X[15],14,0xd8a1e681);
    GG(b,c,d,a,X[4],20,0xe7d3fbc8);
    GG(a,b,c,d,X[9],5,0x21e1cde6);
    GG(d,a,b,c,X[14],9,0xc33707d6);
    GG(c,d,a,b,X[3],14,0xf4d50d87);
    GG(b,c,d,a,X[8],20,0x455a14ed);
    GG(a,b,c,d,X[13],5,0xa9e3e905);
    GG(d,a,b,c,X[2],9,0xfcefa3f8);
    GG(c,d,a,b,X[7],14,0x676f02d9);
    GG(b,c,d,a,X[12],20,0x8d2a4c8a);
    HH(a,b,c,d,X[5],4,0xfffa3942);
    HH(d,a,b,c,X[8],11,0x8771f681);
    HH(c,d,a,b,X[11],16,0x6d9d6122);
    HH(b,c,d,a,X[14],23,0xfde5380c);
    HH(a,b,c,d,X[1],4,0xa4beea44);
    HH(d,a,b,c,X[4],11,0x4bdecfa9);
    HH(c,d,a,b,X[7],16,0xf6bb4b60);
    HH(b,c,d,a,X[10],23,0xbebfbc70);
```

```
    HH(a,b,c,d,X[13],4,0x289b7ec6);
    HH(d,a,b,c,X[0],11,0xeaa127fa);
    HH(c,d,a,b,X[3],16,0xd4ef3085);
    HH(b,c,d,a,X[6],23,0x04881d05);
    HH(a,b,c,d,X[9],4,0xd9d4d039);
    HH(d,a,b,c,X[12],11,0xe6db99e5);
    HH(c,d,a,b,X[15],16,0x1fa27cf8);
    HH(b,c,d,a,X[2],23,0xc4ac5665);
    II(a,b,c,d,X[0],6,0xf4292244);
    II(d,a,b,c,X[7],10,0x432aff97);
    II(c,d,a,b,X[14],15,0xab9423a7);
    II(b,c,d,a,X[5],21,0xfc93a039);
    II(a,b,c,d,X[12],6,0x655b59c3);
    II(d,a,b,c,X[3],10,0x8f0ccc92);
    II(c,d,a,b,X[10],15,0xffeff47d);
    II(b,c,d,a,X[1],21,0x85845dd1);
    II(a,b,c,d,X[8],6,0x6fa87e4f);
    II(d,a,b,c,X[15],10,0xfe2ce6e0);
    II(c,d,a,b,X[6],15,0xa3014314);
    II(b,c,d,a,X[13],21,0x4e0811a1);
    II(a,b,c,d,X[4],6,0xf7537e82);
    II(d,a,b,c,X[11],10,0xbd3af235);
    II(c,d,a,b,X[2],15,0x2ad7d2bb);
    II(b,c,d,a,X[9],21,0xeb86d391);
    A+=a;
    B+=b;
    C+=c;
    D+=d;
    MD[0]=A;
    MD[1]=B;
    MD[2]=C;
    MD[3]=D;
```

5. 得出结果 至此计算已经完毕，然后使用unsigned char对结果进行存储并输出

```
unsigned char digit[16];
    for(int i = 0; i < 4; i++) {
        for(int j = 0; j < 4; j++) {
            digit[i * 4 + j] = MD[i] >> (8 * j);
        }
    }
    for(int i = 0; i < 16; i++) {
       printf("%x", digit[i]);
    }
```

3. 测试结果

```
dongchenyudeMacBook-Pro:信息安全 dongchenyu$ cd "/Users/dongchenyu/Documents/信息安全/" && g++ MD5.cpp -o MD5 && "/Users/dongchenyu/Documents/
信息安全/"MD5
a
cc175b9c0f1b6a831c399e269772661
dongchenyudeMacBook-Pro:信息安全 dongchenyu$ cd "/Users/dongchenyu/Documents/信息安全/" && g++ MD5.cpp -o MD5 && "/Users/dongchenyu/Documents/
信息安全/"MD5
ab
187ef4436122d1cc2f40dc2b92f0eba0
dongchenyudeMacBook-Pro:信息安全 dongchenyu$ cd "/Users/dongchenyu/Documents/信息安全/" && g++ MD5.cpp -o MD5 && "/Users/dongchenyu/Documents/
信息安全/"MD5
abc
90150983cd24fb0d6963f7d28e17f72
dongchenyudeMacBook-Pro:信息安全 dongchenyu$ cd "/Users/dongchenyu/Documents/信息安全/" && g++ MD5.cpp -o MD5 && "/Users/dongchenyu/Documents/
信息安全/"MD5
abcd
e2fc714c4727ee9395f324cd2e7f331f
dongchenyudeMacBook-Pro:信息安全 dongchenyu$ cd "/Users/dongchenyu/Documents/信息安全/" && g++ MD5.cpp -o MD5 && "/Users/dongchenyu/Documents/
信息安全/"MD5
abcde
ab56b4d92b40713acc5af89985d4b786
```