

# 个人信息

学号：1911410

姓名：付文轩

专业：信息安全

## 实验步骤及实验要求

### 实验步骤

#### 1. 算法分析

请参照教材内容，分析MD5算法实现的每一步原理。

#### 2. 算法实现：

利用Visual C++语言，自己编写MD5的实现代码，并检验代码实现的正确性。

#### 3. 雪崩效应检验：

尝试对一个长字符串进行Hash运算，并获得其运算结果。对该字符串进行轻微的改动，比如增加一个空格或标点，比较Hash结果值的改变位数。进行8次这样的测试。

### 实验要求

- 自己编写完整的MD5实现代码，并提交程序和程序流程图。
- 对编好的MD5算法，测试其雪崩效应，要求给出文本改变前和改变后的Hash值，并计算出改变的位数。写出8次测试的结果，并计算出平均改变的位数。

## MD5算法

### 消息填充

首先需要对信息进行填充，使其字节长度与448模512同余，即信息的字节长度扩展至

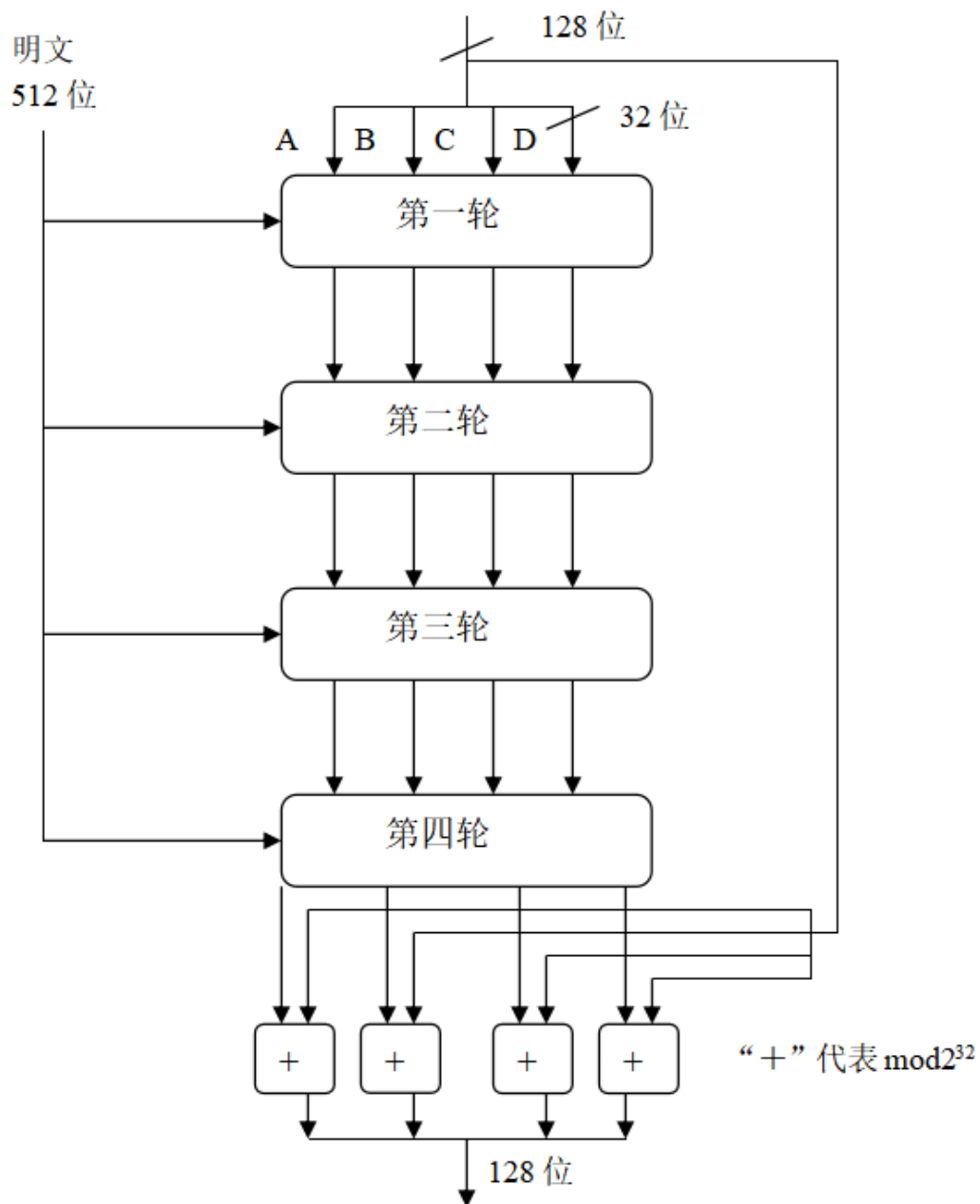
$n * 512 + 448$ ， $n$ 为一个正整数。填充的方法如下：在信息的后面填充第一位为1，其余各位均为0，直到满足上面的条件时才停止用0对信息的填充。然后，再在这个结果后面附加一个以64位二进制表示的填充前信息长度。经过这两步的处理，现在的信息字节长度为

$n * 512 + 448 + 64 = (n + 1) * 512$ ，即长度恰好是512的整数倍，这样做的目的是为了后面处理中对信息长度的要求。

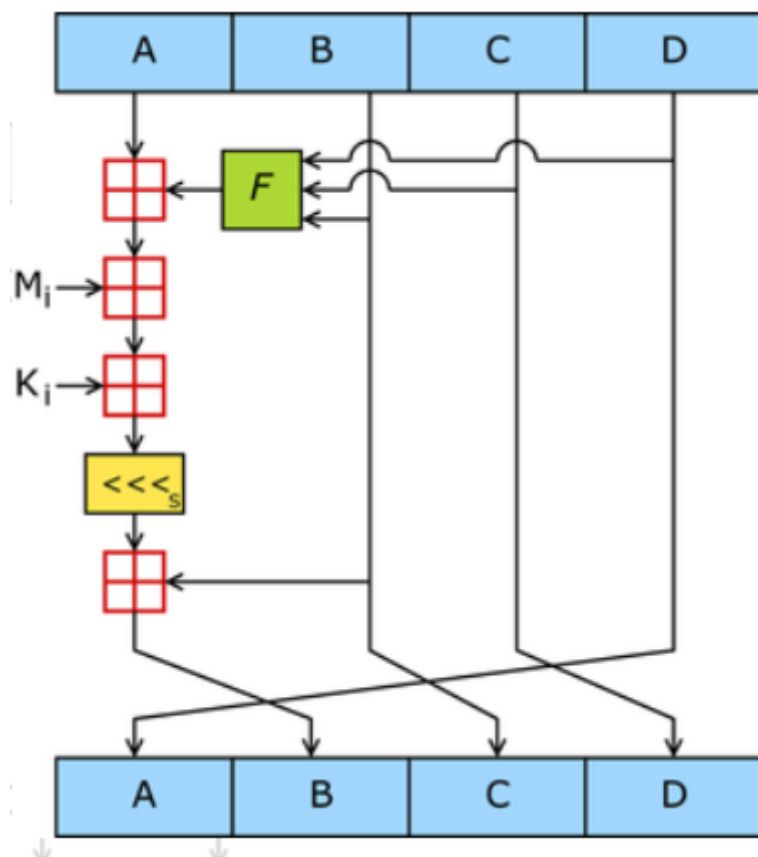
对应代码为：

```
1 // 进行填充
2 messageByte[lengthInByte >> 2] |= 0x80 << ((lengthInByte % 4) * 8);
3
4 // 附加信息
5 messageByte[groupNum * 16 - 2] = lengthInByte * 8;
```

### 计算过程



MD5的计算主要分为4轮，其中每一轮都会进行16次操作，也就是说总共会进行 $4 \times 16$ 次的操作。每次操作对A、B、C、D中的3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，文本的一个子分组（32位）和一个常数。再将所得结果向左循环移S位，并加上A、B、C、D其中之一。最后用该结果取代A、B、C、D其中之一。其中A、B、C、D的初始值为： $A_0 = 0x01234567$ ， $B_0 = 0x89abcdef$ ， $C_0 = 0xfedcba98$ ， $D_0 = 0x76543210$



F：一个非线性函数，一个函数运算一次

Mi：表示一个 32-bits 的输入数据

Ki：表示一个 32-bits 常数，用来完成每次不同的计算

## 逻辑函数

```

1 //基本逻辑函数
2 unsigned int F(unsigned int b, unsigned int c, unsigned int d) {
3     return ((b & c) | ((~b) & (d)));
4 }
5 unsigned int G(unsigned int b, unsigned int c, unsigned int d) {
6     return ((b & d) | (c & (~d)));
7 }
8 unsigned int H(unsigned int b, unsigned int c, unsigned int d) {
9     return (b ^ c ^ d);
10 }
11 unsigned int I(unsigned int b, unsigned int c, unsigned int d) {
12     return (c ^ (b | (~d)));
13 }
14
15 //x循环左移n位
16 unsigned int shift(unsigned int x, unsigned int n) {
17     return ((x << n) | (x >> (32 - n)));
18 }

```

## 算法主要实现代码

```

1 /// <summary>

```

```

2  /// 计算MD5的值
3  /// </summary>
4  /// <param name="message">任意长度的待处理消息</param>
5  /// <returns>128比特MD5字符串</returns>
6  string myGetMD5(string message) {
7      //初始化,, 这里要按照大端字节序的方式进行存储
8      unsigned int A = 0x67452301;
9      unsigned int B = 0xefcdab89;
10     unsigned int C = 0x98badcfe;
11     unsigned int D = 0x10325476;
12
13     // 得到消息的长度, 单位: 字节
14     int lengthInByte = message.length();
15     // 计算分组数量, 同时需要注意: 如果一开始就刚好是message=448(mod 512)也需要进行
16     // 10000填充, 所以算出来的直接+1
17     int groupNum = ((lengthInByte + 8) / 64) + 1;
18
19     // messageByte数组中的每一个元素, 对应消息二进制表示中的32bit (4字节)
20     // 每个分组的长度为 16 * 32bit = 512bit
21     unsigned int* messageByte = new unsigned int[groupNum * 16];
22
23     // 初始化
24     memset(messageByte, 0, sizeof(unsigned int) * groupNum * 16);
25
26     //将string类型变量 保存进 messageByte数组中
27     for (int i = 0; i < lengthInByte; i++)
28     {
29         //每个字中对应存储4个字节的数据
30         messageByte[i / 4] |= message[i] << ((i % 4) * 8);
31     }
32
33     // 进行填充
34     messageByte[lengthInByte >> 2] |= 0x80 << ((lengthInByte % 4) * 8);
35
36     // 附加信息
37     messageByte[groupNum * 16 - 2] = lengthInByte * 8;
38
39     // 临时变量, 用来保存每轮中每一步计算之后的abcd
40     unsigned int a, b, c, d;
41
42     // 循环处理每个分组
43     for (int i = 0; i < groupNum; i++)
44     {
45         a = A;
46         b = B;
47         c = C;
48         d = D;
49         unsigned int g;
50         int k;
51
52         // 对每个分组进行64轮压缩
53         for (int j = 0; j < 64; j++)
54         {
55             // 根据轮数不同, 使用不同的压缩函数
56             if (j < 16)
57             {
58                 // 对B C D进行基本逻辑函数运算
59                 g = F(b, c, d);

```

```

59         // 计算k，以从消息分组中取出对应的字
60         k = j;
61     }
62     else if (j >= 16 && j < 32)
63     {
64         // 对B C D进行基本逻辑函数运算
65         g = G(b, c, d);
66         // 计算k，以从消息分组中取出对应的字
67         k = (1 + 5 * j) % 16;
68     }
69     else if (j >= 32 && j < 48)
70     {
71         // 对B C D进行基本逻辑函数运算
72         g = H(b, c, d);
73         // 计算k，以从消息分组中取出对应的字
74         k = (5 + 3 * j) % 16;
75     }
76     else if (j >= 48 && j < 64)
77     {
78         // 对B C D进行基本逻辑函数运算
79         g = I(b, c, d);
80         // 计算k，以从消息分组中取出对应的字
81         k = (7 * j) % 16;
82     }
83
84     unsigned tempd = d;
85     d = c;
86     c = b;
87     b = b + shift(a + g + messageByte[i * 16 + k] + T[j], s[j]);
88     a = tempd;
89 }
90 // 更新值
91 A = a + A;
92 B = b + B;
93 C = c + C;
94 D = d + D;
95 }
96 //返回16进制形式字符串
97 return Int2HexString(A) + Int2HexString(B) + Int2HexString(C) +
Int2HexString(D);
98 }

```

## 实验计算结果

输入一个随机的字符串： 1a65dsg489121asdg

得到结果为：

```

1  =====MD5计算程序=====
2  请输入一个字符串(为了进行雪崩效应的测试，请输入的字符串长度>=8):
3  1a65dsg489121asdg
4  经过计算可以得到MD5值为:
5  07155a6fa59176c2a1a75581a0590dd8

```

使用工具计算得到：



验证得到实验结果正确

## 雪崩效应测试

### 计算位数不同的函数

```
1  /// <summary>
2  /// 雪崩效应测试时计算不同位数的函数
3  /// </summary>
4  /// <param name="NewSumry">改变过后的字符串</param>
5  /// <param name="Origisumry">原始字符串</param>
6  /// <returns>不同的位数</returns>
7  int getCmpVal(string newMd5, string originMd5) {
8      int indexOfTemp = 0;
9      bitset<128> newmd5;
10     // 使用最笨的办法, switch case把16进制的字符表示转换成2进制的结果
11     for (; indexOfTemp < 32; indexOfTemp++) {
12         int startOfEdit = 127 - 4 * indexOfTemp;
13         switch (newMd5[indexOfTemp]) {
14             case '0':
```

```
15         newmd5[startOfEdit] = 0;
16         newmd5[startOfEdit - 1] = 0;
17         newmd5[startOfEdit - 2] = 0;
18         newmd5[startOfEdit - 3] = 0;
19         break;
20     case '1':
21         newmd5[startOfEdit] = 0;
22         newmd5[startOfEdit - 1] = 0;
23         newmd5[startOfEdit - 2] = 0;
24         newmd5[startOfEdit - 3] = 1;
25         break;
26     case '2':
27         newmd5[startOfEdit] = 0;
28         newmd5[startOfEdit - 1] = 0;
29         newmd5[startOfEdit - 2] = 1;
30         newmd5[startOfEdit - 3] = 0;
31         break;
32     case '3':
33         newmd5[startOfEdit] = 0;
34         newmd5[startOfEdit - 1] = 0;
35         newmd5[startOfEdit - 2] = 1;
36         newmd5[startOfEdit - 3] = 1;
37         break;
38     case '4':
39         newmd5[startOfEdit] = 0;
40         newmd5[startOfEdit - 1] = 1;
41         newmd5[startOfEdit - 2] = 0;
42         newmd5[startOfEdit - 3] = 0;
43         break;
44     case '5':
45         newmd5[startOfEdit] = 0;
46         newmd5[startOfEdit - 1] = 1;
47         newmd5[startOfEdit - 2] = 0;
48         newmd5[startOfEdit - 3] = 1;
49         break;
50     case '6':
51         newmd5[startOfEdit] = 0;
52         newmd5[startOfEdit - 1] = 1;
53         newmd5[startOfEdit - 2] = 1;
54         newmd5[startOfEdit - 3] = 0;
55         break;
56     case '7':
57
58         newmd5[startOfEdit] = 0;
59         newmd5[startOfEdit - 1] = 1;
60         newmd5[startOfEdit - 2] = 1;
61         newmd5[startOfEdit - 3] = 1;
62         break;
63     case '8':
64
65         newmd5[startOfEdit] = 1;
66         newmd5[startOfEdit - 1] = 0;
67         newmd5[startOfEdit - 2] = 0;
68         newmd5[startOfEdit - 3] = 0;
69         break;
70     case '9':
71
72         newmd5[startOfEdit] = 1;
```

```
73         newmd5[startOfEdit - 1] = 0;
74         newmd5[startOfEdit - 2] = 0;
75         newmd5[startOfEdit - 3] = 1;
76         break;
77     case 'A':
78
79         newmd5[startOfEdit] = 1;
80         newmd5[startOfEdit - 1] = 0;
81         newmd5[startOfEdit - 2] = 1;
82         newmd5[startOfEdit - 3] = 0;
83         break;
84     case 'B':
85
86         newmd5[startOfEdit] = 1;
87         newmd5[startOfEdit - 1] = 0;
88         newmd5[startOfEdit - 2] = 1;
89         newmd5[startOfEdit - 3] = 1;
90         break;
91     case 'C':
92
93         newmd5[startOfEdit] = 1;
94         newmd5[startOfEdit - 1] = 1;
95         newmd5[startOfEdit - 2] = 0;
96         newmd5[startOfEdit - 3] = 0;
97         break;
98     case 'D':
99
100         newmd5[startOfEdit] = 1;
101         newmd5[startOfEdit - 1] = 1;
102         newmd5[startOfEdit - 2] = 0;
103         newmd5[startOfEdit - 3] = 1;
104         break;
105     case 'E':
106
107         newmd5[startOfEdit] = 1;
108         newmd5[startOfEdit - 1] = 1;
109         newmd5[startOfEdit - 2] = 1;
110         newmd5[startOfEdit - 3] = 0;
111         break;
112     case 'F':
113
114         newmd5[startOfEdit] = 1;
115         newmd5[startOfEdit - 1] = 1;
116         newmd5[startOfEdit - 2] = 1;
117         newmd5[startOfEdit - 3] = 1;
118         break;
119     case 'a':
120
121         newmd5[startOfEdit] = 1;
122         newmd5[startOfEdit - 1] = 0;
123         newmd5[startOfEdit - 2] = 1;
124         newmd5[startOfEdit - 3] = 0;
125         break;
126     case 'b':
127
128         newmd5[startOfEdit] = 1;
129         newmd5[startOfEdit - 1] = 0;
130         newmd5[startOfEdit - 2] = 1;
```



```

131         newmd5[startOfEdit - 3] = 1;
132         break;
133     case 'c':
134
135         newmd5[startOfEdit] = 1;
136         newmd5[startOfEdit - 1] = 1;
137         newmd5[startOfEdit - 2] = 0;
138         newmd5[startOfEdit - 3] = 0;
139         break;
140     case 'd':
141
142         newmd5[startOfEdit] = 1;
143         newmd5[startOfEdit - 1] = 1;
144         newmd5[startOfEdit - 2] = 0;
145         newmd5[startOfEdit - 3] = 1;
146         break;
147     case 'e':
148
149         newmd5[startOfEdit] = 1;
150         newmd5[startOfEdit - 1] = 1;
151         newmd5[startOfEdit - 2] = 1;
152         newmd5[startOfEdit - 3] = 0;
153         break;
154     case 'f':
155
156         newmd5[startOfEdit] = 1;
157         newmd5[startOfEdit - 1] = 1;
158         newmd5[startOfEdit - 2] = 1;
159         newmd5[startOfEdit - 3] = 1;
160         break;
161     }
162 }
163
164 indexOfTemp = 0;
165 bitset<128> origin;
166 // 使用最笨的办法, switch case把16进制的字符表示转换成2进制的结果
167 for (; indexOfTemp < 32; indexOfTemp++) {
168     int startOfEdit = 127 - 4 * indexOfTemp;
169     switch (originMd5[indexOfTemp]) {
170     case '0':
171         origin[startOfEdit] = 0;
172         origin[startOfEdit - 1] = 0;
173         origin[startOfEdit - 2] = 0;
174         origin[startOfEdit - 3] = 0;
175         break;
176     case '1':
177         origin[startOfEdit] = 0;
178         origin[startOfEdit - 1] = 0;
179         origin[startOfEdit - 2] = 0;
180         origin[startOfEdit - 3] = 1;
181         break;
182     case '2':
183         origin[startOfEdit] = 0;
184         origin[startOfEdit - 1] = 0;
185         origin[startOfEdit - 2] = 1;
186         origin[startOfEdit - 3] = 0;
187         break;
188     case '3':

```

```
189         origin[startOfEdit] = 0;
190         origin[startOfEdit - 1] = 0;
191         origin[startOfEdit - 2] = 1;
192         origin[startOfEdit - 3] = 1;
193         break;
194     case '4':
195         origin[startOfEdit] = 0;
196         origin[startOfEdit - 1] = 1;
197         origin[startOfEdit - 2] = 0;
198         origin[startOfEdit - 3] = 0;
199         break;
200     case '5':
201         origin[startOfEdit] = 0;
202         origin[startOfEdit - 1] = 1;
203         origin[startOfEdit - 2] = 0;
204         origin[startOfEdit - 3] = 1;
205         break;
206     case '6':
207         origin[startOfEdit] = 0;
208         origin[startOfEdit - 1] = 1;
209         origin[startOfEdit - 2] = 1;
210         origin[startOfEdit - 3] = 0;
211         break;
212     case '7':
213
214         origin[startOfEdit] = 0;
215         origin[startOfEdit - 1] = 1;
216         origin[startOfEdit - 2] = 1;
217         origin[startOfEdit - 3] = 1;
218         break;
219     case '8':
220
221         origin[startOfEdit] = 1;
222         origin[startOfEdit - 1] = 0;
223         origin[startOfEdit - 2] = 0;
224         origin[startOfEdit - 3] = 0;
225         break;
226     case '9':
227
228         origin[startOfEdit] = 1;
229         origin[startOfEdit - 1] = 0;
230         origin[startOfEdit - 2] = 0;
231         origin[startOfEdit - 3] = 1;
232         break;
233     case 'A':
234
235         origin[startOfEdit] = 1;
236         origin[startOfEdit - 1] = 0;
237         origin[startOfEdit - 2] = 1;
238         origin[startOfEdit - 3] = 0;
239         break;
240     case 'B':
241
242         origin[startOfEdit] = 1;
243         origin[startOfEdit - 1] = 0;
244         origin[startOfEdit - 2] = 1;
245         origin[startOfEdit - 3] = 1;
246         break;
```

```
247     case 'C':
248
249         origin[startOfEdit] = 1;
250         origin[startOfEdit - 1] = 1;
251         origin[startOfEdit - 2] = 0;
252         origin[startOfEdit - 3] = 0;
253         break;
254     case 'D':
255
256         origin[startOfEdit] = 1;
257         origin[startOfEdit - 1] = 1;
258         origin[startOfEdit - 2] = 0;
259         origin[startOfEdit - 3] = 1;
260         break;
261     case 'E':
262
263         origin[startOfEdit] = 1;
264         origin[startOfEdit - 1] = 1;
265         origin[startOfEdit - 2] = 1;
266         origin[startOfEdit - 3] = 0;
267         break;
268     case 'F':
269
270         origin[startOfEdit] = 1;
271         origin[startOfEdit - 1] = 1;
272         origin[startOfEdit - 2] = 1;
273         origin[startOfEdit - 3] = 1;
274         break;
275     case 'a':
276
277         origin[startOfEdit] = 1;
278         origin[startOfEdit - 1] = 0;
279         origin[startOfEdit - 2] = 1;
280         origin[startOfEdit - 3] = 0;
281         break;
282     case 'b':
283
284         origin[startOfEdit] = 1;
285         origin[startOfEdit - 1] = 0;
286         origin[startOfEdit - 2] = 1;
287         origin[startOfEdit - 3] = 1;
288         break;
289     case 'c':
290
291         origin[startOfEdit] = 1;
292         origin[startOfEdit - 1] = 1;
293         origin[startOfEdit - 2] = 0;
294         origin[startOfEdit - 3] = 0;
295         break;
296     case 'd':
297
298         origin[startOfEdit] = 1;
299         origin[startOfEdit - 1] = 1;
300         origin[startOfEdit - 2] = 0;
301         origin[startOfEdit - 3] = 1;
302         break;
303     case 'e':
304
```

```

305         origin[startOfEdit] = 1;
306         origin[startOfEdit - 1] = 1;
307         origin[startOfEdit - 2] = 1;
308         origin[startOfEdit - 3] = 0;
309         break;
310     case 'f':
311
312         origin[startOfEdit] = 1;
313         origin[startOfEdit - 1] = 1;
314         origin[startOfEdit - 2] = 1;
315         origin[startOfEdit - 3] = 1;
316         break;
317     }
318 }
319
320 bitset<128> result;
321 result = newmd5 ^ origin;
322
323 return result.count();
324 }

```

## 测试雪崩效应的函数

这里我们在改变的时候是直接对原始字符串中的char进行数值上的+1，则通常情况下只会改变1位或者2位，共进行8轮测试，相关代码如下：

```

1     for (int i = 0; i < 8; i++) {
2         cout << "\n\n=====第" << i + 1 << "轮雪崩效应测试
===== \n";
3         cout << "字符串的第" << i + 1 << "个char的值+1，得到新的字符串为: \n";
4         newM = m;
5         newM[i] += 1;
6         cout << newM << endl;
7         // 计算新的md5
8         string newMd5M = myGetMD5(newM);
9         cout << "经过计算可以得到修改过后的字符串MD5值为: \n" << newMd5M << endl;
10
11         changeBit[i] = getCmpval(newMd5M, md5M);
12         sumBit += changeBit[i];
13         cout << "经过比较后发生变化的位数为: ";
14         cout << changeBit[i] << endl;
15     }
16
17     averageBit = sumBit / 8;
18     cout << "\n\n经过计算得到平均改变的位数为: " << averageBit << endl;

```

## 实验结果

得到结果如下：

```

1     =====第1轮雪崩效应测试=====
2     字符串的第1个char的值+1，得到新的字符串为：

```

```
3 2a65dsg489121asdg
4 经过计算可以得到修改过后的字符串MD5值为:
5 b08fa0001f0e349c338d73f18937033d
6 经过比较后发生变化的位数为: 68
7
8
9 =====第2轮雪崩效应测试=====
10 字符串的第2个char的值+1, 得到新的字符串为:
11 1b65dsg489121asdg
12 经过计算可以得到修改过后的字符串MD5值为:
13 31996c4c30defa03f52cc59ef7378246
14 经过比较后发生变化的位数为: 63
15
16
17 =====第3轮雪崩效应测试=====
18 字符串的第3个char的值+1, 得到新的字符串为:
19 1a75dsg489121asdg
20 经过计算可以得到修改过后的字符串MD5值为:
21 69745a92214692fb803e244676ff4d82
22 经过比较后发生变化的位数为: 60
23
24
25 =====第4轮雪崩效应测试=====
26 字符串的第4个char的值+1, 得到新的字符串为:
27 1a66dsg489121asdg
28 经过计算可以得到修改过后的字符串MD5值为:
29 c3d123fdc560f2d65a90003a27233eae
30 经过比较后发生变化的位数为: 65
31
32
33 =====第5轮雪崩效应测试=====
34 字符串的第5个char的值+1, 得到新的字符串为:
35 1a65esg489121asdg
36 经过计算可以得到修改过后的字符串MD5值为:
37 751af42e8f96fa2035b597d5f3bd9e5a
38 经过比较后发生变化的位数为: 53
39
40
41 =====第6轮雪崩效应测试=====
42 字符串的第6个char的值+1, 得到新的字符串为:
43 1a65dtg489121asdg
44 经过计算可以得到修改过后的字符串MD5值为:
45 be8e5364b5f8822453dc40aef331b59d
46 经过比较后发生变化的位数为: 63
47
48
49 =====第7轮雪崩效应测试=====
50 字符串的第7个char的值+1, 得到新的字符串为:
51 1a65dsh489121asdg
52 经过计算可以得到修改过后的字符串MD5值为:
53 1d8c28dfa85e3bdea94ca9bbe60aadca
54 经过比较后发生变化的位数为: 58
55
56
57 =====第8轮雪崩效应测试=====
58 字符串的第8个char的值+1, 得到新的字符串为:
59 1a65dsg589121asdg
60 经过计算可以得到修改过后的字符串MD5值为:
```

61 3bc4aa0ee22700d1c182ae5f8c2a91bc

62 经过比较后发生变化的位数为: 65

63

64

65 经过计算得到平均改变的位数为: 61