MD5加密算法实验报告

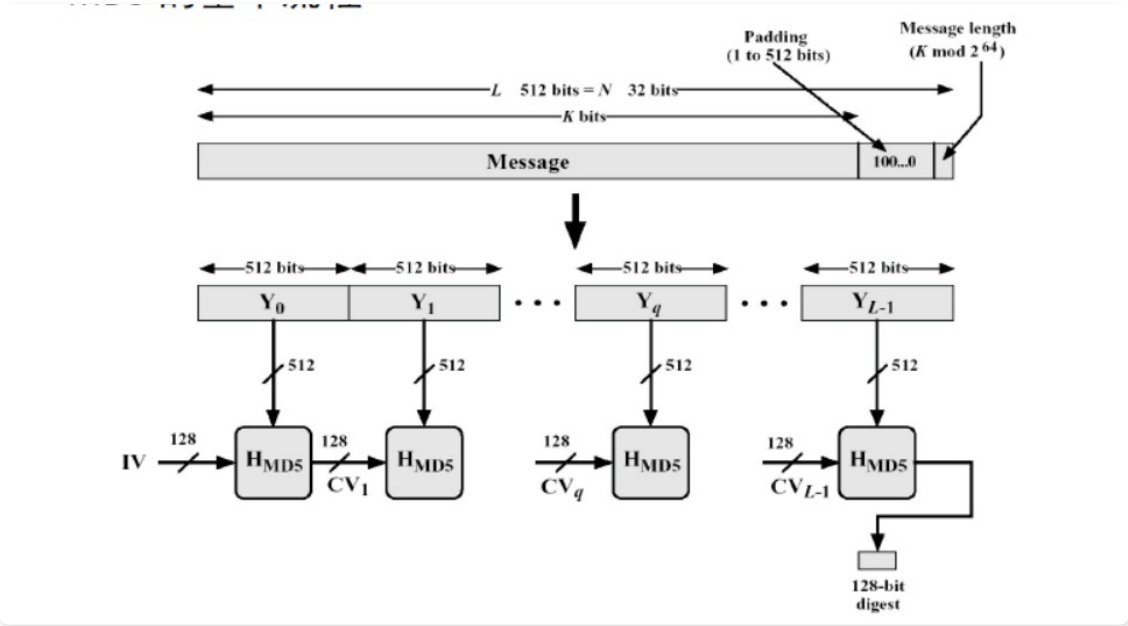
学号：15331061
姓名：邓旺

Md5简介

消息摘要算法第五版（英语：Message-Digest Algorithm 5，缩写为MD5），是一种是广泛使用的散列算法，可以将任意长度的“字节串”变换成一个128bit长度值的密文以达到加密的目的。MD5以其算法的不可逆性以及稳定、快速的特点而被广泛应用于普通数据的错误检查领域。
MD5的典型应用是对一段字节串产生密文，以防止被“篡改”，具体的应用体现包括数字签名。MD5还广泛用于加密和解密技术上，在很多操作系统中，用户的密码是以MD5值（或类似的其它算法）的方式保存的， 用户Login的时候，系统是把用户输入的密码计算成MD5值，然后再去和系统中保存的MD5值进行比较，而系统并不“知道”用户的密码是什么。

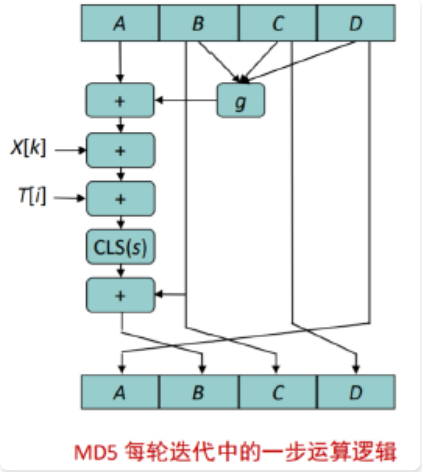
算法原理

算法流程



过程概述

1. 填充
   信息转化为二进制后，计算前先要进行位补， 即将数据扩展至 K * 512 + 448(bit) 的形式，K为整数。 具体补位操作：附一个1在消息后面，后接所要求的多个0至满足以上形式为止。需要注意的是总共最少要补1bit，最多补512bit。
2. 尾部加上部分原信息
   上述填充好的消息尾部附加原始消息的位数的低64位，最后得到一个长度L 是512位整数倍的消息
3. 初始化缓冲区
   初始化一个128位的MD 缓冲区，记为CVq，也表示为4个32位寄存器(A, B, C, D)。
   在这儿需要注意的一点是寄存器采用小端存储(little-endian) 的存储结构，Little-Endian 将低位字节排放在内存的低地址端，因此低字节要写在前面；还要注意的一个点是整数与字节的转换。其中
   A= 0x67452301
   B= 0xEFCDAB89
   C= 0x98BADCFE
   D= 0x10325476
4. 轮转变换



MD5 每轮迭代中的一步运算逻辑

- MD5 从CV 输入128位，从消息分组输入512位，完成4轮循环后，输出128位，用于下一轮输入的CV 值。
- 每轮循环分别固定不同的生成函数F, G, H, I，结合指定的T 表元素T[] 和消息分组的不同部分X[] 做16 次运算，生成下一轮循环的输入。
- 总共有64次迭代运算
- 4轮循环中使用的生成函数(轮函数) g 是一个32位非线性逻辑函数，在相应各轮的定义如下：

轮次	Function g	g(b, c, d)
1	$F(b,c,d)$	$(b\wedge c)\vee(\neg b\wedge d)$
2	$G(b,c,d)$	$(b\wedge d)\vee(c\wedge\neg d)$
3	$H(b,c,d)$	$b\oplus c\oplus d$
4	$I(b,c,d)$	$c\oplus(b\vee\neg d)$

- 每轮循环中的一步运算逻辑
  a = b + ((a + g(b,c,d) + X[k] + T[i]) <<<s)

> 说明：
> a, b, c, d : MD 缓冲区(A, B, C, D) 的当前值。
> g : 轮函数(F, G, H, I 中的一个)。
> <<<s : 将32位输入循环左移(CLS) s 位。
> X[k] : 当前处理消息分组的第k 个32位字，即M(q*16+k) 。
> T[i] : T 表的第i 个元素，32位字。
> + : 模232 加法。

定义：

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

- 各轮迭代中X[k] 之间的关系：
  - 第1轮迭代：X[j] , j = 1..16.
    顺序使用X[0, 1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12,13,14,15]
  - 第2轮迭代：X[(1 + 5j) mod 16], j = 1..16.

顺序使用X[1, 6,11, 0, 5,10,15, 4, 9,14, 3, 8,13, 2, 7,12]
◌ 第3轮迭代：X[(5 + 3j) mod 16]， j = 1..16.
顺序使用X[5, 8,11,14, 1, 4, 7,10,13, 0, 3, 6, 9,12,15, 2]
◌ 第4轮迭代：X[7j mod 16], j = 1..16.
顺序使用X[0, 7,14, 5,12, 3,10, 1, 8,15, 6,13, 4,11, 2, 9]

- 各次迭代运算采用的T 值：
  T[1.. 4] = { 0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee }
  T[5.. 8] = { 0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501 }
  T[9..12] = { 0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be }
  T[13..16] = { 0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821 }
  T[17..20] = { 0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa }
  T[21..24] = { 0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8 }
  T[25..28] = { 0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed }
  T[29..32] = { 0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a }
  T[33..36] = { 0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c }
  T[37..40] = { 0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70 }
  T[41..44] = { 0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05 }
  T[45..48] = { 0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665 }
  T[49..52] = { 0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039 }
  T[53..56] = { 0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1 }
  T[57..60] = { 0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1 }
  T[61..64] = { 0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391 }

- 各次迭代运算采用的左循环移位的s 值：
  s[1..16] = { 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22 }
  s[17..32] = { 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20 }
  s[33..48] = { 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23 }
  s[49..64] = { 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21 }

代码简介

代码主要是按照算法流程来模块化设计的，MD5算法可以很方便的分为几个流程，流程之间的依赖性不是很强，因此很方便一个模块一个模块分别实现。因为算法中涉及到字符串的处理，c处理字符串相比于c语言更加的方便。所以使用了c语言。但是因为这个实验是很典型面向过程式编程，就没有设计额外的类了。

- 缓冲区数据结构
  缓冲区就四个32位寄存器，而unsigned int数据类型长度为4个字节，32位，刚好表示

```
struct CV
{
    unsigned int A;
    unsigned int B;
    unsigned int C;
    unsigned int D;
};
```

- 轮函数
  此处参考[MD5算法C/C++的实现]

```
#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;                   \
    \
}
```

- 将输入数据转换为二进制，补位并填充尾部信息

```
void str_to_bint(string cleartext, int *md5)
{
    int len = cleartext.length();
    for (int i = 0; i < len; i++)
    {
        char temp = cleartext[i];
        for (int j = 0; j < 8; j++)
        {
            md5[8 * i + 7 - j] = temp % 2;
            temp /= 2;
        }
    }
}

//填充函数
int func_padding(int *md5, int len, string cleartext)
{
    int len_o = len % 512;
    md5[len] = 1;
    if (len_o < 448)
        len += 512 - len_o;
    else
        len += 512 + 512 - len_o;
    unsigned int length = cleartext.length() * 8;
    int i = 33;
    int cnt = 0;
    while (length)
    {
        if (length % 2)
            md5[len - i - (3 - cnt) * 8] = 1;
        length /= 2;
        i++;
        if (i % 8 == 0)
            cnt++;
        if (cnt > 3)
            cnt = 0;
        if (i > 64)
            i -= 64;
        else if (i == 32)
            break;
    }
    return len;
}
```

- 将数据转化为小端储存

```
unsigned int ToLittleEndian(int *bin)
{
    unsigned int res = 0;
    for (int i = 7; i >= 0; i--)
        if (bin[i])
            res += pow(2, 7 - i);
    for (int i = 15; i >= 8; i--)
        if (bin[i])
            res += pow(2, 15 - i + 8);
    for (int i = 23; i >= 16; i--)
        if (bin[i])
            res += pow(2, 23 - i + 16);
    for (int i = 31; i >= 24; i--)
        if (bin[i])
            res += pow(2, 31 - i + 24);
    return res;
}
```

- 轮转换过程

```
void Md5Round(CV *cv_block, int *bin)
{
    unsigned int x[16];
    for (int i = 0; i < 16; i++)
    {
        x[i] = Md5Init(bin + i * 32);
    }
    unsigned int a = cv_block->A;
    unsigned int b = cv_block->B;
    unsigned int c = cv_block->C;
    unsigned int d = cv_block->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, 0x2441453);
    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, 0x4881d05);
    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);
    cv_block->A += a;
    cv_block->B += b;
    cv_block->C += c;
    cv_block->D += d;
}
```

实验测试