山东大学网络空间安全学院

网络空间安全创新创业实践

Project 1 SM3生日攻击

姓名：张麟康

学号：201900301107

1 原理概述

1.1 SM3算法综述

SM3密码杂凑算法适用于商用密码应用中的数字签名和验证、消息认证码的生成和验证以及随机数的生成，可以满足多种密码应用的安全需求。

对于一个长度为l比特的消息m，SM3杂凑算法经过填充、迭代压缩和输出选裁生成杂凑值，杂凑值的输出长度为256比特。

1.2 常数与函数

初始值

IV=7380166f 4914b2b9 172442d7 da8a0600 a96f30bc 163138aa e38dee4d b0fb0e4e

常量

$$T_j=\begin{cases}79cc4519 & 0\leqslant j\leqslant 15\\ 7a879d8a & 16\leqslant j\leqslant 63\end{cases}$$

布尔函数

$$FF_j(X,Y,Z)=\begin{cases}X\oplus Y\oplus Z & 0\leqslant j\leqslant 15\\ (X\wedge Y)\vee(X\wedge Z)\vee(Y\wedge Z) & 16\leqslant j\leqslant 63\end{cases}$$

$$GG_j(X,Y,Z)=\begin{cases}X\oplus Y\oplus Z & 0\leqslant j\leqslant 15\\ (X\wedge Y)\vee(\neg X\wedge Z) & 16\leqslant j\leqslant 63\end{cases}$$

式中 X,Y,Z 为字。

置换函数

$$P_0(X)=X\oplus(X<<<9)\oplus(X<<<17)$$
$$P_1(X)=X\oplus(X<<<15)\oplus(X<<<23)$$

式中 X 为字。

1.3 填充

假设消息 m 的长度为 l 比特，则首先将比特“1”添加到消息的末尾，再添加 k 个“0”，k 是满足 $l+1+k\equiv 448 \pmod{512}$ 的最小的非负整数。然后再添加一个 64 位比特串，该比特串是长度 l 的二进制表示。填充后的消息 m' 的比特长度为 512 的倍数。

例如：对消息：01100001 01100010 01100011，其长度 $l=24$，经填充得到比特串：

$$01100001\ \ 01100010\ \ 01100011\ \ 1\ \overbrace{00\cdots00}^{423\text{比特}}\ \overbrace{\underbrace{00\cdots011000}_{l\text{的二进制表示}}}^{64\text{比特}}$$

1.4 迭代压缩

1.4.1 迭代过程

将填充后的消息 m' 按 512 比特进行分组：

$$m' = B^{(0)} B^{(1)} \cdots B^{(n-1)}$$

其中 $n=(l+k+65)/512$。

对 m' 按下列方式迭代：

FOR $i=0$ **TO** $n-1$

$V^{(i+1)} = CF(V^{(i)}, B^{(i)})$

ENDFOR

其中 CF 是压缩函数，$V^{(0)}$ 为 256 比特初始值 IV，$B^{(i)}$ 为填充后的消息分组，迭代压缩的结果为 $V^{(n)}$。

1.4.2 消息扩展

将消息分组 $B^{(i)}$ 按以下方法扩展生成 132 个消息字 $W_0, W_1, \cdots W_{67}, W'_0, W'_1, \cdots W'_{63}$，用于压缩函数 CF：

a) 将消息分组 $B^{(i)}$ 划分为 16 个字 $W_0, W_1, \cdots W_{15}$。

b) **FOR** $j=16$ **TO** 67

$W_j \leftarrow P_1(W_{j-16} \oplus W_{j-9} \oplus (W_{j-3} <<< 15)) \oplus (W_{j-13} <<< 7) \oplus W_{j-6}$

ENDFOR

c) **FOR** $j=0$ **TO** 63

$W'_j = W_j \oplus W_{j+4}$

ENDFOR

1.4.3 压缩函数

令 A, B, C, D, E, F, G, H 为字寄存器，$SS1, SS2, TT1, TT2$ 为中间变量，压缩函数 $V^{(i+1)} = CF(V^{(i)}, B^{(i)})$，$0 \leqslant i \leqslant n-1$。计算过程描述如下：

$ABCDEFGH \leftarrow V^{(i)}$

FOR $j=0$ **TO** 63

$SS1 \leftarrow ((A <<< 12) + E + (T_j <<< (j \bmod 32))) <<< 7$

$SS2 \leftarrow SS1 \oplus (A <<< 12)$

$TT1 \leftarrow FF_j(A, B, C) + D + SS2 + W'_j$

$TT2 \leftarrow GG_j(E, F, G) + H + SS1 + W_j$

$D \leftarrow C$

$C \leftarrow B <<< 9$

$B \leftarrow A$

$A \leftarrow TT1$

$II \leftarrow G$

$G \leftarrow F<<<19$

$F \leftarrow E$

$E \leftarrow P_0(TT2)$

ENDFOR

$V^{(i+1)} \leftarrow ABCDEFGII \oplus V^{(i)}$

其中，字的存储为 big-endian 格式，左边为高有效位，右边为低有效位。

1.5 杂凑值

$ABCDEFGII \leftarrow V^{(n)}$

输出 256 比特的杂凑值 $y = ABCDEFGII$。

1.6 生日攻击原理

生日攻击起源于生日悖论：随机选取n个人，求n个人中两个人的生日相同的概率是多少。经过概率论推导，设有值域为N的k个数，选两个数存在碰撞的概率为 $1-\Pi_{i=1}^{k}\frac{N-i}{N}$，其中$\frac{N-i}{N}$为不碰撞的概率。

那么有如下公式：

$$1-\prod_{i=1}^{k}\frac{N-i}{N}=1-\frac{(N-1)!}{N^k\cdot(N-k)!}$$

根据均值不等式可得：

$$\sqrt[k]{\prod_{i=1}^{k}\frac{N-i}{N}}<\frac{1}{k}\cdot\sum_{i=1}^{k}(1-\frac{i}{N})$$

从而

$$1-\prod_{i=1}^{k}\frac{N-i}{N}>1-\left(\frac{1}{k}\cdot\sum_{i=1}^{k}(1-\frac{i}{N})\right)^k=1-\left[\frac{(1-1/N+1-k/N)\times k}{2\times k}\right]^k=1-\left(1-\frac{k+1}{2N}\right)^k$$

由重要不等式可得：

$$1-\left(1-\frac{k+1}{2N}\right)^k\geq 1-\left[e^{-\frac{k+1}{2N}}\right]^k$$

因此可得：

$$1-\prod_{i=1}^{k}\frac{N-i}{N}>1-\left[e^{-\frac{k+1}{2N}}\right]^{k}$$

故令碰撞概率为$p=1-e^{-\frac{k^2+k}{2N}}=f(k,N)$，当$k$为23、$N$为365时，$f(23,365)$即成功概率约等于0.51。

将此问题进行扩展，假设有一个杂凑函数f，其值域为H，那么寻找到一对碰撞就是找到不同的x和x'使得$f(x)=f(x')$。

根据生日悖论，要想达到超过百分之五十的概率找到一对碰撞，需要尝试的次数是$\sqrt{\frac{\pi}{2}H}$次。

2 具体实现

在本项目中，所有程序均采用Python语言编写，具体实现方案如下。

2.1 SM3算法的具体实现

SM3算法完全依据国密算法标准实现，未经任何优化，具体实现如下。

2.1.1 常量和函数

```
IV = [
    1937774191, 1226093241, 388252375, 3666478592,
    2842636476, 372324522, 3817729613, 2969243214,
]

T_j = [
    2043430169, 2043430169, 2043430169, 2043430169, 2043430169, 2043430169,
    2043430169, 2043430169, 2043430169, 2043430169, 2043430169, 2043430169,
    2043430169, 2043430169, 2043430169, 2043430169, 2055708042, 2055708042,
    2055708042, 2055708042, 2055708042, 2055708042, 2055708042, 2055708042,
    2055708042, 2055708042, 2055708042, 2055708042, 2055708042, 2055708042,
    2055708042, 2055708042, 2055708042, 2055708042, 2055708042, 2055708042,
    2055708042, 2055708042, 2055708042, 2055708042, 2055708042, 2055708042,
    2055708042, 2055708042, 2055708042, 2055708042, 2055708042, 2055708042,
    2055708042, 2055708042, 2055708042, 2055708042, 2055708042, 2055708042,
    2055708042, 2055708042, 2055708042, 2055708042
]
```

```python
def sm3_ff_j(x, y, z, j):
    if 0 <= j and j < 16:
        ret = x ^ y ^ z
    elif 16 <= j and j < 64:
        ret = (x & y) | (x & z) | (y & z)
    return ret

def sm3_gg_j(x, y, z, j):
    if 0 <= j and j < 16:
        ret = x ^ y ^ z
    elif 16 <= j and j < 64:
        #ret = (X | Y) & ((2 ** 32 - 1 - X) | Z)
        ret = (x & y) | ((~ x) & z)
    return ret

def sm3_p_0(x):
    return x ^ (rotl(x, 9 % 32)) ^ (rotl(x, 17 % 32))

def sm3_p_1(x):
    return x ^ (rotl(x, 15 % 32)) ^ (rotl(x, 23 % 32))
```

```python
from random import choice


xor = lambda a, b:list(map(lambda x, y: x ^ y, a, b))

rotl = lambda x, n:((x << n) & 0xffffffff) | ((x >> (32 - n)) & 0xffffffff)

get_uint32_be = lambda key_data:((key_data[0] << 24) | (key_data[1] << 16) | (key_data[2] << 8) | (key_data[3]))

put_uint32_be = lambda n:[((n>>24)&0xff), ((n>>16)&0xff), ((n>>8)&0xff), ((n)&0xff)]

padding = lambda data, block=16: data + [(16 - len(data) % block)for _ in range(16 - len(data) % block)]

unpadding = lambda data: data[:-data[-1]]

list_to_bytes = lambda data: b''.join([bytes((i,)) for i in data])

bytes_to_list = lambda data: [i for i in data]

random_hex = lambda x: ''.join([choice('0123456789abcdef') for _ in range(x)])
```

2.1.2 填充

```python
def padding(msg):
    mlen = len(msg)
    msg.append(0x80)
    mlen += 1
    tail = mlen % 64
    range_end = 56
    if tail > range_end:
        range_end = range_end + 64
    for i in range(tail, range_end):
        msg.append(0x00)
    bit_len = (mlen - 1) * 8
    msg.extend([int(x) for x in struct.pack('>q', bit_len)])
    for j in range(int((mlen - 1) / 64) * 64 + (mlen - 1) % 64, len(msg)):
        global pad
        pad.append(msg[j])
        global pad_str
        pad_str += str(hex(msg[j]))
    return msg
```

2.1.3 迭代压缩

```
def sm3_cf(v_i, b_i):
    w = []
    for i in range(16):
        weight = 0x1000000
        data = 0
        for k in range(i*4,(i+1)*4):
            data = data + b_i[k]*weight
            weight = int(weight/0x100)
        w.append(data)

    for j in range(16, 68):
        w.append(0)
        w[j] = sm3_p_1(w[j-16] ^ w[j-9] ^ (rotl(w[j-3], 15 % 32))) ^ (rotl(w[j-13], 7 % 32)) ^ w[j-6]
        str1 = "%08x" % w[j]
    w_1 = []
    for j in range(0, 64):
        w_1.append(0)
        w_1[j] = w[j] ^ w[j+4]
        str1 = "%08x" % w_1[j]
```

```
    for j in range(0, 64):
        ss_1 = rotl(
            ((rotl(a, 12 % 32)) +
            e +
            (rotl(T_j[j], j % 32))) & 0xffffffff, 7 % 32
        )
        ss_2 = ss_1 ^ (rotl(a, 12 % 32))
        tt_1 = (sm3_ff_j(a, b, c, j) + d + ss_2 + w_1[j]) & 0xffffffff
        tt_2 = (sm3_gg_j(e, f, g, j) + h + ss_1 + w[j]) & 0xffffffff
        d = c
        c = rotl(b, 9 % 32)
        b = a
        a = tt_1
        h = g
        g = rotl(f, 19 % 32)
        f = e
        e = sm3_p_0(tt_2)

        a, b, c, d, e, f, g, h = map(
            lambda x:x & 0xFFFFFFFF ,[a, b, c, d, e, f, g, h])

    v_j = [a, b, c, d, e, f, g, h]
```

```python
def sm3_hash(msg, new_v):
    # print(msg)
    len1 = len(msg)
    reserve1 = len1 % 64
    msg.append(0x80)
    reserve1 = reserve1 + 1
    # 56-64, add 64 byte
    range_end = 56
    if reserve1 > range_end:
        range_end = range_end + 64

    for i in range(reserve1, range_end):
        msg.append(0x00)

    bit_length = (len1) * 8
    bit_length_str = [bit_length % 0x100]
    for i in range(7):
        bit_length = int(bit_length / 0x100)
        bit_length_str.append(bit_length % 0x100)
    for i in range(8):
```

```python
    group_count = round(len(msg) / 64) - 1

    B = []
    for i in range(0, group_count):
        B.append(msg[(i + 1)*64:(i+2)*64])

    V = []
    V.append(new_v)
    for i in range(0, group_count):
        V.append(sm3_cf(V[i], B[i]))

    y = V[i+1]
    result = ""
    for i in y:
        result = '%s%08x' % (result, i)
    return result
```

2.2 生日攻击的具体实现

生日攻击的实现过程非常简单，首先定义缩减输出的SM3杂凑函数便于实践操作，只需简单截取标准SM3实现的前$TRUNC$个字即可（每个字为一个16进制数4比特，因此找到$4 \times TRUNC$比特碰撞）。

```python
def reduced_sm3(m):
    return sm3.SM3(m)[:TRUNC]
```

为了便于寻找碰撞，定义生成随机字符串的方法get_random_input()，该函数利用random模块中的取样方法生成一个长度为N的随机字符串。

```python
def get_random_input():
    random_input = ''.join(random.sample(char_set, N))
    res = ''
    for x in random_input:
        res += str(ord(x))
    return res
```

定义变量MAX_TRIAL表示最大尝试次数，定义一个字典数据结构record用于记录每次生成随机字符串及其对应的哈希值以便于后续寻找碰撞，之后进入循环，生成一个随机字符串并调用缩减输出的SM3算法，如果本次生成的哈希已经在record中且其对应的字符串与本次字符串不同则找到一对碰撞，如果本次生成的哈希在record中且对应的字符串与本次生成的随机字符串相同则表明不是一个满足要求的碰撞。如果哈希值尚未存储到字典中，那么将哈希值和对应的字符串存入字典。

```python
for i in range(MAX_TRIAL):
    # print(trial_times)
    trial_times+=1
    random_input = get_random_input()
    hash = reduced_sm3(random_input)
    if hash not in record.keys():
        try:
            record[hash] = random_input
        except MemoryError:
            print("LOG: MemoryError")
            hashed_dict = {}
    else:
        if record[hash] == random_input:
            print("String Already Used!")
            print("Number of evaluations made so far", trial_times)
        else:
            break
```

当找到满足条件的碰撞后，输入碰撞结果。

```python
print('Collision Results')
print("collision:", colliding_hash)
print("Number of evaluations made: ", str(trial_times))
print("Time Taken: %.2f seconds" % (time.time() - start_time))
print(colliding+' | '+sm3.SM3(colliding))
print(random_input+' | '+sm3.SM3(random_input))
```

最终依次实现了32、36、40、44、48、52比特的生日攻击，具体结果如下图所示。

```
PS F:\course-project-2022> & D:/Python310/python.exe f:/course-project-2022/sm3_btd_atk.py
Collision Results
collision: 193cda2e
Number of evaluations made:  59247
Time Taken: 42.98 seconds
99901191201081047375117871038812181557471109696651688489971071151221136582 72 | 193cda2e920705dab06afc1665ee31d9dcadf30aeb12b031288b81138a9d3578
10810510411483708281741138872118119736611775531029052115514849978410611612186 | 193cda2ec89eced13ac0011a4281a27ebb29aaee8b642978e659b422c7d10701
```

```
PS F:\course-project-2022> & D:/Python310/python.exe f:/course-project-2022/sm3_btd_atk.py
Collision Results
collision: 921cc83ee
Number of evaluations made:  482251
Time Taken: 354.31 seconds
549072105881005699121821024911350808581115119867176988410170108106791178912 2 | 921cc83ee8e155afa5eeefe99bb622281126f05dc85c6d1b2edd82b11338b6ec
837198651188089748110311497821025370671067511711611554721081127666107121906 8 | 921cc83ee77c94d638b52c93831d49cfe7df0f4ab22f7f123e6ec51f37505bbe
```

```
PS F:\course-project-2022> & D:/Python310/python.exe f:/course-project-2022/sm3_btd_atk.py
Collision Results
collision: f2005553af
Number of evaluations made:  1077199
Time Taken: 749.30 seconds
806977121104106539810210090729788891141016811010767491057911784667054521131 19 | f2005553af0971ce2cc56fbc65a61926e3eab1c0559261d1d0089cfff5d5d21a
981001055372104551131101088512011968118117761035286111671149011510612174897973 66 | f2005553afee508a11a588a472ff6a8c314d36e9b4d86f3fd7d5f17a414f7d2
```

```
PS F:\course-project-2022> & D:/Python310/python.exe f:/course-project-2022/sm3_btd_atk.py
Collision Results
collision: 3d3bcda4c7a
Number of evaluations made:  5322767
Time Taken: 3788.42 seconds
789048881029865555087861111216911776109105681061201031041189783101108119574982 | 3d3bcda4c7a07ddf19e21ef4230c4d7e9f26563ffd93283e72a3336f6ba62b9a
121499711981738286871026688100111107787011856116109114771088911510354981207548 | 3d3bcda4c7a87532a3cb01912806826360681ff82a3106ab8a4cbfa880c2195
```

```
PS F:\course-project-2022> & D:/Python310/python.exe f:/course-project-2022/sm3_btd_atk.py
Collision Results
collision: 8491adf4577a
Number of evaluations made:  7629625
Time Taken: 46632.08 seconds
518510086881194811152538097120116846610810210411811471545672110997789105122 81 | 8491adf4577a662d6cbd44b852fca7d1da72c188680b4950abbcc4b47dac438f
114687282746711310648521151179857901218850757683105861201077810289108109119 56 | 8491adf4577a13513864583046220c3b8332d1b8ed8956812d1b4d357277a28
```

```
PS F:\course-project-2022> & D:/Python310/python.exe f:/course-project-2022/sm3_btd_atk.py
Collision Results
collision: 836d7ec96e7be
Number of evaluations made:  12000000
Time Taken: 17368.18 seconds
117110109685211911176991205611675519078571125354831159810865555070113118698 9 | 836d7ec96e7bef942b6686ae37c8d90a8c60a1074bdc9701ce7cfd7e26ad70
117110109685211911176991205611675519078571125354831159810865555070113118698 9 | 836d7ec96e7bef942b6686ae37c8d90a8c60a1074bdc9701ce7cfd7e26ad70
PS F:\course-project-2022>
```