

MD5 算法的程序设计和实现

MD5 算法的程序设计和实现

MD5算法

Introduction

Overall structure

Characteristics Overview

Module Decomposition

Append Padding Bits

Append Length

Initialize MD buffer

Processing message in 16-word block

Auxiliary Functions

The Table

The Algorithm

Output

Source code

利用string和bitset实现

利用string和cstdint实现

运行结果

[Github项目地址](#)

MD5算法

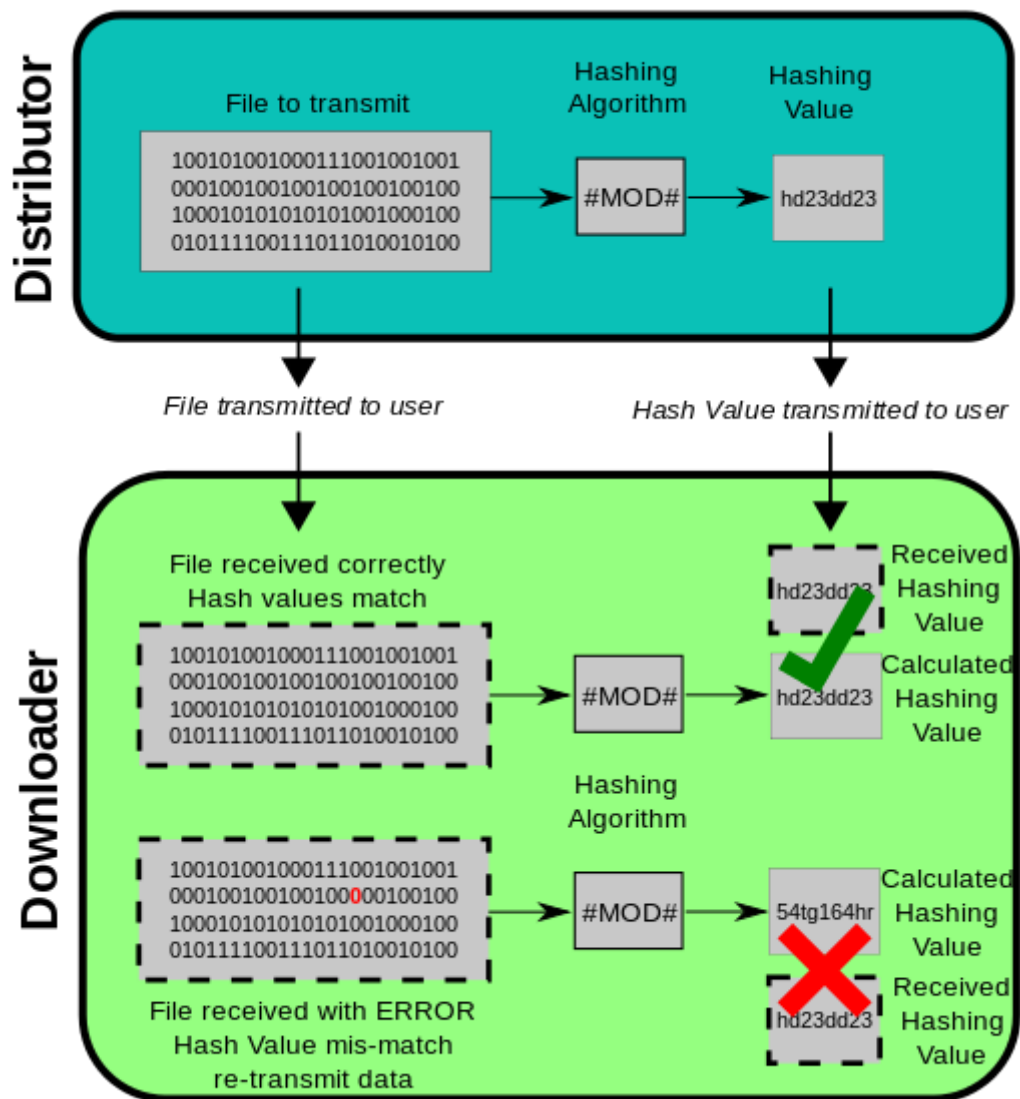
Introduction

MD5消息摘要算法是一个广泛使用的散列函数，它产生一个128位的散列值。尽管MD5最初被设计为用作加密哈希函数，但它被发现存在大量漏洞，不过仍然可以用作校验和，以验证数据的完整性，但只能防止意外损坏。它仍然适用于其他非加密目的，例如，确定分区数据库中特定密钥的分区。

MD5哈希算法是一种单向加密功能，该算法的思想是采用随机数据(文本或二进制)作为输入，并生成固定大小的“散列值”作为输出。它接受任意长度的消息作为输入，但是输出返回的是固定长度的摘要值，以用于认证原始消息。

任何加密哈希函数的一个基本要求是，在计算上不可能找到两个哈希值不同的不同消息。MD5无法满足这一要求;在一台普通的家用电脑上，这种碰撞几秒钟就能发现。

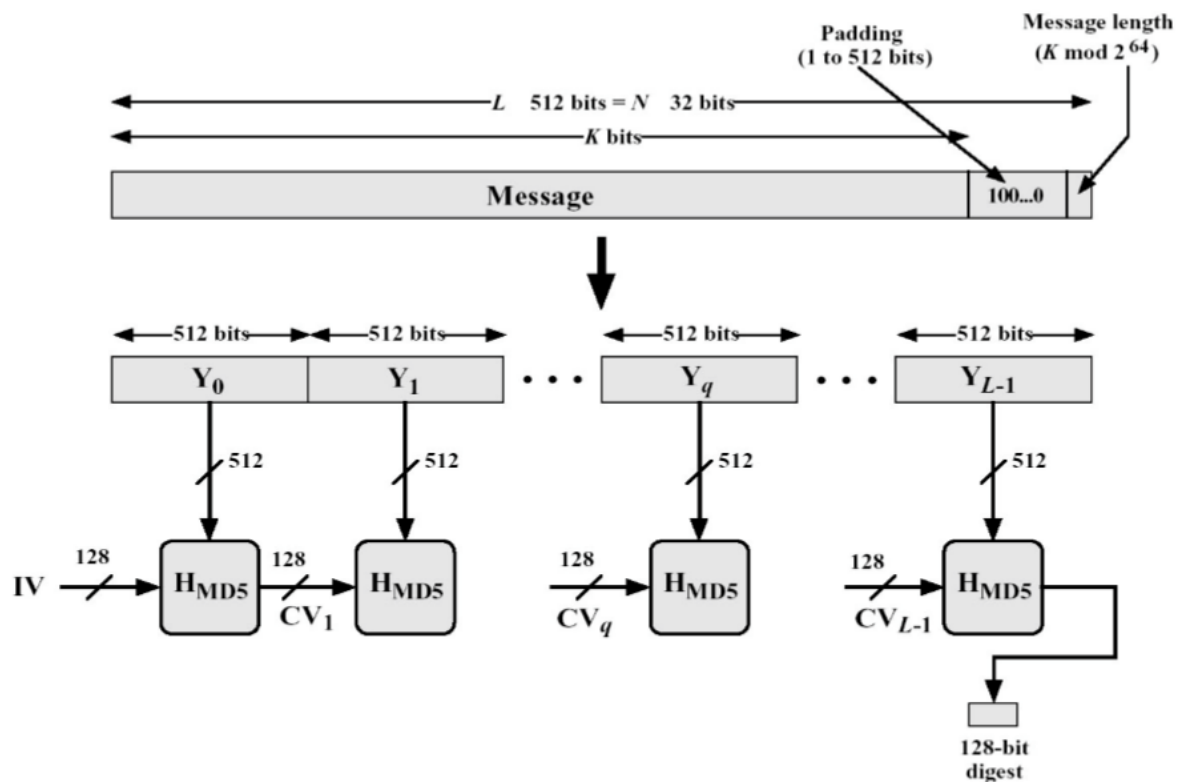
MD5算法适用于数字签名应用，其中大文件必须先以安全的方式“压缩”，然后才能在诸如RSA之类的公钥密码系统下使用私钥（秘密）进行加密。



Overall structure

输入：任意长度的消息

输出：固定 128-bits 的摘要信息



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

Characteristics Overview

Advantage and disadvantage

MD5算法很有用，因为与存储可变长度的大文本相比，比较和存储这些较小的哈希值更容易。MD5算法是一种广泛使用的用于单向哈希的算法，用于验证而不必给出原始值。Unix系统使用MD5算法以128位加密格式存储用户密码。MD5算法广泛用于检查文件的完整性。而且，使用该算法很容易生成原始消息的消息摘要，可以执行任意位数的消息的消息摘要。但是多年来，MD5容易出现哈希冲突弱点，即可以为两个不同的输入创建相同的哈希函数。MD5对这些冲突攻击没有提供任何安全性。SHA (Secure Hash Algorithm, 可产生160位消息摘要，由NSA设计为数字签名算法的一部分) 现已取代MD5。现在可以接受在密码字段生成哈希函数,因为它是不容易产生SHA-1碰撞,直到现在还没有产生碰撞。

Module Decomposition

Append Padding Bits

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 $100\dots0$, $1 \leq P \leq 512$ (即至少要填充1个bit), 使得填充后的消息位数为: $K + P \equiv 448 \pmod{512}$. 注意到当 $K \equiv 448 \pmod{512}$ 时, 需要 $P=512$.

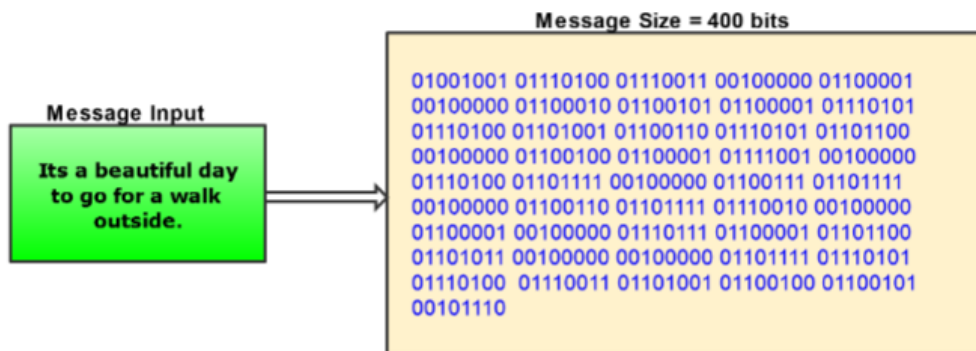


Figure 2: 400 bits original message

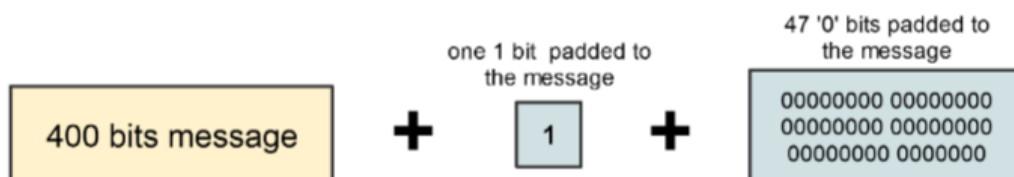


Figure 3: First one '1' bit is added and then '0' bits

如上图，如果我们的消息是400位，那么我们将添加一个“1”位和47个“0”位，这样就得到448位，比512少64位。如果我们的消息的大小是1200位，那么我们将添加一个“1”位和271“0”位，得到1472位。1472+64能被512整除。将至少1位和最多512位填充(或扩展)到原始消息

Append Length

在填充之后，我们获取原始消息K并在上述填充好的消息末尾插入64位，用来记录原始输入的长度。现在消息的长度正好可以被512整除，即 $K + P + 64 \equiv 0 \pmod{512}$

把填充后的消息结果分割为L个512-bit 分组： $Y_0, Y_1, Y_2, Y_3, \dots, Y_{L-1}$

再将分组结果表示成N个32-bit 字 M_0, M_0, \dots, M_{N-1} ($N = L \cdot 16$)

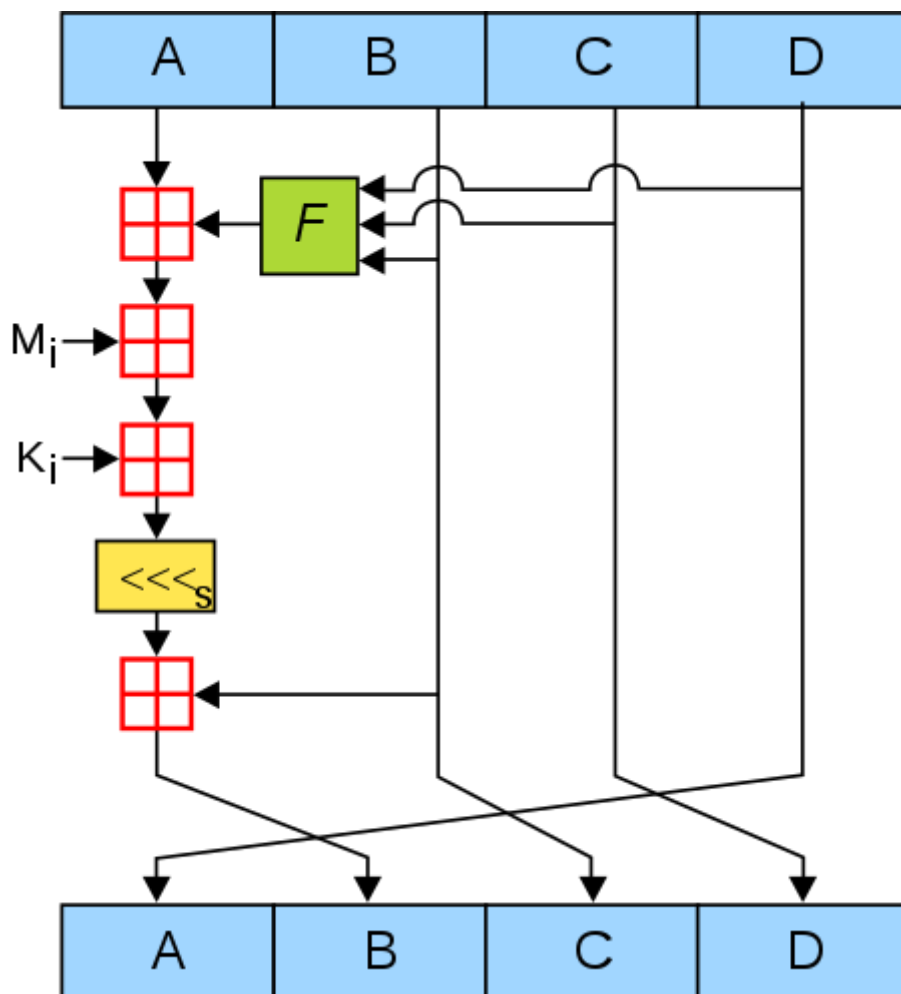
Initialize MD buffer

128-bit缓冲区 (A, B, C, D) 用于计算消息摘要的值。这里的A, B, C, D是32位寄存器，并按以下方式（采用小端存储(little-endian) 的存储结构）初始化

Word A	01	23	45	67
Word B	89	AB	CD	EF
Word C	FE	DC	BA	98
Word D	76	54	32	10

Little-Endian 将低位字节排放在内存的低地址端，高位字节 排放在内存的高地址端。相反Big-Endian 将高位字节排放在内存的低地址端，低位字节排放在内存的高地址端。存 储结构与CPU 体系结构和语言编译器有关。PowerPC 系列 采用Big Endian 方式存储数据，而Intel x86系列则 采用Little Endian 方式存储

Processing message in 16-word block



Auxiliary Functions

辅助函数接受三个32位的输入，然后给出一个32位的输出。辅助函数将逻辑and、or和xor应用于输入，是一个32位非线性逻辑函数。稍后我们将展示如何在4轮循环中使用这4个函数（作为生成函数/轮函数）（每轮有16个操作）。这些辅助函数的功能是

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	$I(b, c, d)$	$c \oplus (b \vee \neg d)$

The Table

T表的生成

$$T[i] = \text{int}(2^{32} * |\sin(i)|)$$

int取整函数，sin正弦函数，以i作为弧度输入

我们将该表表示为K [1 64]。 K [i]是表格的第i个元素。 该表的元素已预先计算

K[0.. 3]	0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee
K[4.. 7]	0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501
K[8..11]	0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be
K[12..15]	0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821
K[16..19]	0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa
K[20..23]	0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8
K[24..27]	0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed
K[28..31]	0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a
K[32..35]	0xfffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c
K[36..39]	0xa4bceea4, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbc70
K[40..43]	0x289b7ec6, 0xeaad127f, 0xd4ef3085, 0x04881d05
K[44..47]	0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665
K[48..51]	0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039
K[52..55]	0x655b59c3, 0x8f0ccc92, 0xffeff47d, 0x85845dd1
K[56..59]	0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1
K[60..63]	0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391

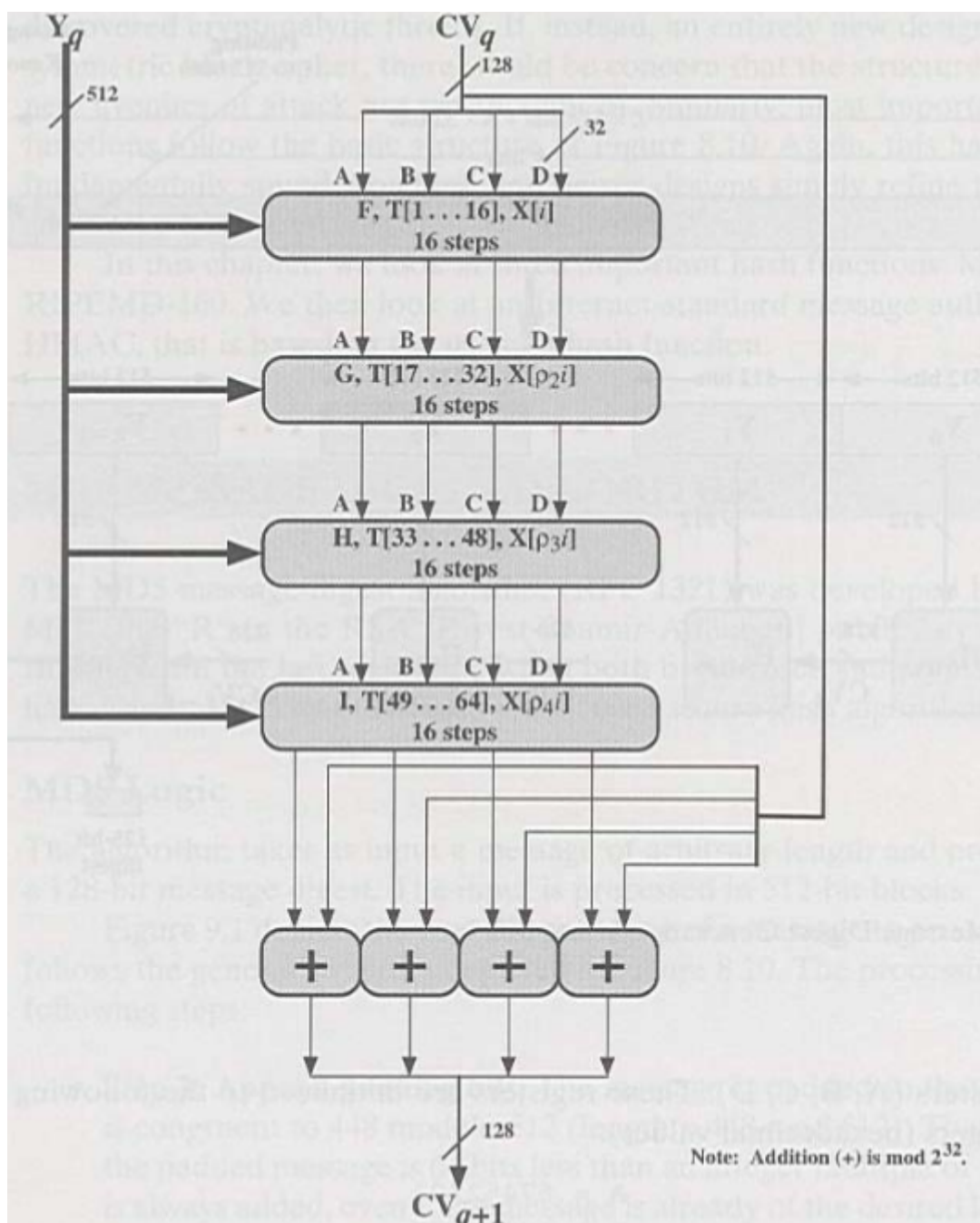
The Algorithm

算法流程：

以512-bit 消息分组为单位，每一分组 Y_q ($q = 0, 1, \dots, L-1$) 经过4个循环的压缩算法，表示为：

$$\begin{aligned} CV_0 &= IV \\ CV_i &= H_{MD5}(CV_{i-1}, Y_i) \end{aligned}$$

输出结果： $MD = CV_L$



回想一下步骤2，我们将消息分成512位的块，然后每个512位的块是16个32位的字。这16个字(每个32位)块表示为 $M[0 \dots N-1]$ 。现在，对于每个字块，我们执行4轮，每轮有16个操作。每一轮都遵循基本模式。

每轮循环中的一次迭代运算逻辑：

(1) 对A迭代： $a \leftarrow b + ((a + g(b, c, d) + X[k] + T[i]) \lll s)$

(2) 缓冲区(A, B, C, D) 作循环轮换：

(A, B, C, D) \rightarrow (D, A, B, C)

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)
- $\lll s$: 将32位输入循环左移(CLS) s位; s 为规定值
- $X[k]$: 当前处理消息分组q 的第k个($k = 0 \dots 15$) 32位字, 即 $M_{q \cdot 16 + k}$
- $T[i]$: T 表的第i个元素, 32位字; T表总共有64个元素, 也称为加法常数
- + : 模 2^{32} 加法

- 压缩函数H_{MD5}
 - HMD5 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K_j和消息分组的不同部分X_j做16次迭代运算，生成下一轮循环的输入。

4轮循环共有64次迭代运算。

- X[k]的确定

设 $j = i - 1$:

- 第1轮迭代: $k = j$.
 - 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: $k = (1 + 5j) \bmod 16$.
 - 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: $k = (5 + 3j) \bmod 16$.
 - 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: $k = 7j \bmod 16$.
 - 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(1..64) 采用的左循环移位的s值

每次循环使用相同的迭代逻辑和4 * 16 次运算的预设参数表

$s[1..16] = \{ 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22 \}$

$s[17..32] = \{ 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20 \}$

$s[33..48] = \{ 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23 \}$

$s[49..64] = \{ 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21 \}$

Output

上述操作完成之后，缓冲区所保存的 (a, b, c, d) 就是我们所需要的结果。

算法结束时，对32位寄存器A的值按照little-endian转换成4个字节，顺序输出其8个16进制数符号；同样分别处理寄存器B, C, D，输出 MD5 值的其他24个16进制数符号。寄存器A, B, C, D联合输出的结果是32个16进制数符号(每个16进制数符号占4 bits，共128 bits)。

Source code

更加具体的代码在附件中给出

利用string和bitset实现

- 因为我们的输入是任意长度的，所以我们使用string来读取输入
- 因为涉及很多位的与或异或操作，所以使用bitset

具体流程：

1. 将读取到的string转为对应的bit串
2. 然后将转化的bit串填充标识/长度
3. 分块，将每块按little-endian重新排列，进行迭代

4. 输出结果：将每一缓冲区的little-endian转换回来，再拼在一起，转换为16进制输出

具体的类定义以及声明

由于我们迭代过程中有4个轮函数，为了代码的简便，我们使用函数数组来存储它们

```
1  #pragma once
2  #include<string>
3  #include<bitset>
4
5  using std::string;
6  using std::bitset;
7
8  class MD5 {
9  private:
10     bitset<32>IV[4] = { 0x67452301,0xefcdab89,0x98badcfe,0x10325476 };
11     bitset<32>MD[4];
12
13     //MD5 encryption related
14     static bitset<32>(*fistleFunc[4])(bitset<32>, bitset<32>, bitset<32>);
15     static unsigned int table_int[64];
16     static int leftShift[16];
17     static int x[64];
18     static inline bitset<32> functionF(bitset<32>b, bitset<32>c,
19     bitset<32>d);
20     static inline bitset<32> functionG(bitset<32>b, bitset<32>c,
21     bitset<32>d);
22     static inline bitset<32> functionH(bitset<32>b, bitset<32>c,
23     bitset<32>d);
24     static inline bitset<32> functionI(bitset<32>b, bitset<32>c,
25     bitset<32>d);
26     static inline bitset<32> loopLeftShift(bitset<32>input,int n);
27
28     //radix and character position related
29     static string dec264bitsbin(int input);
30     static string string2bit(string &plain);
31     static string littleEndian(string input);
32     static string deLittleEndian(bitset<32> &input);
33     static string bits2hex(string &input);
34
35     void chunkProcess(string &chunk);
36     void padding(string &input);
37     void transform(string plain);
38 public:
39     explicit MD5(string plain);
40     string getCipher();
41 };
```

按little-endian（便于MD5处理）排列以及Big-Endian（更符合我们的输入和输出习惯）排列

```

1 string MD5::littleEndian(string input) {
2     return input.substr(24, 8) + input.substr(16, 8) + input.substr(8, 8) +
    input.substr(0, 8);
3 }
4
5 string MD5::deLittleEndian(bitset<32>&input) {
6     string temp = input.to_string();
7     return temp.substr(24, 8) + temp.substr(16, 8) + temp.substr(8, 8) +
    temp.substr(0, 8);
8 }

```

填充标识和长度

```

1 void MD5::padding(string &input) {
2     string fill = "1";
3     int origin_len = input.length();
4
5     //Calculate the length required to fill '0'
6     int len = (input.length() % 512 - 448);
7     len = (len >= 0) ? 512 - len : -len;
8     for (int i = 1; i < len; i++) {
9         fill += "0";
10    }
11
12    //Fill in the "10..." string and the original string bit length in
    little endian order
13    input += fill + dec264bitsbin(origin_len);
14 }

```

块的迭代函数处理

后面发现需要模 2^{32} 的加法，但是bitset是没有这样的运算符，所以我们需要先通过to_ulong()转成unsigned long之后再做法

```

1 void MD5::chunkProcess(string &trunk)
2 {
3     bitset<32> Message[16];
4     for (int i = 0; i < 16; i++) {
5         Message[i] = bitset<32>(MD5::littleEndian(trunk.substr(32 * i,
6         32)));
7     }
8
9     //Iterative operation
10    bitset<32> MD[4] = { IV[0], IV[1], IV[2], IV[3] };
11    for (int i = 0; i < 64; i++) {
12        //Fixed different generation functions for each round of loop
13        bitset<32> fistleres = MD5::fistlerFunc[i / 16](MD[1], MD[2],
14        MD[3]);
15
16        //Combine the specified T table element with the different part of
17        the message group
18        bitset<32> tempRes = (fistleres.to_ulong() + MD[0].to_ulong() +
19        Message[MD5::X[i]].to_ulong() + MD5::table_int[i])&0xffffffff;
20
21        MD[0] = MD[3];
22        MD[3] = MD[2];
23    }
24 }

```

```

19     MD[2] = MD[1];
20     MD[1] = MD[1].to_ulong() + MD5::loopLeftShift(tempRes,
MD5::leftShift[4*(i/16)+(i%16)%4]).to_ulong();
21 }
22
23     IV[0] = (unsigned int)MD[0].to_ulong() + (unsigned
int)IV[0].to_ulong();
24     IV[1] = (unsigned int)MD[1].to_ulong() + (unsigned
int)IV[1].to_ulong();
25     IV[2] = (unsigned int)MD[2].to_ulong() + (unsigned
int)IV[2].to_ulong();
26     IV[3] = (unsigned int)MD[3].to_ulong() + (unsigned
int)IV[3].to_ulong();
27
28 }

```

利用string和cstdint实现

- 因为我们的输入是任意长度的，所以我们使用string来读取输入
- 因为我们涉及到模加法运算，而且很多字节（Little-endian）操作，所以使用uint8_t, uint16_t, uint32_t.....

具体流程：

- 将读取的string转为char再转为uint8_t，填充到大小为512bits的buffer中，一旦buffer满了马上对它进行迭代处理
- 将需要padding的“10.....”和原消息位长度填充到buffer中一旦buffer满了马上对它进行迭代处理
- 通过std::cout << std::hex将最后的结果以十六进制输出

具体的类声明和定义

```

1  #pragma once
2  #pragma once
3
4  #include<cstdint>
5  #include<string>
6
7  using std::string;
8
9  #define BLOCKSIZE (512 / 8)
10
11  class MD5 {
12  private:
13      uint32_t IV[4] = { 0x67452301,0xefcdab89,0x98badcfe,0x10325476 };
14      uint32_t MD[4];
15
16      //MD5 encryption related
17      static uint32_t(*fistleFunc[4])(uint32_t, uint32_t, uint32_t);
18      static uint32_t table_int[64];
19      static int leftShift[16];
20      static int x[64];
21      static inline uint32_t functionF(uint32_t b, uint32_t c, uint32_t d);
22      static inline uint32_t functionG(uint32_t b, uint32_t c, uint32_t d);
23      static inline uint32_t functionH(uint32_t b, uint32_t c, uint32_t d);
24      static inline uint32_t functionI(uint32_t b, uint32_t c, uint32_t d);
25      static inline uint32_t loopLeftShift(uint32_t input, int n);
26

```

```

27 //radix and character position related
28 static void littleEndian(uint32_t output[], uint8_t input[], int size);
29 static void deLittleEndian(uint8_t output[], uint32_t input[], int
size);
30
31 void chunkProcess(uint8_t trunk[]);
32 void padding(uint32_t len[2]);
33 void transform(string plain);
34 void fillAndEncrypt(uint8_t plain[], unsigned int len);
35
36 uint8_t buffer[BLOCKSIZE] = { '0' };//Record the last not aligned block
37 uint64_t messageLen;
38 public:
39 explicit MD5(string plain);
40 void getCipher();
41 };

```

按little-endian（便于MD5处理）排列以及Big-Endian（更符合我们的输入和输出习惯）排列

```

1 //Associated with radix
2 void MD5::littleEndian(uint32_t output[],uint8_t input[],int size) {
3     for (int i = 0, j = 0; i < size; i += 4,j++) {
4         output[j] = ((uint32_t)input[i] | (uint32_t)input[i + 1] << 8 |
(uint32_t)input[i + 2] << 16 | (uint32_t)input[i + 3] << 24);
5     }
6 }
7
8 void MD5::deLittleEndian(uint8_t output[],uint32_t input[],int size) {
9     for (int i = 0, j = 0; i < size; i++, j += 4) {
10         output[j] = input[i] & 0xff;
11         output[j + 1] = (input[i] >> 8) & 0xff;
12         output[j + 2] = (input[i] >> 16) & 0xff;
13         output[j + 3] = (input[i] >> 24) & 0xff;
14     }
15 }

```

边填充数据到buffer边检验是否需要将当前缓冲区加密

```

1 void MD5::fillAndEncrypt(uint8_t plain[], unsigned int len){
2     unsigned int curPos = messageLen / 8 % BLOCKSIZE;
3     unsigned int firstToFill = BLOCKSIZE - curPos;
4     unsigned int copyLen = 0;
5
6     messageLen += len << 3;
7
8     //If the length exceeds the gap in the current buffer
9     if (len >= firstToFill) {
10         for (int i = curPos,j=0; i < BLOCKSIZE; i++,j++) {
11             buffer[i] = plain[j];
12             copyLen++;
13         }
14         chunkProcess(buffer);
15         curPos = 0;
16
17         for (int i = firstToFill; i + BLOCKSIZE <= len; i+=BLOCKSIZE) {
18             chunkProcess(&plain[i]);

```

```

19         copyLen += BLOCKSIZE;
20     }
21 }
22
23 for (unsigned int i = curPos, j = copyLen; j < len; i++, j++) {
24     buffer[i] = plain[j];
25 }
26 }

```

填充'10.....'和长度

```

1 void MD5::padding(uint32_t len[2]) {
2     static uint8_t paddingTable[64] = {
3         0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
4         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
5         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
6     };
7
8     uint8_t lenBits[64] = { '0' };
9     deLittleEndian(lenBits, len, 2);
10
11     unsigned int padLen = (len[0] % 512);
12     padLen = (padLen < 448) ? (448 - padLen) : (448 + 512 - padLen);
13     fillAndEncrypt(paddingTable, padLen / 8);
14     fillAndEncrypt(lenBits, 64);
15
16 }

```

块的迭代函数处理

```

1 void MD5::chunkProcess(uint8_t trunk[])
2 {
3     uint32_t Message[16];
4     MD5::littleEndian(Message, trunk, BLOCKSIZE);
5
6     uint32_t MD[4] = { IV[0], IV[1], IV[2], IV[3] };
7     for (int i = 0; i < 64; i++) {
8         uint32_t fistleres = MD5::fistlerFunc[i / 16](MD[1], MD[2], MD[3]);
9         uint32_t tempRes = fistleres + MD[0] + Message[MD5::X[i]] +
MD5::table_int[i];
10
11         MD[0] = MD[3];
12         MD[3] = MD[2];
13         MD[2] = MD[1];
14         MD[1] = MD[1] + MD5::loopLeftShift(tempRes, MD5::leftShift[4 * (i /
15 16) + (i % 16) % 4]);
16     }
17
18     IV[0] = MD[0] + IV[0];
19     IV[1] = MD[1] + IV[1];
20     IV[2] = MD[2] + IV[2];
21     IV[3] = MD[3] + IV[3];
22 }

```

运行结果

- 运行方法

main函数通过

```
1 getline(cin, input);
2 cout << MD5(input).getCipher() << endl;
```

从控制台读取字符串加密后输出

由于输出结果时没有设置暂停，所以可以通过[重定向输出到文本文件中](#)查看结果

- 根据Wikipedia上的例子

```
1 MD5("The quick brown fox jumps over the lazy dog") =
2 9e107d9d372bb6826bd81d3542a419d6
3
4 MD5("The quick brown fox jumps over the lazy dog.") =
5 e4d909c290d0fb1ca068ffaddf22cbd0
6
7 MD5("") =
8 d41d8cd98f00b204e9800998ecf8427e
```

- 根据自己寻找的一个长字符串

```
1 The MD5 algorithm is specified for messages consisting of any number of
  bits; it is not limited to multiples of eight bits (octets, bytes). Some
  MD5 implementations such as md5sum might be limited to octets, or they
  might not support streaming for messages of an initially undetermined
  length.
```

The MD5 algorithm is specified for messages consisting of any number of bits; it is not limited to multiples of eight bits (octets, bytes). Some MD5 implementations such as md5sum might be limited to octets, or they might not support streaming for messages of an initially undetermined length.


32位大写 A65C8A69E56A8D65EA70C251512A5DCE

32位小写 a65c8a69e56a8d65ea70c251512a5dce


16位大写 E56A8D65EA70C251

16位小写 e56a8d65ea70c251

bitset版本的MD5:

 E:\VSprogram\MD5\Debug\MD5.exe

```
The quick brown fox jumps over the lazy dog
9e107d9d372bb6826bd81d3542a419d6
```

 E:\VSprogram\MD5\Debug\MD5.exe

```
The quick brown fox jumps over the lazy dog.
e4d909c290d0fb1ca068ffaddf22cbd0
```

```
E:\VSprogram\MD5\Debug\MD5.exe
^Z
d41d8cd98f00b204e9800998ecf8427e

E:\VSprogram\MD5\Debug\MD5.exe
The MD5 algorithm is specified for messages consisting of any number of bits; it is not limited to multiples of eight bi
ts (octets, bytes). Some MD5 implementations such as md5sum might be limited to octets, or they might not support stream
ing for messages of an initially undetermined length.
a65c8a69e56a8d65ea70c251512a5dce
```

cstdint版本的MD5:

```
E:\VSprogram\MD5_3\Debug\MD5_3.exe
The quick brown fox jumps over the lazy dog
9e107d9d372bb6826bd81d3542a419d6

E:\VSprogram\MD5_3\Debug\MD5_3.exe
The quick brown fox jumps over the lazy dog.
e4d909c290d0fb1ca068ffaddf22cbd0

E:\VSprogram\MD5_3\Debug\MD5_3.exe
^Z
d41d8cd98f00b204e9800998ecf8427e

E:\VSprogram\MD5_3\Debug\MD5_3.exe
The MD5 algorithm is specified for messages consisting of any number of bits; it is not limited to multiples of eight bi
ts (octets, bytes). Some MD5 implementations such as md5sum might be limited to octets, or they might not support stream
ing for messages of an initially undetermined length.
a65c8a69e56a8d65ea70c251512a5dce
```