Experimental report——the Rho method of reduced SM3

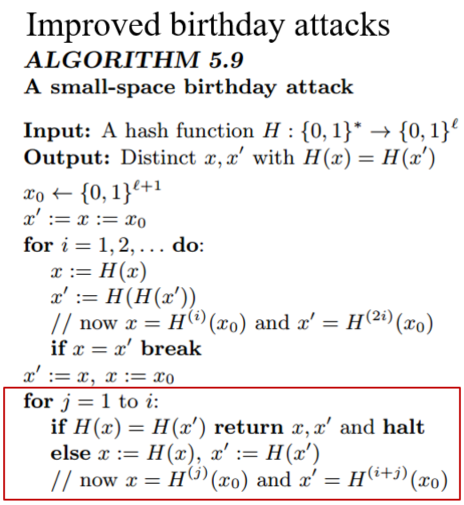
姓名：李祥方

学号：201900460041

1 前置知识

$SM3$**的定义：**SM3密码杂凑算法是中国国家密码管理局2010年公布的中国商用密码杂凑算法标准。具体算法标准原始文本参见参考文献[1]。该算法于2012年发布为密码行业标准(GM/T 0004-2012)，2016年发布为国家密码杂凑算法标准(GB/T 32905-2016)。SM3适用于商用密码应用中的数字签名和验证，是在[SHA-256]基础上改进实现的一种算法，其安全性和SHA-256相当。SM3和MD5的迭代过程类似，也采用Merkle-Damgard结构。消息分组长度为512位，摘要值长度为256位。整个算法的执行过程可以概括成四个步骤：消息填充、消息扩展、迭代压缩、输出结果。

$therhomethod$:



Improved birthday attacks

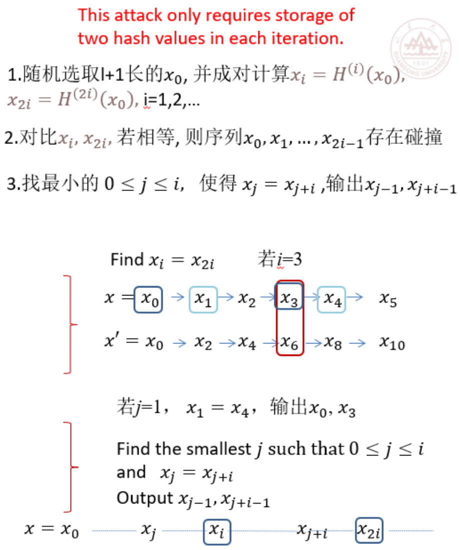
ALGORITHM 5.9

A small-space birthday attack

Input: A hash function $H : \{0,1\}^* \to \{0,1\}^\ell$

Output: Distinct x, x' with $H(x) = H(x')$

```
$x_0 \leftarrow \{0,1\}^{\ell+1}$
$x' := x := x_0$
for $i = 1, 2, \ldots$ do:
    $x := H(x)$
    $x' := H(H(x'))$
    // now $x = H^{(i)}(x_0)$ and $x' = H^{(2i)}(x_0)$
    if $x = x'$ break
$x' := x$, $x := x_0$
for $j = 1$ to $i$:
    if $H(x) = H(x')$ return $x, x'$ and halt
    else $x := H(x)$, $x' := H(x')$
    // now $x = H^{(j)}(x_0)$ and $x' = H^{(i+j)}(x_0)$
```



This attack only requires storage of two hash values in each iteration.

1.随机选取l+1长的x_0, 并成对计算$x_i = H^{(i)}(x_0)$, $x_{2i} = H^{(2i)}(x_0)$, i=1,2,...

2.对比x_i, x_{2i}, 若相等, 则序列$x_0, x_1, \ldots, x_{2i-1}$存在碰撞

3.找最小的 $0 \le j \le i$，使得 $x_j = x_{j+i}$,输出x_{j-1}, x_{j+i-1}

Find $x_i = x_{2i}$ 若i=3

若j=1， $x_1 = x_4$，输出x_0, x_3

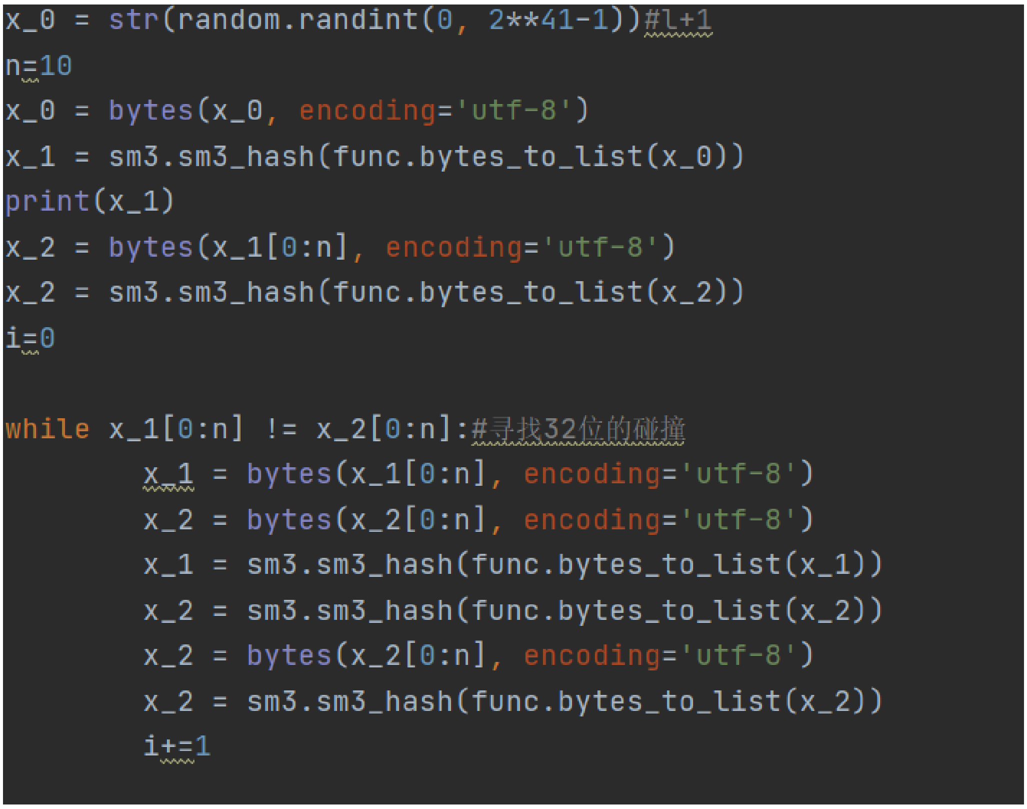
Find the smallest j such that $0 \le j \le i$ and $x_j = x_{j+i}$

Output x_{j-1}, x_{j+i-1}

$x = x_0 \to x_1 \to x_2 \to x_3 \to x_4 \to x_5$
$x' = x_0 \to x_2 \to x_4 \to x_6 \to x_8 \to x_{10}$
$x = x_0$
x_j
x_i
x_{j+i}
x_{2i}

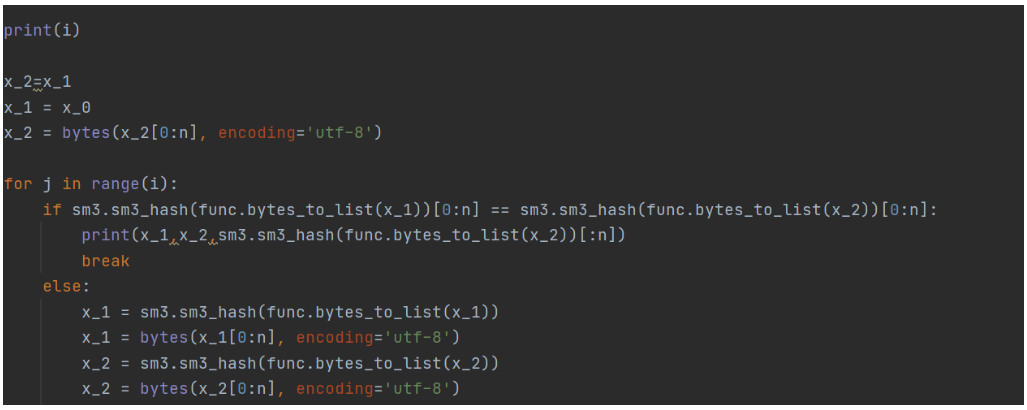
2 实验过程

根据以上前置知识，编写了以下的代码，代码寻找的是**40**位的碰撞.



```
x_0 = str(random.randint(0, 2**41-1))#l+1
n=10
x_0 = bytes(x_0, encoding='utf-8')
x_1 = sm3.sm3_hash(func.bytes_to_list(x_0))
print(x_1)
x_2 = bytes(x_1[0:n], encoding='utf-8')
x_2 = sm3.sm3_hash(func.bytes_to_list(x_2))
i=0

while x_1[0:n] != x_2[0:n]:#寻找32位的碰撞
        x_1 = bytes(x_1[0:n], encoding='utf-8')
        x_2 = bytes(x_2[0:n], encoding='utf-8')
        x_1 = sm3.sm3_hash(func.bytes_to_list(x_1))
        x_2 = sm3.sm3_hash(func.bytes_to_list(x_2))
        x_2 = bytes(x_2[0:n], encoding='utf-8')
        x_2 = sm3.sm3_hash(func.bytes_to_list(x_2))
        i+=1
```



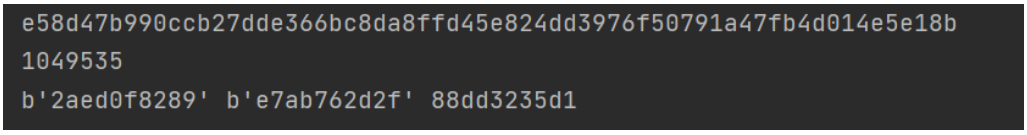
```
print(i)

x_2=x_1
x_1 = x_0
x_2 = bytes(x_2[0:n], encoding='utf-8')

for j in range(i):
    if sm3.sm3_hash(func.bytes_to_list(x_1))[0:n] == sm3.sm3_hash(func.bytes_to_list(x_2))[0:n]:
        print(x_1,x_2,sm3.sm3_hash(func.bytes_to_list(x_2))[:n])
        break
    else:
        x_1 = sm3.sm3_hash(func.bytes_to_list(x_1))
        x_1 = bytes(x_1[0:n], encoding='utf-8')
        x_2 = sm3.sm3_hash(func.bytes_to_list(x_2))
        x_2 = bytes(x_2[0:n], encoding='utf-8')
```

3 实验结果

经过代码的运行找到了**40**位的碰撞,如下图：



```
e58d47b990ccb27dde366bc8da8ffd45e824dd3976f50791a47fb4d014e5e18b
1049535
b'2aed0f8289' b'e7ab762d2f' 88dd3235d1
```