Web 安全第二次作业：MD5算法程序设计与实现

陈铭涛

16340024

## 算法原理概述

MD5算法是一种广泛使用的 Hash 算法，常用于确保信息传输的完整性与一致性。对于输入的任一不定长度信息， MD5算法在对最后一块进行填充后用512个比特对其进行分组，使用压缩函数将其生成4个32比特的数据，合并为128比特的信息摘要。MD5实现的基本过程为填充，分块，缓冲区初始化，循环压缩，得出结果。

## 算法总体流程

1. Padding：对于给出的长度为 K 比特的数据，MD5将其填充 P 比特的数据，数据为100…0，填充后 K+P 模512后为448。填充的数据量范围为1至512位比特。完成上步后在数据再填充 K 按 little endian 排列的64位数据，最终得到的数据长度 K+P+64模512后为0.
2. 初始化：初始化32比特寄存器 A，B，C，D作为初始向量，数据如下：

A=0x67452301

B=0xEFCDAB89

C=0x98BADCFE

D=0x10325476

要注意的是需要按 little endian 排列字节数据，即低位字节放在内存的低地址位置。

1. 压缩函数：将消息按512比特分组为单位与 CV 向量传入压缩函数，压缩函数进行4轮16次迭代的循环输出128位下一组的 CV 值，最后一组数据输出的 CV 值即为 MD5的结果。
2. 循环迭代：压缩函数输入的128位 CV 值划分为各32位的 a, b, c, d 值进行循环迭代，每轮循环各使用如下的对应生成函数 g结合对应的 T 表元素与消息的不同部分 X进行16次迭代运算：

第一轮循环：

第二轮循环：

第三轮循环：

第四轮循环：

每一轮的迭代运算为*:*

其中 X[k] 表示消息组中的第 k 个32位字，T[i]表示 T 表的第 i个元素，<<<s 表示将数据循环左移 s 位，位移的量来自于 s 值表。

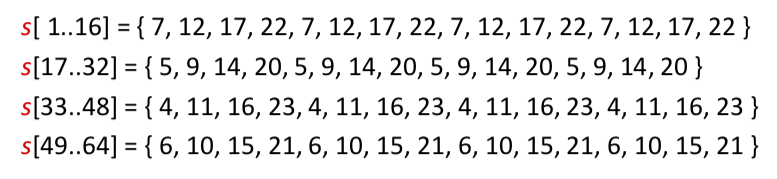
运算后将b, c, d, a 分别轮换为a, b, c, d。

最后一个分组完成循环迭代后的结果即为 MD5输出的结果。

1. 数据表：上面使用的X[k]中 k 的确定方法为：对于每轮的第 i 次迭代（i=1..16），令j=i-1, 则对于第一轮迭代 *k=j*，第二轮迭代*k=(1+5j) mod 16*，第三轮迭代*k=(5+3j) mod 16*，第四轮迭代*k = 7j mod 16*。

T 表的生成方式为：对于 T 表中的第 i 个元素，.

循环左移的 s 值表如下：



## 代码实现简述

提交的代码使用 C++语言编写，包含的文件分别为：

* md5.cpp, md5.h 分别为 MD5类的实现与定义，为程序的主要部分
* md5test.cpp, md5test.h 为进行测试的函数的实现与定义，用于测试与辅助显示 md5结果。
* main.cpp 为程序主函数，其使用方法参考了 macOS 系统终端自带的 md5命令使用方法。
* MD5类为主要的内容，其包含的各方法与介绍如下：

std::array<unsigned char, 16> **sum**(const std::vector<unsigned char> &data);

对于输入的字节 vector 进行md5计算，返回16字节（128比特）的 md5结果。

std::array<unsigned char, 16> **sum**(const std::string &data);

对传入的字符串进行 md5计算，返回16字节（128比特）的 md5结果。

std::array<unsigned char, 16> **md5Sum**(std::vector<unsigned char> data)

上面两个函数实际调用的函数，包含了 md5算法的全过程。

void **padding**(std::vector<unsigned char> & data);

对传入的消息数据根据算法的方法原地进行填充操作。

uint32\_t **circularLeftShift**(uint32\_t data, unsigned int c);

对传入的32位数据进行循环左移 c 位

std::array<unsigned char, 16> **HMD5**(const std::array<unsigned char, 64> &data, const std::array<unsigned char, 16> &cv);

压缩函数，传入数据与 cv 值，其中包含了4轮16次循环迭代操作，返回128位运算后的结果。

auto F = [](uint32\_t b, uint32\_t c, uint32\_t d) { return (b & c) | (~b & d); };

auto G = [](uint32\_t b, uint32\_t c, uint32\_t d) { return (b & d) | (c & ~d); };

auto H = [](uint32\_t b, uint32\_t c, uint32\_t d) { return b ^ c ^ d; };

auto I = [](uint32\_t b, uint32\_t c, uint32\_t d) { return c ^ (b | ~d); };

array<function<**uint32\_t**(uint32\_t, uint32\_t, uint32\_t)>, 4> g = {F, G, H, I};

该4个函数为 HMD5函数实现中使用的4个轮函数 g。

* md5test 中包含的函数介绍如下：

void **md5StrTest**(const std::string& input)

将传入的字符串进行 md5运算，结果以16进制形式打印至标准输出

void **md5FileTest**(const char\* filename)

传入文件名，使用二进制模式读取文件并进行 md5运算，结果以16进制输出。

void **md5RunTest**();

使用预先设定好的一组字符串进行md5运算测试，输出结果。

std::string **bytesToHexStr**(std::array<unsigned char, 16> data);

将输入的128比特数据转为16进制形式的字符串。

## 程序编译

程序使用 cmake 进行编译，在 Windows，Mac和 Linux 系统下编译指令分别如下：

*# Linux, macOS*

cmake **.**

make

*#Windows*

cmake -G "MinGW Makefiles" **.**

mingw32-make

编译完成后将会把可执行文件输出至 bin 文件夹下，提交的代码目录下的 bin 文件夹已包含这三个系统下的可执行文件。

程序的使用方法模仿了 macOS 命令行下的 md5工具，具体如下：

./md5 [-x] [-s string] [files ...]

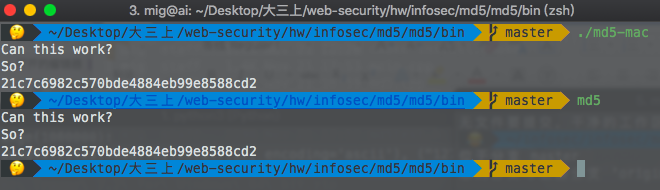
其中各参数作用如下：

* -x 会使程序执行测试，使用预先定义的一组字符串进行 md5运算结果的测试。
* -s 后接字符串，程序会对该字符串进行运算并输出结果，该参数可重复指定
* 程序读取到第一个非选项的参数时将会进入文件读取模式，将剩余的参数视为文件名并读取文件进行运算输出结果。
* 若未指定任何参数，程序将会从标准输入中读取输入，并在用户输入 EOF 后输出结果。

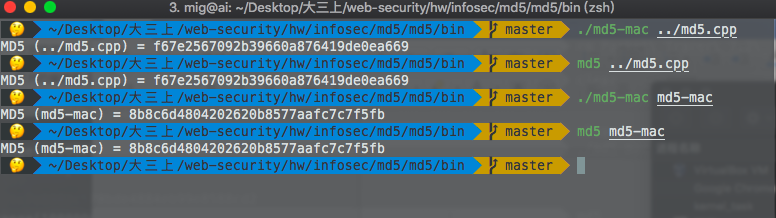
## 程序运行结果

在macOS 10.14.1 系统下进行程序的运行测试，与系统自带 md5工具进行结果对比，程序运行的效果如下：

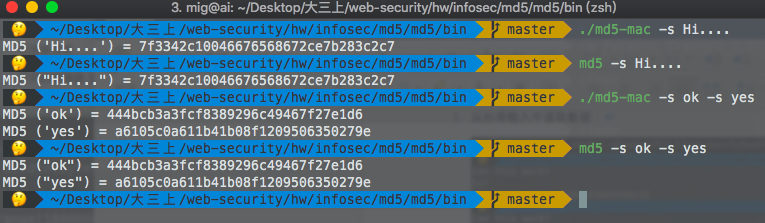
1. 从标准输入中读取数据：



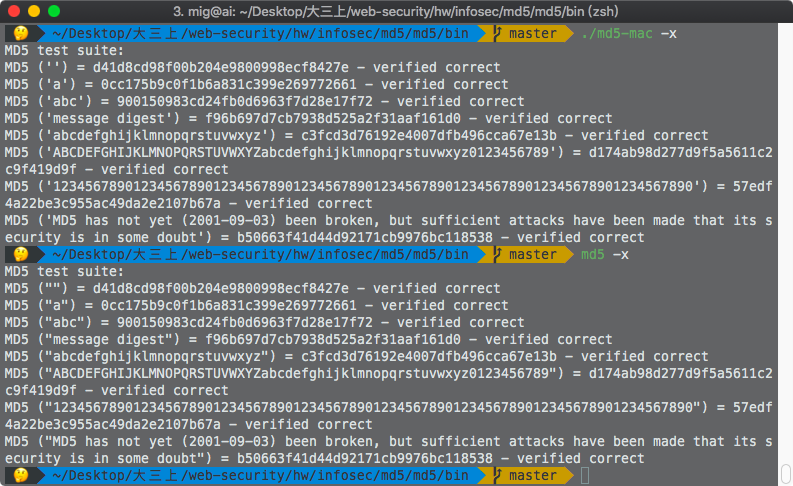
1. 对代码文件以及可执行文件自身计算 md5 值：



1. 从命令行读取字符串：



1. 使用-x 执行测试：



程序在macOS 10.14.1，Ubuntu 18.04.1 LTS以及 Windows 10下测试运行正常，输出结果与各系统自带的md5工具输出的结果相同。