山东大学网络空间安全学院

创新创业实践 课程实验报告

学号：202100141038	姓名：沈文慧	班级：21 密码 2 班
实验题目： implement the Rho method of reduced SM3		
实验学时：	实验日期：2023-07	

问题分析： implement the Rho method of reduced SM3

本实验要求利用 Rho method 破解 SM3 算法找到 nbit 碰撞的原像，该实验设计 f 函数为 f:H(x),即 Wi=H(Wi-1)(除第一次输入信息 m 外，f 函数输入输出均为 256bit)
Polladr rho method to fing collision：利用了生日悖论，使碰撞的复杂度降到 O(n--√n)级别，同时能有效避免内存过大;
其思想是：利用 f 函数随机游走，构造出随机序列，可以发现，该序列发展到一定程度，会得到与之前相同的元素，形成环。因此可以应用到哈希函数中。
如何找环？如何找到进入环的元素（即找到哈希碰撞）？可以通过 Floyd 判圈法，规定两个指针，其中一个按照序列一步一步走，另外一个两步两步走。当两者相遇时，即找到了一个环，此时找到进入环的元素的两个前驱，即可找到一对碰撞。
如何找到刚进入环的元素？定义两个指针，一个指向初始信息 m，另外一个指向上一步两个指针相遇的点 h，两个指针都是一步一步走，最后相遇的点即为第一次进入环的点。
值得注意的是，要避免信息 m 在环上，因此初始值要大于 256bit。

硬件环境：Windows 11，X64

软件环境：visual studio code

实验步骤与内容：

1. 实验代码：

```
import math
import random
import time
import string


#SM3 算法
iv_1='7380166f4914b2b9172442d7da8a0600a96f30bc163138aae38dee4db0fb0e4e'
T_all=['79cc4519','7a879d8a']


def str2byte(m): #str 转换成 byte 数组的 int 值
    m_mid=len(m)
```

```
    m_byte=[]
    m_bytearray=m.encode('utf-8')
    for i in range(m_mid):
        m_byte.append(m_bytearray[i])
    return int(m_byte[0])

def FF(x,y,z,j):
    if j<=15:
        return x^y^z
    else:
        return (x&y)|(x&z)|(y&z)

def GG(x,y,z,j):
    if j<=15:
        return x^y^z
    else:
        return (x&y)|(~x&z)


def leftmov(m,num):   #向左移动
    num=num%32
    m=bin(m).replace('0b','').zfill(32)
    m_left=m[num:]+m[:num]
    return int(m_left,2)

def P0(x):
    return x^leftmov(x,9)^leftmov(x,17)
def P1(x):
    return x^leftmov(x,15)^leftmov(x,23)


#消息填充
def msgcut(msg,mem):   #数据按间距分组划分 iv 向量
    lenth=len(msg)
    num=int(lenth/mem)
    tool_ary=[]
    for i in range(num):
        tool_ary.append(msg[i*mem:mem+i*mem])
```

```
    return tool_ary

def msgpop(m):
    m=str(m) #输入的内容作为 str
    lenth=len(m)
    mid_msg=""
    for i in range(lenth):
        mid_msg+=bin(str2byte(m[i])).replace('0b','').zfill(8)
    l_msg=len(mid_msg)
    k=512-(64+(l_msg+1))%512
    msg=mid_msg+'1'+k*'0'
    add_len=bin(l_msg).replace('0b','').zfill(64)
    msg=msg+add_len
    msg=msgcut(msg,512)
    return msg

#消息扩展和迭代压缩
def msgexpd(IV,m):
    B1=msgcut(m, 32)
    B2=[]
    T=[]
    T.append(int(T_all[0],16))
    T.append(int(T_all[1],16))
    for i in range(16):
        B1[i]=int(B1[i],2)
    for j in range(16,68):
        num=B1[j-16]^B1[j-9]^leftmov(B1[j-3],15)
        mid=P1(num)
        re=mid^leftmov(B1[j-13],7)^B1[j-6]
        B1.append(re)
    for k in range(64):
        mid=B1[k]^B1[k+4]
        B2.append(mid)

    #B1 是 W,B2 是 W'
    #print(B1,'**********',B2)
    iv=[]
    for i in range(8):
```

```
        iv.append(int(IV[i],16))
    A=iv[0]
    B=iv[1]
    C=iv[2]
    D=iv[3]
    E=iv[4]
    F=iv[5]
    G=iv[6]
    H=iv[7]
    toolnum=2**32
    for j in range(64):
        if j<=15:
            mid1=(leftmov(A,12)+E+leftmov(T[0],j))%toolnum
        else:
            mid1=(leftmov(A,12)+E+leftmov(T[1],j))%toolnum
        SS1=leftmov(mid1,7)#SS1
        SS2=(SS1^leftmov(A,12))%toolnum
        TT1=(FF(A,B,C,j)+D+SS2+B2[j])%toolnum
        TT2=(GG(E,F,G,j)+H+SS1+B1[j])%toolnum
        D=C
        C=leftmov(B,9)
        B=A
        A=TT1
        H=G
        G=leftmov(F,19)
        F=E
        E=P0(TT2)

    v_new=[]
    v_new.append(A^iv[0])
    v_new.append(B^iv[1])
    v_new.append(C^iv[2])
    v_new.append(D^iv[3])
    v_new.append(E^iv[4])
    v_new.append(F^iv[5])
    v_new.append(G^iv[6])
    v_new.append(H^iv[7])
    return v_new
```

```
def sm3(msg): #sm3 整合
    mid1=msgpop(msg)
    n=len(mid1)
    temp_IV=iv_1
    for i in mid1:
        if i !='':
            mid_IV=msgcut(temp_IV,8) #向量
            temp_IV=msgexpd(mid_IV,i)
            final=""
            miwen=""
            for k in range(8):
                final+=bin(temp_IV[k]).replace('0b','').zfill(32)
            for j in range(64):
                mid_str=final[4*j:4+4*j]
                mid_num=int(mid_str,2)
                miwen+=hex(mid_num).replace('0x','')
            temp_IV=miwen
    out=""
    for a in range(64):
        out+=bin(int(temp_IV[a],16)).replace('0b','').zfill(4)
    return out

#rho 方法

def rho(n):
    max_num=2**n
    x=random.randint(0,max_num)
    temp=[]
    temp_num=[]
    x_sm3=sm3(x)
    temp_num.append(int(x_sm3,2))##
    sm3_mid=x_sm3[0:n]
    temp.append(sm3_mid)#
    rho_sm3=sm3(x_sm3)
    temp_num.append(int(rho_sm3,2))##
    rho_mid=rho_sm3[0:n]
    if rho_mid in temp:
```

```
            print('前',n,'位相等,find x1:',x,' and x2:',x_sm3)
            return x,x_sm3
        else:   #进入循环，找“环”
            temp.append(rho_mid)#
            while True:
                rho_new=sm3(rho_sm3)
                temp_num.append(int(rho_new,2))##
                new_mid=rho_new[0:n]
                if new_mid in temp:
                    loc=temp.index(new_mid)
                    x1=rho_sm3
                    #x1=int(rho_sm3,2)
                    x2=temp_num[loc-1]
                    x2=bin(x2).replace('0b','').zfill(256)
                    break
                else:
                    temp.append(new_mid)
                    rho_sm3=rho_new
                    continue
        return x1,x2

def func(n):
        x1,x2=rho(n)
        while x1==x2:
            x1,x2=rho(n)
        print('前',n,'位相等,find x1:',x1,'and x2:',x2)

t1=time.time()
func(n) #需要满足前 n 位相等，并求出产生碰撞的两个数
t2=time.time()
print('耗时：',t2-t1)
```