信息安全作业 3

190110429-何为

1. 给定消息“000……000”（512 位二进制数），通过编程计算其 MD5 值。请给出主要计算步骤和中间结果（16 进制）。

答:

代码如下:

```
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):
    ciphertext: str

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

        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]
        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
```

```python
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

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))
```

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

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

@property
def group_processing(self):
    M = []
    for i in range(0, 512, 32):
        num = ""
        # 获取每一段的标准十六进制形式
        for j in range(0, len(self.ciphertext[i:i + 32]), 4):
            temp = self.ciphertext[i:i + 32][j:j + 4]
            temp = hex(int(temp, 2))
            num += temp[2]
        # 对十六进制进行小端排序
        num_tmp = ""
        for j in range(8, 0, -2):
            temp = num[j - 2:j]
            num_tmp += temp

        num = ""
        for i in range(len(num_tmp)):
            num += int2bin(int(num_tmp[i], 16), 4)
        M.append(num)

    print(M)

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

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

    for j in range(0, 16, 4):
        self.HH(M[self.m[32 + j]], self.s[32 + j], self.T[32 + j])
```

```python
            self.HH(M[self.m[32 + j + 1]], self.s[32 + j + 1], self.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])
                print(answer)

        return answer


MD5 = MD5("0" * 512)
# 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))
```

主要计算步骤：
获取每一段的标准十六进制形式，对十六进制进行小端排序

```python
M = []
for i in range(0, 512, 32):
    num = ""
    # 获取每一段的标准十六进制形式
    for j in range(0, len(self.ciphertext[i:i + 32]), 4):
        temp = self.ciphertext[i:i + 32][j:j + 4]
        temp = hex(int(temp, 2))
        num += temp[2]
    # 对十六进制进行小端排序
    num_tmp = ""
    for j in range(8, 0, -2):
        temp = num[j - 2:j]
        num_tmp += temp

    num = ""
    for i in range(len(num_tmp)):
        num += int2bin(int(num_tmp[i], 16), 4)
    M.append(num)

print(M)
```

得到的消息分组如下:

M = ['00110000001100000011000000110000', '00110000001100000011000000110000', '00110000001100000011000000110000', '00110000001100000011000000110000', '00110000001100000011000000110000', '00110000001100000011000000110000', '00110000001100000011000000110000', '00110000001100000011000000110000', '00110000001100000011000000110000', '00110000001100000011000000110000', '00110000001100000011000000110000', '00110000001100000011000000110000', '00110000001100000011000000110000', '00110000001100000011000000110000', '00110000001100000011000000110000', '00110000001100000011000000110000']

四轮运算:

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

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

for j in range(0, 16, 4):
    self.HH(M[self.m[32 + j]], self.s[32 + j], self.T[32 + j])
    self.HH(M[self.m[32 + j + 1]], self.s[32 + j + 1], self.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])
```

最后一次计算得链接变量 buffer 为:
A:0x529b2ab3
B:0xb54edd2a
C:0xebf455db
D:0x27936c9f
小端（倒序）级联运算结果:

```
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])
        print(answer)

return answer
```

最终结果:
b32a9b522add4eb5db55f4eb9f6c9327

2. 有人说“所有的散列函数都存在产生碰撞的问题，很不安全”，你认为正确与否，为什么？
答:
我认为不正确。

虽然所有散列函数确实都存在碰撞的情况，就如生日悖论所说，只要摘要的长度有限，遍历足够多的随机输出，总会发生碰撞现象。

但是生日攻击只依赖于消息摘要的长度，即 Hash 值的长度，没有利用 Hash 函数的结构和任何代数弱性质，因此增长 Hash 值的长度就可以抵御生日攻击——即，让解密的代价远远超过加密的代价，这样就能增加散列函数的安全性。