

密码学实验四实验报告

姓名：孟衍璋 学号：16337183

一、实验目的

实现安全 hash 算法。

二、实验内容

将自己的学号进行 hash 并给出结果。

三、实验及算法原理

本次实验需要实现 SHA-1 算法，这是一个具有 160 比特消息摘要的迭代 hash 函数。SHA-1 建立在对比特串的面向字的操作，每一个字由 32 比特（或者 8 个 16 进制数）组成。SHA-1 用到了 6 个面向字的操作，分别如下：

X 和 Y 的逻辑“和”：

```
string AND(string a, string b)
{
    string ans;
    for (unsigned int i = 0; i < a.size(); i++)
    {
        if (a[i] == '1' && b[i] == '1')
            ans += '1';
        else
            ans += '0';
    }
    return ans;
}
```

X 和 Y 的逻辑“或”：

```

string OR(string a, string b)
{
    string ans;
    for (unsigned int i = 0; i < a.size(); i++)
    {
        if (a[i] == '0' && b[i] == '0')
            ans += '0';
        else
            ans += '1';
    }
    return ans;
}

```

X 和 Y 的逻辑“异或”:

```

string XOR(string a, string b)
{
    string ans;
    for (unsigned int i = 0; i < a.size(); i++)
    {
        if (a[i] == b[i])
            ans += '0';
        else
            ans += '1';
    }
    return ans;
}

```

X 的逻辑“补”:

```

string NOT(string a)
{
    string ans;
    for (unsigned int i = 0; i < a.size(); i++)
    {
        if (a[i] == '0')
            ans += '1';
        else
            ans += '0';
    }
    return ans;
}

```

模 2^{32} 整数加:

```

string ADD_mod32(string a, string b)
{
    char carry = '0';
    string ans;
    for (int i = a.size() - 1; i >= 0; i--)
    {
        int count = 0;
        if (a[i] == '1')
            count++;
        if (b[i] == '1')
            count++;
        if (carry == '1')
            count++;
        switch (count)
        {
            case 0:
                ans += '0';
                carry = '0';
                continue;
            case 1:
                ans += '1';
                carry = '0';
                continue;
            case 2:
                ans += '0';
                carry = '1';
                continue;
            case 3:
                ans += '1';
                carry = '1';
                continue;
        }
    }
    reverse(ans.begin(), ans.end());
    return ans;
}

```

X 循环左移 s 个位置 ($0 \leq s \leq 31$):

```

string ROTL(string a, int i)
{
    string ans;
    string start = a.substr(0, i);
    string last = a.substr(i, 32 - i);
    ans = last;
    ans += start;
    return ans;
}

```

实现了这些面向字的操作之后，接下来要考虑的就是 SHA-1 所用的

填充算法，代码实现如下：

```
string SHA_1_PAD(string x)
{
    int d = (447 - x.size()) % 512;
    string l = DecToBin(x.size());
    while (l.size() < 64)
        l.insert(0, "0");
    string y = x;
    y += '1';
    for (int i = 0; i < d; ++i)
        y += '0';
    y += l;
    return y;
}
```

其中用到的将十进制数转换为二进制数的函数如下：

```
string DecToBin(int a)
{
    string a_bin;
    for (int i = 0; a > 0; i++)
    {
        a_bin += to_string(a % 2);
        a /= 2;
    }
    reverse(a_bin.begin(), a_bin.end());
    return a_bin;
}
```

接下来还要实现如下函数：

按如下方式定义 f_0, \dots, f_{79} ：

$$f_t(B, C, D) = \begin{cases} (B \wedge C) \vee ((\neg B) \wedge D) & 0 \leq t \leq 19 \\ B \oplus C \oplus D & 20 \leq t \leq 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & 40 \leq t \leq 59 \\ B \oplus C \oplus D & 60 \leq t \leq 79 \end{cases}$$

代码实现如下：

```

string f0_19(string B, string C, string D)
{
    string ans;
    string temp1 = AND(B, C);
    string temp2 = NOT(B);
    string temp3 = AND(temp2, D);
    ans = OR(temp1, temp3);
    return ans;
}

string f20_39(string B, string C, string D)
{
    string ans;
    string temp1 = XOR(B, C);
    ans = XOR(temp1, D);
    return ans;
}

string f40_59(string B, string C, string D)
{
    string ans;
    string temp1 = AND(B, C);
    string temp2 = AND(B, D);
    string temp3 = AND(C, D);
    string temp4 = OR(temp1, temp2);
    ans = OR(temp4, temp3);
    return ans;
}

string f60_79(string B, string C, string D)
{
    string ans;
    string temp1 = XOR(B, C);
    ans = XOR(temp1, D);
    return ans;
}

```

实现了上述函数之后，便可以开始依次按照 SHA-1 算法的步骤实现加密。

首先，将明文转换为二进制形式，其函数实现代码如下：

```

string message_bin(char a)
{
    int temp = int(a);
    string ans;
    while (temp != 0)
    {
        int m = temp % 2;
        ans = char(m + '0') + ans;
        temp /= 2;
    }
    int len = ans.length();
    for (int j = 0; j < 8 - len; j++) {
        ans = '0' + ans;
    }
    return ans;
}

```

将转换为二进制后的明文串进行填充，将所得到的 y 分为 512 比特一组：

```

string x;
for (unsigned int i = 0; i < message.size(); i++)
    x += message_bin(message[i]);

string y = SHA_1_PAD(x); //y为512比特

int group_size = y.size() / 512;
string M[1000];
for (int i = 0; i < group_size; ++i)
{
    M[i] = y.substr(i*512,512);
}

```

定义常数字符串，因为函数的接口都是字符串的二进制表示，所以这里需要将十六进制表示转化为二进制表示的函数，实现代码如下：

```
string HexToBin_assist(char ch)
{
    string ans;
    switch (ch)
    {
        case '0':
            ans = "0000";
            break;
        case '1':
            ans = "0001";
            break;
        case '2':
            ans = "0010";
            break;
        case '3':
            ans = "0011";
            break;
        case '4':
            ans = "0100";
            break;
        case '5':
            ans = "0101";
            break;
        case '6':
            ans = "0110";
            break;
        case '7':
            ans = "0111";
            break;
        case '8':
            ans = "1000";
            break;
    }
}
```

```

    case '9':
        ans = "1001";
        break;
    case 'A':
        ans = "1010";
        break;
    case 'B':
        ans = "1011";
        break;
    case 'C':
        ans = "1100";
        break;
    case 'D':
        ans = "1101";
        break;
    case 'E':
        ans = "1110";
        break;
    case 'F':
        ans = "1111";
        break;
    }
    return ans;
}

```

```

string HexToBin(string hex)
{
    string bin;
    for (unsigned int i = 0; i < hex.size(); i++)
    {
        bin += HexToBin_assist(hex[i]);
    }
    return bin;
}

```

定义常数字符串：

```

string K0_19 = HexToBin("5A827999");
string K20_39 = HexToBin("6ED9EBA1");
string K40_59 = HexToBin("8F1BBCDC");
string K60_79 = HexToBin("CA62C1D6");

```

```

string H0 = HexToBin("67452301");
string H1 = HexToBin("EFCDA89");
string H2 = HexToBin("98BADCFE");
string H3 = HexToBin("10325476");
string H4 = HexToBin("C3D2E1F0");



```


之后按照下列步骤使用之前定义的操作函数进行实现：

```

for  $i \leftarrow 1$  to  $n$ 
    令  $M_i = W_0 \parallel W_1 \parallel \dots \parallel W_{15}$ , 其中每个  $W_i$  是一个字
    for  $t \leftarrow 16$  to 79
        do  $W_t \leftarrow \text{ROTL}^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})$ 
         $A \leftarrow H_0$ 
         $B \leftarrow H_1$ 
         $C \leftarrow H_2$ 
         $D \leftarrow H_3$ 
         $E \leftarrow H_4$ 
        for  $t \leftarrow 0$  to 79
            do {
                 $\text{temp} \leftarrow \text{ROTL}^5(A) + f_t(B, C, D) + E + W_t + K_t$ 
                 $E \leftarrow D$ 
                 $D \leftarrow C$ 
                do {
                     $C \leftarrow \text{ROTL}^{30}(B)$ 
                     $B \leftarrow A$ 
                }
                 $A \leftarrow \text{temp}$ 
                 $H_0 \leftarrow H_0 + A$ 
                 $H_1 \leftarrow H_1 + B$ 
                 $H_2 \leftarrow H_2 + C$ 
                 $H_3 \leftarrow H_3 + D$ 
                 $H_4 \leftarrow H_4 + E$ 
            }
    
```

四、 实验代码

 sha1_operation.h	2018/10/29 9:28	C Header File	5 KB
 sha1.cpp	2018/10/29 10:08	CPP 文件	3 KB

五、 运行截图

```

The hash value is:
d43cdb31c1438520beb2e86eeb77eb255822df17
请按任意键继续. . .

```

六、 结果分析

16337183

散列/哈希算法：

SHA1

SHA224

SHA256

SHA384

SHA512

MD5

HmacSHA1

HmacSHA224

HmacSHA256

HmacSHA384

HmacSHA512

HmacMD5

PBKDF2

哈希值

d43cdb31c1438520beb2e86eeb77eb255822df17

在线解密工具运行之后所得的结果与我的结果相符。

七、总结

这次实验是实现 SHA-1，只需要参考书上的步骤，按照步骤来便可以依次实现。相比较之下比上一次实验 AES 的实现要简单许多，步骤没有这么繁杂，而且有了上一次实验的经验，很多之前踩过的坑也都避开了，不过还是有很多地方需要注意。

在做这个实验的过程中，首先将需要实现的函数列出来，对于 SHA-1，主要是那 6 个面向字的操作的函数，填充算法，还有在压缩过程中的函数 f。只要理清了思路，便可以以较快的速度实现出来。

本次实验最开始刚写完可以运行的代码之后，运行的结果与网站上验证的结果并不相符，一定是在之前的函数有哪一步有些小问题，经过检查之后发现是在实现模 2^{32} 整数加的函数中，将进位置为 0 的情况没有考虑进去，只写了进位置为 1 的情况，所以出现了错误。修改之后所得结果便是正确的。

总而言之，这次实验让我清楚了 SHA-1 的实现过程，对 hash 函数有了更深的理解。收获还是蛮丰富的。