信息安全第二次作业 MD5算法的设计与实现

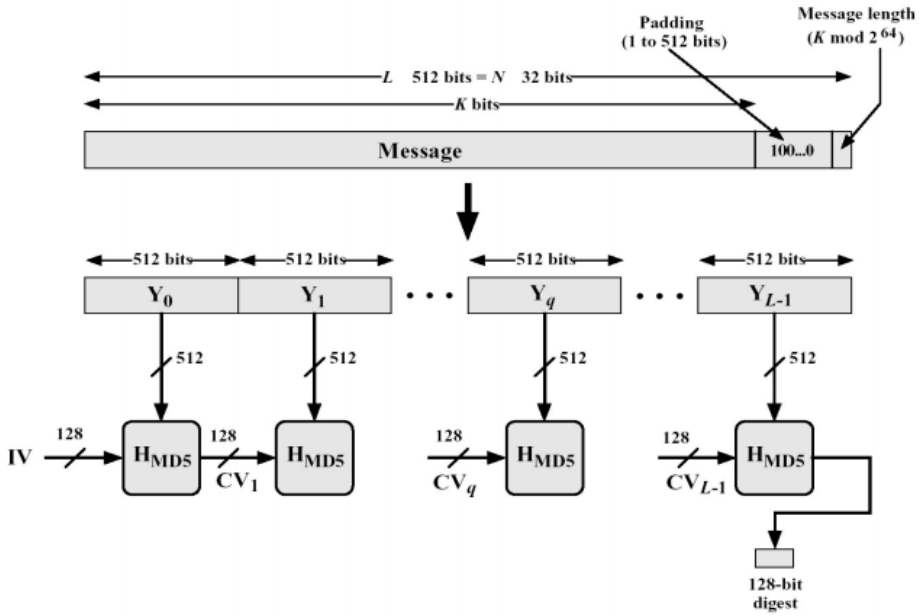
数据科学与计算机学院 17343094 彭湃

在上一次的作业中，我们实现了简易的`DES`算法，而在这次的作业中，我们要求实现简易的`MD5`算法，下面我将从`算法原理概述`、`总体结构`、`模块分解`、`数据结构`、`程序设计源代码`、`运行结果`几个方面来完成我们的程序设计与分析。

一、算法原理概述

- `MD5`算法是一种被广泛使用的`hash`算法，用于确保信息传输的完整性和一致性。
- `MD5`算法使用`little-endian`(小端模式)，输入任意不定长度信息，以`512-bit`进行分组，生成四个`32-bit`数据，最后联合输出固定`128-bit`的信息摘要。
- 我们简要可以将`MD5`算法的实现过程概括为：填充 -> 分块 -> 缓冲区初始化 -> 循环压缩 -> 得出结果。

基本流程图如下：



二、总体结构

我们本次的项目主要由三个部分组成，分别是`MD5.h` 、 `MD5.cpp`两个部分组成，(万年不喜欢写头文件，这次是实在看不过去一堆表和`define`在主文件……)

- `MD5.h`：主要是用来防止相关的表文件和相关宏定义语句。
  - `Order_Table[]`：四轮迭代时候的顺序表。
  - `S[]`：四次迭代运算采用的左循环移位的`s`值 。
  - `T[]`：压缩函数用到的T值表。
  - 相关宏定义：`F`、`G`、`H`、`I`、`LS`等。
  - 函数声明：`cpp`文件中函数的预声明。
- `MD5.c`：主要又由两个部分组成，一个是我们的具体实现部分，一个是检测部分(`main`函数)。
  - `void Init(const char *msg)`

    对输入的信息进行填充、初始化。
  - `HMD5(void *buffer)`

    装入标准的幻数`A B C D`，主循环等。
  - `main()`

    可以理解为前端，接收我们想要处理的明文，进行操作后，拼接`A B C D`得到结果。

三、模块分解

1.填充模块

其实填充模块的流程很简单：

- 对获得的数据求其位数，我们定义为`K`. 同时定义填充的位数`P`。

```
unsigned int K = strlen(msg);
unsigned int P = 0;
```

- 因为要求 得填充后的消息长度为 `N * 512 + 448`，所以我们需要对`K`值进行判断。
  - 如果`K`直接满足这个条件，那么`p`直接就是512.

```
if(K*8%512==448)
{
    P = 512;
}
```

  - 否则

```
P = 448 - (K*8)%512
```

- 然后我们向上述填充好的消息尾部附加 K 值的低64位 (即 `K mod 264`)。

2.分组模块

我们将我们第一步填充好的信息进行分组，使得我们得到`N`个`32bit`的字。

```
buffer[msgLen-5] = (K*8&0xFF000000)>>24;
buffer[msgLen-6] = (K*8&0x00FF0000)>>16;
buffer[msgLen-7] = (K*8&0x0000FF00)>>8;
buffer[msgLen-8] = K*8&0x000000FF;
```

3.初始化模块

- 首先我们需要初始化一个128-bit 的 `MD` 缓冲区，记为 `CVq`，表示成4个32-bit寄存器 (A, B, C, D);

  ```
  unsigned int A, B, C, D;
  for (int i = 0;i < L;i++){
      A = MD[0];//CVq
      B = MD[1];
      C = MD[2];
      D = MD[3];
  }
  ```

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

按照ppt上给的这张表，并且采用小段存储的方式作为初始向量`IV`.

```
unsigned int *MD = new unsigned int[4]
{0x67452301,0xEFCDAB89,0x98BADCFE,0x10325476};
```

> 所谓小端存储，就是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 。

- 接下来的迭代等将会在`MD`缓冲区进行，总控流程如下， 每一分组 Yq (q = 0, 1, ..., L-1) 都要经过4个循环的压缩算法。

  ```
  HMD5(&buffer[i*64]);
  MD[0] += A;
  MD[1] += B;
  MD[2] += C;
  MD[3] += D;
  ```

4.主循环模块

这个模块是整个函数最麻烦的部分。

- 每轮压缩有64次迭代，1-16次使用F函数，17-32为G，33-48为H，49-64为I。
- 生成函数所生成的值结合T表对应元素+消息的某部分运算。
- 详细来说就是`a = a ← b + ((a + g(b, c, d) + X[k] + T[i]) <<<s)`缓冲区轮换：`(B, C, D, A) ← (A, B, C, D)`.
- 然后得出的输出结果作为下一次的输入，所以直接使用寄存器存储 。

`F G H I`和左移我们写在宏定义里面。

```
#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 LS(x,n) ((x << n) | (x >> (32-n)))
```

5.检测模块

这个其实就是我们的`main`主函数，我们读入我们想要加密的信息。

```
char msg[32];
scanf("%s", msg);
```

对其进行`MD5`加密。

```
Init(msg);
```

```
for(int i = 0;i < 4;i++){
        a = (MD[i]&0xFF000000)>>24;
        b = (MD[i]&0x00FF0000)>>16;
        c = (MD[i]&0x0000FF00)>>8;
        d = MD[i]&0x000000FF;
}
```

最后可以得到32位的结果，我们再与标准答案进行对比即可。

```
for(int i = 0;i < 4;i++){
    printf("%x%x%x%x", d,c,b,a);
}
```

四、数据结构

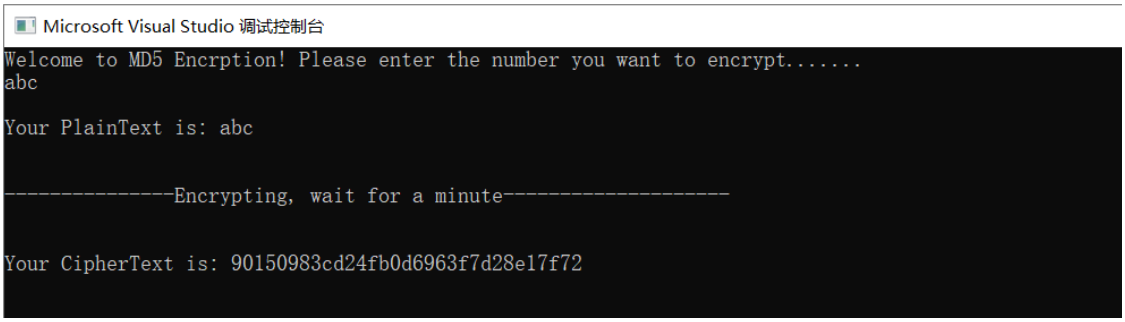
- `unsigned int`：各类表结构。
- `char`：最后处理的明文。
- `16进制数据`：最后输出的密文。

五、c++源代码

- src/MD5.h
- src/MD5.cpp

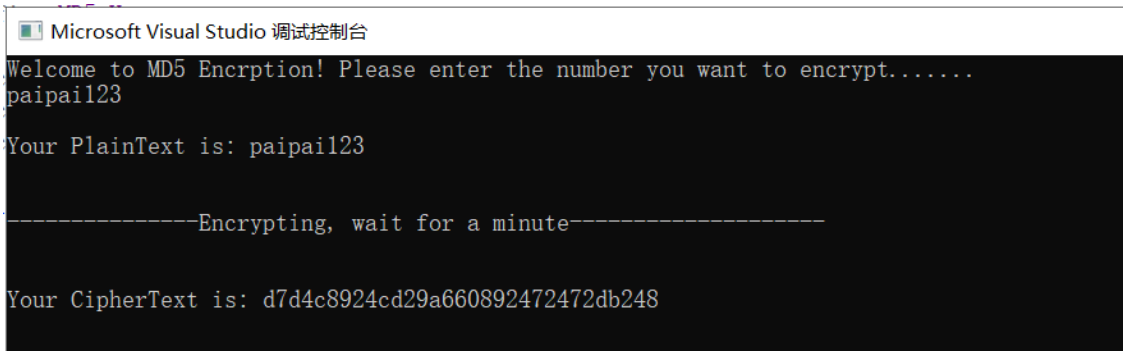
六、运行结果

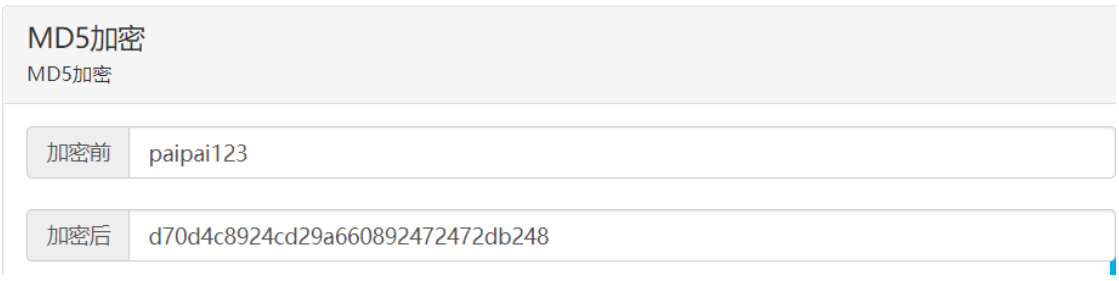
- 运行平台：`Virsual Studio2017`
- 第一组测试：



与我们通过平台进行加密的结果一致。

- 第二组测试：





结果一致。

至此我们的实验结束。