

哈希函数，MD5，碰撞攻击和随机数

18308045 Zhengyang Gu

December 26, 2020

Contents

1	摘要	3
2	哈希函数	3
2.1	定义	3
2.2	安全性	3
2.2.1	原像抗性	3
2.2.2	次原像抗性	4
2.2.3	碰撞抗性	5
3	MD5	5
3.1	算法描述	5
3.1.1	Step 1: Append Padding Bits	5
3.1.2	Step 2: Append Length	6
3.1.3	Step 3: Initialize MD Buffer	6
3.1.4	Step 4: Process Message in 512-bit blocks	6
3.1.5	Step 5: Output	7
3.2	MD5 碰撞	7

4	实验	8
4.1	创建两个碰撞的文件	8
4.2	利用 MD5 碰撞实现攻击	9
4.3	MD5 实现随机数	12
5	感想	13

1 摘要

本文将简要介绍哈希函数——它的定义、安全性，实现了历史上出现的经典的哈希函数——MD5，利用 MD5 碰撞实现一次攻击，绘制 MD5 实现随机数的分布。通过 MD5 的三个实验以更好地理解哈希函数的安全性。

2 哈希函数

哈希函数是应用十分广泛的函数或说思想，亦是诸多应用的基石。小到日常编程时字典、集合的数据结构、伪随机数生成等的实现，大到信息安全领域消息认证码、数字签名等的实现，统统都用到了哈希函数。

2.1 定义

哈希函数的一种常见定义如下：

Definition 1 *A hash function is any function that can be used to map data of arbitrary size to fixed-size values.*

将任意长度的数据映射到固定长度的数据，也就意味着它有压缩数据的功能，这就可以催生出很多应用。如比较两个文件是否相等，无需打开两个文档文件进行逐字比较，这些文件的计算出的哈希值将使所有者立即知道它们是否不同。

2.2 安全性

由于鸽笼原理，哈希函数是一定会发生碰撞的，也就是说一定存在 $x \neq x'$ 使 $H(x) = H(x')$ 。考虑一个极端的情况：

$$H(x) = 0$$

它确实确实将任意长度的数据 x 映射成了固定长度 1 的数据 0，也就是说它是一个哈希函数。但是它是一个任意 $x \neq x'$ 都会发生碰撞的。如果它用于比对文件那显然是不好的，因为它会判断任何两个文件都是相同的，因而哈希函数是有优劣区分的。而在信息安全领域给出了一套相对具体的标准来评价一个哈希函数的安全性。

2.2.1 原像抗性

原像性又称单向性。它的一种常见定义如下：

Definition 2 *Given y , it is computationally infeasible to find any input x such that $y = H(x)$.*

其现实意义在于，假设一个网站如果在数据库存储 $(username, password)$ ，它通过比对用户输入的 $password$ 和数据库中对应用户 $username$ 的 $password$ 来验证用户。一旦这个数据库被攻击者破获，攻击者就可以得知任意 $(username, password)$ 对，即可以通过对应的 $(username, password)$ 对登录任意账户。但是假设这里有一个满足原像抗性的哈希函数 H ，这个网站就可以改为在数据库存储 $(username, H(password))$ 对，通过比对用户输入的 $password$ 的哈希值和数据库中对应用户 $username$ 的 $H(password)$ 来验证用户。即使这个数据库被攻击者破获，由于原像抗性攻击者难以通过 $H(password)$ 来得知 $password$ ，这样攻击者仍然无法登录账户。

2.2.2 次原像抗性

次原像抗性又称弱碰撞抗性。它的一种常见定义如下：

Definition 3 *Given x , it is computationally infeasible to find another input $x' \neq x$ such that $H(x) = H(x')$.*

它和原像抗性的不同地方在于，在攻击者知晓 $H(x)$ 的基础上还假设攻击者知晓 x 。有些哈希函数如：

$$H(x) = x^e \mod N$$

这里 $N = pq$ 但是 p, q 是未知的大质数，且有 $\phi(N) = \phi(p)\phi(q) = (p-1)(q-1)$ ， $\gcd\{e, \phi(N)\} = 1$ 且 d 已知。这个函数满足原像抗性，但是不满足次原像抗性。因为对它的原像攻击是 RSA 问题，最有效的方法是先对 N 质因数分解成 p, q 再用：

$$de \equiv 1 \mod (p-1)(q-1)$$

算 d ，最后根据欧拉定理：

$$x \equiv x^{k\phi(N)+1} \equiv H(x)^d \mod N$$

来获得 x 。而大质因数分解是很难的，因而这个哈希函数满足原像抗性。但是攻击者可以令 $x' = xN + x$ 这样就有：

$$H(x') \equiv (xN + x)^e \equiv x \equiv H(x) \mod N$$

即找到了碰撞，来发起次原像攻击。假设在数字签名的情境下，发布者发布文件 $document$ 和它的哈希值 $H(document)$ ，别人可以通过计算 $document$ 的哈希值并与 $H(document)$ 比对，就可验证文件有没有被修改。由于 $document$ 和 $H(document)$ 对于攻击者都是可见的，如果这个哈希函数 H 不满足次原像抗性，攻击者可以轻松的找到一个 $document' \neq document$ 使得 $H(document') = H(document)$ 这样攻击者便可轻松的伪造 $document$ 骗过验证。

2.2.3 碰撞抗性

碰撞抗性又称强碰撞抗性。它的一种常见定义如下：

Definition 4 *It is computationally infeasible to find any two distinct inputs x and x' such that $H(x) = H(x')$.*

这是一种比原像攻击和次原像攻击更加容易实现的攻击，但是它成功攻击的危害性也弱于前两种攻击。假设哈希函数 H 将数据映射成长度为 n bits 的数据，由于每一位正确或错误的概率都是 $\frac{1}{2}$ ，所以对它暴力破解的平均时间复杂度是：

$$O(\lim_{m \rightarrow +\infty} \sum_{i=1}^m i(1 - (\frac{1}{2})^n)^{i-1}(\frac{1}{2})^n) = O(\lim_{m \rightarrow +\infty} 2^n - (1 - (\frac{1}{2})^n)^m(2^n - m - 2)) = O(2^n)$$

而用暴力找到一个碰撞可以转换为生日问题。遍历时第 m 次迭代找到至少一个碰撞的概率是：

$$p = \begin{cases} 1 - \prod_{i=1}^{m-1} (1 - \frac{i}{2^n}), & m \leq 2^n \\ 1, & m > 2^n \end{cases} \approx 1 - e^{-\frac{m(m-1)}{2^{n+1}}}$$

因此有：

$$m \approx \sqrt{2^{n+1} \ln(\frac{1}{1-p})}$$

就有时间复杂度：

$$O(\sqrt{\frac{\pi}{2}} 2^n) = O(2^{\frac{n}{2}})$$

证明略。因此它比前两种攻击的暴力破解更容易实现，但是它的危害更弱。假设一个 Windows 系统的员工，发现了一对 b, b' 使得 $H(b) = H(b')$ 其中 b 是微软要发布的补丁而 b' 是一个恶意软件，那么该员工就可以通过发布补丁来传播恶意软件。但是前提是找到的 b 和 b' 刚好是补丁和恶意软件。

3 MD5

MD5 是一个非常经典的哈希算法，由美国密码学家罗纳德·李维斯特（Ronald Linn Rivest）设计，于 1992 年公开，用以取代 MD4 算法。但后来该算法被多次证明是不安全的。

3.1 算法描述

3.1.1 Step 1: Append Padding Bits

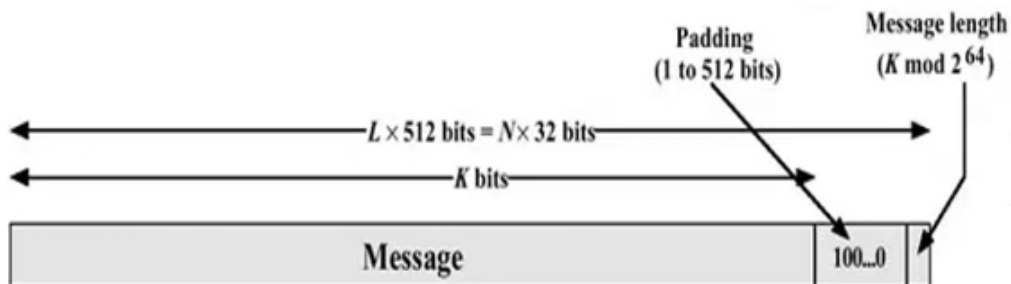
用 10...0 填充输入消息且填充位数为 $1 - 512$ ，使其长度 K 满足：

$$K \equiv 448 \pmod{512}$$

这样分成 L 块，其中前面每一块 512 位，最后一块还差 64 位达到 512 位。

3.1.2 Step 2: Append Length

用 $K \bmod 2^{64}$ 填充最后一块的最后 64 位，使其每一块达到 512 块，这样延长后的消息长度 $512L$ 位。



3.1.3 Step 3: Initialize MD Buffer

用一个 128 位的缓冲区来存储中间结果和最终结果。这个 128 位缓冲使用 4 个 32 位的寄存器 P, Q, R, S 实现。缓冲区被初始化为：

$$P = 01, 23, 45, 67$$

$$Q = 89, AB, CD, EF$$

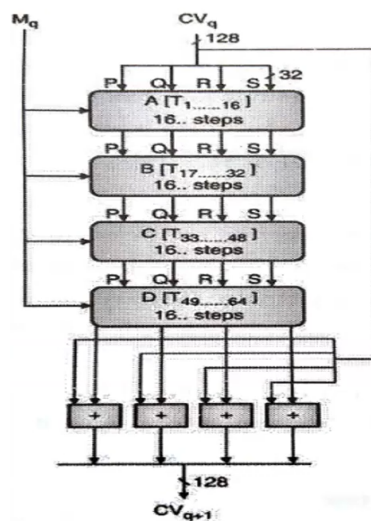
$$R = FE, DC, BA, 98$$

$$S = 76, 54, 32, 10$$

初始值也被称为 IV (initial values)。

3.1.4 Step 4: Process Message in 512-bit blocks

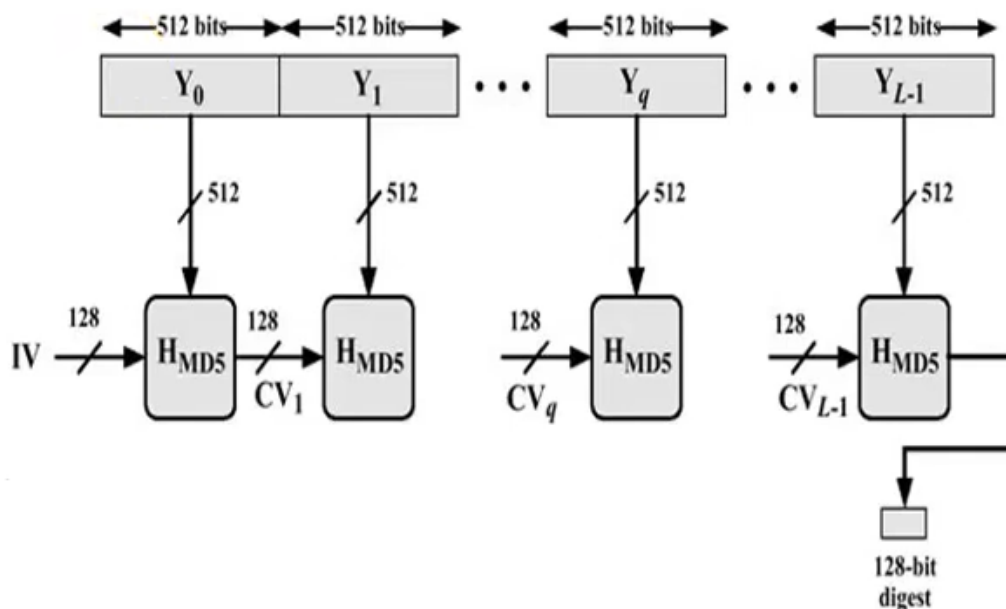
如图进行 4 轮处理：



这四轮有和 SHA 类似的结构，但是在逻辑函数原语 A, B, C, D 有所不同每一轮接收 512 位的块，处理并产生 128 位输出，第 q 轮 128 位输出加上 CV_q (current values) 产生 CV_{q+1} 。

3.1.5 Step 5: Output

在处理完全部 L 个块后，最后产生的 128 位消息摘要作为输出。



3.2 MD5 碰撞

王小云、于洪波在 2005 年发表的文章描述了一种算法，来找到一对不同的数据拥有相同的 MD5 哈希值。其中一对著名的数据对分别是：

```
1 d131dd02c5e6eec4693d9a0698aff95c2fcb58712467eab4004583eb8fb7f89
2 55ad340609f4b30283e488832571415a085125e8f7cdc99fd91dbdf280373c5b
3 d8823e3156348f5bae6dacd436c919c6dd53e2b487da03fd02396306d248cda0
4 e99f33420f577ee8ce54b67080a80d1ec69821bcb6a8839396f9652b6ff72a70
```

和

```
1 d131dd02c5e6eec4693d9a0698aff95c2fcb58712467eab4004583eb8fb7f89
2 55ad340609f4b30283e488832571415a085125e8f7cdc99fd91dbdf280373c5b
3 d8823e3156348f5bae6dacd436c919c6dd53e2b487da03fd02396306d248cda0
4 e99f33420f577ee8ce54b67080a80d1ec69821bcb6a8839396f9652b6ff72a70
```

它们的 MD5 哈希值都是

```
1 79054025255fb1a26e4bc422aef54eb4
```

假设 f 就是前面提到的 A、B、C、D 四个步骤，王小云和于洪波提出的方法，对于任意给定的 CV_q ，都可以实现找到四个块 M, M', N, N' 使得，有 $f(f(CV_q, M), M') = f(f(CV_q, N), N')$ 。非常重要的一点在于由于 CV_q 是任意的，我们可以基于此构造出任意的碰撞的数据。这样就说明 MD5 是不满足碰撞抗性的。运用这个方法可以构造一些有实际意义的攻击，比如利用分支结构：

```
1 Program 1: if (data1 == data1) then { good_program } else { evil_program }
```

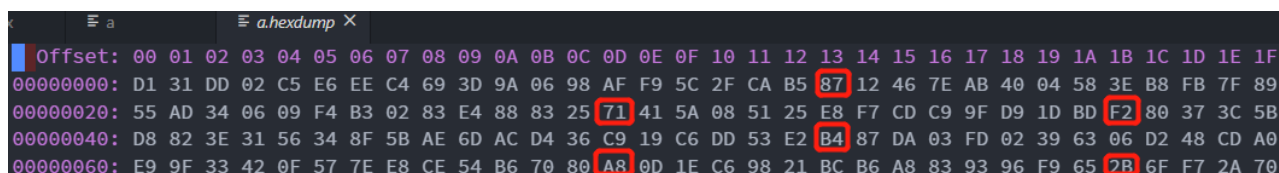
```
1 Program 2: if (data2 == data1) then { good_program } else { evil_program }
```

4 实验

4.1 创建两个碰撞的文件

运行 write.py 创建二进制文件 a、b，分别为：

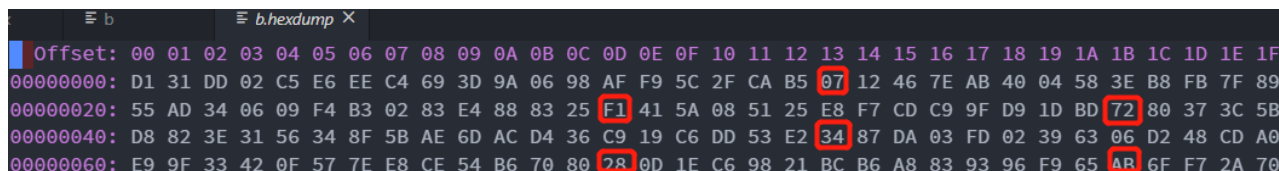
```
1 d131dd02c5e6eec4693d9a0698aff95c2fcab58712467eab4004583eb8fb7f89
2 55ad340609f4b30283e488832571415a085125e8f7cdc99fd91dbdf280373c5b
3 d8823e3156348f5bae6dacd436c919c6dd53e2b487da03fd02396306d248cda0
4 e99f33420f577ee8ce54b67080a80d1ec69821bcb6a8839396f9652b6ff72a70
```



```
Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
00000000: D1 31 DD 02 C5 E6 EE C4 69 3D 9A 06 98 AF F9 5C 2F CA B5 87 12 46 7E AB 40 04 58 3E B8 FB 7F 89
00000020: 55 AD 34 06 09 F4 B3 02 83 E4 88 83 25 71 41 5A 08 51 25 E8 F7 CD C9 9F D9 1D BD F2 80 37 3C 5B
00000040: D8 82 3E 31 56 34 8F 5B AE 6D AC D4 36 C9 19 C6 DD 53 E2 B4 87 DA 03 FD 02 39 63 06 D2 48 CD A0
00000060: E9 9F 33 42 0F 57 7E E8 CE 54 B6 70 80 A8 0D 1E C6 98 21 BC B6 A8 83 93 96 F9 65 2B 6F F7 2A 70
```

和

```
1 d131dd02c5e6eec4693d9a0698aff95c2fcab58712467eab4004583eb8fb7f89
2 55ad340609f4b30283e488832571415a085125e8f7cdc99fd91dbdf280373c5b
3 d8823e3156348f5bae6dacd436c919c6dd53e2b487da03fd02396306d248cda0
4 e99f33420f577ee8ce54b67080a80d1ec69821bcb6a8839396f9652b6ff72a70
```



```
Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
00000000: D1 31 DD 02 C5 E6 EE C4 69 3D 9A 06 98 AF F9 5C 2F CA B5 07 12 46 7E AB 40 04 58 3E B8 FB 7F 89
00000020: 55 AD 34 06 09 F4 B3 02 83 E4 88 83 25 F1 41 5A 08 51 25 E8 F7 CD C9 9F D9 1D BD 72 80 37 3C 5B
00000040: D8 82 3E 31 56 34 8F 5B AE 6D AC D4 36 C9 19 C6 DD 53 E2 34 87 DA 03 FD 02 39 63 06 D2 48 CD A0
00000060: E9 9F 33 42 0F 57 7E E8 CE 54 B6 70 80 28 0D 1E C6 98 21 BC B6 A8 83 93 96 F9 65 4B 6F F7 2A 70
```

使用编译 md5.c 生成的 md5.exe 计算其哈希值：

```
> ./md5 a b
79054025255fb1a26e4bc422aef54eb4
79054025255fb1a26e4bc422aef54eb4
```


发现两个不同的数据哈希值是一样的，都是：

```
1 79054025255fb1a26e4bc422aef54eb4
```

4.2 利用 MD5 碰撞实现攻击

先使用 NASM 语法编写 hello_world_raw.asm，其中包括分支结构：

```
1 loop:
2     mov cl, [data1+bx]
3     cmp cl, [data2+bx]
4     jnz evil
5     inc bx
6     cmp bx, 0x80
7     jz good
8     jmp loop
```

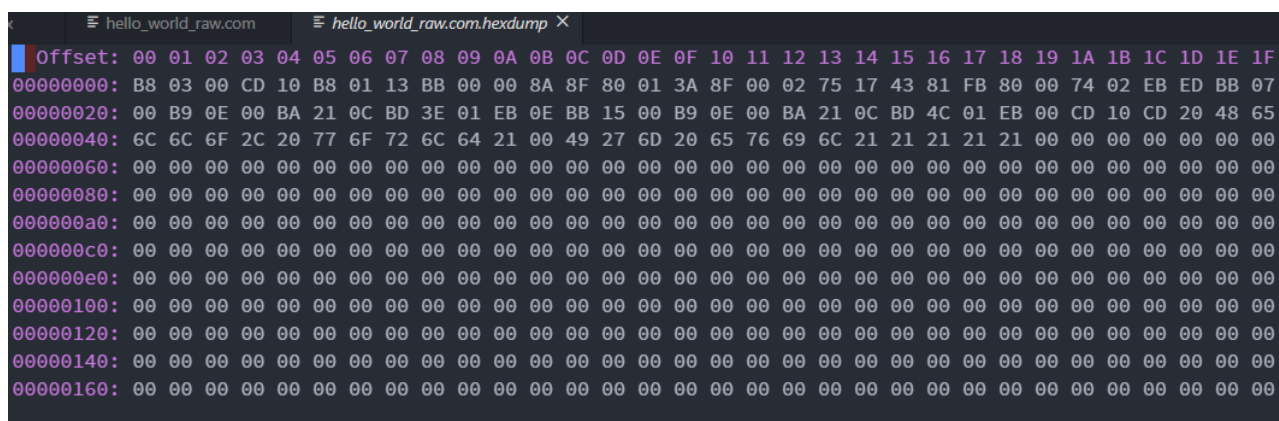
在 data1、data2 前面补充 0，使其前面长度是 512 位即 64 比特的倍数：

```
1 times 0x80-($-$$) db 0
```

将 data1、data2 全部初始化为 0（为了确保 data1、data2 的位置正确，以保证可以正确利用其标签）：

```
1 data1: times 0x80-($-$$) db 0
2 data2: times 0x80-($-$$) db 0
```

编译 hello_world.asm，得 hello_world_raw.com：



```
Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
00000000: B8 03 00 CD 10 B8 01 13 BB 00 00 8A 8F 80 01 3A 8F 00 02 75 17 43 81 FB 80 00 74 02 EB ED BB 07
00000020: 00 B9 0E 00 BA 21 0C BD 3E 01 EB 0E BB 15 00 B9 0E 00 BA 21 0C BD 4C 01 EB 00 CD 10 CD 20 48 65
00000040: 6C 6C 6F 2C 20 77 6F 72 6C 64 21 00 49 27 6D 20 65 76 69 6C 21 21 21 21 21 21 00 00 00 00 00 00
00000060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000120: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000140: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000160: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

运行 delete.py 把 hello_world_raw.com 的 data1、data2 去掉，以留出位置给真正的 data1、data2，生成新文件叫 hello_world.com：

```

hello_world.com  hello_world.com.hexdump X
Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
00000000: B8 03 00 CD 10 B8 01 13 BB 00 00 8A 8F 80 01 3A 8F 00 02 75 17 43 81 FB 80 00 74 02 EB ED BB 07
00000020: 00 B9 0E 00 BA 21 0C BD 3E 01 EB 0E BB 15 00 B9 0E 00 BA 21 0C BD 4C 01 EB 00 CD 10 CD 20 48 65
00000040: 6C 6C 6F 2C 20 77 6F 72 6C 64 21 00 49 27 6D 20 65 76 69 6C 21 21 21 21 21 00 00 00 00 00 00 00
00000060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

使用下载的工具 fastcoll.exe 在 hello_world.com 后加后缀，快速生成两个不同的，哈希值却相同的文件 hello_world_msg1.com, hello_world_msg2.com:

```

> ./fastcoll .\hello_world.com
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: '.\hello_world_msg1.com' and '.\hello_world_msg2.com'
Using prefixfile: '.\hello_world.com'
Using initial value: f3d0131c0383307e60f6c82a738c6d1b

Generating first block: ..
Generating second block: S00.....
Running time: 0.777 s

```

```

hello_world_msg1.com  hello_world_msg1.com.hexdump X
Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
00000000: B8 03 00 CD 10 B8 01 13 BB 00 00 8A 8F 80 01 3A 8F 00 02 75 17 43 81 FB 80 00 74 02 EB ED BB 07
00000020: 00 B9 0E 00 BA 21 0C BD 3E 01 EB 0E BB 15 00 B9 0E 00 BA 21 0C BD 4C 01 EB 00 CD 10 CD 20 48 65
00000040: 6C 6C 6F 2C 20 77 6F 72 6C 64 21 00 49 27 6D 20 65 76 69 6C 21 21 21 21 21 00 00 00 00 00 00 00
00000060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000080: D3 B4 C5 97 07 DD F3 4A E4 51 D2 37 7B 81 99 97 D1 7D 88 F7 8B C5 31 E7 58 0A 81 C1 0A 32 44 BD
000000a0: E5 D8 EB CF E2 3F 0C 60 EF 21 E0 7C 91 85 ED 58 74 48 67 CF FA 26 4C E2 2E 93 86 6E 9F E5 0C 03
000000c0: CC 05 D7 C5 52 DC B4 16 79 C7 E6 C2 28 A0 DD 95 3D F2 95 B6 45 EE 3A FD F4 B4 E2 46 4B 65 FD BC
000000e0: 35 10 3B 77 5B 11 DE 25 96 34 0D 62 96 78 FE 5B D2 30 0A 35 95 C8 23 E7 AA 14 C4 DC 6D A7 1D 1B

```

```

hello_world_msg2.com  hello_world_msg2.com.hexdump X
Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
00000000: B8 03 00 CD 10 B8 01 13 BB 00 00 8A 8F 80 01 3A 8F 00 02 75 17 43 81 FB 80 00 74 02 EB ED BB 07
00000020: 00 B9 0E 00 BA 21 0C BD 3E 01 EB 0E BB 15 00 B9 0E 00 BA 21 0C BD 4C 01 EB 00 CD 10 CD 20 48 65
00000040: 6C 6C 6F 2C 20 77 6F 72 6C 64 21 00 49 27 6D 20 65 76 69 6C 21 21 21 21 21 00 00 00 00 00 00 00
00000060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000080: D3 B4 C5 97 07 DD F3 4A E4 51 D2 37 7B 81 99 97 D1 7D 88 77 8B C5 31 E7 58 0A 81 C1 0A 32 44 BD
000000a0: E5 D8 EB CF E2 3F 0C 60 EF 21 E0 7C 91 05 EE 58 74 48 67 CF FA 26 4C E2 2E 93 86 EE 9F E5 0C 03
000000c0: CC 05 D7 C5 52 DC B4 16 79 C7 E6 C2 28 A0 DD 95 3D F2 95 36 45 EE 3A FD F4 B4 E2 46 4B 65 FD BC
000000e0: 35 10 3B 77 5B 11 DE 25 96 34 0D 62 96 F8 FD 5B D2 30 0A 35 95 C8 23 E7 AA 14 C4 5C 6D A7 1D 1B

```

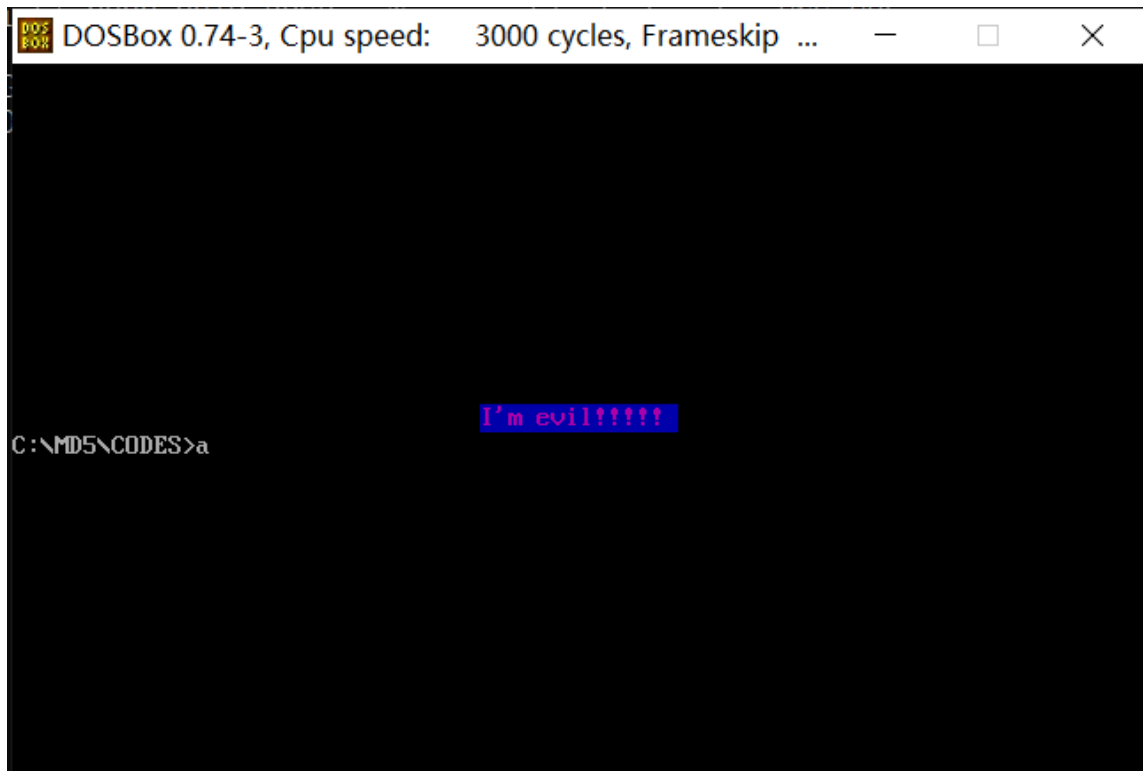
运行 datacopy.py 把 msg1 的 data1 复制到 msg1 和 msg2 的 data2 的位置，且分别重命名成 good.com 和 evil.com:

```
good.com good.com.hexdump X
Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
00000000: B8 03 00 CD 10 B8 01 13 BB 00 00 8A 8F 80 01 3A 8F 00 02 75 17 43 81 FB 80 00 74 02 EB ED BB 07
00000020: 00 B9 0E 00 BA 21 0C BD 3E 01 EB 0E BB 15 00 B9 0E 00 BA 21 0C BD 4C 01 EB 00 CD 10 CD 20 48 65
00000040: 6C 6C 6F 2C 20 77 6F 72 6C 64 21 00 49 27 6D 20 65 76 69 6C 21 21 21 21 21 00 00 00 00 00 00 00
00000060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000080: D3 B4 C5 97 07 DD F3 4A E4 51 D2 37 7B 81 99 97 D1 7D 88 F7 8B C5 31 E7 58 0A 81 C1 0A 32 44 BD
000000a0: E5 D8 EB CF E2 3F 0C 60 EF 21 E0 7C 91 85 ED 58 74 48 67 CF FA 26 4C E2 2E 93 86 6E 9F E5 0C 03
000000c0: CC 05 D7 C5 52 DC B4 16 79 C7 E6 C2 28 A0 DD 95 3D F2 95 B6 45 EE 3A FD F4 B4 E2 46 4B 65 FD BC
000000e0: 35 10 3B 77 5B 11 DE 25 96 34 0D 62 96 78 FE 5B D2 30 0A 35 95 C8 23 E7 AA 14 C4 DC 6D A7 1D 1B
00000100: D3 B4 C5 97 07 DD F3 4A E4 51 D2 37 7B 81 99 97 D1 7D 88 F7 8B C5 31 E7 58 0A 81 C1 0A 32 44 BD
00000120: E5 D8 EB CF E2 3F 0C 60 EF 21 E0 7C 91 85 ED 58 74 48 67 CF FA 26 4C E2 2E 93 86 6E 9F E5 0C 03
00000140: CC 05 D7 C5 52 DC B4 16 79 C7 E6 C2 28 A0 DD 95 3D F2 95 B6 45 EE 3A FD F4 B4 E2 46 4B 65 FD BC
00000160: 35 10 3B 77 5B 11 DE 25 96 34 0D 62 96 78 FE 5B D2 30 0A 35 95 C8 23 E7 AA 14 C4 DC 6D A7 1D 1B
```

```
evil.com evil.com.hexdump X
Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
00000000: B8 03 00 CD 10 B8 01 13 BB 00 00 8A 8F 80 01 3A 8F 00 02 75 17 43 81 FB 80 00 74 02 EB ED BB 07
00000020: 00 B9 0E 00 BA 21 0C BD 3E 01 EB 0E BB 15 00 B9 0E 00 BA 21 0C BD 4C 01 EB 00 CD 10 CD 20 48 65
00000040: 6C 6C 6F 2C 20 77 6F 72 6C 64 21 00 49 27 6D 20 65 76 69 6C 21 21 21 21 21 00 00 00 00 00 00 00
00000060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000080: D3 B4 C5 97 07 DD F3 4A E4 51 D2 37 7B 81 99 97 D1 7D 88 77 8B C5 31 E7 58 0A 81 C1 0A 32 44 BD
000000a0: E5 D8 EB CF E2 3F 0C 60 EF 21 E0 7C 91 05 EE 58 74 48 67 CF FA 26 4C E2 2E 93 86 EE 9F E5 0C 03
000000c0: CC 05 D7 C5 52 DC B4 16 79 C7 E6 C2 28 A0 DD 95 3D F2 95 36 45 EE 3A FD F4 B4 E2 46 4B 65 FD BC
000000e0: 35 10 3B 77 5B 11 DE 25 96 34 0D 62 96 F8 FD 5B D2 30 0A 35 95 C8 23 E7 AA 14 C4 5C 6D A7 1D 1B
00000100: D3 B4 C5 97 07 DD F3 4A E4 51 D2 37 7B 81 99 97 D1 7D 88 F7 8B C5 31 E7 58 0A 81 C1 0A 32 44 BD
00000120: E5 D8 EB CF E2 3F 0C 60 EF 21 E0 7C 91 85 ED 58 74 48 67 CF FA 26 4C E2 2E 93 86 6E 9F E5 0C 03
00000140: CC 05 D7 C5 52 DC B4 16 79 C7 E6 C2 28 A0 DD 95 3D F2 95 B6 45 EE 3A FD F4 B4 E2 46 4B 65 FD BC
00000160: 35 10 3B 77 5B 11 DE 25 96 34 0D 62 96 78 FE 5B D2 30 0A 35 95 C8 23 E7 AA 14 C4 DC 6D A7 1D 1B
```

使用 dosbox（或其他 dos 模拟器）运行 good.com 和 evil.com:

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip ...
Hello, world!
C:\MD5\C0DES>
```



使用 md5.exe 查看两个文件的哈希值：

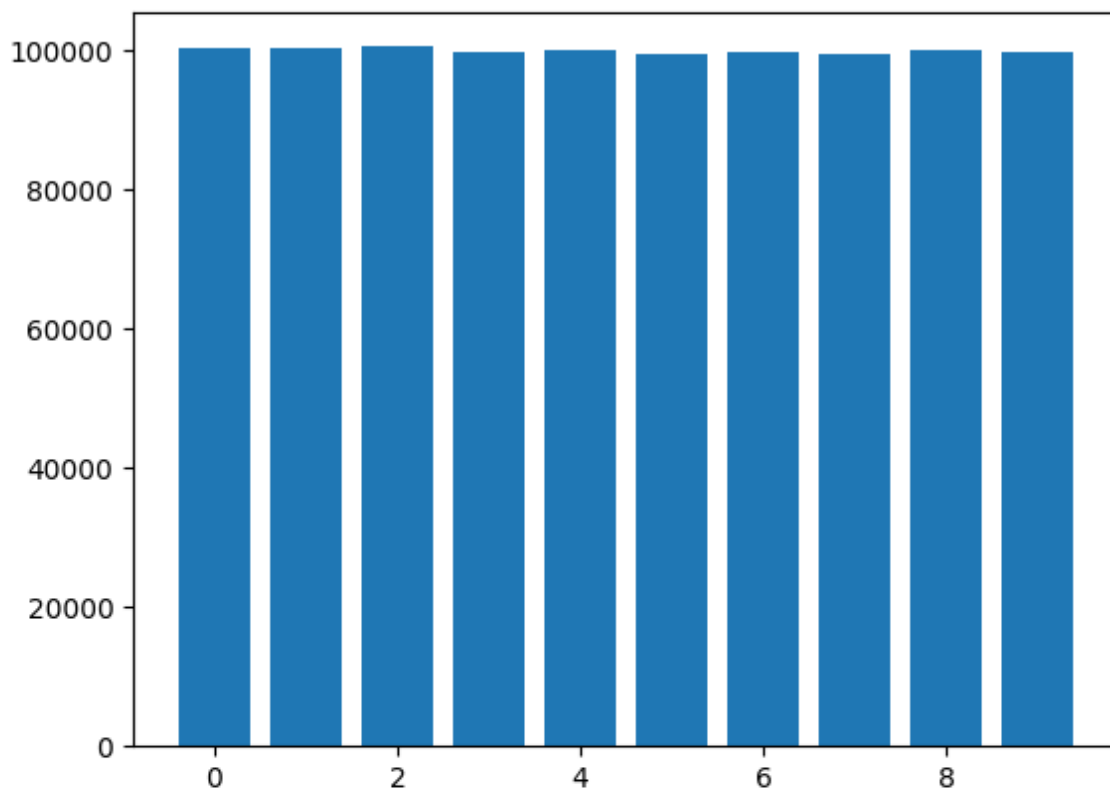
```
> ./md5 good.com evil.com
5d3643299c52331740c5b89fc662e646
5d3643299c52331740c5b89fc662e646
```

4.3 MD5 实现随机数

使用 python 实现，md5 使用 python 库，实现 md5rand.py 如下：

```
1 from hashlib import md5
2 import matplotlib.pyplot as plt
3
4
5 domain = 10
6 sample = 1000000
7
8
9 y = [0] * domain
10 for i in range(sample):
11     y[int(md5(i.to_bytes(64, "little")).hexdigest(), 16) % domain] += 1
12 plt.bar(range(domain), y)
13 plt.show()
```

每次采样生成 0-9 的随机整数，总共采样 1000000 次，最终采样结果：



5 感想

利用 MD5 碰撞可以轻易发动危害比较大的攻击，而碰撞攻击在三种攻击中的危害却又是最小的，因而可见哈希函数的安全性分析是非常重要的。另外 MD5 虽然已经多次被证明是不安全的哈希函数，但是仍可以应用于一些对安全性要求不高的场景，如随机整数生成。因此 MD5 作为一个哈希函数在一些场景下并没有过时。