第5单元　消息摘要

5.1　知识讲解

5.1.1　MD5算法的特点

按照MD5算法生成的消息摘要包含128个二进制位。

任意两组数据经过MD5运算后，生成相同摘要的概率极小。

即使在算法和程序已知的情况下，也无法从MD5摘要中反推出原始数据。

MD5算法的典型应用就是防止数据在传输过程中被篡改：

- 在传输之前利用MD5算法生成数据的消息摘要，而后传输数据
- 对接收到的数据再次利用MD5算法生成消息摘要，若二者完全相同则证明数据未被篡改

Linux系统自带计算和校验MD5摘要的命令行工具md5sum：

- md5sum 原始数据文件 >消息摘要文件
- md5sum -c 消息摘要文件

5.1.2　MD5算法的内容

1. 消息的填充与分割

MD5算法以512位为单位对消息进行分组，每个分组都是一个512位的数据块。为此：

- 首先对原始消息进行填充，填充的方法是在消息的最后填充一位1和若干位0。填充后的消息长度对512取余的结果等于448
- 然后在填充部分的后面追加一个64位的原始消息长度

例如，原始消息的长度为512+512+440=1464位：

```
|<--512-->|<--512-->|<--440-->|<---8--->|<---64-->|
+---------+---------+---------+---------+---------+
|         |         |         | 10...00 |   1464  |
+---------+---------+---------+---------+---------+
                    |<------------512------------>|
```

例如，原始消息的长度为512+512+448=1472位：

```
|<--512-->|<--512-->|<--448-->|<---64-->|<--448-->|<---64-->|
+---------+---------+---------+---------+---------+---------+
|         |         |         | 10...00 | 00...00 |   1472  |
+---------+---------+---------+---------+---------+---------+
                    |<-------512------->|<-------512------->|
```

例如：原始消息的长度为512+512+456=1480位：

```
|<--512-->|<--512-->|<--456-->|<---56-->|<--448-->|<---64-->|
+---------+---------+---------+---------+---------+---------+
|         |         |         | 10...00 | 00...00 |   1480  |
+---------+---------+---------+---------+---------+---------+
                    |<-------512------->|<-------512------->|
```

经过填充后的消息，其长度刚好是512位的整数倍，便于以512位为一组进行分割。

2. 分组的循环运算

1) 初始状态

$$A_0 = 0x67452301$$
$$B_0 = 0xefcdab89$$
$$C_0 = 0x98badcfe$$
$$D_0 = 0x10325476$$

初始状态用于初始化系统的当前状态。

2) 基本运算

$$\overline{X}$$
$$X \wedge Y$$
$$X \vee Y$$
$$X \oplus Y$$
$$X <<< S$$

3) 非线性函数

$$F(a,b,c) = (a \wedge b) \vee (\overline{a} \wedge c)$$
$$G(a,b,c) = (a \wedge c) \vee (b \wedge \overline{c})$$
$$H(a,b,c) = a \oplus b \oplus c$$
$$I(a,b,c) = b \oplus (a \vee \overline{c})$$

4) 循环运算

每个512位的消息分组需要依次经历四轮计算，每计算一个分组即更新一次当前状态。该状态的最终值将构成消息摘要。

每轮每次计算中都会用到一个常量(t)，该值取自$2^{32}|sin(i)|\ (i = 1, 2, \ldots, 64)$的整数部分。

A. 开始

状态向量$[A, B, C, D]$的值取自当前状态。

将一个512位的消息分组划分为16个子分组，每个子分组32位，记作：X_1、X_2、$\ldots$、X_{16}。

B. 第一轮

使用如下表达式计算16次，更新状态向量$[A, B, C, D]$的值：

$$FF(a,b,c,d,x,t,s)\ \{$$
$$a = b + ((a + F(b,c,d) + x + t)\ <<< s)\ \}$$

FF	a	b	c	d	x	t	s
1	A	B	C	D	X_1	0xd76aa478	7
2	D	A	B	C	X_2	0xe8c7b756	12
3	C	D	A	B	X_3	0x242070db	17
4	B	C	D	A	X_4	0xc1bdceee	22
5	A	B	C	D	X_5	0xf57c0faf	7
6	D	A	B	C	X_6	0x4787c62a	12
7	C	D	A	B	X_7	0xa8304613	17

8	B	C	D	A	X_8	0xfd469501	22
9	A	B	C	D	X_9	0x698098d8	7
10	D	A	B	C	X_{10}	0x8b44f7af	12
11	C	D	A	B	X_{11}	0xffff5bb1	17
12	B	C	D	A	X_{12}	0x895cd7be	22
13	A	B	C	D	X_{13}	0x6b901122	7
14	D	A	B	C	X_{14}	0xfd987193	12
15	C	D	A	B	X_{15}	0xa679438e	17
16	B	C	D	A	X_{16}	0x49b40821	22

C. 第二轮

使用如下表达式计算16次，更新状态向量$[A, B, C, D]$的值：

$$GG(a, b, c, d, x, t, s)\ \{$$
$$a = b + ((a + G(b, c, d) + x + t)\ <<<\ s)\ \}$$

GG	a	b	c	d	x	t	s
1	A	B	C	D	X_2	0xf61e2562	5
2	D	A	B	C	X_7	0xc040b340	9
3	C	D	A	B	X_{12}	0x265e5a51	14
4	B	C	D	A	X_1	0xe9b6c7aa	20
5	A	B	C	D	X_6	0xd62f105d	5
6	D	A	B	C	X_{11}	0x02441453	9
7	C	D	A	B	X_{16}	0xd8a1e681	14
8	B	C	D	A	X_5	0xe7d3fbc8	20
9	A	B	C	D	X_{10}	0x21e1cde6	5
10	D	A	B	C	X_{15}	0xc33707d6	9
11	C	D	A	B	X_4	0xf4d50d87	14
12	B	C	D	A	X_9	0x455a14ed	20
13	A	B	C	D	X_{14}	0xa9e3e905	5
14	D	A	B	C	X_3	0xfcefa3f8	9
15	C	D	A	B	X_8	0x676f02d9	14
16	B	C	D	A	X_{13}	0x8d2a4c8a	20

D. 第三轮

使用如下表达式计算16次，更新状态向量$[A, B, C, D]$的值：

$$HH(a,b,c,d,x,t,s)\ \{$$
$$a = b + ((a + H(b,c,d) + x + t)\ <<<\ s)\ \}$$

HH	a	b	c	d	x	t	s
1	A	B	C	D	X_6	0xfffa3942	4
2	D	A	B	C	X_9	0x8771f681	11
3	C	D	A	B	X_{12}	0x6d9d6122	16
4	B	C	D	A	X_{15}	0xfde5380c	23
5	A	B	C	D	X_2	0xa4beea44	4
6	D	A	B	C	X_5	0x4bdecfa9	11
7	C	D	A	B	X_8	0xf6bb4b60	16
8	B	C	D	A	X_{11}	0xbebfbc70	23
9	A	B	C	D	X_{14}	0x289b7ec6	4
10	D	A	B	C	X_1	0xeaa127fa	11
11	C	D	A	B	X_4	0xd4ef3085	16
12	B	C	D	A	X_7	0x04881d05	23
13	A	B	C	D	X_{10}	0xd9d4d039	4
14	D	A	B	C	X_{13}	0xe6db99e5	11
15	C	D	A	B	X_{16}	0x1fa27cf8	16
16	B	C	D	A	X_3	0xc4ac5665	23

E. 第四轮

使用如下表达式计算16次，更新状态向量$[A, B, C, D]$的值：

$$II(a,b,c,d,x,t,s)\ \{$$
$$a = b + ((a + I(b,c,d) + x + t)\ <<<\ s)\ \}$$

II	a	b	c	d	x	t	s
1	A	B	C	D	X_1	0xf4292244	6
2	D	A	B	C	X_8	0x432aff97	10
3	C	D	A	B	X_{15}	0xab9423a7	15
4	B	C	D	A	X_6	0xfc93a039	21
5	A	B	C	D	X_{13}	0x655b59c3	6
6	D	A	B	C	X_4	0x8f0ccc92	10
7	C	D	A	B	X_{11}	0xffeff47d	15
8	B	C	D	A	X_2	0x85845dd1	21
9	A	B	C	D	X_9	0x6fa87e4f	6

10	D	A	B	C	X_{16}	0xfe2ce6e0	10
11	C	D	A	B	X_{7}	0xa3014314	15
12	B	C	D	A	X_{14}	0x4e0811a1	21
13	A	B	C	D	X_{5}	0xf7537e82	6
14	D	A	B	C	X_{12}	0xbd3af235	10
15	C	D	A	B	X_{3}	0x2ad7d2bb	15
16	B	C	D	A	X_{10}	0xeb86d391	21

F. 结束
将状态向量$[A, B, C, D]$的值累加进当前状态。

3. 消息摘要的生成

MD5算法针对消息中的每个512位分组循环计算，每计算一个分组更新一次当前状态，直至计算完最后一个分组。这时只要将当前状态中的四个分量A、B、C、D按照从低字节到高字节的顺序拼接成一个128位的消息摘要即可。

5.2 实训案例

5.2.1 基于MD5算法的文件摘要

在Linux平台上编写应用程序，正确实现MD5算法。

程序不仅能够为任意长度的字符串生成MD5摘要，而且可以为任意大小的文件生成MD5摘要。

程序还可以利用MD5摘要验证文件的完整性：

- 程序用计算得出的测试文件摘要和手动输入的原始文件摘要，进行一致性比对
- 程序用计算得出的测试文件摘要和md5sum命令输出的测试文件摘要，进行一致性比对

5.2.2 程序清单

1. 声明Md5类

```
// md5.h
// 声明Md5类

#pragma once

#include <stdint.h>
#include <string>
using namespace std;

// 基于MD5算法的消息摘要
class Md5 {
public:
    // 构造函数
    Md5(void);

    // 开始
    void begin(void);
    // 追加
    void append(const void* buf, size_t len);
    void append(const char* str);
    // 结束
    string end(void);

    // 文本摘要
    string text(const char* text);
    // 文件摘要
    string file(const char* file);

private:
    // 循环左移位
    uint32_t LS(uint32_t x, uint32_t s) const;

    // 非线性函数
    uint32_t F(uint32_t a, uint32_t b, uint32_t c) const;
    uint32_t G(uint32_t a, uint32_t b, uint32_t c) const;
    uint32_t H(uint32_t a, uint32_t b, uint32_t c) const;
    uint32_t I(uint32_t a, uint32_t b, uint32_t c) const;

    // 表达式
    void FF(uint32_t* a, uint32_t b, uint32_t c, uint32_t d,
        uint32_t x, uint32_t t, uint32_t s) const;
    void GG(uint32_t* a, uint32_t b, uint32_t c, uint32_t d,
        uint32_t x, uint32_t t, uint32_t s) const;
    void HH(uint32_t* a, uint32_t b, uint32_t c, uint32_t d,
        uint32_t x, uint32_t t, uint32_t s) const;
    void II(uint32_t* a, uint32_t b, uint32_t c, uint32_t d,
        uint32_t x, uint32_t t, uint32_t s) const;

    // 更新状态
    void update(void);

    static const uint8_t  padding[64]; // 填充
    static const uint32_t s[4][4];     // 移位

    uint32_t t[64];      // 常量
    uint64_t nbits;      // 输入位数
    uint32_t state[4];   // 当前状态
    uint8_t  packet[64]; // 消息分组
};
```

2. 实现Md5类

```cpp
// md5.cpp
// 实现Md5类

#include <math.h>
#include <string.h>
#include <sstream>
#include <fstream>
#include <iomanip>
using namespace std;

#include "md5.h"

// 填充
const uint8_t Md5::padding[64] = {0x80};

// 移位
const uint32_t Md5::s[4][4] = {
    {7, 12, 17, 22}, {5,  9, 14, 20}, {4, 11, 16, 23}, {6, 10, 15, 21}};

// 构造函数
Md5::Md5(void) {
    double d = (uint64_t)1 << 32; // 2^32
    for(int i = 0; i < 64; ++i)
        t[i] = d * fabs(sin(i + 1));
}

// 开始
void Md5::begin(void) {
    nbits = 0;

    // 用初始状态初始化当前状态
    state[0] = 0x67452301;
    state[1] = 0xefcdab89;
    state[2] = 0x98badcfe;
    state[3] = 0x10325476;
}

// 追加
void Md5::append(const void* buf, size_t len) {
    // 输入字节数对64取余即消息分组中的字节数
    size_t i = nbits >> 3 & 0x3f;

    // 输入位数累加
    nbits += len << 3;

    // 对消息缓冲区中的每个字节...
    for (const uint8_t* byte = (const uint8_t*)buf; len; --len) {
        // 填入消息分组
        packet[i++] = *byte++;

        // 若消息分组满
        if (i == 64) {
            // 更新状态
            update();
            // 清空消息分组
            i = 0;
        }
    }
}

void Md5::append(const char* str) {
    if (str)
        append(str, strlen(str));
}

// 结束
string Md5::end(void) {
    // 输入字节数对64取余即消息分组中的字节数
    size_t i = nbits >> 3 & 0x3f;
```

```
    uint64_t bits = nbits;
    /*
       追加填充块和长度块

       +-------+------------+-------+
       | XXXXX |   PADDING  |  BITS |
       +-------+------------+-------+
       |<- i ->|<- padlen ->|       |
       |<------- 56 ------->|<- 8 ->|
       |<----------- 64 ----------->|

       +------------------------+------------------------+-------+
       | XXXXXXXXXXXXXXXXXXXXXX |         PADDING        |  BITS |
       +------------------------+------------------------+-------+
       |<--------- i ---------->|<------- padlen ------->|       |
       |<--------------------- 120 --------------------->|<- 8 ->|
       |<------------------------ 128 ------------------------->|
    */
    size_t padlen = i < 56 ? 56 - i : 120 - i;
    append(padding, padlen);
    append(&bits, 8);

    // 格式化消息摘要(128位即16字节)为16进制字符串(32个字符)
    ostringstream oss;
    oss << hex << setfill('0');
    for (i = 0; i < 16; ++i)
        oss << setw(2) << (unsigned int)((uint8_t*)state)[i];
    return oss.str();
}

// 文本摘要
string Md5::text(const char* text) {
    begin();
    append(text);
    return end();
}

// 文件摘要
string Md5::file(const char* file) {
    ifstream ifs (file, ios::binary);
    if (! ifs)
        return "";

    begin();

    char buf[1024];
    while (ifs.read(buf, sizeof(buf)))
        append(buf, sizeof(buf));

    if (! ifs.eof()) {
        ifs.close();
        return "";
    }

    append(buf, ifs.gcount());

    ifs.close();
    return end();
}

// 循环左移位
uint32_t Md5::LS(uint32_t x, uint32_t s) const {
    return x << s | x >> (32 - s);
}
//
// 非线性函数
//
uint32_t Md5::F(uint32_t a, uint32_t b, uint32_t c) const {
    return (a & b) | (~a & c);
}
```

```
uint32_t Md5::G(uint32_t a, uint32_t b, uint32_t c) const {
    return (a & c) | (b & ~c);
}

uint32_t Md5::H(uint32_t a, uint32_t b, uint32_t c) const {
    return a ^ b ^ c;
}

uint32_t Md5::I(uint32_t a, uint32_t b, uint32_t c) const {
    return b ^ (a | ~c);
}
//
// 表达式
//
void Md5::FF(uint32_t* a, uint32_t b, uint32_t c, uint32_t d,
    uint32_t x, uint32_t t, uint32_t s) const {
    *a = b + LS(*a + F(b, c, d) + x + t, s);
}

void Md5::GG(uint32_t* a, uint32_t b, uint32_t c, uint32_t d,
    uint32_t x, uint32_t t, uint32_t s) const {
    *a = b + LS(*a + G(b, c, d) + x + t, s);
}

void Md5::HH(uint32_t* a, uint32_t b, uint32_t c, uint32_t d,
    uint32_t x, uint32_t t, uint32_t s) const {
    *a = b + LS(*a + H(b, c, d) + x + t, s);
}

void Md5::II(uint32_t* a, uint32_t b, uint32_t c, uint32_t d,
    uint32_t x, uint32_t t, uint32_t s) const {
    *a = b + LS(*a + I(b, c, d) + x + t, s);
}

// 更新状态
void  Md5::update(void) {
    // 状态向量的值取自当前状态
    uint32_t a = state[0];
    uint32_t b = state[1];
    uint32_t c = state[2];
    uint32_t d = state[3];

    // 将一个512位的消息分组划分为16个子分组，每个子分组32位
    uint32_t* x = (uint32_t*)packet;

    // 第一轮
    FF(&a, b, c, d, x[ 0], t[ 0], s[0][0]);
    FF(&d, a, b, c, x[ 1], t[ 1], s[0][1]);
    FF(&c, d, a, b, x[ 2], t[ 2], s[0][2]);
    FF(&b, c, d, a, x[ 3], t[ 3], s[0][3]);
    FF(&a, b, c, d, x[ 4], t[ 4], s[0][0]);
    FF(&d, a, b, c, x[ 5], t[ 5], s[0][1]);
    FF(&c, d, a, b, x[ 6], t[ 6], s[0][2]);
    FF(&b, c, d, a, x[ 7], t[ 7], s[0][3]);
    FF(&a, b, c, d, x[ 8], t[ 8], s[0][0]);
    FF(&d, a, b, c, x[ 9], t[ 9], s[0][1]);
    FF(&c, d, a, b, x[10], t[10], s[0][2]);
    FF(&b, c, d, a, x[11], t[11], s[0][3]);
    FF(&a, b, c, d, x[12], t[12], s[0][0]);
    FF(&d, a, b, c, x[13], t[13], s[0][1]);
    FF(&c, d, a, b, x[14], t[14], s[0][2]);
    FF(&b, c, d, a, x[15], t[15], s[0][3]);

    // 第二轮
    GG(&a, b, c, d, x[ 1], t[16], s[1][0]);
    GG(&d, a, b, c, x[ 6], t[17], s[1][1]);
    GG(&c, d, a, b, x[11], t[18], s[1][2]);
    GG(&b, c, d, a, x[ 0], t[19], s[1][3]);
    GG(&a, b, c, d, x[ 5], t[20], s[1][0]);
```

```
    GG(&d, a, b, c, x[10], t[21], s[1][1]);
    GG(&c, d, a, b, x[15], t[22], s[1][2]);
    GG(&b, c, d, a, x[ 4], t[23], s[1][3]);
    GG(&a, b, c, d, x[ 9], t[24], s[1][0]);
    GG(&d, a, b, c, x[14], t[25], s[1][1]);
    GG(&c, d, a, b, x[ 3], t[26], s[1][2]);
    GG(&b, c, d, a, x[ 8], t[27], s[1][3]);
    GG(&a, b, c, d, x[13], t[28], s[1][0]);
    GG(&d, a, b, c, x[ 2], t[29], s[1][1]);
    GG(&c, d, a, b, x[ 7], t[30], s[1][2]);
    GG(&b, c, d, a, x[12], t[31], s[1][3]);

    // 第三轮
    HH(&a, b, c, d, x[ 5], t[32], s[2][0]);
    HH(&d, a, b, c, x[ 8], t[33], s[2][1]);
    HH(&c, d, a, b, x[11], t[34], s[2][2]);
    HH(&b, c, d, a, x[14], t[35], s[2][3]);
    HH(&a, b, c, d, x[ 1], t[36], s[2][0]);
    HH(&d, a, b, c, x[ 4], t[37], s[2][1]);
    HH(&c, d, a, b, x[ 7], t[38], s[2][2]);
    HH(&b, c, d, a, x[10], t[39], s[2][3]);
    HH(&a, b, c, d, x[13], t[40], s[2][0]);
    HH(&d, a, b, c, x[ 0], t[41], s[2][1]);
    HH(&c, d, a, b, x[ 3], t[42], s[2][2]);
    HH(&b, c, d, a, x[ 6], t[43], s[2][3]);
    HH(&a, b, c, d, x[ 9], t[44], s[2][0]);
    HH(&d, a, b, c, x[12], t[45], s[2][1]);
    HH(&c, d, a, b, x[15], t[46], s[2][2]);
    HH(&b, c, d, a, x[ 2], t[47], s[2][3]);

    // 第四轮
    II(&a, b, c, d, x[ 0], t[48], s[3][0]);
    II(&d, a, b, c, x[ 7], t[49], s[3][1]);
    II(&c, d, a, b, x[14], t[50], s[3][2]);
    II(&b, c, d, a, x[ 5], t[51], s[3][3]);
    II(&a, b, c, d, x[12], t[52], s[3][0]);
    II(&d, a, b, c, x[ 3], t[53], s[3][1]);
    II(&c, d, a, b, x[10], t[54], s[3][2]);
    II(&b, c, d, a, x[ 1], t[55], s[3][3]);
    II(&a, b, c, d, x[ 8], t[56], s[3][0]);
    II(&d, a, b, c, x[15], t[57], s[3][1]);
    II(&c, d, a, b, x[ 6], t[58], s[3][2]);
    II(&b, c, d, a, x[13], t[59], s[3][3]);
    II(&a, b, c, d, x[ 4], t[60], s[3][0]);
    II(&d, a, b, c, x[11], t[61], s[3][1]);
    II(&c, d, a, b, x[ 2], t[62], s[3][2]);
    II(&b, c, d, a, x[ 9], t[63], s[3][3]);

    // 将状态向量的值累加进当前状态
    state[0] += a;
    state[1] += b;
    state[2] += c;
    state[3] += d;
}
```

3. 测试Md5类

```cpp
// md5_test.cpp
// 测试Md5类

#include <stdlib.h>
#include <string.h>
#include <iostream>
using namespace std;

#include "md5.h"

int main(int argc, char* argv[]) {
    if (argc < 3)
        goto escape;

    if (! strcmp(argv[1], "-t")) { // 文本摘要
        Md5 md5;
        cout << md5.text(argv[2]) << endl;
    }
    else if (! strcmp(argv[1], "-f")) { // 文件摘要
        Md5 md5;
        cout << md5.file(argv[2]) << endl;
    }
    else
        goto escape;

    return EXIT_SUCCESS;

escape:
        cerr << "Usage: " << argv[0] << " -t <text>" << endl;
        cerr << "Usage: " << argv[0] << " -f <file>" << endl;
        return EXIT_FAILURE;
}
```

4. 测试Md5类构建脚本

```
# md5_test.mak
# 测试Md5类构建脚本

PROJ   = md5_test
OBJS   = md5_test.o md5.o
CXX    = g++
LINK   = g++
RM     = rm -rf
CFLAGS = -c -g -Wall -I.

$(PROJ): $(OBJS)
    $(LINK) $^ -o $@

.cpp.o:
    $(CXX) $(CFLAGS) $^

clean:
    $(RM) $(PROJ) $(OBJS)
```

5.3 扩展提高

5.3.1 Linux口令与MD5算法

1. 口令文件：/etc/passwd

早期Linux系统将用户的登录口令加密后保存在口令文件/etc/passwd文件中。该文件中的每一行对应一个用户，并用冒号分隔为7个字段，其中第二个字段即为加密后的用户口令。

一般情况下，口令文件/etc/passwd允许所有用户读取，但只允许root用户写入。攻击者可以轻易读取任何用户的口令密文，然后通过逆向破解获取其明文，以该用户的身份登录系统。

2. 影子文件：/etc/shadow

现代版本的Linux系统在这方面做了更安全的改进。口令文件/etc/passwd中每一行的第二个字段不再存放加密后的用户口令，而是一个表示该用户是否有口令的标志字符“x”，而真正的口令保存在另一个仅允许root用户访问的影子文件中。

与口令文件/etc/passwd类似，影子文件/etc/shadow中也是一行对应一个用户，并用冒号分隔为9个字段，但其中第二个字段所存放的并非加密后的用户口令，而是用户口令的MD5摘要。

登录验证时，根据用户输入的口令计算其MD5摘要，与影子文件中与该用户对应的口令摘要进行比较，二者完全一致即表明输入的口令正确，允许登录，否则拒绝其登录系统。

因为影子文件中保存的仅仅是用户口令的MD5摘要，而摘要算法本身保证了其不可能被反推出原始数据。因此即使攻击者获得了影子文件中的信息，也无法反解出用户的真实口令。

5.3.2　字典攻击与MD5变换算法

1. 字典攻击

MD5算法本身的不可逆性保证了不可能通过数学方法从消息摘要反推原始消息。因此攻击者通常会使用字典攻击的手段破解MD5算法产生的消息摘要。

所谓字典攻击，就是事先收集大量原始消息及其MD5摘要，保存在数据库中，逐个与待破解消息摘要进行比对，直至找到与之匹配的记录，相同的摘要必源自相同的消息。

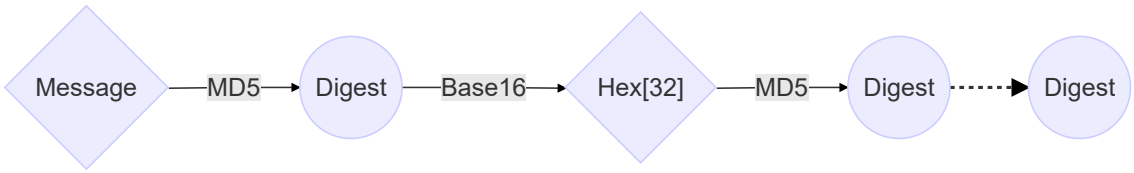
目前收集MD5字典的网站有很多，比如：

- https://hashtoolkit.com
- http://www.xmd5.org

2. MD5变换算法

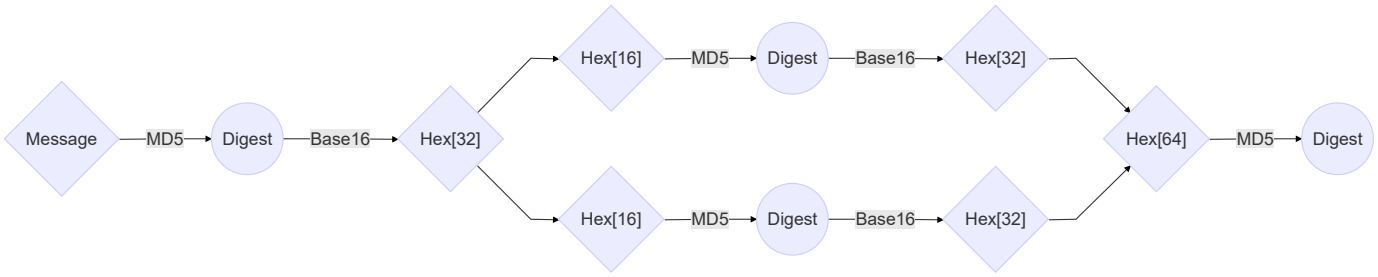
1) 多重摘要

根据原始消息生成MD5摘要以后，再用同样的算法生成摘要的摘要，甚至摘要的摘要的摘要……，以此躲避基于常规MD5算法的字典攻击。如下图所示：



2) 拆分合并

先由原始消息得到MD5摘要(128位即16字节)，将其表示为包含32个字符的十六进制字符串(Base16)。将该字符串拆分成两个子串，每个子串包含16个字符。分别对两个子串计算MD5摘要，得到两个各包含32个字符的十六进制字符串(Base16)。将这两个字符串合并为一个包含64个字符的字符串，最后再进行一次MD5计算，得到最终摘要。如下图所示：


Message
MD5
Digest
Base16
Hex[32]
Hex[16]
MD5
Digest
Base16
Hex[32]
Hex[16]
MD5
Digest
Base16
Hex[32]
Hex[64]
MD5
Digest

达内集团◇C++教研部◇闵卫