

MD5 Collision Attack

简介

一个安全的单向哈希函数需要满足两个特性：单向特性和抗碰撞特性。单向属性确保给定一个哈希值 h ，找到一个输入 M 在计算上是不可行的，这样的 $\text{hash}(M)=h$ 。抗碰撞特性保证了找到两个不同的输入在计算上是不可行的 $M1$ 和 $M2$ ，从而生成 $\text{hash}(M1) = \text{hash}(M2)$ 。

一些广泛使用的单向哈希函数很难保持抗碰撞性能。在2004年的加密货币的会议上，王晓云和合著者展示了对MD5的碰撞攻击。2017年2月，CWI阿姆斯特丹和谷歌研究公司宣布了冲击攻击，打破了SHA1的抗碰撞特性。虽然许多学生在理解单向属性的重要性方面没有什么困难，但他们不容易理解为什么抗碰撞特性是必要的，以及这些攻击会造成什么影响。

本实验室的学习目标是让学生真正了解碰撞攻击的影响，并亲眼所见，如果一个广泛使用的单向哈希函数的抗碰撞特性被破坏，会造成什么损害。为了实现这个目标，学生需要对MD5哈希函数发起实际的碰撞攻击。使用这些攻击，学生应该能够创建两个不同的程序，共享相同的MD5散列，但有完全不同的行为。本实验室涵盖了以下所描述的一些主题：

- 单向哈希函数，MD5
- 抗碰撞性能
- 沙漏碰撞袭击

环境：实验室使用了一个名为“Fast MD5 Collision Generation”的工具，这是由马克·史蒂文斯编写的。二进制文件的名称在虚拟机中称为md5colgen，它安装在/usr/bin文件夹中。如果它不在那里，你可以从实验室的网站

上下载它(在Labsetup.zip)。如果您有兴趣将该工具安装到您自己的机器上，您可以直接从 <https://www.win.tue.nl/hashclash/> 下载。

任务1:使用相同的MD5散列生成两个不同的文件

在此任务中，我们将生成具有相同MD5哈希值的两个不同文件。这两个文件的开头部分需要是相同的，即，它们共享相同的prefix。我们可以使用md5colgen程序来实现这一点，它允许我们提供任何任意内容的prefix文件。该程序的工作方式如图1所示。下面的命令对于一个给定的文件prefix.txt生成两个输出文件 out1.bin 和 out2.bin，先创建文件：

```
md5collgen -p prefix.txt -o out1.bin out2.bin
```



图1：从前缀生成的MD5碰撞

```
echo "this is md5 lab" > prefix.txt
```

```
md5collgen -p prefix.txt -o out1.bin out2.bin
```

```
# seed @ VM in ~/md5 [7:21:44]
$ echo "this is md5 lab" > prefix.txt

# seed @ VM in ~/md5 [7:23:42]
$ md5collgen -p prefix.txt -o out1.bin out2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1.bin' and 'out2.bin'
Using prefixfile: 'prefix.txt'
Using initial value: ed028c51d63cb39a09956bbf4c7046cb

Generating first block: .....
Generating second block: W...
Running time: 5.78362 s
```

我们可以使用diff命令来检查输出文件是否不同。我们还可以使用md5sum命令来检查每个输出文件的MD5哈希，请参阅以下命令。

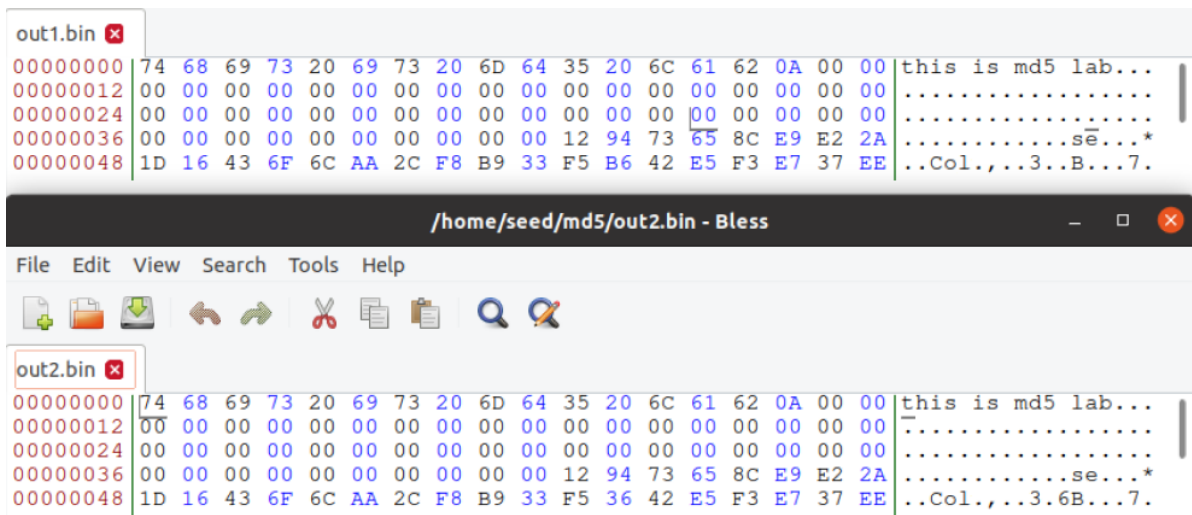
```
diff out1.bin out2.bin
md5sum out1.bin
md5sum out2.bin
```

```
# seed @ VM in ~/md5 [7:41:14]
$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ

# seed @ VM in ~/md5 [7:41:27] C:1
$ md5sum out1.bin ; md5sum out2.bin
110f20d7cc0aeeb99fe9bca82216aa56 out1.bin
110f20d7cc0aeeb99fe9bca82216aa56 out2.bin
```

可见两个输出文件相同

自1。箱和出2。bin是二进制的，我们不能使用cat之类的文本查看器程序来查看它们；我们需要使用二进制编辑器来查看（和编辑）它们。我们已经在虚拟机中安装了一个名为bless的十六进制编辑器软件。请使用这样的编辑器来查看这两个输出文件，并描述您的观察结果。



这两个bin文件

此外，您还应回答以下问题：

1. Q1:如果prefix文件的长度不是64的倍数，那么会发生什么呢？

它将用零填充。

先创建了一个文件 `him.txt`

```
echo "hello" > him.txt
```

并使用

```
truncate -s 10 him.txt
```

截断它，取前十个字符，不足的填充0。运行

```
md5collgen -p him.txt -o hi1 hi2
```

```
# seed @ VM in ~/md5 [7:55:58]
$ echo "hello" > him.txt

# seed @ VM in ~/md5 [7:57:33]
$ truncate -s 10 him.txt

# seed @ VM in ~/md5 [7:58:02]
$ md5collgen -p him.txt -o hi1 hi2
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

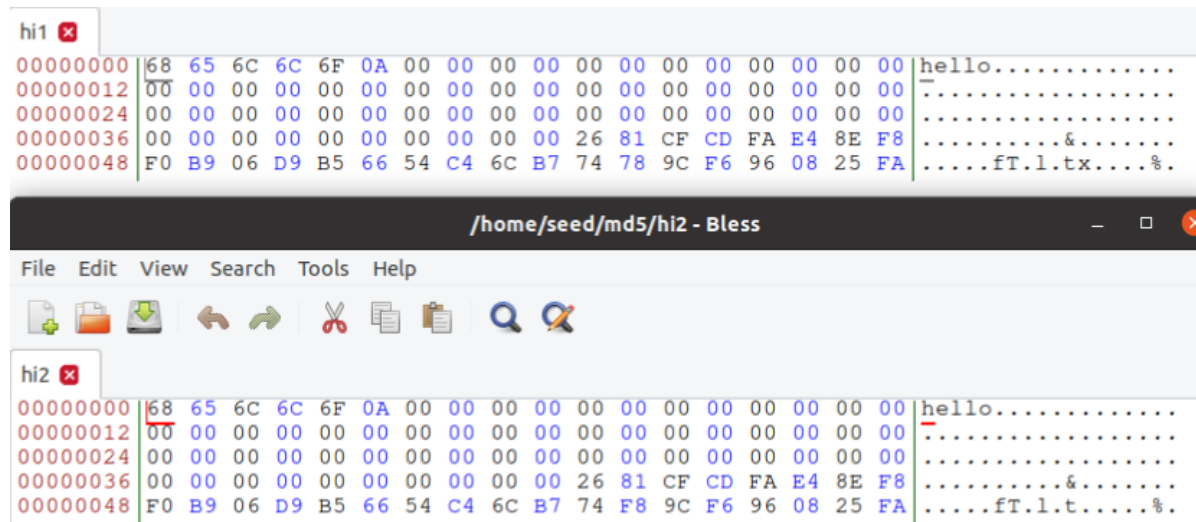
Using output filenames: 'hi1' and 'hi2'
Using prefixfile: 'him.txt'
Using initial value: 91ele88548a4bef54d09a7c23bd03682

Generating first block: .....
Generating second block: S00.....
Running time: 4.44382 s
```

查看结果使用

```
bless hi1
```

我们可以看到前64个（十六进制的40）已填充零。这是因为MD5处理大小64字节的块。



2. 创建一个恰好为64字节的文件，然后再次运行碰撞工具，看看会发生什么。

前64个没有观察到零填充，但会填充接下来的 64 字节，即从字节偏移 40 到 7F。

向文件中输入63个字符，带上本来的结束符刚好64个，为了确保万无一失，可以使用truncate截取，前提是你的字符数超过64个，否则truncate也会自动填充0，造成干扰。生成文件

```
echo "$(python3 -c 'print("A"*63)')' >> prefix_64.txt
truncate -s 64 prefix_64.txt
md5collgen -p prefix_64.txt -o pf1 pf2
```

```
# seed @ VM in ~/md5 [8:15:19]
$ echo "$(python3 -c 'print("A"*63)')' >> prefix_64.txt
truncate -s 64 prefix_64.txt
md5collgen -p prefix_64.txt -o pf1 pf2
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)
```

```
Using output filenames: 'pf1' and 'pf2'
Using prefixfile: 'prefix_64.txt'
Using initial value: d3fcd94d23f15d68c7ce05dc545a880f
```

```
Generating first block: .....
Generating second block: S11.....
Running time: 5.16373 s
```

命令，查看输出信息bless pf1 pf2

```
pf1 x
00000000 0A 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 .AAAAAAAAAAAAAAAAAAAA
00000012 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAAA
00000024 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAAA
00000036 41 41 41 41 41 41 41 41 41 41 41 41 FE 25 EA A4 7B 55 1E AC AAAAAAAAAAA.%..{U..
00000048 27 BD D2 61 7B F1 D1 F2 E5 43 A8 51 93 B2 B7 42 61 DE '...a{....C.Q...Ba.
0000005a 83 41 52 12 AD 0F AD C0 FC D1 82 47 52 87 0B 20 52 5A .AR.....GR.. RZ
0000006c 0F A8 6B 53 2B 7C 41 1B 8C 75 EE F7 AE A0 B6 B1 35 27 ..kS+|A..u.....5'
0000007e AE EA 03 7C 2C F9 61 62 2E C6 F9 FD 8C EC 5E B1 C1 48 ...|,.ab.....^..H
00000090 A3 B6 E4 04 07 9E 0B 5E 6C D9 8F 5F 4B 20 2D C1 F2 18 .....^l..._K -...
000000a2 B1 2D 96 27 7B 96 2C 98 6B D1 04 00 00 0F 8E D9 EE 35 .-.'{.,.k.....5
000000b4 EA 18 F3 6F 39 7E 5D C1 EF 11 BA 00 ...o9~].....

/home/seed/md5/pf2 - Bless

File Edit View Search Tools Help

pf2 x
00000000 0A 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 .AAAAAAAAAAAAAAAAAAAA
00000012 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAAA
00000024 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAAA
00000036 41 41 41 41 41 41 41 41 41 41 41 41 FE 25 EA A4 7B 55 1E AC AAAAAAAAAAA.%..{U..
00000048 27 BD D2 61 7B F1 D1 F2 E5 43 A8 D1 93 B2 B7 42 61 DE '...a{....C.....Ba.
0000005a 83 41 52 12 AD 0F AD C0 FC D1 82 47 52 87 0B 20 52 5A .AR.....GR.. RZ
0000006c 0F 28 6C 53 2B 7C 41 1B 8C 75 EE F7 AE A0 B6 31 35 27 .(1S+|A..u.....15'
0000007e AE EA 03 7C 2C F9 61 62 2E C6 F9 FD 8C EC 5E B1 C1 48 ...|,.ab.....^..H
00000090 A3 B6 E4 84 07 9E 0B 5E 6C D9 8F 5F 4B 20 2D C1 F2 18 .....^l..._K -...
000000a2 B1 2D 96 27 7B 96 2C 98 6B D1 04 80 FF 0E 8E D9 EE 35 .-.'{.,.k.....5
000000b4 EA 18 F3 6F 39 7E 5D 41 EF 11 BA 00 ...o9~]A....
```

3. md5colgen为这两个输出文件生成的数据（128字节）是否完全不同？请识别所有不同的字节。

不，不是所有字节都不同。在上一次情况下，字节仅在个别位置处不同。在多次试验之后，发现这些差异的地方不是固定的。

1	0A 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	1	0A 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
2	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	2	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
3	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	3	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
4	93 B2 B7 42 61 DE 83 41 52 12 AD 0F AD C0 FC D1 82 47 52 87 0B 20 52 5A 0F A8 6B 53	4	93 B2 B7 42 61 DE 83 41 52 12 AD 0F AD C0 FC D1 82 47 52 87 0B 20 52 5A 0F 28 6C 53
5	2B 7C 41 1B 8C 75 EE F7 AE A0 B6 31 35 27 AE EA 03 7C 2C F9 61 62 2E C6 F9 FD 8C EC	5	2B 7C 41 1B 8C 75 EE F7 AE A0 B6 31 35 27 AE EA 03 7C 2C F9 61 62 2E C6 F9 FD 8C EC
6	5E B1 C1 48 A3 B6 E4 84 07 9E 0B 5E 6C D9 8F 5F 4B 20 2D C1 F2 18 B1 2D 96 27 7B 96	6	5E B1 C1 48 A3 B6 E4 84 07 9E 0B 5E 6C D9 8F 5F 4B 20 2D C1 F2 18 B1 2D 96 27 7B 96
7	2C 98 6B D1 04 00 00 0F 8E D9 EE 35 EA 18 F3 6F 39 7E 5D C1 EF 11 BA 00	7	2C 98 6B D1 04 80 FF 0E 8E D9 EE 35 EA 18 F3 6F 39 7E 5D 41 EF 11 BA 00

任务2:了解MD5的属性

在这个任务中，我们将尝试理解MD5算法的一些特性。这些特性对我们在这个实验室进行进一步的任务很重要。

MD5是一个相当复杂的算法，但从非常高的层面，它不是那么复杂。如图2所示，MD5将输入数据划分为64个字节的块，然后在这些块上迭代计算散列。MD5算法的核心是一个压缩函数，它接受两个输入，一个64字节的数据块和上一次迭代的结果。压缩函数产生一个128位的IHV，它代表“中级哈希值”(Intermediate Hash Value)；然后将此输出输入到下一个迭代中。如果当前的迭代是最后一个迭代，则IHV将是最终的哈希值。第一次迭代的IHV输入(IHV0)是一个固定的值。

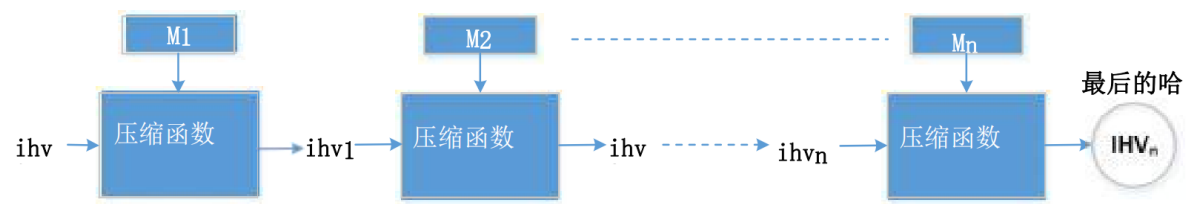


图2：MD5算法的工作原理

基于 MD5 算法的工作原理，我们可以推导出一个属性：- 给定两个输入 M , N 如果 $MD5(M) = MD5(N)$, 那么对于任何输入 T , $MD5(M || T) = MD5(N || T)$ 。因此，将特定的suffix添加到具有相同 MD5 散列的任何两个不同消息中，通过连接原始消息和suffix消息，得到两个新的更长消息，这两个消息也具有相同的 MD5 散列。

也就是说，如果输入m和n具有相同的哈希，则将相同的suffix添加到它们将导致两个具有相同哈希值的输出。此属性不仅适用于MD5哈希算法，还适用于许多其他哈希算法。您在此任务中的工作是设计实验，以证明这一属性持有MD5。

您可以使用cat命令将两个文件（二进制文件或文本文件）连接为一个文件

```
cat file1 file2 > file3
```

向文件写入信息，并生成两份文件

```
echo "skprimin" > prefix.txt
md5collgen -p prefix.txt -o f1 f2
```

```
# seed @ VM in ~/md5 [8:41:18]
$ echo "skprimin" > prefix.txt
md5collgen -p prefix.txt -o f1 f2
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)
```

```
Using output filenames: 'f1' and 'f2'
Using prefixfile: 'prefix.txt'
Using initial value: 035f06452b7ed1dd3db3512f55247fa7
```

```
Generating first block: .
Generating second block: S01.....
Running time: 1.33336 s
```

创建suffix，并添加

```
echo "btfjade" > suffix

cat f1 suffix > fs1
cat f2 suffix > fs2
```

```
# seed @ VM in ~/md5 [8:42:31]
$ echo "btfjade" > suffix

cat f1 suffix > fs1
cat f2 suffix > fs2
```

比较

```
md5sum f1 ; md5sum f2 ; md5sum fs1 ; md5sum fs2;
```

我们可以看到MD5哈希值保持相同。

在此任务中，您将得到以下C程序。您的工作是创建这个程序的两个不同版本，这样它们的xyz数组的内容就不同了，但可执行文件的哈希值是相同的。

```
#include <stdio.h>  
  
unsigned char xyz[200] = {  
    /*此数组的实际内容取决于您 */  
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',  
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',  
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',  
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',  
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',  
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',  
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',  
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',  
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',  
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',  
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',  
    'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',  
};  
  
int main()  
{  
    int i;  
    for (i = 0; i < 200; i++)  
    {  
        printf("%x", xyz[i]);  
    }  
    printf("\n");  
}
```

您可以选择在源代码级别工作，即生成上述C程序的两个版本，这样在编译后，它们对应的可执行文件具有相同的MD5哈希值。然而，直接在二进制级别上工作可能更容易。您可以在xyz数组中放一些

任意的值，将上面的代码编译为二进制代码。然后，您可以使用十六进制编辑器工具直接在二进制文件中修改xyz数组的内容。

查找数组的内容存储在二进制文件中的位置并不容易。但是，如果我们用一些固定的值来填充数组，我们可以很容易地在二进制文件中找到它们。例如，下面的代码用0x41填充数组，这是字母A的ASCII值。在二进制文件中找到200个a并不难。

指南：在数组内部，我们可以找到两个位置，在那里我们可以将可执行文件分为三个部分：一个prefix、一个128字节的区域和一个suffix。该prefix的长度需要是64个字节的倍数。有关该文件的分割方式的说明，请参见图3。

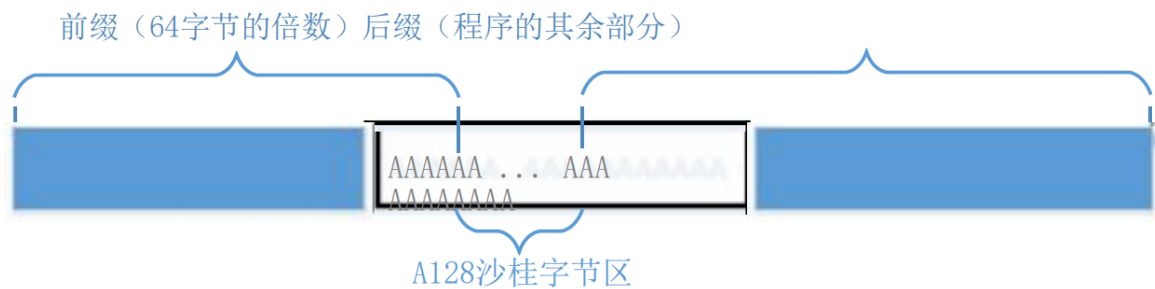


图3：将可执行文件分成三部分。

我们可以在prefix上运行md5colgen来生成两个具有相同MD5哈希值的输出。让我们使用P和Q来表示这些输出的第二部分（每个部分有128个字节）（即，prefix后面的部分）。因此，我们有以下内容：

$$\text{MD5 (prefix k P)} = \text{MD5 (prefix k Q)}$$

基于MD5的属性，我们知道，如果我们在上面的两个输出中附加相同的suffix，所得到的数据也将具有相同的哈希值。基本上，以下内容对任何suffix都适用：

$$\text{MD5 (prefix k P k suffix)} = \text{MD5 (prefix k Q k suffix)}$$

因此，我们只需要使用P和Q来替换数组的128个字节（在两个分法点之间），并且我们将能够创建两个具有相同哈希值的二进制程序。它们的结果是不同的，因为它们每个人都打印出自己的数组，其中有不同的内容。

工具：您可以使用bless来查看二进制可执行文件，并找到数组的位置。对于分割一个二进制文件，我们可以使用一些工具来从一个特定的位置分割一个文件。头部和尾部的命令就是如此有用的工具。你可以看看他们的手册来学习如何使用它们。我们给出以下三个例子：

```
head -c 3200 a.out > prefix
tail -c 100 a.out > suffix
tail -c +3300 a.out > suffix
```

上面的第一个命令保存了a.out的第3200字节到prefix。第二个命令将最后100个字节保存到suffix。第三个命令将3300个字节的数据保存a.out的结尾到suffix。通过这两个命令，我们可以将二进制文件划分为从任何位置的碎片。如果我们需要一起粘合某些作品，我们可以使用CAT命令。

如果您使用Wells将数据块从一个二进制文件复制到另一个二进制文件到另一个文件中，请菜单项“编辑 -> 选择范围”非常方便，因为您可以使用起始点和a选择一块数据块范围，而不是手动计算选择了多少字节。

生成前缀

编译

```
gcc t3.c -o a.out
```

```
# seed @ VM in ~/md5 [10:28:30]
```

```
$ gcc t3.c -o a.out
```

查看a.out文件，发现前缀在0x3020出结束

```
00002f8f 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 08 40 00 00 .....@..
0000300c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00003020 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAAAA
00003034 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAAAA
00003048 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAAAA
0000305c 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAAAA
00003070 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAAAA
00003084 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAAAA
00003098 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAAAA
000030ac 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAAAA
000030c0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAAAA
000030d4 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAAAA
000030e8 47 43 43 3a 20 28 55 62 75 6e 74 75 20 39 2e 33 2e 30 2d 31 GCC: (Ubuntu 9.3.0-1
000030cf 37 75 62 75 6e 74 75 31 7e 32 30 2e 30 34 29 20 39 2e 33 2e 7ubuntu1~20.04) 9.3.
00003100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

取64 (0x40) 的倍数0x3040。十进制12352

```
head -c 12352 a.out > prefix
md5collgen -p prefix -o a b
```

```
# seed @ VM in ~/md5 [20:39:49]
```

```
$ head -c 12352 a.out > prefix
```

```
md5collgen -p prefix -o a b
```

```
MD5 collision generator v1.5
```

```
by Marc Stevens (http://www.win.tue.nl/hashclash/)
```

```
Using output filenames: 'a' and 'b'
```

```
Using prefixfile: 'prefix'
```

```
Using initial value: b5f608a646f364988fa0af20a2f3a19d
```

```
Generating first block: .....
```

```
Generating second block: W.
```

```
Running time: 7.41992 s
```

添加后缀

再次查看a.out，发现原来的后缀于0x30e8开始，但我们需要再截取一部分数组的内容

```
000030c0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAAAA
000030d4 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAAAA
000030e8 47 43 43 3a 20 28 55 62 75 6e 74 75 20 39 2e 33 2e 30 2d 31 GCC: (Ubuntu 9.3.0-1
000030cf 37 75 62 75 6e 74 75 31 7e 32 30 2e 30 34 29 20 39 2e 33 2e 7ubuntu1~20.04) 9.3.
00003100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00003110 30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00003124 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 03 00 01 00 .....
00003138 18 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000314c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

保留原始二进制文件中 $12352 + 128 = 12,480$ 之后的字节作为suffix，实际上需要加一

```
tail -c +12481 a.out > suffix
```


bleess 查看发现在此中间0x3040 → 0x30bf有 128个字符，正是其自动生成的部分

```
0000300a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00003020 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAAAA
00003036 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAA:F.v*.csc...
0000304c 1C 26 8C 2F C1 34 5A 57 D2 9F D6 94 3C CE 40 8A A8 1A BC F6 69 AA .&./4ZW...<.@....i.
00003062 CB FB CA AB 47 BF 67 24 54 3B C5 ED 65 FF 3D 04 B0 CE 28 A5 75 04 ....G.g$T;..e=...(u.
00003078 7A 40 B7 32 44 0C DA F9 01 29 4B F3 79 C2 F3 EA 4F 6F 10 E6 EE 59 z@.2D....)K.y...Oo...Y
0000308e 6B D3 4B F0 27 47 89 56 C3 4D F6 24 E2 36 4D 25 7E BE 75 BF BF 83 k.K.'G.V.M$.6M%~.u...
000030a4 27 32 F5 46 DA 89 55 92 99 18 8E FD 1C ED C0 EA 76 C9 63 21 89 1C '2.F..U.....v.c!...
000030ba 8A 78 0A E2 E9 E9 41 41 41 41 41 41 41 41 41 41 41 41 41 41 .x...AAAAAAAAAAAAAAAA
000030d0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAAAA
000030e6 41 41 47 43 43 3A 20 28 55 62 75 6E 74 75 20 39 2E 33 2E 30 2D 31 AAGCC: (Ubuntu 9.3.0-1
000030cf 37 75 62 75 6E 74 75 31 7E 32 30 2E 30 34 29 20 39 2E 33 2E 30 00 7ubuntu1~20.04) 9.3.0.
```

我们对结果MD5一下

```
echo $(./m) | md5sum;echo $(./n) | md5sum
```

```
# seed @ VM in ~/md5 [21:05:13]
$ echo $(./m) | md5sum;echo $(./n) | md5sum
fcb138f1267d396c7f261d819a5d183a -
e53b817825ec85be27917b829207b9c5 -
```

发现内容实际上还是有差别的

但是两个可执行文件的MD5是相同的

```
md5sum m; md5sum n
```

```
# seed @ VM in ~/md5 [21:05:31]
$ md5sum m; md5sum n
b2b68a781cfd6d0639289094850beffe m
b2b68a781cfd6d0639289094850beffe n
```

任务4：使两个程序表现不同

在以前的任务中，我们已成功创建了两个具有相同MD5哈希的程序，但其行为是不同的。但是，它们的差异仅在他们打印出的数据中；它们仍然执行相同的指令序列。在这项任务中，我们希望实现更具意义和更有意义的东西。

假设您已创建一个软件，这些软件是好事。您将软件发送到可信任的权限以获得认证。该权限对您的软件进行了全面的测试，并得出结论，您的软件确实做得很好。权威将向您展示您的证明，指出您的计划是好的。为了防止您在获得证书后更改程序，您的程序的MD5哈希值也包含在证书中；证书由权限签署，因此您无法在证书或您的程序上更改任何内容，而不会呈现签名无效。

您想通过管理权限获得您的恶意软件，但如果您只需将恶意软件发送到权限，因此您有零机会实现这一目标。但是，您注意到该权限使用MD5来生成散列值。你有一个想法。您计划准备两个不同的程序。一个程序将始终执行良好的指令并做好事，而其他程序将执行恶意指令并导致损坏。您可以获得这两个程序以共享相同的MD5哈希值。然后，您将良性版本发送给证书的权限。由于此版本做得好，它将通过证书，并且您将获得包含良性计划的哈希值的证书。由于您的恶意程序具有相同的哈希值，因此该证书对您的恶意计划也有效。因此，您已成功获得了对恶意计划的有效证明。如果其他人相信当局发出的证书，他们将下载您的恶意计划。

此任务的目的是启动上述攻击。即，您需要创建两个共享相同MD5哈希的程序。但是，一个程序将始终执行良性说明，而其他程序将执行恶意指令。在您的工作中，执行良性/恶意指令并不重要；它表明这两个程序执行的指令是不同的。

指导方针。创建两个完全不同的程序，产生相同的MD5哈希值非常硬。MD5CollGen产生的两个哈希碰撞程序需要共享相同的预先预定x;此外，正如我们从上一个任务中看到的那样，如果我们需要将一些有意义的SUF X添加到MD5CollGen产生的输出，则添加到两个程序的SUF x也需要相同。这些是我们使用的MD5碰撞生成程序的局限性。虽然还有其他更复杂和更高级的工具，可以提升一些限制，例如接受两个不同的预先xes，它们需要更多的计算能力，因此它们超出了此实验室的范围。我们需要在限制内生成两个不同的程序。

有很多方法可以实现上述目标。我们提供一种方法作为参考，但鼓励学生提出自己的想法。教师可以考虑获得自己的想法的奖励学生。在我们的方法中，我们创建了两个阵列x和y.我们比较这两个阵列的内容;如果他们是相同的，良性代码被执行;否则，执行恶意代码。请参阅以下伪代码：

```
Array X;  
Array Y;  
main()  
{  
    if(X's contents and Y's contents are the same)  
        run benign code;  
    else  
        run malicious code;  
    return;  
}
```

我们可以使用一些值初始化阵列x和y，这些值可以帮助我们在exe可变的二进制文件中找到他们的位置。我们的工作更改这两个阵列的内容，因此我们可以生成具有相同MD5哈希的两个不同的版本。在一个版本中，x和y的内容相同，因此执行良性代码;在其他版本中，x和y的内容是不同的，因此执行恶意代码。我们可以使用类似于任务3中使用的技术来实现这一目标。

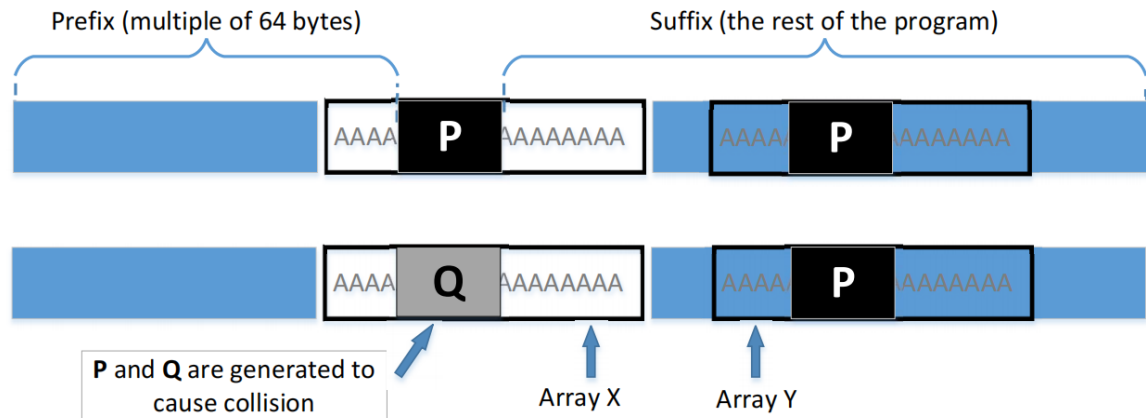


图4一种具有不同行为的两个哈希碰撞程序的方法

从图4中，我们知道这两个二进制文件具有相同的MD5哈希值，只要P和Q被相应地生成。在第一个版本中，我们使数组x和y的内容相同，而在第二个版本中，我们使它们的内容不同。因此，我们唯一需要改变的是这两个数组的内容，并且不需要改变程序的逻辑。

构建C程序

填充数组脚本，a、b数组分别填充100个'A'：

```
python3 -c "print('\n'+'\n','\n'.join(x for x in ['A']*200)+'\n')"
```

```
#include<stdio.h>
```



```
# seed @ VM in ~/md5 [21:01:13]
$ bless b.out
```

a数组第一个元素在0x3020, 不是64的倍数了, 为了能方便定位, 还要加上0x20 = 32个'A'也包括进去。

[illegible]

截取第一个数组之前的部分以及32个 **a** , $0x3020 + 32 = 12352$, 并生成两个不同的文件

```
head -c 12352 b.out > prefix
md5collgen -p prefix -o p q
```

```
# seed @ VM in ~/md5 [22:09:00] C:130
$ head -c 12352 b.out > prefix
md5collgen -p prefix -o p q
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)
```

```
Using output filenames: 'p' and 'q'
Using prefixfile: 'prefix'
Using initial value: 43083ff8a04572a7b796d81cda5867c8
```

```
Generating first block: .....
Generating second block: S00.....
Running time: 6.80648 s
```

截取后缀

查看发现在0x31c8处开始后面内容，在此截取后部分内容

[illegible]

0x31c8 = 12,744, 加一为12745

```
tail -c +12745 b.out > suffix
```

填充中部

把其中一个新前缀p的后32(字符A) + 128(md5填充物) = 160个字节截取出来:

```
tail -c 160 p > middle
```

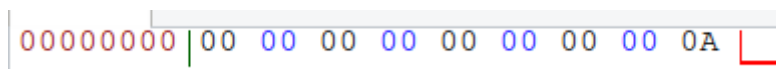
```
# seed @ VM in ~/md5 [22:10:41]
$ tail -c 160 p > middle
```


数组的长度是200, $200 - 160 = 40$, 所以还需要填充40个字节给a和b数组,

两个数组相差 $0x3100 - 0x3020 = 0xE0 = 224$ 字节, 不是200字节, 因此需要在a数组和b数组之间插入24个 `0x00` :

使用python生成40个字符串, 由于python生成时会在末尾自动加上 `0A`, 因此我们需要截取生成的文件

```
python3 -c "print('\x00'*40)" > tmp
```



A hex dump visualization of the file 'tmp'. It shows a sequence of 40 null bytes (00) followed by a 0A byte. The first 8 null bytes are grouped together, and the final 0A byte is highlighted with a red bracket.

```
head -c 40 tmp > m40
```

```
head -c 24 tmp > m24
```

```
# seed @ VM in ~/md5 [22:11:34]
$ python3 -c "print('\x00'*40)" > tmp

# seed @ VM in ~/md5 [22:13:00]
$ head -c 40 tmp > m40
head -c 24 tmp > m24
```

开始拼接

前缀	a数组末尾填充	数组间隔 24	中缀主体 192 (数组b主体)	a数组末尾填充	后缀
p	m40	m24	middle	m40	suffix
q	m40	m24	middle	m40	suffix

```
cat p m40 m24 middle m40 suffix > s
```

```
cat q m40 m24 middle m40 suffix > k
```

```
# seed @ VM in ~/md5 [22:13:39]
$ cat p m40 m24 middle m40 suffix > s
cat q m40 m24 middle m40 suffix > k
```

赋权运行

直接运行可能会出现权限问题

```
# seed @ VM in ~/md5 [21:58:19]
$ ./s; ./k
zsh: permission denied: ./s
zsh: permission denied: ./k
```

```
chmod +x s
chmod +x k

./s; ./k
```

运行发现两个文件的运行结果不同

```
# seed @ VM in ~/md5 [22:17:25]
$ chmod +x s
chmod +x k

# seed @ VM in ~/md5 [22:17:40]
$ ./s; ./k
benign code!
malicious code!
```

然而两个文件的md5值却是相同的

```
md5sum s; md5sum k
```

```
# seed @ VM in ~/md5 [22:17:47]
$ md5sum s; md5sum k
c31c2ca8f718047c92af87af40ebd5f2  s
c31c2ca8f718047c92af87af40ebd5f2  k
```