南昌大学实验报告

学生姓名：丁俊　　学　　号：8003119100　专业班级：信息安全 193 班
实验类型：□ 验证 □ 综合 □ 设计 □ 创新　实验日期：2021.12.17　实验成绩：

一、实验项目名称

MD5 算法的实现

二、实验目的

1、掌握 MD5 加密散列算法的原理
2、用代码实现 MD5 加密过程

三、实验基本原理

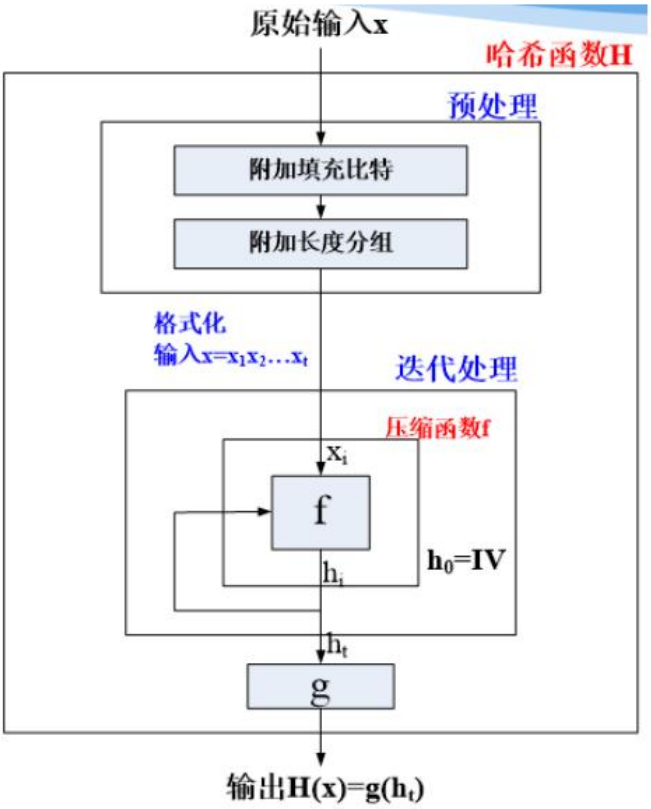
算法流程

1、附加位填充

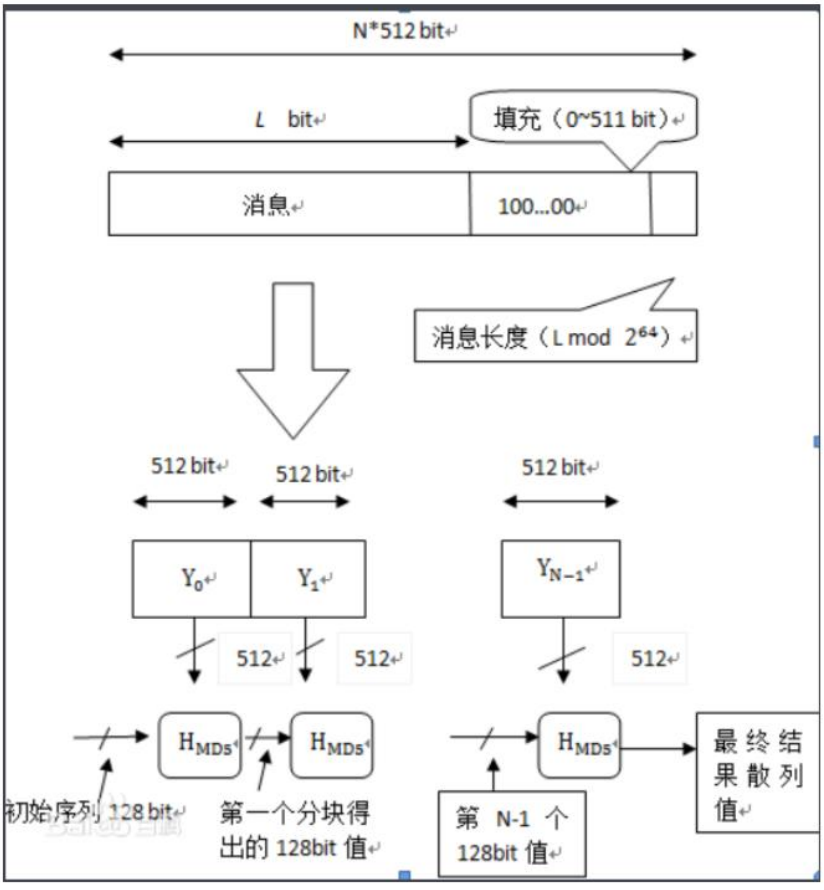
2、初始化链接变量

3、分组处理

4、步函数的运算



注：g通常是恒等映射$g(h_t)=h_t$



四、主要仪器设备及耗材

Window10、pycharm

五、实验步骤

定义一个 MD5 类存储初始向量和对于的 F、G、H、I 函数，以及对应的 T 序列和循环左移的位数。

1、初始化链接向量

使用 4 个 32 位的寄存器 A， B，C， D 存放 4 个固定的 32 位整型参数，用于第一轮迭代，这里需要注意，书本上的值是直接给你的，但是没有倒过来，也就是大端和小端的转换问题。

2、分组处理

与分组密码分组处理相似，有 4 轮步骤，将 512 比特的消息分组平均分为 16 个子分组，每个子分组有 32 比特，参与每一轮的的 16 步运算，每步输入是 4 个 32 比特的链接变量和一个 32 位的的消息子分组，经过这样的 64 步之后得到 4 个寄存器的值分别与输入的链接变量进行模加，关键代码如下，为了能够保存一下一开始 A，B，C，D 这四个初始变量的值，所以就先找四个变量把他们的值暂存一下，为最后一步的模加做准备。

完整代码：

```
    1.
def int2bin(n, count=24):
    """returns the binary of integer n, using count number of digits"
""
    return "".join([str((n >> y) & 1) for y in range(count - 1, -1, -
1)])


class MD5(object):
    # 初始化密文
    def __init__(self, message):
        self.message = message
        self.ciphertext = ""

        # 初始向量
        self.A = 0x67452301
```

```
        self.B = 0xEFCDAB89
        self.C = 0x98BADCFE
        self.D = 0x10325476
        self.init_A = 0x67452301
        self.init_B = 0xEFCDAB89
        self.init_C = 0x98BADCFE
        self.init_D = 0x10325476
        '''
        self.A = 0x01234567
        self.B = 0x89ABCDEF
        self.C = 0xFEDCBA98
        self.D = 0x76543210
         '''

        self.T = [0xD76AA478, 0xE8C7B756, 0x242070DB, 0xC1BDCEEE, 0xF57C0FAF, 0x4787C62A, 0xA8304613, 0xFD469501,
                  0x698098D8, 0x8B44F7AF, 0xFFFF5BB1, 0x895CD7BE, 0x6B901122, 0xFD987193, 0xA679438E, 0x49B40821,
                  0xF61E2562, 0xC040B340, 0x265E5A51, 0xE9B6C7AA, 0xD62F105D, 0x02441453, 0xD8A1E681, 0xE7D3FBC8,
                  0x21E1CDE6, 0xC33707D6, 0xF4D50D87, 0x455A14ED, 0xA9E3E905, 0xFCEFA3F8, 0x676F02D9, 0x8D2A4C8A,
                  0xFFFA3942, 0x8771F681, 0x6D9D6122, 0xFDE5380C, 0xA4BEEA44, 0x4BDECFA9, 0xF6BB4B60, 0xBEBFBC70,
                  0x289B7EC6, 0xEAA127FA, 0xD4EF3085, 0x04881D05, 0xD9D4D039, 0xE6DB99E5, 0x1FA27CF8, 0xC4AC5665,
                  0xF4292244, 0x432AFF97, 0xAB9423A7, 0xFC93A039, 0x655B59C3, 0x8F0CCC92, 0xFFEFF47D, 0x85845DD1,
                  0x6FA87E4F, 0xFE2CE6E0, 0xA3014314, 0x4E0811A1, 0xF7537E82, 0xBD3AF235, 0x2AD7D2BB, 0xEB86D391]
        # 循环左移的位移位数
        self.s = [7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22,
                  5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20,
                  4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23,
                  6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21]
        self.m = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
                  1, 6, 11, 0, 5, 10, 15, 4, 9, 14, 3, 8, 13, 2, 7, 12,
```

```
                  5, 8, 11, 14, 1, 4, 7, 10, 13, 0, 3, 6, 9, 12, 15, 2,
                  0, 7, 14, 5, 12, 3, 10, 1, 8, 15, 6, 13, 4, 11, 2, 9]

    # 附加填充位
    def fill_text(self):
        for i in range(len(self.message)):
            c = int2bin(ord(self.message[i]), 8)
            self.ciphertext += c

        if len(self.ciphertext) % 512 != 448:
            if (len(self.ciphertext) + 1) % 512 != 448:
                self.ciphertext += '1'
            while len(self.ciphertext) % 512 != 448:
                self.ciphertext += '0'

        length = len(self.message) * 8
        if length <= 255:
            length = int2bin(length, 8)
        else:
            length = int2bin(length, 16)
            temp = length[8:12] + length[12:16] + length[0:4] + length[4:8]
            length = temp

        self.ciphertext += length
        while len(self.ciphertext) % 512 != 0:
            self.ciphertext += '0'

    # 分组处理（迭代压缩）
    def circuit_shift(self, x, amount):
        x &= 0xFFFFFFFF
        return ((x << amount) | (x >> (32 - amount))) & 0xFFFFFFFF

    def change_pos(self):
        a = self.A
        b = self.B
        c = self.C
        d = self.D
        self.A = d
        self.B = a
        self.C = b
        self.D = c
```

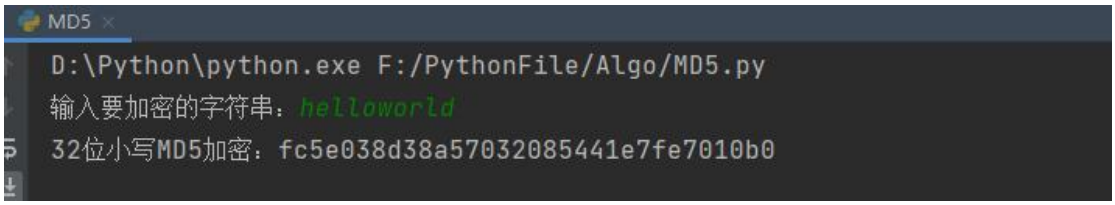
```
    ## F、G、H、I 函数
    def FF(self, mj, s, ti):
        mj = int(mj, 2)
        temp = self.F(self.B, self.C, self.D) + self.A + mj + ti
        temp = self.circuit_shift(temp, s)
        self.A = (self.B + temp) % pow(2, 32)
        self.change_pos()

    def GG(self, mj, s, ti):
        mj = int(mj, 2)
        temp = self.G(self.B, self.C, self.D) + self.A + mj + ti
        temp = self.circuit_shift(temp, s)
        self.A = (self.B + temp) % pow(2, 32)
        self.change_pos()

    def HH(self, mj, s, ti):
            mj = int(mj, 2)
            temp = self.H(self.B, self.C, self.D) + self.A + mj + ti
            temp = self.circuit_shift(temp, s)
            self.A = (self.B + temp) % pow(2, 32)
            self.change_pos()

       def II(self, mj, s, ti):
            mj = int(mj, 2)
            temp = self.I(self.B, self.C, self.D) + self.A + mj + ti
            temp = self.circuit_shift(temp, s)
            self.A = (self.B + temp) % pow(2, 32)
            self.change_pos()

       def F(self, X, Y, Z):
            return (X & Y) | ((~X) & Z)

       def G(self, X, Y, Z):
            return (X & Z) | (Y & (~Z))

       def H(self, X, Y, Z):
            return X ^ Y ^ Z

       def I(self, X, Y, Z):
            return Y ^ (X | (~Z))

       def group_processing(self):
            M = []
            for i in range(0, 512, 32):
```

```
128.                 num = ""
129.                 # 获取每一段的标准十六进制形式
130.                 for j in range(0, len(self.ciphertext[i:i + 32]), 4):
131.                     temp = self.ciphertext[i:i + 32][j:j + 4]
132.                     temp = hex(int(temp, 2))
133.                     num += temp[2]
134.                 # 对十六进制进行小端排序
135.                 num_tmp = ""
136.                 for j in range(8, 0, -2):
137.                     temp = num[j - 2:j]
138.                     num_tmp += temp
139.
140.                 num = ""
141.                 for i in range(len(num_tmp)):
142.                     num += int2bin(int(num_tmp[i], 16), 4)
143.                 M.append(num)
144.
145.             # print(M)
146.
147.             for j in range(0, 16, 4):
148.                 self.FF(M[self.m[j]], self.s[j], self.T[j])
149.                 self.FF(M[self.m[j + 1]], self.s[j + 1], self.T[j + 1])
150.                 self.FF(M[self.m[j + 2]], self.s[j + 2], self.T[j + 2])
151.                 self.FF(M[self.m[j + 3]], self.s[j + 3], self.T[j + 3])
152.
153.             for j in range(0, 16, 4):
154.                 self.GG(M[self.m[16 + j]], self.s[16 + j], self.T[16 +
    j])
155.                 self.GG(M[self.m[16 + j + 1]], self.s[16 + j + 1], sel
    f.T[16 + j + 1])
156.                 self.GG(M[self.m[16 + j + 2]], self.s[16 + j + 2], sel
    f.T[16 + j + 2])
157.                 self.GG(M[self.m[16 + j + 3]], self.s[16 + j + 3], sel
    f.T[16 + j + 3])
158.
159.             for j in range(0, 16, 4):
160.                 self.HH(M[self.m[32 + j]], self.s[32 + j], self.T[32 +
    j])
161.                 self.HH(M[self.m[32 + j + 1]], self.s[32 + j + 1], sel
    f.T[32 + j + 1])
```

```
                self.HH(M[self.m[32 + j + 2]], self.s[32 + j + 2], self.T[32 + j + 2])
                self.HH(M[self.m[32 + j + 3]], self.s[32 + j + 3], self.T[32 + j + 3])

            for j in range(0, 16, 4):
                self.II(M[self.m[48 + j]], self.s[48 + j], self.T[48 + j])
                self.II(M[self.m[48 + j + 1]], self.s[48 + j + 1], self.T[48 + j + 1])
                self.II(M[self.m[48 + j + 2]], self.s[48 + j + 2], self.T[48 + j + 2])
                self.II(M[self.m[48 + j + 3]], self.s[48 + j + 3], self.T[48 + j + 3])

            self.A = (self.A + self.init_A) % pow(2, 32)
            self.B = (self.B + self.init_B) % pow(2, 32)
            self.C = (self.C + self.init_C) % pow(2, 32)
            self.D = (self.D + self.init_D) % pow(2, 32)
            '''
            print("A:{}".format(hex(self.A)))
            print("B:{}".format(hex(self.B)))
            print("C:{}".format(hex(self.C)))
            print("D:{}".format(hex(self.D)))
            '''
            answer = ""
            for register in [self.A, self.B, self.C, self.D]:
                register = hex(register)[2:]
                for i in range(8, 0, -2):
                    answer += str(register[i - 2:i])

            return answer


    # MD5 = MD5("iscbupt")
    # MD5 = MD5("Beijing University of Posts and Telecommunications")

    # MD5 = MD5("njupt.information security")
    message = input("输入要加密的字符串：")
    MD5 = MD5(message)
    MD5.fill_text()
    result = MD5.group_processing()
    print("32 位小写 MD5 加密：{}".format(result))
```

六、实验数据及处理结果

加密结果和在线网站解密结果一致



七、思考讨论题或体会或对改进实验的建议

八、参考资料

现代密码学第 4 版