



南开大学
Nankai University

南 开 大 学

网 络 安 全 技 术

MD5 加密算法

学院：网络安全学院

年级：2021 级

班级：信息安全 1 班

学号：2111252

姓名：李佳豪

手机号：13191110713

2024 年 5 月 8 日

目录

一、 实验目的	1
二、 实验内容	1
三、 MD5 原理	1
(一) 初始化 MD5 状态变量	1
(二) 处理消息块	2
(三) 填充	2
(四) 附加长度	2
(五) 处理每个 512 位块	2
(六) 输出最终 MD5 哈希	3
四、 实验步骤	3
(一) 程序总览	3
(二) MD5 加密部分	4
1. 总览	4
2. 基本函数	5
3. API 调用顺序	6
4. 重载 Update 函数	8
(三) 命令解析部分	10
五、 实验结果	12
(一) md5 测试样例	12
(二) 完整程序	12
1. -h 打印 help	12
2. -t 测试样例	13
3. -s 计算任意字符串的 Hash	13
4. -f 计算文件 Hash	13
5. -c 校验文件	13
六、 实验遇到的问题及其解决方法	14
(一) 数据类型	14
(二) 大小端	14
(三) 流式传入、计算	14
七、 Linux md5 加密密码	15
八、 实验结论	15

一、 实验目的

在讨论了传统的对称加密算法 DES 原理与实现技术的基础上，本章将以典型的非对称密码体系中 RSA 算法为例，以基于 TCP 协议的聊天程序加密为任务，系统地进行非对称密码体系 RSA 算法原理与应用编程技术的讨论和训练。通过练习达到以下的训练目的：

1. 深入理解 MD5 算法的基本原理。
2. 掌握利用 MD5 算法生成数据摘要的所有计算过程。
3. 掌握 Linux 系统中检测文件完整性的基本方法。
4. 熟悉 Linux 系统中文件的基本操作方法。

二、 实验内容

1. 准确地实现 MD5 算法的完整计算过程。
2. 对于任意长度的字符串能够生成 128 位 MD5 摘要。
3. 对于任意大小的文件能够生成 128 位 MD5 摘要。
4. 通过检查 MD5 摘要的正确性来检验原文件的完整性。

补充：由于在 windows 下调试更方便，且 linux 和 Windows 下 socket 编程原理相同，因此延续上一次、上上次实验采用 windows。

三、 MD5 原理

MD5 算法包括以下步骤

1. **初始化 MD5 状态变量**
2. **处理消息块**
3. **填充**
4. **附加长度**
5. **处理每个 512 位块**
6. **输出最终 MD5 哈希**

(一) 初始化 MD5 状态变量

MD5 使用以下数值初始化四个 32 位状态变量：

$$\begin{aligned} A &= 0x67452301 \\ B &= 0xEFCDAB89 \\ C &= 0x98BADCFE \\ D &= 0x10325476 \end{aligned} \tag{1}$$

(二) 处理消息块

消息以 512 位块（64 字节）为单位进行处理。如果消息长度不是 512 的倍数，则需要填充。

(三) 填充

消息的填充方式如下：

- 在消息末尾添加一个 ‘1’ 比特。
- 添加 ‘0’ 比特，直到消息长度模 512 的结果为 448。

(四) 附加长度

原始消息的长度（以比特为单位）以 64 位小端整数的形式附加。填充后消息总长度应为 512 的倍数。

(五) 处理每个 512 位块

每个 512 位块的处理方式如下：

1. 将块划分为十六个 32 位字 M_i , $0 \leq i \leq 15$ 。
2. 定义四个辅助函数 F、G、H 和 I:

$$\begin{aligned} F(x, y, z) &= (x \wedge y) \vee (\neg x \wedge z) \\ G(x, y, z) &= (x \wedge z) \vee (\neg z \wedge y) \\ H(x, y, z) &= x \oplus y \oplus z \\ I(x, y, z) &= y \oplus (x \vee \neg z) \end{aligned} \quad (2)$$

3. 使用当前状态值初始化四个变量 A、B、C 和 D。
4. 执行 64 次主循环迭代：

- 第 1 轮： $0 \leq i \leq 15$ ，使用函数 F 和下列循环左移位数 s ：

$$s = [7, 12, 17, 22] \quad (3)$$

- 第 2 轮： $16 \leq i \leq 31$ ，使用函数 G 和下列循环左移位数 s ：

$$s = [5, 9, 14, 20] \quad (4)$$

- 第 3 轮： $32 \leq i \leq 47$ ，使用函数 H 和下列循环左移位数 s ：

$$s = [4, 11, 16, 23] \quad (5)$$

- 第 4 轮： $48 \leq i \leq 63$ ，使用函数 I 和下列循环左移位数 s ：

$$s = [6, 10, 15, 21] \quad (6)$$

5. 每轮后更新状态变量：

$$\begin{aligned} A &= B + ((A + F(B, C, D) + M[k] + T[i]) \lll s) \\ B &= C \\ C &= D \\ D &= A \end{aligned} \quad (7)$$

(六) 输出最终 MD5 哈希

经过所有块的处理后，将状态变量 A 、 B 、 C 、 D 拼接成最终的 128 位 MD5 哈希值。

四、 实验步骤

(一) 程序总览

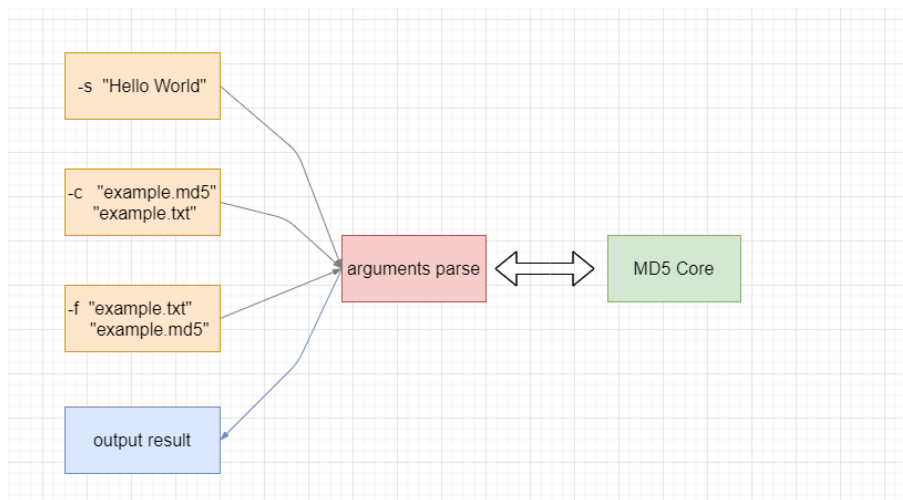


图 1: 程序总览

本次实验流程较为简答，程序提供简单的参数列表，接受用户输入，进而解析命令，调用 MD5 模块核心算法以得到结果输出。

(二) MD5 加密部分

1. 总览

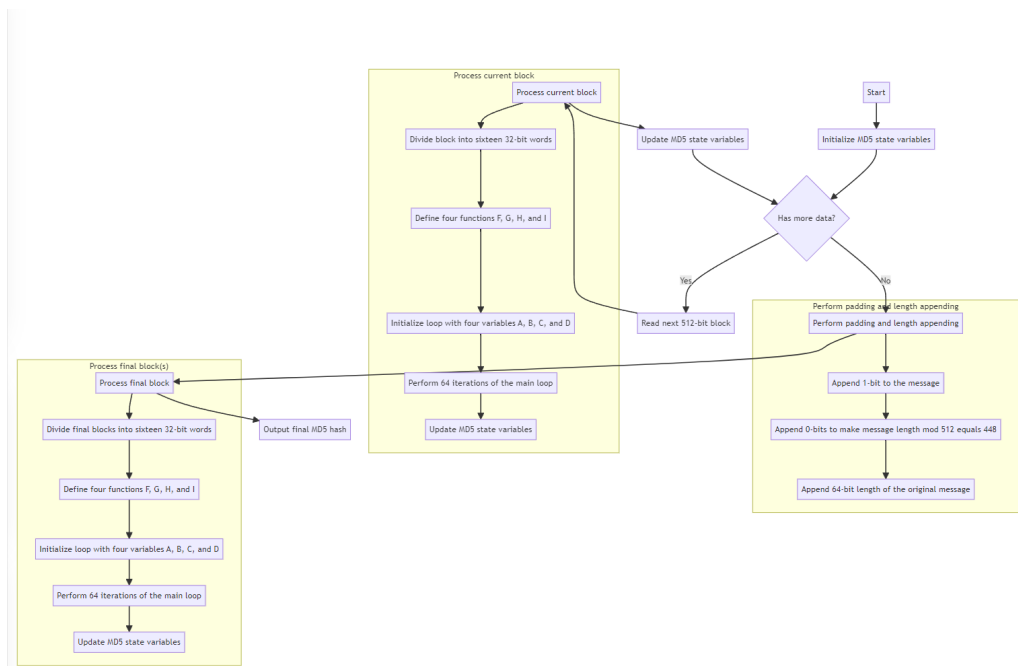


图 2: MD5 flow Chart

MD5 模块提供所有接口如下:

MD5 API

```

1 typedef unsigned int uint;
2 typedef unsigned char BYTE;
3
4 uint F(uint x, uint y, uint z);
5 uint G(uint x, uint y, uint z);
6 uint H(uint x, uint y, uint z);
7 uint I(uint x, uint y, uint z);
8
9 uint RotateLeft(uint x, int s);
10
11 uint FF(uint a, uint b, uint c, uint d, uint x, int s, uint ti);
12 uint GG(uint a, uint b, uint c, uint d, uint x, int s, uint ti);
13 uint HH(uint a, uint b, uint c, uint d, uint x, int s, uint ti);
14 uint II(uint a, uint b, uint c, uint d, uint x, int s, uint ti);
15
16 // For each 512 bits ...
17 void Transform(const BYTE blk[64], uint state[4]);
18 //重置
19 void reset();
20
21 void printState();
22

```

```

23 int getBufferIndex();
24
25 void _Update(const BYTE* stream, size_t len, bool isPrint = false);
26
27 void int2ByteArray(BYTE out[], uint input[], int len);
28 //最后的padding
29 void finalize();
30
31 std::string getMD5AsHex(BYTE digest[16], bool isPrint = false);
32
33 void printMD5AsASCII(uint md5[4]);
34
35 void test();
36 /*
37 input: test text
38 cout: MD5('') = 32 digits HEX's val
39 */
40 std::string Update(const std::string text, bool isPrint=false);
41 /*
42 input: fliePath outputPath(default:nullptr)
43 cout: MD5('') = 32 digits HEX's val
44 */
45 std::string Update(const char* filePath, const char* outputPath, bool isPrint=false);

```

2. 基本函数

上面已经介绍过基本函数，下面展示程序中的基本函数实现
其中，Transform 函数负责对 64 字节数据块的进行 MD5 运算

```

1  uint F(uint x, uint y, uint z) {
2      return (x & y) | ((~x) & z);
3  }
4  uint G(uint x, uint y, uint z) {
5      return (x & z) | (y & (~z));
6  }
7  uint H(uint x, uint y, uint z) {
8      return x^y^z;
9  }
10 uint I(uint x, uint y, uint z) {
11     return y ^ (x | ~z);
12 }
13 uint RotateLeft(uint x, int s) {
14     return (x << s) | (x >> (32 - s));
15 }
16 uint FF(uint a, uint b, uint c, uint d, uint x, int s, uint ti) {
17     return b + RotateLeft((a + F(b, c, d) + x + ti), s);
18 }
19 uint GG(uint a, uint b, uint c, uint d, uint x, int s, uint ti) {
20     return b + RotateLeft((a + G(b, c, d) + x + ti), s);

```

```

21 }
22
23 uint HH(uint a, uint b, uint c, uint d, uint x, int s, uint ti) {
24     return b + RotateLeft((a + H(b, c, d) + x + ti), s);
25 }
26 uint II(uint a, uint b, uint c, uint d, uint x, int s, uint ti) {
27     return b + RotateLeft((a + I(b, c, d) + x + ti), s);
28 }
29 void Transform(const BYTE blk[64], uint state[4]) {
30     uint a = state[0], b = state[1], c = state[2], d = state[3], x[16];
31     ...
32     // 第一次运算FF
33     a = FF(a, b, c, d, x[0], 7, 0xd76aa478L);
34     ...
35
36     state[0] += a;
37     state[1] += b;
38     state[2] += c;
39     state[3] += d;
40 }

```

3. API 调用顺序

MD5 计算流程核心调用是 `_Update()`, `finalize()`, 程序设计考虑到大文件以流式传入计算, 每 512bit 进行 `transform()` 计算

1. 在 MD5 计算前, 首先初始化函数 `reset()`

`blockBuffer` 是一个 512bit 的数组, 因为 md5 算法中每 512bit 为一个 block 为单位做一系列运算
`digest` 存储结果

`state` 数组存储初始的 A、B、C、D

`reset()`

```

1 void reset() {
2     memset(blockBuffer, 0, sizeof(blockBuffer));
3     memset(count, 0, sizeof(count));
4     memset(digest, 0, sizeof(digest));
5     state[0] = origin_a;    state[1] = origin_b;
6     state[2] = origin_c;    state[3] = origin_d;
7
8     //std::cout << "MD5 INIT"<<std::endl;
9 }

```

2. `_Update()`

函数用于更新 MD5 的状态变量, 将新的数据流整合进现有状态 (代码参考了课程报告, 并且在其基础上进行改进, 详细逻辑见下面代码)

`update()`


```

1 void _Update(const BYTE* stream, size_t len, bool isPrint) {
2
3     int ind = getBufferIndex();
4     count[0] += ((uint)len << 3);
5     count[1] += ((uint)len >> 29);
6     if (count[0] < ((uint)len << 3)) //判断是否进位
7         count[1]++;
8
9     int partLen = 64 - ind; //blk 里剩下的
10    if (len < partLen) {
11        //stream加上buffer也不够512
12        memcpy(&blockBuffer[ind], stream, len);
13        //printState();
14        return ;
15    }
16
17    /* 64 一组进行transform*/
18    memcpy(&blockBuffer[ind], stream, partLen);
19    Transform(blockBuffer, state);
20
21    int i = partLen;
22    for (i; i < len; i += 64)
23        Transform(&stream[i], state);
24
25    memcpy(blockBuffer, &stream[i - 64], len + 64 - i);
26    if(isPrint)
27        printState();
28 }

```

3. finalize()

函数是 MD5 算法的最后一步,用于在 MD5 哈希运算的最后阶段,将数据进行填充和附加长度,然后生成最终的 MD5 哈希值

int2ByteArray 和 int2Bytearray 函数分别负责将 int 形数组->BYTE 数组、BYTE 数组->int 数组,这里注意 int 拆成 4 个 BYTE 时, **int 的低位在 BYTE 数组的低位**。

finalize()

```

1 void finallize() {
2
3     /* 以64位二进制表示的填充前的信息长度 */
4     BYTE bit_count[8];
5     int2ByteArray(bit_count, count, 8);
6     /* length of padding: count % 512 = 448 */
7     int indx = getBufferIndex();
8     int padLen = (indx < 56) ? (56 - indx) : (120 - indx);
9
10    _Update(PADDING, padLen);
11    _Update(bit_count, 8);
12 }

```

```

13         int2ByteArray(digest , state , 16);
14     }

```

4. 重载 Update 函数

重载两个 *Update* 函数，其中均调用 *_Update*

其中一个 *Update* 函数接受 *String*，输出该 *String* 的 Md5，另一个 *Update* 函数接受文件地址，输出文件的 MD5。

- *string Update*

1. 将 *string* 类型的 *text* 转为 *BYTE* 类型 (*typedef unsigned char BYTE*，即无符号 *word* 类型)
2. *reset()* 初始化相关参数
3. *_Update()* 和 *finalize* 函数做填充和 md5 计算。
4. *getMD5AsHex()* 函数负责将 *BYTE* 类型的 *digest* 转为 十六进制字符。

```

                                string Update
1  std::string Update(const std::string text, bool isPrint) {
2
3      const int l = text.length();
4      BYTE* tp = new BYTE[l];
5      for (int i = 0; i < l; i++)
6          tp[i] = text[i];
7
8      reset();
9      _Update(tp, l);
10     finalize();
11
12     std::string s = getMD5AsHex(digest);
13     if(isPrint)
14         std::cout << "MD5(" << "\"" << text << "\"'=" << s << std::endl;
15     return s;
16 }

```

- *File Update*

- 打开文件后，通过模拟文件流，每一次读取最多 `sizeof(buffer) = 1024BYTE` 的内容，使用 *_Update()* 函数更新当前的 md5 程序状态。
- 最后输出结果到指定路径

```

                                File Update
1
2  std::string Update(const char* filePath, const char* outPath, bool isPrint) {
3
4      std::ifstream file(filePath, std::ifstream::binary);
5      if (!file) {

```

```
6         std::cerr << "File_could_not_be_opened:" << filePath << std::endl;
7         return "";
8     }
9     reset();
10
11     char buffer[1024]; // Define a buffer to hold file parts
12     BYTE ubuffer[1024]; // BYTE temp buffer
13     while (file.read(buffer, sizeof(buffer)) || file.gcount()) {
14         // 将 char 数组转换为 unsigned char 数组
15         for (size_t i = 0; i < 1024; ++i)
16             ubuffer[i] = static_cast<BYTE>(ubuffer[i]);
17         _Update(ubuffer, file.gcount());
18     }
19
20     finalize();
21     std::string s = getMD5AsHex(digest, false);
22
23     if (isPrint)
24         std::cout << "MD5_of_file:" << filePath << "\"_is_" << s << std::endl;
25
26     if (!outPath)
27         return s;
28
29     // output 文件
30     std::ofstream md5file(outPath);
31     if (md5file) {
32         md5file << s;
33         md5file.close();
34
35         std::cout << ".md5_outpath:" << outPath << "\"<<std::endl;
36     }
37     else {
38         std::cerr << "MD5_file_could_not_be_opened:" << outPath << std::endl;
39     }
40
41     return s;
42 }
```

(三) 命令解析部分

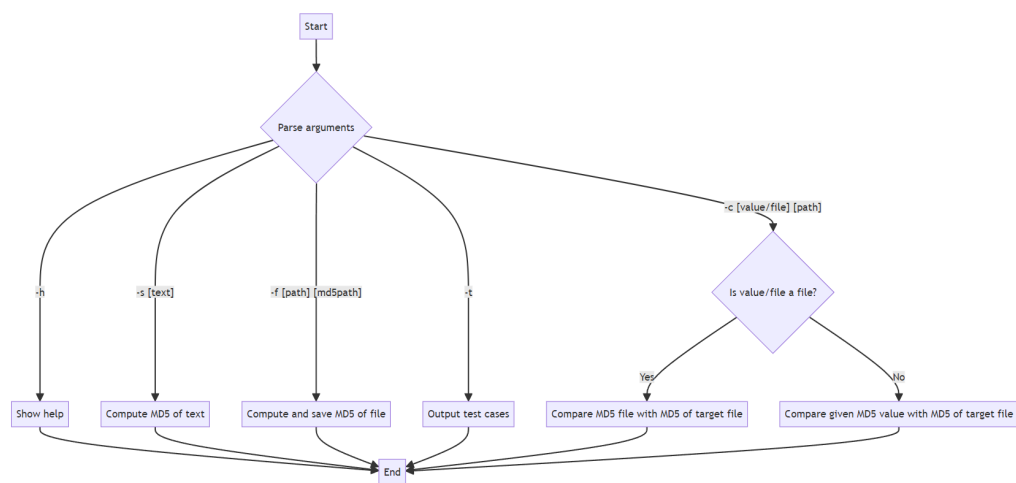


图 3: args func

Command	Description
-h	展示帮助信息, 展示指令用法和可用的命令类型.
-s [text]	计算提供的 MD5 值, 直接输出给定的字符串 MD5 Hash 值 Example:md5.exe -s "hello world"
-f [path] [md5path]	计算 [path] 的 Hash 值并且保存结果到 [md5path]. Example: md5.exe -f input.txt output.md5
-t	输出测试样例. 展示预计算的测试样例和他们的 MD5 值
-c [md5value/file] [path]	比较 MD5 和对应 [path] md5 文件的 Hash 值.
[md5value]	比较给定的 MD5 值和对应 [path] 文件的 MD5. Example: md5.exe -c d41d8cd98f00b204e9800998ecf8427e input.txt
[md5file]	比较 MD5 值和对应文件 [md5file] 的值 [path]. Example: md5.exe -c stored.md5 input.txt

表 1: MD5 Program Commands

args.cpp

```

1 // args.cpp
2 void printHelp();
3 std::string rtrim(const std::string& s);
4 std::string computeMD5(const std::string text);
5 void handleFile(const char* filepath, const char* md5path);
6 void handleCompare(const char* targetPath);
7 void handleCompare(const char* md5OrFilePath, const char* targetPath);
8 void printWatermark();
9 int main(int argc, char* argv[]) {

```

```
10     if (argc < 2) {
11         printWatermark();
12         printHelp();
13         return 1;
14     }
15
16     if (strcmp(argv[1], "-h") == 0) {
17         printWatermark();
18         printHelp();
19     }
20     else if (strcmp(argv[1], "-t") == 0) {
21         printWatermark();
22         test();
23     }
24     else if (strcmp(argv[1], "-s") == 0) {
25         if (argc < 3) {
26             std::cerr << "No_text_provided_for_s_option." << std::endl;
27             return 1;
28         }
29         printWatermark();
30         computeMD5(argv[2]);
31     }
32     else if (strcmp(argv[1], "-f") == 0) {
33         if (argc < 4) {
34             std::cerr << "Insufficient_arguments_for_f_option." << std::endl;
35             return 1;
36         }
37         printWatermark();
38         handleFile(argv[2], argv[3]);
39     }
40     else if (strcmp(argv[1], "-c") == 0) {
41         printWatermark();
42         if (argc == 3) {
43             handleCompare(argv[2]);
44             return 1;
45         }
46         handleCompare(argv[2], argv[3]);
47     }
48     else {
49         std::cerr << "Unknown_option:" << argv[1] << std::endl;
50         printHelp();
51     }
52 }
```

五、 实验结果

(一) md5 测试样例

测试代码:

```

1  std::string text = "a";
2  const int l = text.length();
3  BYTE* tp = new BYTE[l];
4
5  for (int i = 0; i < l; i++)
6      tp[i] = text[i];
7
8  reset();
9  _Update(tp, l);
10 finallize();
11 std::string s = getMD5AsHex(digest);
12
13 std::cout << "MD5(" << "\"" << text << "\"')=" << s << std::endl;

```

```

std::string text = "a";
const int l = text.length();
BYTE* tp = new BYTE[l];

for (int i = 0; i < l; i++)
{
    tp[i] = text[i];
}

reset();
_Update(tp, l);
finallize();
std::string s = getMD5AsHex(digest);

std::cout << "MD5(" << "\"" << text << "\"')=" << s << std::endl;

```

Microsoft Visual Studio 调试控制台

State:
---Count: 512 bits
---state:
-----A b975c10c
-----B a8b6f1c0
-----C e299c331
-----D 61267769
MD5('a')=0cc175b9c0f1b6a831c399e269772661
G:\A大三下\C++\MD5\x64\Debug\MD5.exe (进程 38364) 已退出, 代码为 0。
按任意键关闭此窗口。

图 4: 测试

(二) 完整程序

1. -h 打印 help

./MD5.exe -h

```

PS G:\A大三下\C++\MD5\x64\Debug> .\MD5.exe -h
=====
MD5 Hash Generator
Created by: Fond
=====

Usage: md5.exe [option]
Options:
-h          Show help information
-s [text]   Compute MD5 hash of the provided text
-f [path] [md5path] Compute MD5 hash of the file at [path] and save it to [md5path]
-t          Output test cases
-c [md5value/file] [path] Compare MD5 value or file with MD5 of file at [path]
PS G:\A大三下\C++\MD5\x64\Debug>

```

图 5: -h

2. -t 测试样例

```
./MD5.exe -t
```

```
PS G:\A大三下\C++\MD5\x64\Debug> .\MD5.exe -t
=====
MD5 Hash Generator
Created by: Fond
=====
MD5('')=d41d8cd98f00b204e9800998ecf8427e
MD5('a')=0cc175b9c0f1b6a831c399e269772661
MD5('abc')=900150983cd24fb0d6963f7d28e17f72
MD5('abc')=900150983cd24fb0d6963f7d28e17f72
MD5('Md5 Test example')=506cfd0da24c7e45e7eeeb461d5cf383
MD5('ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789')=7e72a2430880e9bf357110a29d5925a8
MD5('HHHHHH, This is NKU Cyber security')=e657bf90cf7acf035d9c68953c3c9179
PS G:\A大三下\C++\MD5\x64\Debug>
```

图 6: -t

3. -s 计算任意字符串的 Hash

```
./MD5.exe -s "This is a test !! "
```

```
PS G:\A大三下\C++\MD5\x64\Debug> .\MD5.exe -s "This is a test !! "
=====
MD5 Hash Generator
Created by: Fond
=====
MD5('This is a test !! ')=7769da15f9e17d0e867839269a2c341f
PS G:\A大三下\C++\MD5\x64\Debug>
```

图 7: -s

4. -f 计算文件 Hash

```
./MD5.exe -f "G:/A 大三下/Certificate Award for Undergradu.pdf" "G:/A 大三下/text.md5"
```

```
PS G:\A大三下\C++\MD5\x64\Debug> .\MD5.exe -f "G:\A大三下\Certificate Award for Undergradu.pdf" "G:\A大三下\text.md5"
=====
MD5 Hash Generator
Created by: Fond
=====
MD5 of file: 'G:\A大三下\Certificate Award for Undergradu.pdf' is fcddc17b866489332f71d6b6e2b813e2
.md5 outpath: 'G:\A大三下\text.md5'
MD5 of file: 'G:\A大三下\Certificate Award for Undergradu.pdf' is: fcddc17b866489332f71d6b6e2b813e2
PS G:\A大三下\C++\MD5\x64\Debug>
```

图 8: f

5. -c 校验文件

给定.md5 文件校验

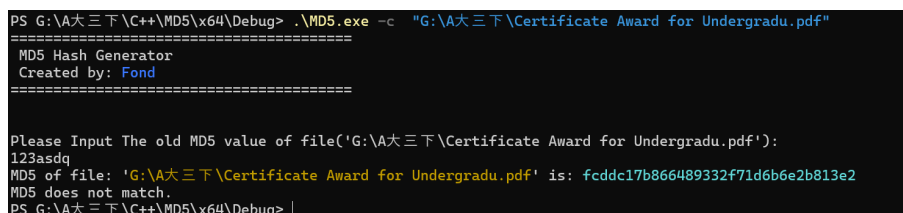
```
./MD5.exe -c "G:/A 大三下/text.md5" "G:/A 大三下/Certificate Award for Undergradu.pdf"
```

```
PS G:\A大三下\C++\MD5\x64\Debug> .\MD5.exe -c "G:\A大三下\text.md5" "G:\A大三下\Certificate Award for Undergradu.pdf"
=====
MD5 Hash Generator
Created by: Fond
=====
The old MD5 value of file('G:\A大三下\Certificate Award for Undergradu.pdf') in ('G:\A大三下\text.md5') is
fcddc17b866489332f71d6b6e2b813e2
MD5 of file: 'G:\A大三下\Certificate Award for Undergradu.pdf' is: fcddc17b866489332f71d6b6e2b813e2
MD5 match successful.
PS G:\A大三下\C++\MD5\x64\Debug>
```

图 9: -c

手动输入 hash, 校验文件

```
./MD5.exe -c "G:/A 大三下/Certificate Award for Undergradu.pdf"
```



```
PS G:\A大三下\C++\MD5\x64\Debug> .\MD5.exe -c "G:\A大三下\Certificate Award for Undergradu.pdf"
=====
MD5 Hash Generator
Created by: Fond
=====

Please Input The old MD5 value of file('G:\A大三下\Certificate Award for Undergradu.pdf'):
123asdq
MD5 of file: 'G:\A大三下\Certificate Award for Undergradu.pdf' is: fcddc17b866489332f71d6b6e2b813e2
MD5 does not match.
PS G:\A大三下\C++\MD5\x64\Debug> |
```

图 10: -c

六、 实验遇到的问题及其解决方法

(一) 数据类型

问题介绍:

使用 `char` 数组记录数据时(临时存储数据的 `buffer`、运算的中间变量), 结果是错误的, 需要 `unsigned char` 才行, 原因是 `char` 类型在 C++ 中是带符号的, 因此引起负值溢出。

解决方案:

使用 `unsigned char` 数组来存储数据, 以避免数据转换和计算中的问题。

```
typedef unsigned int uint;
typedef unsigned char BYTE;
```

(二) 大小端

问题介绍:

MD5 使用小端序进行字节存储。`int` 转为四个 `byte` 时, `int` 的低位需要对应到 `BYTE` 数组的低位:

解决方案:

在将 `int` 转换为字节数组时, 需要确保 `int` 的低位字节对应到 `BYTE` 数组的低位。可以通过手动分配每个字节来实现。详细见函数 `int2ByteArray()` 函数。

(三) 流式传入、计算

问题介绍:

一开始在读课程样例代码中, 看到给的 `Update` 函数时, 看了很久才理解其写法:

MD5 算法针对大文件的处理使用流式输入方式, 逐渐读取部分文件数据, 进一步将这部分数据拆分 512 位的块进行逐块处理, 直到文件读取完毕再调用 `finalize()` 函数进行收尾工作、负责填充数据和附加长度信息, 然后计算最终的 MD5 哈希值。这确保了对任意大小文件的有效处理。

解决方案:

我也是学习了课程提供的样例代码中的相关部分才得知这一点, 并且自己进行编写, 在对应文件的 `Update API` 中, 我依此模拟文件流、实现流式计算 MD5

七、 Linux md5 加密密码

在课程报告中有提到 Linux 中如何在 GRUB 中设置明文密码和 md5 加密密码，然后再将加密字符串保存在配置文件中，就可以进一步提高密码的安全性，因此未授权用户就不能轻易地引导系统，从而获得用户口令。

八、 实验结论

这次实验最重要的部分是理解 MD5 的原理，通过设计和实现一个基于 C++ 的 MD5 算法程序，考虑到各个算法细节实现对任意大小文件、字符串的 MD5 加密。在实验过程中，我们发现使用 *unsigned char* 而非 *char* 来存储数据是确保计算结果正确的关键。通过使用小端序将 *int* 类型转换为字节数组，并在 *_Update()* 函数中以流式处理方式逐块更新状态，最后使用 *finalize()* 函数进行数据填充和附加长度，我们得以实现对大文件的有效校验，验证了程序的正确性和稳定性。此外，程序成功地实现了字符串、文件的 MD5 哈希计算和完整性比较功能，表明它可以作为一个实用的文件完整性检测工具。

收获更大的，是其文件流传入数据、分步更新 MD5 计算状态的思想，在程序设计时，需要考虑各种条件、各种因素。

然而，实验结果也反映出 MD5 算法在文件完整性检测方面的局限性。由于 MD5 容易受到碰撞攻击，我们建议在生产环境中使用更安全的哈希算法（如 SHA-256 或 SHA-512）来确保文件的完整性与可靠性。此外，对于更高级的文件完整性检测需求，可结合 *tripwire* 或 *aide* 等工具，实现对整个文件系统的实时监控。