

西安电子科技大学

物联网安全实验课程 实验报告

实验名称 MD5 加密算法

物联网工程 1803041 班

姓名 魏红旭 学号 18030400014

同作者

实验日期 2021 年 5 月 25 日

成 绩

指导教师评语：

指导教师：

年月日

实验报告内容基本要求及参考格式

- 一、实验目的
- 二、实验所用仪器（或实验环境）
- 三、实验基本原理及步骤（或方案设计及理论计算）
- 四、实验数据记录（或仿真及软件设计）
- 五、实验结果分析及回答问题（或测试环境及测试结果）

一、实验目的:

编程实现 MD5 算法, 深入理解 MD5 加密解密原理

二、实验所用仪器 (或实验环境)

计算机科学与技术学院实验中心, 可接入 Internet 网台式机 44 台。

三、实验基本原理及要求

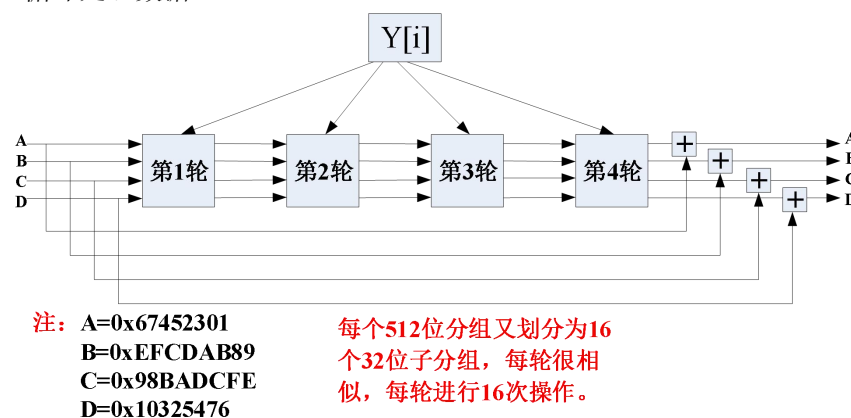
实验原理:

第一步: 添加填充位, 如果输入明文的长度(bit)对 512 求余的结果不等于 448, 就需要填充使得对 512 求余的结果等于 448。填充的方法是填充一个 1 和 n 个 0。填充完后, 信息的长度就为 $N*512+448(\text{bit})$

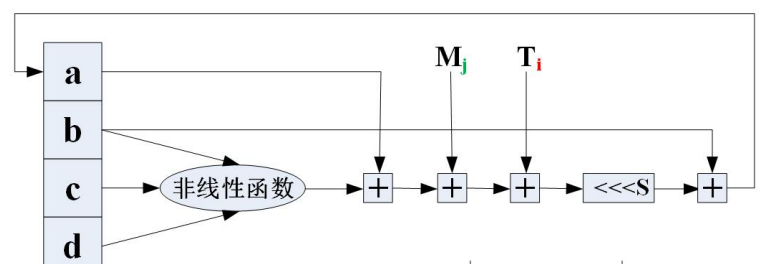
第二步: 填充长度, 在第一步结果之后再填充上原消息的长度, 可用来进行的存储长度为 64 位。如果消息长度大于 2^{64} , 则只使用其低 64 位的值, 即 (消息长度 对 2^{64} 取模)。在此步骤进行完毕后, 最终消息长度就是 $N*512+448+64=(N+1)*512$ 。

第三步, 初始化缓冲区, 一个 128bit 的缓冲区可用于保存 hash 函数中间和最终结果。可表示为 4 个 32bit 的寄存器 (A,B,C,D)。其中 $A=(01234567)_{16}$, $B=(89ABCDEF)_{16}$, $C=(FEDCBA98)_{16}$, $D=(76543210)_{16}$ 。如果在程序中定义应该是 $A=0X67452301L$, $B=0XEFCDA89L$, $C=0X98BADCFEL$, $D=0X10325476L$

第四步: 循环处理数据



其中每一轮操作为



第五步, 级联输出, 给出 MD5 算法加密后的密文。

实验要求:

明文自定义 (例如, 可以是一句英文名言名句), 利用 MD5 算法对其加密, 给出实验过程以及加密结果

四、实验步骤及实验数据记录：（要有文字描述和必要截图）

- 算法说明：

MD5 以 512 位分组来处理输入的信息，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 位分组级联后将生成一个 128 位散列值。

- 代码结构：



- 具体实现步骤：

1. 我们第一步要做的就是填充，如果输入信息的长度(bit)对 512 求余的结果 不等于 448，就需要填充使得对 512 求余的结果等于 448。填充的方法是填充一个 1 和 n 个 0。填充完后，信息的长度就为 $N \times 512 + 448$ (bit)；代码如下图所示：

```
private String Main_process() {  
    byte [] inputBytes=str.getBytes();  
    int str_length=inputBytes.length;    //字符串字节长度  
    int group_number=0;                  //分组个数，每组512位（64字节）  
    group_number=str_length/64;          //字节长度/64字节  
    long []groups=null;                  //每组按4字节继续细分
```

2. 将输入信息的原始长度 b(bit)表示成一个 64-bit 的数字，把它添加到上一步的结果后面(在 32 位的机器上，这 64 位将用 2 个字来表示并且低位在前)。当遇到 b 大于 2^{64} 这种极少见的情况时，b 的高位被截去，仅使用 b 的低 64 位。经过上面两步，数据就被填补成长度为 512(bit)的倍数。也

就是说，此时的数据长度是 16 个字(32byte)的整数倍。代码如下图所示：

```
for(int i=0;i<group_number;i++){ //处理分组，循环运算，循环次数为分组个数
    groups=Create_NewGroup(inputBytes, index: i*64);
    Transform(groups);           //处理分组，核心算法
}
```

```
private long[] Create_NewGroup(byte[] inputBytes,int index){
    long [] temp=new long[16];           //将每一个512位的分组再细分成16个小组，每个小组64位（8个字节）
    for(int i=0;i<16;i++){
        temp[i]=DealHi(inputBytes[4*i+index])|
            (DealHi(inputBytes[4*i+1+index]))<<8|
            (DealHi(inputBytes[4*i+2+index]))<<16|
            (DealHi(inputBytes[4*i+3+index]))<<24;
    }
    return temp;
}
```

3. 装入标准的幻数（四个整数），用一个四个字的缓冲器(A, B, C, D)来计算报文摘要，A,B,C,D 分别是 32 位的寄存器，初始化使用的是十六进制表示的数字，注意低字节在前：

word A: 01 23 45 67

word B: 89 ab cd ef

word C: fe dc ba 98

word D: 76 54 32 10

```
//定义标准幻数
private static final long A=0x67452301L;
private static final long B=0xefcdab89L;
private static final long C=0x98badcfeL;
private static final long D=0x10325476L;
```

4. 四轮循环运算：循环的次数是分组的个数（N+1）

(1) 定义 4 个辅助函数，每个函数的输入是三个 32 位的字，输出是一个 32 位的字：(&是与,|是或,~是非,^是异或)，代码如下图所示：

$$F(X,Y,Z)=(X\&Y)|((\sim X)\&Z)$$
$$G(X,Y,Z)=(X\&Z)|(Y\&(\sim Z))$$
$$H(X,Y,Z)=X\wedge Y\wedge Z$$
$$I(X,Y,Z)=Y\wedge(X|(\sim Z))$$

```

private long F(long x, long y, long z) {
    return (x & y) | ((~x) & z);
}
private long G(long x, long y, long z) {
    return (x & z) | (y & (~z));
}
private static long H(long x, long y, long z) {
    return x ^ y ^ z;
}
private long I(long x, long y, long z) {
    return y ^ (x | (~z));
}
}

```

(2) 设 M_j 表示消息的第 j 个子分组（从 0 到 15）：代码如下图所示：

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

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

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

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

```

private long FF(long a, long b, long c, long d, long x, long s,
    long ac) {
    a += (F(b, c, d) & 0xFFFFFFFFL) + x + ac;
    a = ((a & 0xFFFFFFFFL) << s) | ((a & 0xFFFFFFFFL) >>> (32 - s));
    a += b;
    return (a & 0xFFFFFFFFL);
}
private long GG(long a, long b, long c, long d, long x, long s,
    long ac) {
    a += (G(b, c, d) & 0xFFFFFFFFL) + x + ac;
    a = ((a & 0xFFFFFFFFL) << s) | ((a & 0xFFFFFFFFL) >>> (32 - s));
    a += b;
    return (a & 0xFFFFFFFFL);
}
private long HH(long a, long b, long c, long d, long x, long s,
    long ac) {
    a += (H(b, c, d) & 0xFFFFFFFFL) + x + ac;
    a = ((a & 0xFFFFFFFFL) << s) | ((a & 0xFFFFFFFFL) >>> (32 - s));
    a += b;
    return (a & 0xFFFFFFFFL);
}
private long II(long a, long b, long c, long d, long x, long s,
    long ac) {
    a += (I(b, c, d) & 0xFFFFFFFFL) + x + ac;
    a = ((a & 0xFFFFFFFFL) << s) | ((a & 0xFFFFFFFFL) >>> (32 - s));
    a += b;
    return (a & 0xFFFFFFFFL);
}
}

```



```

//第一轮
a = FF(a, b, c, d, groups[0], S11, ac: 0xd76aa478L);
d = FF(d, a, b, c, groups[1], S12, ac: 0xe8c7b756L);
c = FF(c, d, a, b, groups[2], S13, ac: 0x242070dbL);
b = FF(b, c, d, a, groups[3], S14, ac: 0xc1bdceeeL);
a = FF(a, b, c, d, groups[4], S11, ac: 0xf57c0fafL);
d = FF(d, a, b, c, groups[5], S12, ac: 0x4787c62aL);
c = FF(c, d, a, b, groups[6], S13, ac: 0xa8304613L);
b = FF(b, c, d, a, groups[7], S14, ac: 0xfd469501L);
a = FF(a, b, c, d, groups[8], S11, ac: 0x698098d8L);
d = FF(d, a, b, c, groups[9], S12, ac: 0x8b44f7afL);
c = FF(c, d, a, b, groups[10], S13, ac: 0xffff5bb1L);
b = FF(b, c, d, a, groups[11], S14, ac: 0x895cd7beL);
a = FF(a, b, c, d, groups[12], S11, ac: 0x6b901122L);
d = FF(d, a, b, c, groups[13], S12, ac: 0xfd987193L);
c = FF(c, d, a, b, groups[14], S13, ac: 0xa679438eL);
b = FF(b, c, d, a, groups[15], S14, ac: 0x49b40821L);
//第二轮
a = GG(a, b, c, d, groups[1], S21, ac: 0xf61e2562L);
d = GG(d, a, b, c, groups[6], S22, ac: 0xc040b340L);
c = GG(c, d, a, b, groups[11], S23, ac: 0x265e5a51L);
b = GG(b, c, d, a, groups[0], S24, ac: 0xe9b6c7aaL);
a = GG(a, b, c, d, groups[5], S21, ac: 0xd62f105dL);
d = GG(d, a, b, c, groups[10], S22, ac: 0x2441453L);
c = GG(c, d, a, b, groups[15], S23, ac: 0xd8a1e681L);
b = GG(b, c, d, a, groups[4], S24, ac: 0xe7d3fbc8L);

```

5. 每轮循环后，将 A, B, C, D 分别加上 a, b, c, d，然后进入下一循环

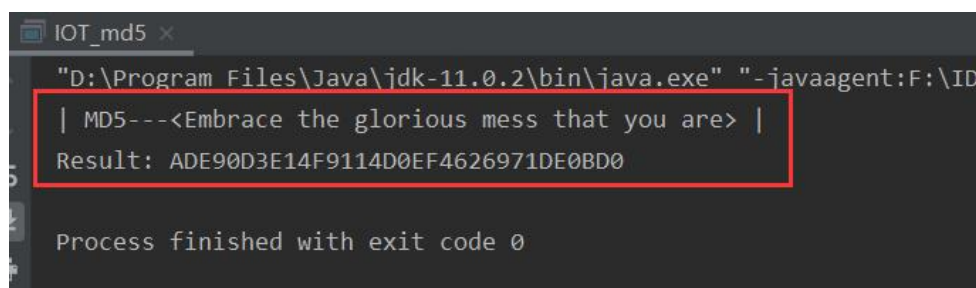
```

result[0] += a; //加到先前计算结果中
result[1] += b;
result[2] += c;
result[3] += d;
result[0]=result[0]&0xFFFFFFFFL;
result[1]=result[1]&0xFFFFFFFFL;
result[2]=result[2]&0xFFFFFFFFL;
result[3]=result[3]&0xFFFFFFFFL;

```

五、实验结果分析及实验总结与体会

● 实验结果：



```
IOT_md5 x
"D:\Program Files\Java\jdk-11.0.2\bin\java.exe" "-javaagent:F:\ID
| MD5---<Embrace the glorious mess that you are> |
Result: ADE90D3E14F9114D0EF4626971DE0BD0
Process finished with exit code 0
```

开始我们传入的字符串为：“Embrace the glorious mess that you are”，最后得到的 MD5 值为“ADE90D3E14F9114D0EF4626971DE0BD0”。

● 实验总结

任何消息经过散列函数处理后，都会产生一个唯一的散列值，这个散列值可以用来验证消息的完整性，MD5 也正是利用了这个特点，通过对 MD5 算法原理的学习及实验，我对散列函数压缩性、容易计算、抗修改性的特点有了更加深刻的体会，对于 MD5 算法的验证基本上只有借助计算机才可以进行，因为 MD5 算法在计算量上相对来说还是比较复杂的，通过计算机我们可以很简单的进行求和运算，当完成最后一个明文的分组运算时，A、B、C、D 中的数值就是最后的结果（即散列函数值）。

六、源代码

```
1. class MD5{
2.     //初始化 MD5 参数
3.     static final int S11 = 7;
4.     static final int S12 = 12;
5.     static final int S13 = 17;
6.     static final int S14 = 22;
7.     static final int S21 = 5;
8.     static final int S22 = 9;
9.     static final int S23 = 14;
10.    static final int S24 = 20;
11.    static final int S31 = 4;
12.    static final int S32 = 11;
13.    static final int S33 = 16;
14.    static final int S34 = 23;
```

```

15.     static final int S41 = 6;
16.     static final int S42 = 10;
17.     static final int S43 = 15;
18.     static final int S44 = 21;
19.     //定义标准幻数
20.     private static final long A=0x67452301L;
21.     private static final long B=0xefcdab89L;
22.     private static final long C=0x98badcfeL;
23.     private static final long D=0x10325476L;
24.
25.     private long [] result={A,B,C,D}; //存储结果
26.     static final String hexs[]={ "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "A",
    "B", "C", "D", "E", "F"};
27.     private String str;
28.
29.     public MD5(String str) {
30.         this.str=str;
31.         String result=Main_process();
32.         System.out.println("| MD5---<"+str+">+" | \nResult: "+result);
33.     }
34.
35.     private String Main_process() {
36.         byte [] inputBytes=str.getBytes();
37.         int str_length=inputBytes.length; //字符串字节长度
38.         int group_number=0; //分组个数，每组 512 位（64 字节）
39.         group_number=str_length/64; //字节长度/64 字节
40.         long []groups=null; //每组按 4 字节继续细分
41.
42.         for(int i=0;i<group_number;i++){ //处理分组，循环运算，循环次数位分组个
    数
43.             groups=Create_NewGroup(inputBytes,i*64);
44.             Transform(groups); //处理分组，核心算法
45.         }
46.
47.         int rest=str_length%64; //处理 512 位分组后的余数
48.         byte [] tempBytes=new byte[64];
49.         if(rest<=56){
50.             for(int i=0;i<rest;i++)
51.                 tempBytes[i]=inputBytes[str_length-rest+i];
52.             if(rest<56){
53.                 tempBytes[rest]=(byte)(1<<7);
54.                 for(int i=1;i<56-rest;i++)
55.                     tempBytes[rest+i]=0;
56.             }

```



```

57.         long len=(long)(str_length<<3);
58.         for(int i=0;i<8;i++){
59.             tempBytes[56+i]=(byte)(len&0xFFL);
60.             len=len>>8;
61.         }
62.         groups=Create_NewGroup(tempBytes,0);
63.         Transform(groups);
64.     }else{
65.         for(int i=0;i<rest;i++)
66.             tempBytes[i]=inputBytes[str_length-rest+i];
67.         tempBytes[rest]=(byte)(1<<7);
68.         for(int i=rest+1;i<64;i++)
69.             tempBytes[i]=0;
70.         groups=Create_NewGroup(tempBytes,0);
71.         Transform(groups);
72.
73.         for(int i=0;i<56;i++)
74.             tempBytes[i]=0;
75.         long len=(long)(str_length<<3);
76.         for(int i=0;i<8;i++){
77.             tempBytes[56+i]=(byte)(len&0xFFL);
78.             len=len>>8;
79.         }
80.         groups=Create_NewGroup(tempBytes,0);
81.         Transform(groups);
82.     }
83.     return ToHexString();
84. }
85.
86. private String ToHexString(){ //Hash 值转换成十六进制的字符串
87.     String resStr="";
88.     long temp=0;
89.     for(int i=0;i<4;i++){
90.         for(int j=0;j<4;j++){
91.             temp=result[i]&0x0FL;
92.             String a=hexs[(int)(temp)];
93.             result[i]=result[i]>>4;
94.             temp=result[i]&0x0FL;
95.             resStr+=hexs[(int)(temp)]+a;
96.             result[i]=result[i]>>4;
97.         }
98.     }
99.     return resStr;
100. }

```

```

101.     private void Transform(long[] groups) { //对分组进行处理，每个分组 64 字
节
102.         long a = result[0], b = result[1], c = result[2], d = result[3];
103.         //第一轮
104.         a = FF(a, b, c, d, groups[0], S11, 0xd76aa478L);
105.         d = FF(d, a, b, c, groups[1], S12, 0xe8c7b756L);
106.         c = FF(c, d, a, b, groups[2], S13, 0x242070dbL);
107.         b = FF(b, c, d, a, groups[3], S14, 0xc1bdceeeL);
108.         a = FF(a, b, c, d, groups[4], S11, 0xf57c0fafL);
109.         d = FF(d, a, b, c, groups[5], S12, 0x4787c62aL);
110.         c = FF(c, d, a, b, groups[6], S13, 0xa8304613L);
111.         b = FF(b, c, d, a, groups[7], S14, 0xfd469501L);
112.         a = FF(a, b, c, d, groups[8], S11, 0x698098d8L);
113.         d = FF(d, a, b, c, groups[9], S12, 0x8b44f7afL);
114.         c = FF(c, d, a, b, groups[10], S13, 0xfffff5bb1L);
115.         b = FF(b, c, d, a, groups[11], S14, 0x895cd7beL);
116.         a = FF(a, b, c, d, groups[12], S11, 0x6b901122L);
117.         d = FF(d, a, b, c, groups[13], S12, 0xfd987193L);
118.         c = FF(c, d, a, b, groups[14], S13, 0xa679438eL);
119.         b = FF(b, c, d, a, groups[15], S14, 0x49b40821L);
120.         //第二轮
121.         a = GG(a, b, c, d, groups[1], S21, 0xf61e2562L);
122.         d = GG(d, a, b, c, groups[6], S22, 0xc040b340L);
123.         c = GG(c, d, a, b, groups[11], S23, 0x265e5a51L);
124.         b = GG(b, c, d, a, groups[0], S24, 0xe9b6c7aaL);
125.         a = GG(a, b, c, d, groups[5], S21, 0xd62f105dL);
126.         d = GG(d, a, b, c, groups[10], S22, 0x2441453L);
127.         c = GG(c, d, a, b, groups[15], S23, 0xd8a1e681L);
128.         b = GG(b, c, d, a, groups[4], S24, 0xe7d3fbc8L);
129.         a = GG(a, b, c, d, groups[9], S21, 0x21e1cde6L);
130.         d = GG(d, a, b, c, groups[14], S22, 0xc33707d6L);
131.         c = GG(c, d, a, b, groups[3], S23, 0xf4d50d87L);
132.         b = GG(b, c, d, a, groups[8], S24, 0x455a14edL);
133.         a = GG(a, b, c, d, groups[13], S21, 0xa9e3e905L);
134.         d = GG(d, a, b, c, groups[2], S22, 0xfcefa3f8L);
135.         c = GG(c, d, a, b, groups[7], S23, 0x676f02d9L);
136.         b = GG(b, c, d, a, groups[12], S24, 0x8d2a4c8aL);
137.         //第三轮
138.         a = HH(a, b, c, d, groups[5], S31, 0xffffa3942L);
139.         d = HH(d, a, b, c, groups[8], S32, 0x8771f681L);
140.         c = HH(c, d, a, b, groups[11], S33, 0x6d9d6122L);
141.         b = HH(b, c, d, a, groups[14], S34, 0xfde5380cL);
142.         a = HH(a, b, c, d, groups[1], S31, 0xa4beea44L);
143.         d = HH(d, a, b, c, groups[4], S32, 0x4bdecfa9L);

```

```

144.         c = HH(c, d, a, b, groups[7], S33, 0xf6bb4b60L);
145.         b = HH(b, c, d, a, groups[10], S34, 0xbefb7c70L);
146.         a = HH(a, b, c, d, groups[13], S31, 0x289b7ec6L);
147.         d = HH(d, a, b, c, groups[0], S32, 0xeeaa127faL);
148.         c = HH(c, d, a, b, groups[3], S33, 0xd4ef3085L);
149.         b = HH(b, c, d, a, groups[6], S34, 0x4881d05L);
150.         a = HH(a, b, c, d, groups[9], S31, 0xd9d4d039L);
151.         d = HH(d, a, b, c, groups[12], S32, 0xe6db99e5L);
152.         c = HH(c, d, a, b, groups[15], S33, 0x1fa27cf8L);
153.         b = HH(b, c, d, a, groups[2], S34, 0xc4ac5665L);
154.         //第四轮
155.         a = II(a, b, c, d, groups[0], S41, 0xf4292244L);
156.         d = II(d, a, b, c, groups[7], S42, 0x432aff97L);
157.         c = II(c, d, a, b, groups[14], S43, 0xab9423a7L);
158.         b = II(b, c, d, a, groups[5], S44, 0xfc93a039L);
159.         a = II(a, b, c, d, groups[12], S41, 0x655b59c3L);
160.         d = II(d, a, b, c, groups[3], S42, 0x8f0ccc92L);
161.         c = II(c, d, a, b, groups[10], S43, 0xffeff47dL);
162.         b = II(b, c, d, a, groups[1], S44, 0x85845dd1L);
163.         a = II(a, b, c, d, groups[8], S41, 0x6fa87e4fL);
164.         d = II(d, a, b, c, groups[15], S42, 0xfe2ce6e0L);
165.         c = II(c, d, a, b, groups[6], S43, 0xa3014314L);
166.         b = II(b, c, d, a, groups[13], S44, 0x4e0811a1L);
167.         a = II(a, b, c, d, groups[4], S41, 0xf7537e82L);
168.         d = II(d, a, b, c, groups[11], S42, 0xbd3af235L);
169.         c = II(c, d, a, b, groups[2], S43, 0x2ad7d2bbL);
170.         b = II(b, c, d, a, groups[9], S44, 0xeb86d391L);
171.
172.         result[0] += a; //加到先前计算结果中
173.         result[1] += b;
174.         result[2] += c;
175.         result[3] += d;
176.         result[0]=result[0]&0xFFFFFFFFL;
177.         result[1]=result[1]&0xFFFFFFFFL;
178.         result[2]=result[2]&0xFFFFFFFFL;
179.         result[3]=result[3]&0xFFFFFFFFL;
180.     }
181.     private long DealHi(byte b){ //需要对符号位进行处理
182.         return b < 0 ? b & 0x7F + 128 : b;
183.     }
184.     private long[] Create_NewGroup(byte[] inputBytes,int index){
185.         long [] temp=new long[16]; //将每一个 512 位的分组再细
            分成 16 个小组，每个小组 64 位（8 个字节）
186.         for(int i=0;i<16;i++){

```

```

187.         temp[i]=DealHi(inputBytes[4*i+index])|
188.             (DealHi(inputBytes[4*i+1+index]))<<8|
189.             (DealHi(inputBytes[4*i+2+index]))<<16|
190.             (DealHi(inputBytes[4*i+3+index]))<<24;
191.     }
192.     return temp;
193. }
194. private long F(long x, long y, long z) {
195.     return (x & y) | ((~x) & z);
196. }
197. private long G(long x, long y, long z) {
198.     return (x & z) | (y & (~z));
199. }
200. private static long H(long x, long y, long z) {
201.     return x ^ y ^ z;
202. }
203. private long I(long x, long y, long z) {
204.     return y ^ (x | (~z));
205. }
206. private long FF(long a, long b, long c, long d, long x, long s,
207.                 long ac) {
208.     a += (F(b, c, d)&0xFFFFFFFFL) + x + ac;
209.     a = ((a&0xFFFFFFFFL)<< s) | ((a&0xFFFFFFFFL) >>> (32 - s));
210.     a += b;
211.     return (a&0xFFFFFFFFL);
212. }
213. private long GG(long a, long b, long c, long d, long x, long s,
214.                 long ac) {
215.     a += (G(b, c, d)&0xFFFFFFFFL) + x + ac;
216.     a = ((a&0xFFFFFFFFL) << s) | ((a&0xFFFFFFFFL) >>> (32 - s));
217.     a += b;
218.     return (a&0xFFFFFFFFL);
219. }
220. private long HH(long a, long b, long c, long d, long x, long s,
221.                 long ac) {
222.     a += (H(b, c, d)&0xFFFFFFFFL) + x + ac;
223.     a = ((a&0xFFFFFFFFL) << s) | ((a&0xFFFFFFFFL) >>> (32 - s));
224.     a += b;
225.     return (a&0xFFFFFFFFL);
226. }
227. private long II(long a, long b, long c, long d, long x, long s,
228.                 long ac) {
229.     a += (I(b, c, d)&0xFFFFFFFFL) + x + ac;
230.     a = ((a&0xFFFFFFFFL) << s) | ((a&0xFFFFFFFFL) >>> (32 - s));

```

```
231.         a += b;
232.         return (a&0xFFFFFFFFL);
233.     }
234. }
235. public class IOT_md5 {
236.     public static void main(String []args){
237.         String str="Embrace the glorious mess that you are";
238.         MD5 md=new MD5(str);
239.     }
240. }
```