

# MD5 算法的程序设计和实现报告

17343128 幸贊

## 1. 算法原理概述

MD5 是一个不可逆的字符串变换算法，他将任意长度的字节串变换为一个 128bit 的大整数在一些初始化处理后，MD5 以 512 位分组来处理输入文本，每一分组又划分为 16 个 32 位子分组。算法的输出由四个 32 位分组组成，将它们级联形成一个 128 位散列值。大概的 MD5 框架就这样，其他具体步骤会在下面的模块分析一一解释。

## 2. 总体结构与模块分解

程序的总体结构可以分为四个模块，一是函数以及变量的声明，二是 FGHI 等等的宏定义，三是 MD5 各个步骤函数的实现，最后是主函数的测试。

### (1) 函数及变量的声明

```
} typedef struct  
{  
    unsigned int count[2];  
    unsigned int state[4];  
    unsigned char buffer[64];  
}MD5_CTX;  
  
void MD5Init(MD5_CTX *context);  
void MD5Transform(unsigned int state[4], unsigned char block[64]);  
void MD5Encode(unsigned char *output, unsigned int *input, unsigned int len);  
void MD5Decode(unsigned int *output, unsigned char *input, unsigned int len);  
void MD5Update(MD5_CTX *context, unsigned char *input, unsigned int inputlen);  
void MD5Final(MD5_CTX *context, unsigned char digest[16]);  
  
} unsigned char PADDING[] = { 0x80,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 };
```

其中声明 MD5\_CTX 的结构体来存储信息，其中 buffer 为输入信息的缓冲区，512 位，state 为 4 个 32 位的数，count 存放原始信息的长度。然后是 6 个函数，下面会一一讲解。最后是 PADDING 缓冲区 64 个字节。

## (2) 各种宏定义

```
#define F(x,y,z) ((x & y) | (~x & z))
#define G(x,y,z) ((x & z) | (y & ~z))
#define H(x,y,z) (x^y^z)
#define I(x,y,z) (y ^ (x | ~z))
#define ROTATE_LEFT(x,n) ((x << n) | (x >> (32-n)))
#define FF(a,b,c,d,x,s,ac) \
{ \
    a += F(b,c,d) + x + ac; \
    a = ROTATE_LEFT(a,s); \
    a += b; \
}
#define GG(a,b,c,d,x,s,ac) \
{ \
    a += G(b,c,d) + x + ac; \
    a = ROTATE_LEFT(a,s); \
    a += b; \
}
#define HH(a,b,c,d,x,s,ac) \
{ \
    a += H(b,c,d) + x + ac; \
    a = ROTATE_LEFT(a,s); \
    a += b; \
}
#define II(a,b,c,d,x,s,ac) \
{ \
    a += I(b,c,d) + x + ac; \
    a = ROTATE_LEFT(a,s); \
    a += b; \
}
```

宏定义都是 MD5 算法规定的，这个没什么好说的。

## (3) 各个函数实现，即 MD5 算法实现步骤

```
void MD5Init(MD5_CTX *context)
{
    context->count[0] = 0;
    context->count[1] = 0;
    context->state[0] = 0x67452301;
    context->state[1] = 0xEFCDAB89;
    context->state[2] = 0x98BADCFE;
    context->state[3] = 0x10325476;
}
```

先是 init，即初始化 MD5 结构，初始位数为 0，然后下面的 4 个 32 位变量初始化是算法规定的。

```
void MD5Update(MD5_CTX *context, unsigned char *input, unsigned int inputlen)
{
    unsigned int i = 0, index = 0, partlen = 0;
    index = (context->count[0] >> 3) & 0x3F;
    partlen = 64 - index;
    context->count[0] += inputlen << 3;
    if (context->count[0] < (inputlen << 3))
        context->count[1]++;
    context->count[1] += inputlen >> 29;

    if (inputlen >= partlen)
    {
        memcpy(&context->buffer[index], input, partlen);
        MD5Transform(context->state, context->buffer);
        for (i = partlen; i + 64 <= inputlen; i += 64)
            MD5Transform(context->state, &input[i]);
        index = 0;
    }
    else
    {
        i = 0;
    }
    memcpy(&context->buffer[index], &input[i], inputlen - i);
}
```

然后是 update 函数，主要计算已有的信息模 64 的余数，然后差多少可以为 512bits 的倍数，并且将输入缓冲区中的不足填充满 512bits 的剩余内容填充到 context->buffer 中，以后再作处理

。

```

void MD5Encode(unsigned char *output, unsigned int *input, unsigned int len)
{
    unsigned int i = 0, j = 0;
    while (j < len)
    {
        output[j] = input[i] & 0xFF;
        output[j + 1] = (input[i] >> 8) & 0xFF;
        output[j + 2] = (input[i] >> 16) & 0xFF;
        output[j + 3] = (input[i] >> 24) & 0xFF;
        i++;
        j += 4;
    }
}

void MD5Decode(unsigned int *output, unsigned char *input, unsigned int len)
{
    unsigned int i = 0, j = 0;
    while (j < len)
    {
        output[i] = (input[j]) |
            (input[j + 1] << 8) |
            (input[j + 2] << 16) |
            (input[j + 3] << 24);
        i++;
        j += 4;
    }
}

```

然后是 encode 和 decode 两个函数，一个编码一个解码，encode 将 4 字节的整数 copy 到字符形式的缓冲区中，decode 这一个把字符形式的缓冲区中的数据 copy 到 4 字节的整数中。

```

void MD5Transform(unsigned int state[4], unsigned char block[64])
{
    unsigned int a = state[0];
    unsigned int b = state[1];
    unsigned int c = state[2];
    unsigned int d = state[3];
    unsigned int x[64];
    MD5Decode(x, block, 64);
    FF(a, b, c, d, x[0], 7, 0xd76aa478); /* 1 */
    FF(d, a, b, c, x[1], 12, 0xe8c7b756); /* 2 */
    FF(c, d, a, b, x[2], 17, 0x242070db); /* 3 */
    FF(b, c, d, a, x[3], 22, 0xc1bdcee5); /* 4 */
    FF(a, b, c, d, x[4], 7, 0xf57c0faf); /* 5 */
    FF(d, a, b, c, x[5], 12, 0x4787c62a); /* 6 */
    FF(c, d, a, b, x[6], 17, 0xa8304613); /* 7 */
    FF(b, c, d, a, x[7], 22, 0xfd469501); /* 8 */
    FF(a, b, c, d, x[8], 7, 0x698098d8); /* 9 */
    FF(d, a, b, c, x[9], 12, 0x8b44f7af); /* 10 */
    FF(c, d, a, b, x[10], 17, 0xffff5bb1); /* 11 */
    FF(b, c, d, a, x[11], 22, 0x895cd7be); /* 12 */
    FF(a, b, c, d, x[12], 7, 0x6b901122); /* 13 */
    FF(d, a, b, c, x[13], 12, 0xfd987193); /* 14 */
    FF(c, d, a, b, x[14], 17, 0xa679438e); /* 15 */
    FF(b, c, d, a, x[15], 22, 0x49b40821); /* 16 */

    /* Round 2 */
    GG(a, b, c, d, x[1], 5, 0xf61e2562); /* 17 */
    GG(d, a, b, c, x[6], 9, 0xc040b340); /* 18 */
    GG(c, d, a, b, x[11], 14, 0x265e5a51); /* 19 */
    GG(b, c, d, a, x[0], 20, 0xe9b6c7aa); /* 20 */
    GG(a, b, c, d, x[5], 5, 0xd62f105d); /* 21 */
    GG(d, a, b, c, x[10], 9, 0x2441453); /* 22 */
    GG(c, d, a, b, x[15], 14, 0xd8a1e681); /* 23 */
    GG(b, c, d, a, x[4], 20, 0xe7d3fbc8); /* 24 */
    GG(a, b, c, d, x[9], 5, 0x21e1cde6); /* 25 */
}

```

```

GG(d, a, b, c, x[2], 9, 0xfcefa3f8); /* 30 */
GG(c, d, a, b, x[7], 14, 0x676f02d9); /* 31 */
GG(b, c, d, a, x[12], 20, 0x8d2a4c8a); /* 32 */

/* Round 3 */
HH(a, b, c, d, x[5], 4, 0xffffa3942); /* 33 */
HH(d, a, b, c, x[8], 11, 0x8771f681); /* 34 */
HH(c, d, a, b, x[11], 16, 0x6d9d6122); /* 35 */
HH(b, c, d, a, x[14], 23, 0xfde5380c); /* 36 */
HH(a, b, c, d, x[1], 4, 0xa4beea44); /* 37 */
HH(d, a, b, c, x[4], 11, 0x4bdecfa9); /* 38 */
HH(c, d, a, b, x[7], 16, 0xf6bb4b60); /* 39 */
HH(b, c, d, a, x[10], 23, 0xbebfb70); /* 40 */
HH(a, b, c, d, x[13], 4, 0x289b7ec6); /* 41 */
HH(d, a, b, c, x[0], 11, 0xeaa127fa); /* 42 */
HH(c, d, a, b, x[3], 16, 0xd4ef3085); /* 43 */
HH(b, c, d, a, x[6], 23, 0x4881d05); /* 44 */
HH(a, b, c, d, x[9], 4, 0xd9d4d039); /* 45 */
HH(d, a, b, c, x[12], 11, 0xe6db99e5); /* 46 */
HH(c, d, a, b, x[15], 16, 0x1fa27cf8); /* 47 */
HH(b, c, d, a, x[2], 23, 0xc4ac5665); /* 48 */

/* Round 4 */
II(a, b, c, d, x[0], 6, 0xf4292244); /* 49 */
II(d, a, b, c, x[7], 10, 0x432aff97); /* 50 */
II(c, d, a, b, x[14], 15, 0xab9423a7); /* 51 */
II(b, c, d, a, x[5], 21, 0xfc93a039); /* 52 */
II(a, b, c, d, x[12], 6, 0x655b59c3); /* 53 */
II(d, a, b, c, x[3], 10, 0x8f0ccc92); /* 54 */
II(c, d, a, b, x[10], 15, 0xffeff47d); /* 55 */
II(b, c, d, a, x[1], 21, 0x85845dd1); /* 56 */
II(a, b, c, d, x[8], 6, 0x6fa87e4f); /* 57 */
II(d, a, b, c, x[15], 10, 0xfe2ce6e0); /* 58 */
II(c, d, a, b, x[6], 15, 0xa3014314); /* 59 */
II(b, c, d, a, x[13], 21, 0x4e0811a1); /* 60 */

II(d, c, d, a, x[1], 21, 0x05045001); /* 50 */
II(a, b, c, d, x[8], 6, 0x6fa87e4f); /* 57 */
II(d, a, b, c, x[15], 10, 0xfe2ce6e0); /* 58 */
II(c, d, a, b, x[6], 15, 0xa3014314); /* 59 */
II(b, c, d, a, x[13], 21, 0x4e0811a1); /* 60 */
II(a, b, c, d, x[4], 6, 0xf7537e82); /* 61 */
II(d, a, b, c, x[11], 10, 0xbd3af235); /* 62 */
II(c, d, a, b, x[2], 15, 0x2ad7d2bb); /* 63 */
II(b, c, d, a, x[9], 21, 0xeb86d391); /* 64 */
state[0] += a;
state[1] += b;
state[2] += c;
state[3] += d;

```

然后就是 MD5 算法标配的 4 轮 16 次共 64 次计算，计算后将最后的信息存在 state 中，即 4 个 32 位信息。



```

void MD5Final(MD5_CTX *context, unsigned char digest[16])
{
    unsigned int index = 0, padlen = 0;
    unsigned char bits[8];
    index = (context->count[0] >> 3) & 0x3F;
    padlen = (index < 56) ? (56 - index) : (120 - index);
    MD5Encode(bits, context->count, 8);
    MD5Update(context, PADDING, padlen);
    MD5Update(context, bits, 8);
    MD5Encode(digest, context->state, 16);
}

```

最后是 final 函数，将需要被转换的信息拷贝，然后模 64，再补上原始信息的长度刚好 512bits，最后将结果存在 digest 数组中就 ok 了。

以上便是 MD5 算法的概述以及主要实现。

#### (4) 主函数

```

int main(){
    int i;
    //unsigned char encrypt[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
    // unsigned char encrypt[] = "aaaaaabbbbb";
    //unsigned char encrypt[] = "ssssscdwdwd";
    // unsigned char encrypt[] = "qwertyuiop";
    // unsigned char encrypt[] = "zxcvbnm123";
    // unsigned char encrypt[] = "wertyu11";
    unsigned char encrypt[] = "hahahahaha";
    unsigned char decrypt[16];
    MD5_CTX md5;
    MD5Init(&md5);
    MD5Update(&md5, encrypt, strlen((char *)encrypt));
    MD5Final(&md5, decrypt);
    printf("加密前:%s\n加密后:", encrypt);
    for (i = 0; i<16; i++){
        printf("%02x", decrypt[i]);
    }
    return 0;
}

```

主函数主要就是一个测试功能。

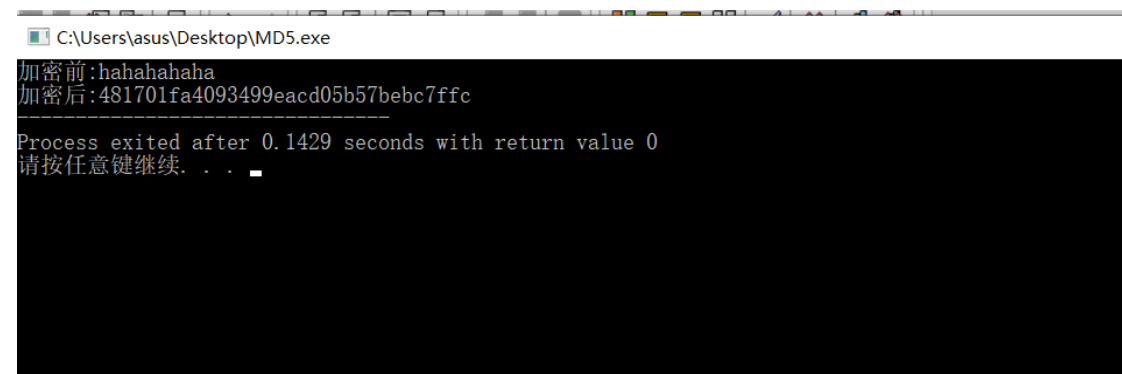
### 3. 数据结构

基本没用什么数据结构，主要是 char 和 unsigned 类型变量，然后就是自己定义的结构体 MD5\_CTX 来存信息。

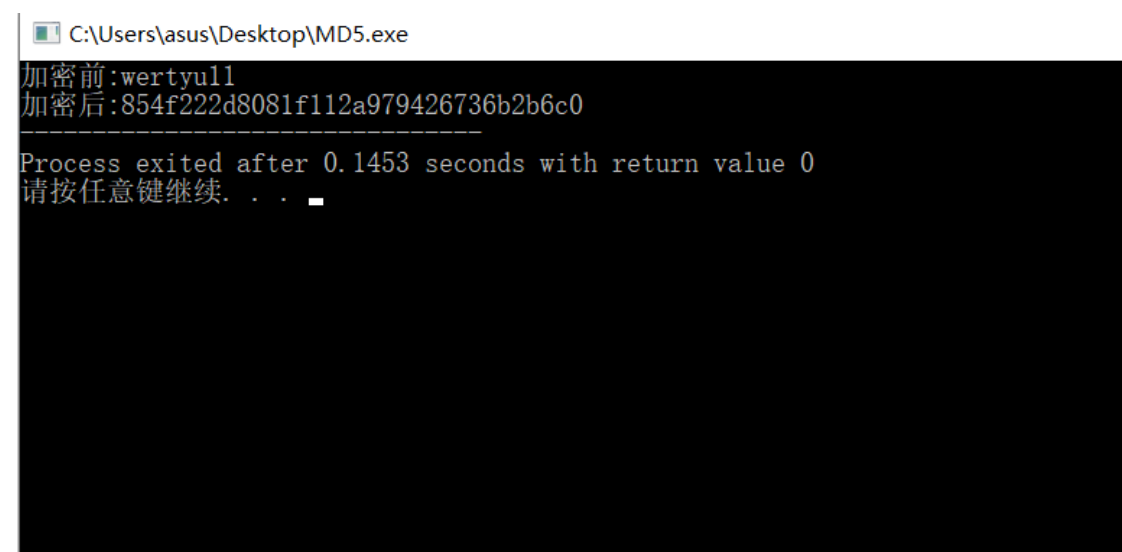
#### 4. C 语言源代码

见压缩包

#### 5. 编译运行结果



```
C:\Users\asus\Desktop\MD5.exe
加密前:hahahahaha
加密后:481701fa4093499eacd05b57bebc7ffc
-----
Process exited after 0.1429 seconds with return value 0
请按任意键继续. . .
```



```
C:\Users\asus\Desktop\MD5.exe
加密前:wertyull
加密后:854f222d8081f112a979426736b2b6c0
-----
Process exited after 0.1453 seconds with return value 0
请按任意键继续. . .
```



C:\Users\asus\Desktop\MD5.exe

加密前:zxcvbnml23

加密后:0062d21c1a97acad0fff653df56849b2

-----  
Process exited after 0.1483 seconds with return value 0

请按任意键继续. . .

C:\Users\asus\Desktop\MD5.exe

加密前:qwertyuiop

加密后:6eea9b7ef19179a06954edd0f6c05ceb

-----  
Process exited after 0.1672 seconds with return value 0

请按任意键继续. . .

```
C:\Users\asus\Desktop\MD5.exe
加密前:ssssscdwdwd
加密后:907f1a1022a5f3af2e037644a56d4834
-----
Process exited after 0.1523 seconds with return value 0
请按任意键继续. . .
```

```
C:\Users\asus\Desktop\MD5.exe
加密前:aaaaaabbbbb
加密后:c892cee7f5ff96b071f7d00ada251df0
-----
Process exited after 0.159 seconds with return value 0
请按任意键继续. . .
```

```
C:\Users\asus\Desktop\MD5.exe
加密前:ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
加密后:f29939a25efabaef3b87e2cbfe641315
-----
Process exited after 0.1419 seconds with return value 0
请按任意键继续. . .
```

以上便是本次实验的全部内容，通过本次实验，对加密算法有了全新的认识，之前的 DES 是既可以加密又可以解密，但是这次的 MD5 因为使用了哈希散列，所以无法解密，因为一个密码可以对应无数个源码，根本找不到源码，也让我重新认识了加密算法，收获很大，希望在以后的实验中能更深入的了解密码学！