

实验报告（哈希函数）

【实验目的】

- 1、理解密码学 Hash 函数的安全性质、攻击方法、设计思想，掌握常用的密码学 Hash 函数的实现流程；
- 2、理解消息认证码的设计思想，掌握基于 Hash 函数的消息认证码 HMAC 的实现方法。

【实验环境】

Python 3.9.7 64-bit

【实验内容】

一、SHA-1

1、算法流程

- 报文填充（填充 100…… + 64bit 报文长度至 bit 长度整除 512）；
- 报文分组（512 bit 为一组）；
- 迭代（压缩函数 f ：输入 512 bit 报文组和 160 bit 缓冲区，输出 160 bit 进入下一缓冲区）；
- 最后一组压缩函数的输出作为报文的 SHA1 值。

函数调用关系图：

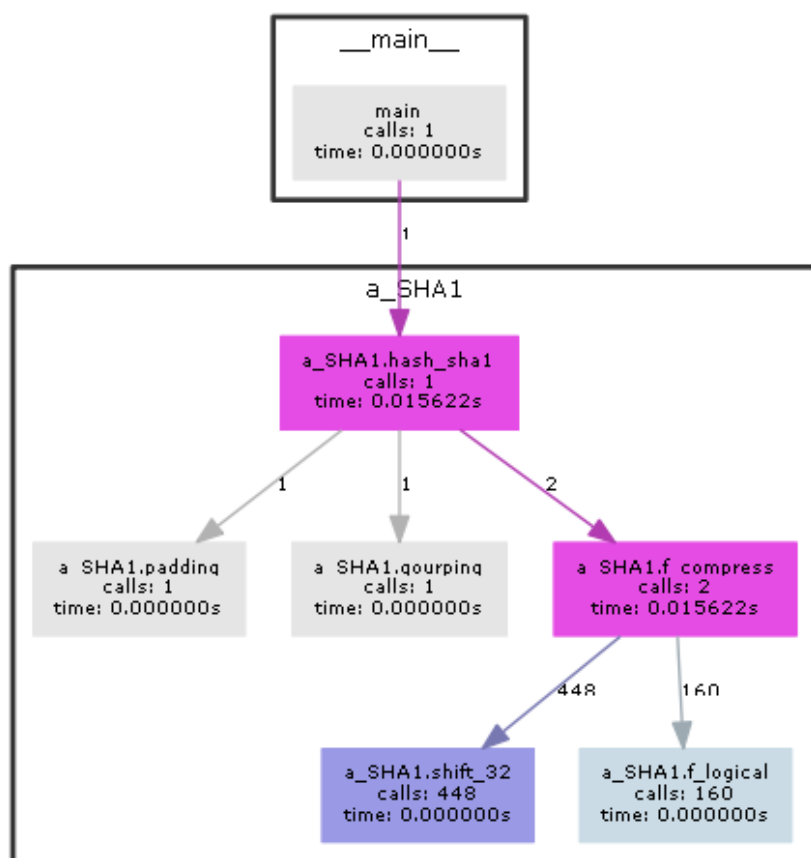


图 1 SHA1 函数调用关系图

表 1 SHA1 函数说明

函数名	函数功能
hash_sha1()	SHA1 主函数
padding()	报文填充
grouping()	报文分组
f_compress()	压缩函数
shift_32()	32 bit 数的循环移位
f_logical()	原始逻辑函数

2、测试样例及结果截图

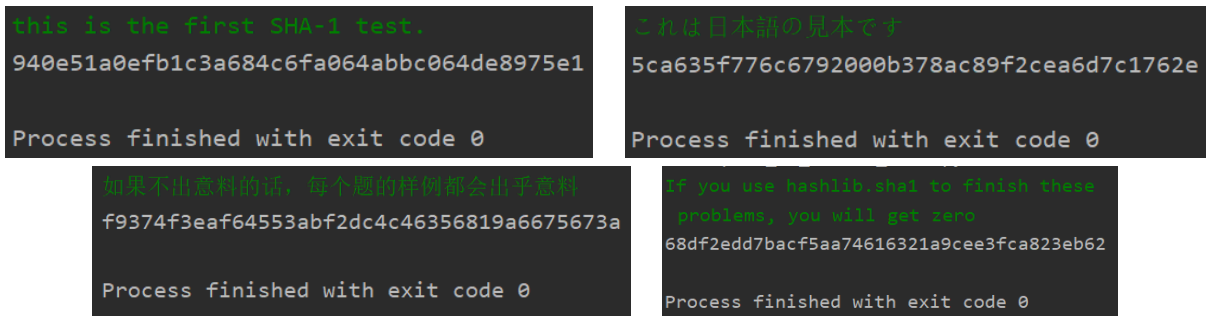


图 2 SHA1 样例截图

3、讨论与思考

- 笔者对本次实验的代码风格做了全面的改变，以求提高程序效率和程序可读性：
 - 变量名和函数名力求清楚，中间变量均按照附件命名。
 - 全程采用 int 类型存储，摒弃了以前字符串的存储方式。
 - 大量采用逻辑运算和位运算，填充操作直接移位+或运算，分组操作直接移位+与运算。
 - JetBrain 产品中直接 Ctrl+Alt+L 代码格式规范化。
- 通过与 Python 中 hashlib.sha1() 的比较，该代码在运行效率和运行内存上都有提升。

23334	2022-05-21 19:49:01	Accepted	Python	43ms	10816KB	
23333	2022-05-21 19:47:22	Accepted	Python	40ms	8488KB	

图 3 该函数（下）与 hashlib.sha1()（上）比较

二、HMAC-SHA1

1、算法流程

- 密钥 K 右边填充 0 至 512 bit 得到 K^+ ;
- $S_i = K^+ \oplus ipad$, $S_0 = K^+ \oplus opad$;
- $HMAC = SHA1[S_0 || SHA1(S_i || M)]$

函数调用关系图:

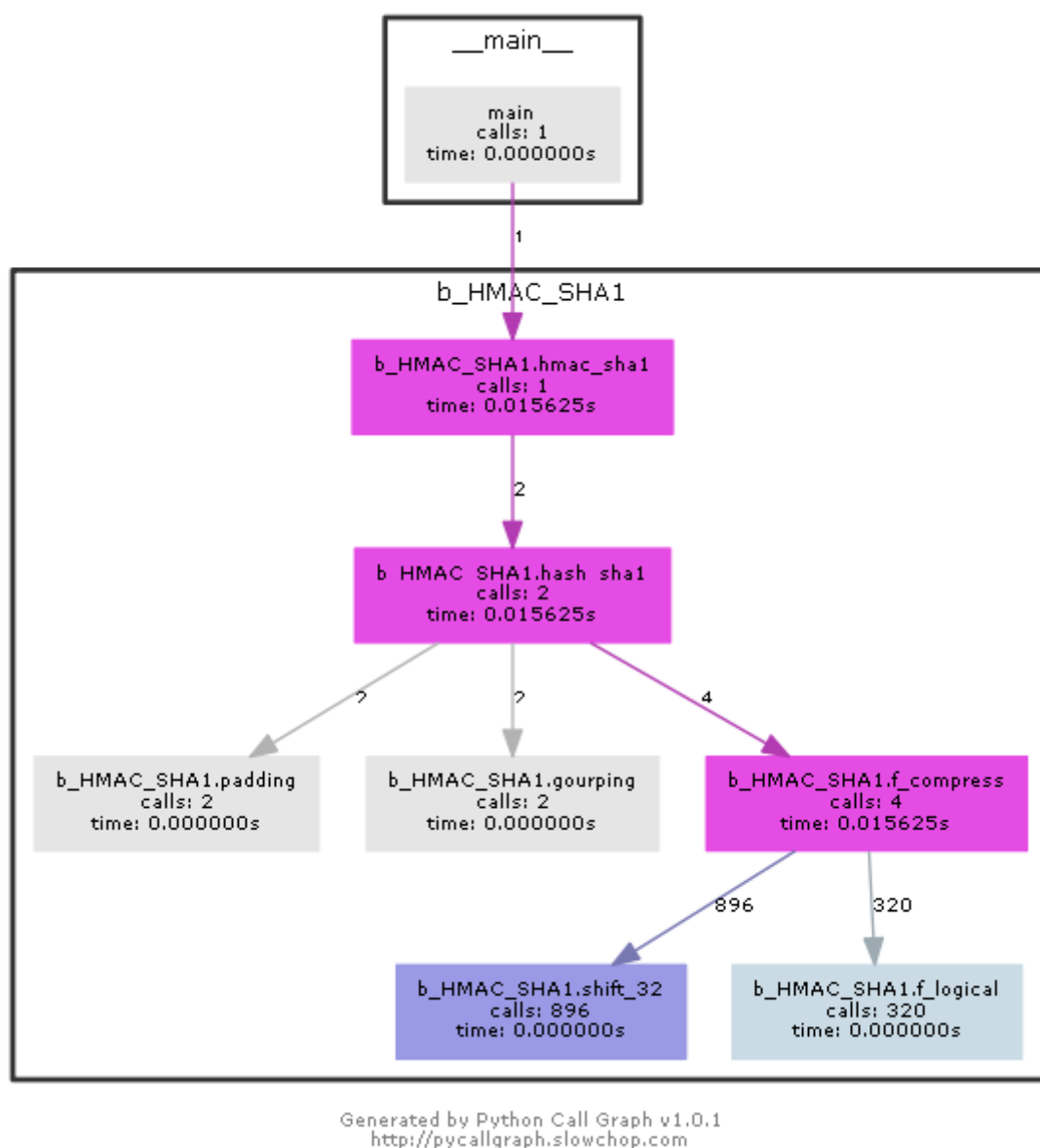


图 4 SHA1-HMAC 函数调用关系图

2、测试样例及结果截图

```
000102030405060708090A0B0C0D0E0F10111213
this is the first HMAC test.
5429b4bc6b12a5dc8d0fe73560fc092ec473e
Process finished with exit code 0
```

```
0123
Sample message for keylen<blocklen
cb5891545bb7e0267d87892cca82a3083c0a391e
Process finished with exit code 0
```

```
000102030405060708090a0b0c0d0e0f1011121a
日哦增增G，特招蓝色哦流灵，哦流灵，自解器，呼唤千组红精灵，深海的取第，拥抱海里里装兵，叠错
流，去素材，六千四直接攻击，原始生命态来竟，巨大哦流呼唤自速，鬼香鞋，魔知香鞋，样鞋，叠出
鞋，藏身在马组里，淘气精灵，让不死之身觉醒，巨大哦流，他自南都送给你，哦哦都~书呆神骑，无路
号在此降临，卫星闪亮，天下第一，要赖皮！
11094da1838e85be5f3e56b59e99c7a2c61d22d7

Process finished with exit code 0

1987
we're no strangers to love you know the rules and so do I a full commitment's
what I'm thinking of you wouldn't get this from any other guy I just wanna tell
you how I'm feeling gotta make you understand never gonna give you up never
gonna let you down never gonna run around and desert you never gonna make you
cry never gonna say good bye never gonna tell a lie and hurt you
f163cf4bb600ed1e09ae8cfdfa69a3a5b40e44b7

Process finished with exit code 0
```

图 5 SHA1-HMAC 样例截图

3、讨论与思考

- SHA1-HMAC 调用 SHA1 的函数时促使我找到了写 SHA1 时的 bug。
- 报文填充的边界条件需要认真思考（报文 bit 长度 $\equiv 448 \pmod{512}$ ）。
- 密钥 K 可以大于 512 bit（当然这次实验里没提这一点），常用的处理方式是将 K 取 Hash 后再填充。
- 附件里面的框图有误： $\text{Hash}(S_i || M)$ 后没有填充这一步。

三、SM3 - 密码杂凑算法

1、算法流程

- 报文填充、报文分组（同 SHA1）；
- 报文扩展（生成 $W[68]$, $W'[64]$ 用于压缩函数）；
- 压缩函数 CF ：输入 512 bit 报文组和 256 bit 缓冲区，输出 160 bit；
- 迭代：上个压缩函数的输出进入缓冲区作为下个压缩函数的输入；
- 最后一组压缩函数的输出作为报文的 SM3 杂凑值。

流程图：

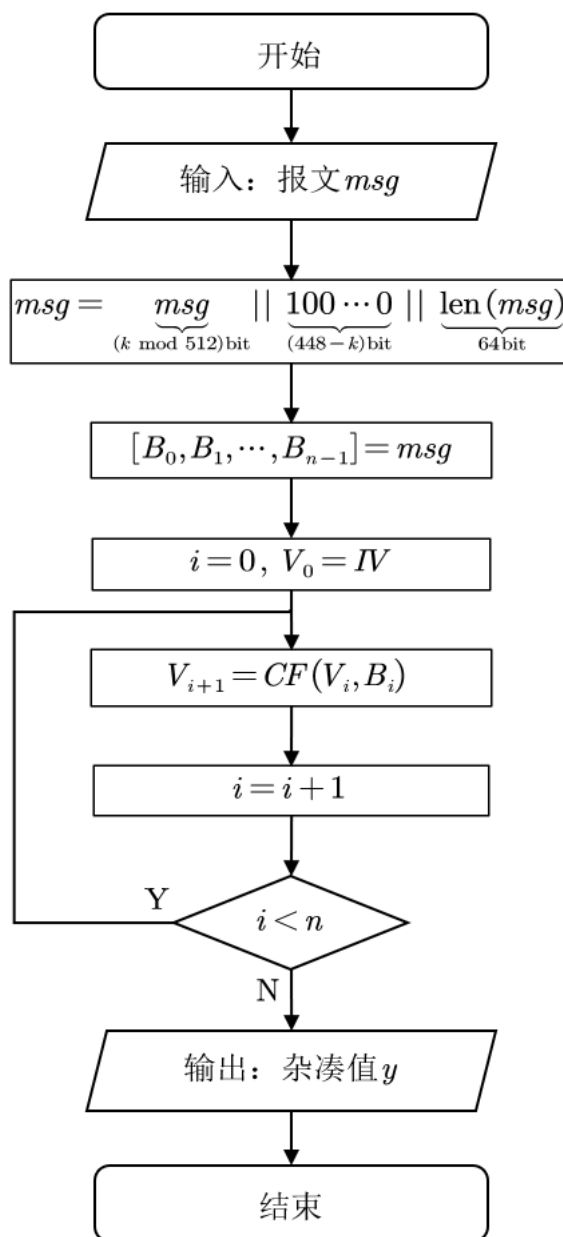


图 6 SM3 杂凑算法流程图

伪代码:

Algorithm 1: SM3压缩函数

Input: 报文块 B_i , 压缩函数缓存值 V_i

Output: 迭代缓存值 V_{i+1}

```

1. function “ $CF(B_i, V_i)$ ”
2.   /* 报文扩展 */
3.   set  $W[68], W'[63]$ 
4.    $[W_0, W_1, \dots, W_{15}] \leftarrow B_i$ 
5.   for  $j \leftarrow 16$  to  $67$  do
6.      $W_j \leftarrow P_1(W_{j-16} \oplus W_{j-9}(W_{j-3} \ll 15)) \oplus (W_{j-13} \ll 7) \oplus W_{j-6}$ 
7.   end for
8.   for  $j \leftarrow 0$  to  $63$  do
9.      $W'_j \leftarrow W_j \oplus W_{j-4}$ 
10.  end for
11.  /* 压缩 */
12.  set  $[A, B, C, D, E, F, G, H] \leftarrow V_i$ 
13.  for  $j \leftarrow 0$  to  $63$  do
14.     $SS1 \leftarrow ((A \ll 12) + E + (T_j \ll j)) \ll 7$ 
15.     $SS2 \leftarrow SS1 \oplus (A \ll 12)$ 
16.     $TT1 \leftarrow FF_j(A, B, C) + D + SS2 + W'_j$ 
17.     $TT2 \leftarrow GG_j(E, F, G) + H + SS1 + W_j$ 
18.     $D \leftarrow C$ 
19.     $C \leftarrow B \ll 9$ 
20.     $B \leftarrow A$ 
21.     $A \leftarrow TT1$ 
22.     $H \leftarrow G$ 
23.     $G \leftarrow F \ll 19$ 
24.     $F \leftarrow E$ 
25.     $E \leftarrow P_0(TT2)$ 
26.  end for
27.   $V_{i+1} \leftarrow [A, B, C, D, E, F, G, H] \oplus V_i$ 
28.  return  $V_{i+1}$ 
29. end function

```

Algorithm 2: SM3杂凑算法**Input:** 报文 msg **Output:** 杂凑值 y

```

1. function "hash_sm3( $msg$ )"
2.    $msg \leftarrow msg || 10 \cdots 0 || msg\_len$  // 填充至长度为512倍数
3.    $[B_0, B_1, \cdots, B_{n-1}] \leftarrow msg$  // 512bit一组, 分为 $n$ 组
4.   set  $V_0$ 
5.   for  $i \leftarrow 0$  to  $n - 1$  do
6.      $V_{i+1} \leftarrow CF(V_i, B_i)$ 
7.   end for
8.   return  $y \leftarrow V_n$ 
9. end function

```

函数调用关系图:

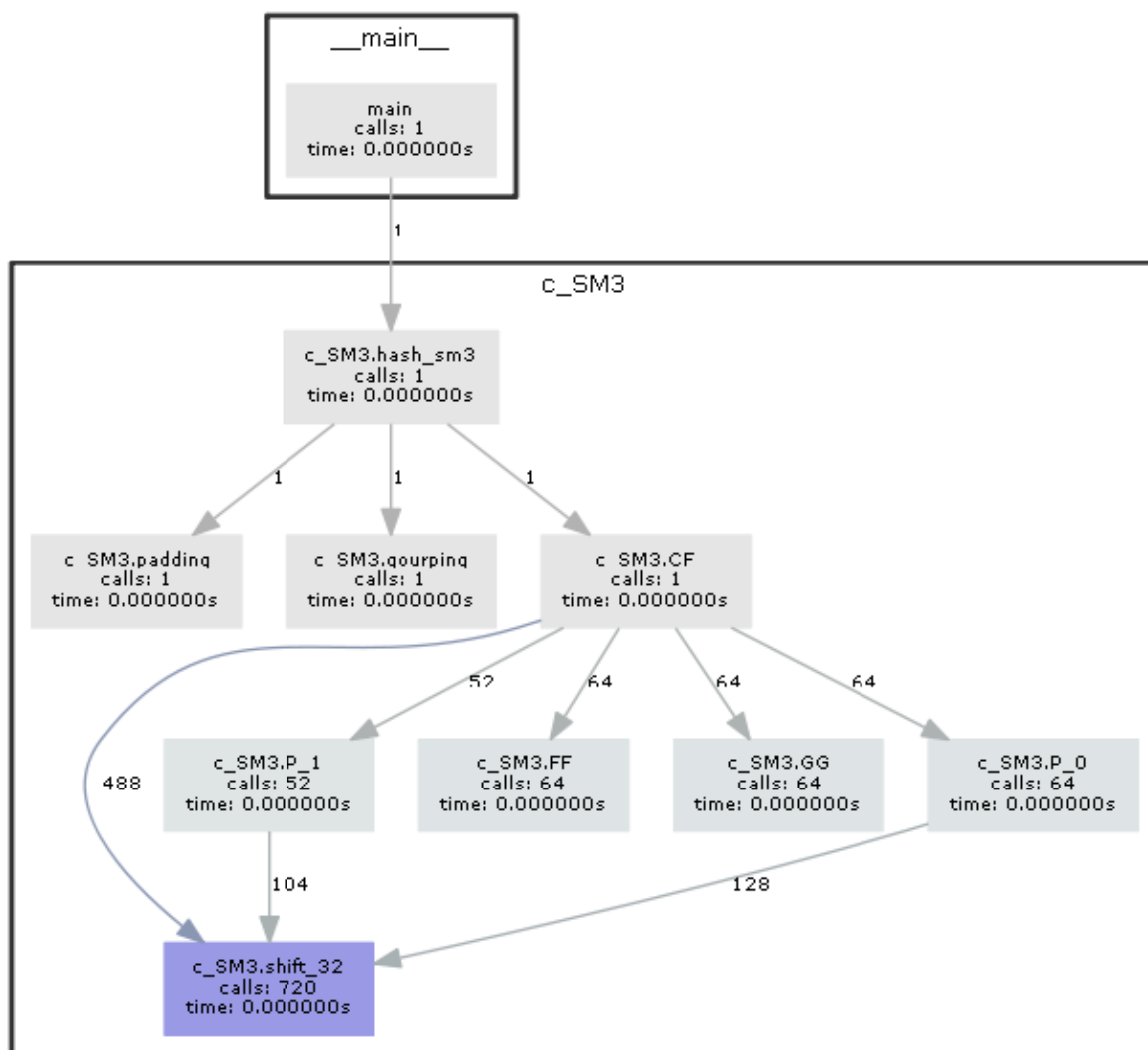


图 7 SM3 函数调用关系图

表 2 SM3 函数说明

函数名	函数功能
hash_sm3()	SM3 主函数
padding()	报文填充
grouping()	报文分组
CF()	压缩函数
P_0(), P_1()	置换函数
FF(), GG()	布尔函数
shift_32()	32 bit 数的循环移位

2、测试样例及结果截图

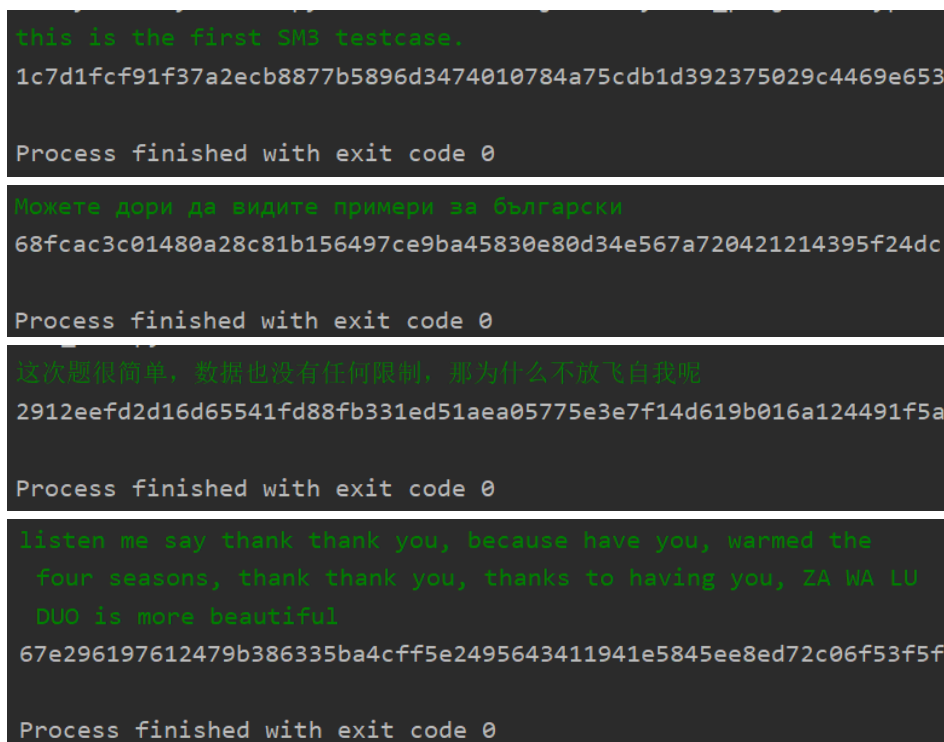


图 8 SM3 样例截图

3、讨论与思考

- SM3 密码杂凑算法是国家密码管理局 2010 年公布的中国商用密码杂凑算法标准。
- 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 bit，摘要值长度为 256 bit。

四、【选做】SHA-1 第一类生日攻击

1、算法流程

- 简单而粗鲁：
- 循环遍历 $1, 2, 3, \dots$ ，将其转化为字符串（直接 `str()`）；
- 判断其 SHA1 值是否匹配目标 SHA1 值的前 n 位，是就输出。

2、测试样例及结果截图

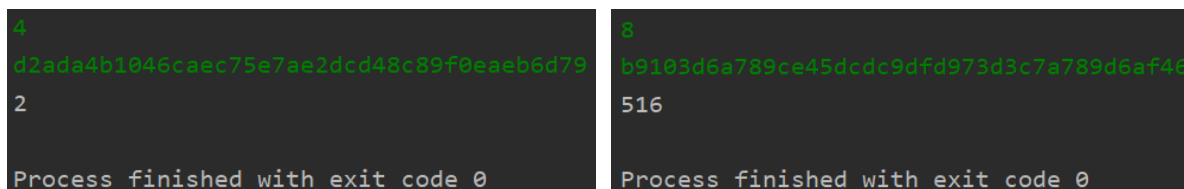


图 9 SHA1 第一类生日攻击测试样例

3、讨论与思考

- 第一类生日攻击理论推导思路很清晰，但在代码的实现上我们考虑的重点则是如何去“试”，更具体一点，放到平台上面去测试，如何提高程序效率以在规定时间内通过测试。
- 为了避免重复，采取遍历整数的方式。
- 为了提高程序运行，尽可能的减少了选取测试报文的操作，直接 `str()`选取的整数。
- OJ 给的测试样例在遍历到 1 000 000 量级时可以得到结果
- 该题可以调 `hashlib.sha1()`，而用我自己写的会超时，这也说明了“我比库还快”是狂妄之言，但运行内存确实要比库小。
- 最后一组测试样例 $n = 24$ ，也就意味着破解平均需要 $2^{24-1} = 8.38861 \times 10^6$ 次，而 1×10^6 即攻击成功说明也有运气成分。

五、【选做】SHA-1 第二类生日攻击

1、算法流程

- 生成三个原始报文（《道德经》前三章）；
- 生成变形报文：
 - ◆ 算法 1：分别在每个字与字，字与符号间加 1~ n 个“空格”；
 - ◆ 算法 2：在任意两个位置插入 1~ n 个“空格”；
 - ◆ 同时用算法 1、算法 2 生成变形报文存储在 txt 中

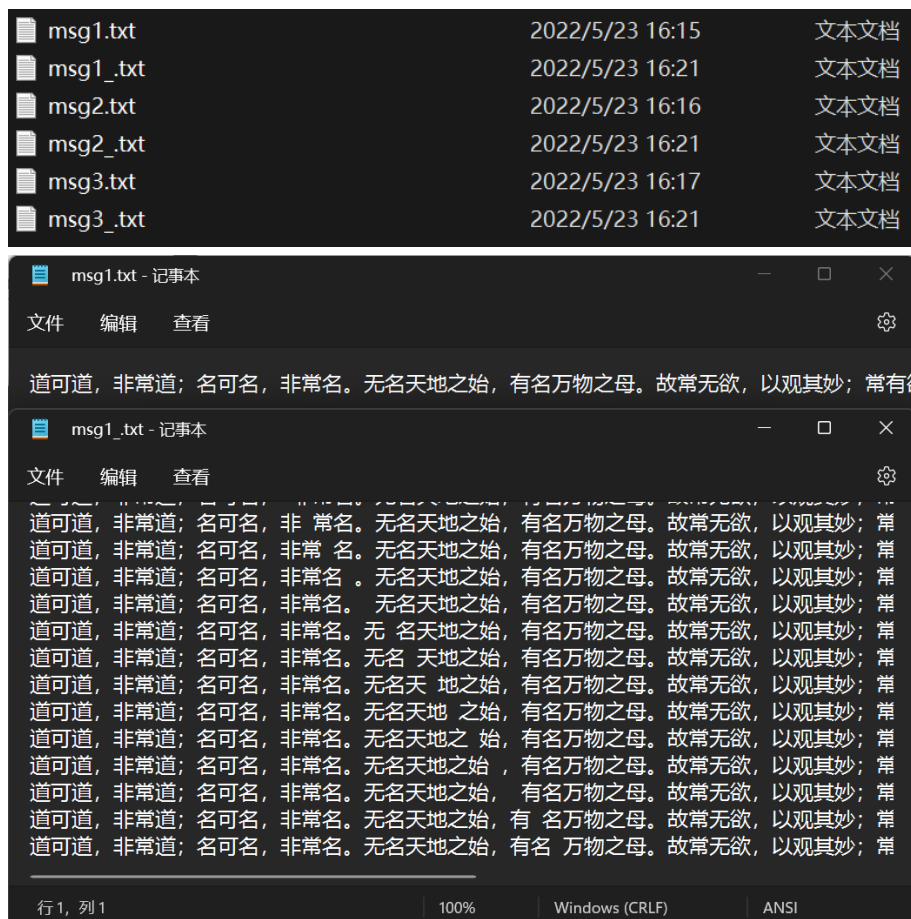


图 10 原始报文及变形报文文件

- 比较每一个变形报文，搜索到前 6 位（24 bit）相同的即输出；
- 三个报文，两两一组，生成三对前 24 bit 哈希值相同的不同变形报文。

2、测试样例及结果截图

```
原始报文1: 道可道，非常道；名可名，非常名。无名天地之始，有名万物之母。故常无欲，以观其妙；常有欲，以观其徼（jiào）。此两者同出而异名，同谓之玄，玄之又玄，众妙之门。
变形报文1: 道可道，非常道；名可名，非常名。无名天地之始，有名万物之母。故常无欲，以观其妙；常有欲，以观其徼（jiào）。此两者同出而异名，同谓之玄，玄之又玄，众妙之门。
原始报文2: 天下皆知美之为美，斯恶（è）已；皆知善之为善，斯不善已。故有无相生，难易相成，长短相较，高下相倾，音声相和（hè），前后相随。是以圣人处无为之事，行不言之教，万物作焉而不辞，生而不有，为而不恃，功成而弗居。夫（fú）唯弗居，是以不去。
变形报文2: 天下皆知美之为美，斯恶（è）已；皆知善之为善，斯不善已。故有无相生，难易相成，长短相较，高下相倾，音声相和（hè），前后相随。是以圣人处无为之事，行不言之教，万物作焉而不辞，生而不有，为而不恃，功成而弗居。夫（fú）唯弗居，是以不去。
变形报文1的SHA1值: b161e6130b8922269ef092f9464549c0ca113bd1
变形报文2的SHA1值: b161e69076942269c52d9dc106a3023be01387e6

Process finished with exit code 0
```

```
原始报文1: 道可道, 非常道; 名可名, 非常名。无名天地之始, 有名万物之母。故常无欲, 以观其妙; 常有欲, 以观其徼(jiào)。此两者同出而异名, 同谓之玄, 玄之又玄, 众妙之门。
变形报文1: 道可道, 非常道; 名可名, 非常名。无名天地之始, 有名万物之母。故常 无欲, 以观其妙; 常有欲, 以观其徼(jiào)。此两者同出而异名, 同谓之玄, 玄之又玄, 众妙之门。
原始报文2: 不尚贤, 使民不争; 不贵难得之货, 使民不为盗; 不见(xiàn)可欲, 使民心不乱。是以圣人之治, 虚其心, 实其腹; 弱其志, 强其骨。常使民无知无欲, 使夫(fú)智者不敢为也。为无为, 则无不治。
变形报文2: 不尚贤, 使民不争; 不贵难得之货, 使民不为盗; 不 见(xiàn)可欲, 使民心不乱。是以圣人之治, 虚其心, 实其腹; 弱其志, 强其骨。常使民无知无欲, 使夫(fú)智者不敢为也。为无为, 则无不治。
变形报文1的SHA1值: a9f163aee67668c5b633713bd8722317f40e5643
变形报文2的SHA1值: a9f163d79c8b12e4c7e9bfab113dab6bad4699a5

Process finished with exit code 0

原始报文1: 天下皆知美之为美, 斯恶(è)已; 皆知善之为善, 斯不善已。故有无相生, 难易相成, 长短相较, 高下相倾, 音声相和(hè), 前后相随。是以圣人处无为之事, 行不言之教, 万物作焉而不辞, 生而不有, 为而不恃, 功成而弗居。夫(fú)唯弗居, 是以不去。
变形报文1: 天下皆知美之为美, 斯恶(è)已; 皆知善之为善, 斯不善已。 故有无相生, 难易相成, 长短相较, 高下相倾, 音声相和(hè), 前后相随。是以圣人处无为之事, 行不言之教, 万物作焉 而不辞, 生而不有, 为而不恃, 功成而弗居。夫(fú)唯弗居, 是以不去。
原始报文2: 不尚贤, 使民不争; 不贵难得之货, 使民不为盗; 不见(xiàn)可欲, 使民心不乱。是以圣人之治, 虚其心, 实其腹; 弱其志, 强其骨。常使民无知无欲, 使夫(fú)智者不敢为也。为无为, 则无不治。
变形报文2: 不尚贤, 使民不争; 不贵难得之货, 使民不为盗; 不见(xiàn)可欲, 使民心不乱。是以圣人之治, 虚其心, 实其腹; 弱其志, 强其骨。常使民无知无欲, 使夫(fú)智者不敢为也。为无为, 则无不治。
变形报文1的SHA1值: 8cc107fb61cae7ddeddaf1e9659551975b8eac8b
变形报文2的SHA1值: 8cc107de83071f14345d1a30be764ffe4d1fb98d

Process finished with exit code 0
```


评测编号↓	提交时间	提交状态	代码语言	最大运行时间	最大运行内存	详细信息
23616	2022-05-23 15:34:55	Accepted	Python	21ms	8432KB	

图 11 选作二样例截图

3、讨论与思考

- 最初将填充空格的最长长度 n 设为1, 即字与字间填的均为1个“空格”。
- 这种情况下, 报文1与报文2, 报文2与报文3能在一分钟之内找到SHA1值前24 bit相同的; 但报文1与报文3搜寻完后未能找到。
- 将报文3的空格最长长度设为2, 产生更多变形报文后, 报文1与报文3也搜寻到了目标报文对。
- 第二类生日攻击能在 $2^{m/2}$ 次后能够有超过一半的概率找到报文对, 在本实验中, $m=24$, 即在寻找4096组后有超过一半的概率找到报文对。查找发现报文1生成了3160个变形报文, 报文2生成了6903个变形报文, 这与假设的概率相符, 而报文1的变形较少, 也解释了报文1与报文3在第一次未找到配对的问题。

【收获与建议】**1、收获**

- ~~加深了对哈希函数的理解。~~
- ~~提高了对 Python 的熟练度。~~
- ~~巩固了排版能力。~~
- 收获了劳累与繁忙。

2、建议

- 无。

【思考题】

- 无