

```
seed@VM: ~/Desktop
[03/14/24]seed@VM:~/Desktop$ 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: 0123456789abcdeffedcba9876543210

Generating first block: ..
Generating second block: S11.....
.....
Running time: 10.71 s
[03/14/24]seed@VM:~/Desktop$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
[03/14/24]seed@VM:~/Desktop$ md5sum out1.bin
e37d7c8ca0c06f3c3a566f02c1aa544d  out1.bin
[03/14/24]seed@VM:~/Desktop$ md5sum out2.bin
e37d7c8ca0c06f3c3a566f02c1aa544d  out2.bin
[03/14/24]seed@VM:~/Desktop$
```

```
[03/14/24]seed@VM:~/Desktop$ cat prefix.txt
[03/14/24]seed@VM:~/Desktop$
```

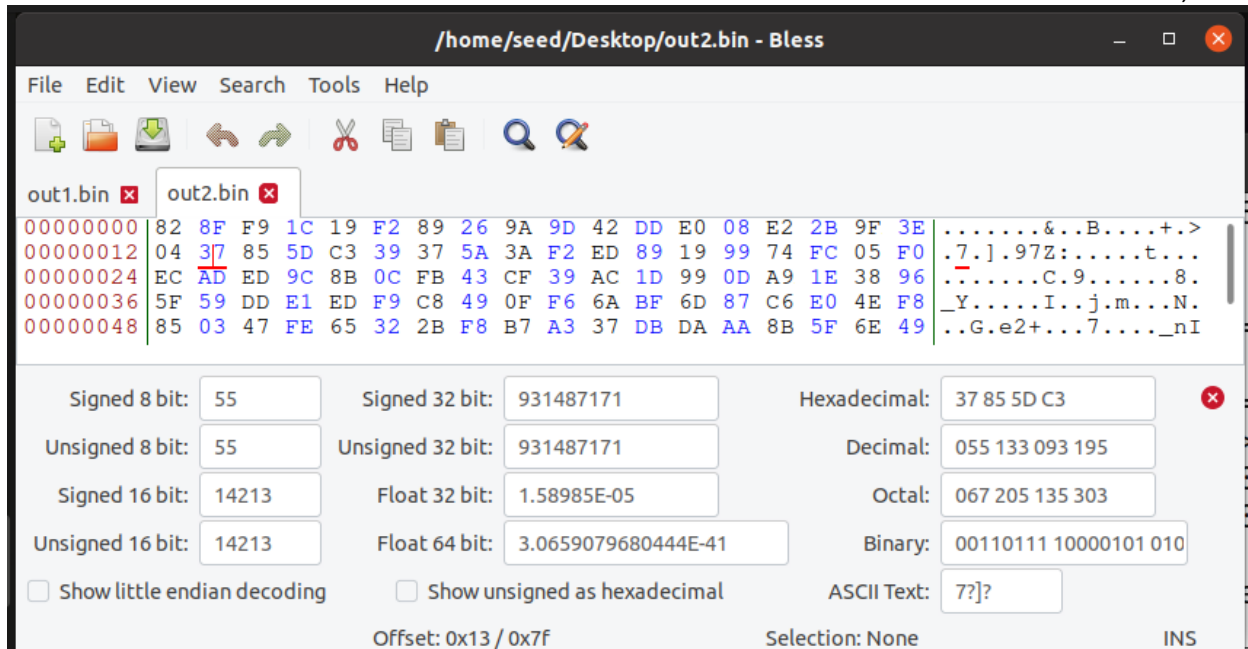
/home/seed/Desktop/out1.bin - Bless

File Edit View Search Tools Help

out1.bin x out2.bin x

00000000	82 8F F9 1C 19 F2 89 26 9A 9D 42 DD E0 08 E2 2B 9F 3E&...B....+.>
00000012	04 B7 85 5D C3 39 37 5A 3A F2 ED 89 19 99 74 FC 05 F0	...].97Z:.....t...
00000024	EC AD ED 9C 8B 0C FB 43 CF B9 AB 1D 99 0D A9 1E 38 96C.....8.
00000036	5F 59 DD E1 ED 79 C8 49 0F F6 6A BF 6D 87 C6 E0 4E F8	_Y...y.I...j.m...N.
00000048	85 03 47 FE 65 32 2B F8 B7 A3 37 5B DA AA 8B 5F 6E 49	..G.e2+...7[..._nI

Signed 8 bit:	-73	Signed 32 bit:	-1215996477	Hexadecimal:	B7 85 5D C3
Unsigned 8 bit:	183	Unsigned 32 bit:	3078970819	Decimal:	183 133 093 195
Signed 16 bit:	-18555	Float 32 bit:	-1.58985E-05	Octal:	267 205 135 303
Unsigned 16 bit:	46981	Float 64 bit:	-3.0659079680444E-41	Binary:	10110111 10000101 010
<input type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal		ASCII Text: ????	
Offset: 0x13 / 0x7f				Selection: None INS	



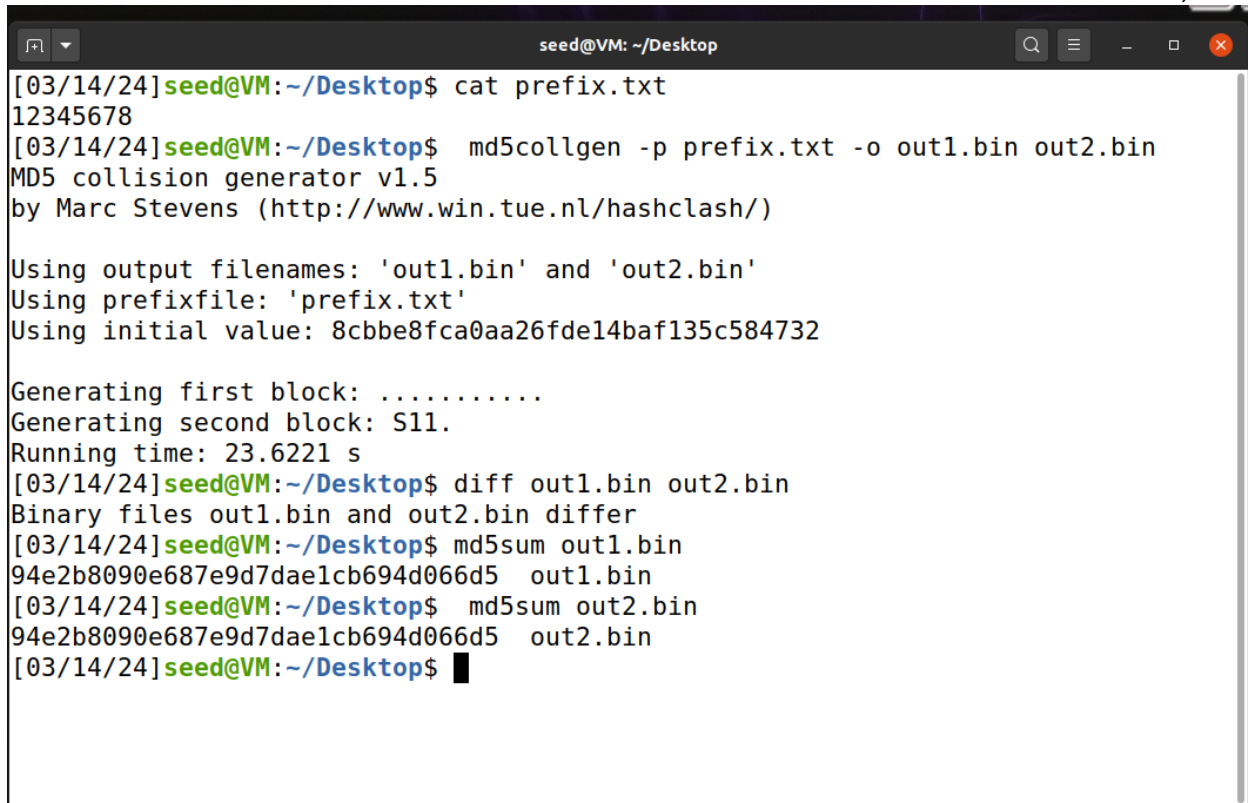
Notice the difference between the two out files, it is slight, only a couple different bits.

Question 1. If the length of your prefix file is not multiple of 64, what is going to happen?

The md5collgen program may not work properly because it expects the prefix to be padded to a multiple of 64 bytes.

Question 2. Create a prefix file with exactly 64 bytes, and run the collision tool again, and see what happens?

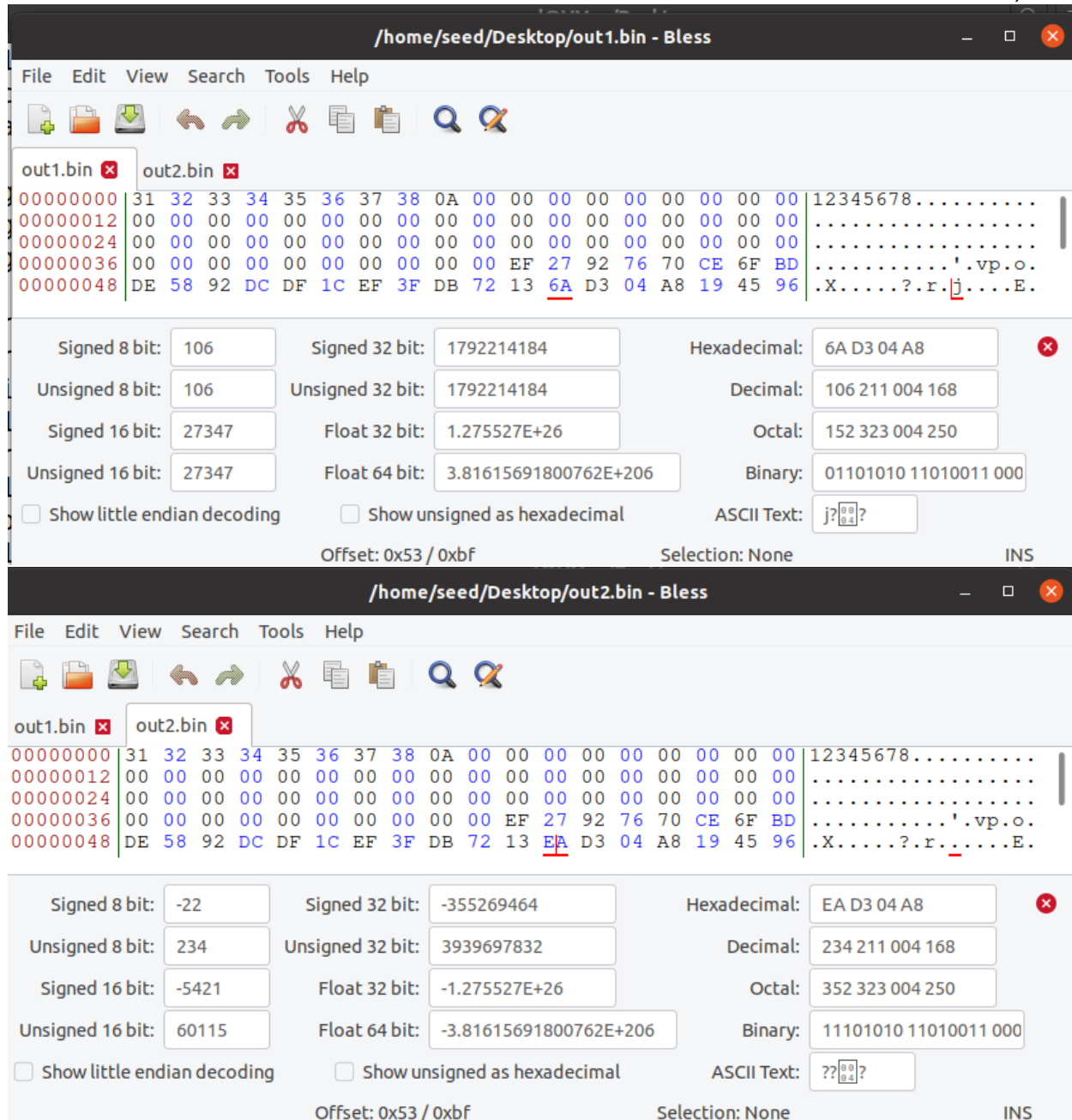
```
[03/14/24]seed@VM:~/Desktop$ cat prefix.txt
12345678
```

A terminal window titled 'seed@VM: ~/Desktop' showing the execution of the 'md5collgen' tool. The user first cat's 'prefix.txt' which contains '12345678'. Then they run 'md5collgen -p prefix.txt -o out1.bin out2.bin'. The tool outputs its version (v1.5), author (Marc Stevens), and the initial value used (8cbb8fca0aa26fde14baf135c584732). It shows progress for generating the first and second blocks, and the total running time (23.6221 s). Finally, it uses 'diff' to show the files differ and 'md5sum' to confirm both files have the same MD5 hash: 94e2b8090e687e9d7dae1cb694d066d5.

```
seed@VM: ~/Desktop
[03/14/24]seed@VM:~/Desktop$ cat prefix.txt
12345678
[03/14/24]seed@VM:~/Desktop$ 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: 8cbb8fca0aa26fde14baf135c584732

Generating first block: .....
Generating second block: S11.
Running time: 23.6221 s
[03/14/24]seed@VM:~/Desktop$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
[03/14/24]seed@VM:~/Desktop$ md5sum out1.bin
94e2b8090e687e9d7dae1cb694d066d5  out1.bin
[03/14/24]seed@VM:~/Desktop$ md5sum out2.bin
94e2b8090e687e9d7dae1cb694d066d5  out2.bin
[03/14/24]seed@VM:~/Desktop$
```



We can see that the first 64 bytes are the exact same, which is the prefix we gave.

Question 3. Are the data (128 bytes) generated by md5collgen completely different for the two output files? Please identify all the bytes that are different.

No, as you can see in the screenshot above, only one bit is different and there's proper padding with 0s.

1. By providing a scenario, explain why using MD5 for digital signatures is not a strong defense against non-repudiation attacks.

Let's say someone wants to tamper with a document and then claim that it was signed by another person using MD5 for digital signatures. This person gets a legitimate document signed by the other individual and its corresponding MD5 hash. They then modify the content of the document but keep the same MD5 hash. Because of MD5's vulnerability to collision attacks, they can generate a different document with the same MD5 hash as the original one. They can now create a fraudulent document with the modified content and the same MD5 hash as the original, claiming that the other person signed it. This lack of resistance in MD5 makes it unreliable in many cases.

2. Code signing is the process of digitally signing executables and scripts to confirm the software author and guarantee that the code has not been altered or corrupted since it was signed. Imagine Adam has published a program along with its MD5 hash on a trusted website where his code is verified. Explain how he can release a malicious version of this program and trick users to trust it.

Let's say a developer creates a program along with its MD5 hash on a trusted website, ensuring users that the code is genuine and unaltered. Later, this developer can release a malicious version of the program and trick users into trusting it by simply updating the MD5 hash on the website along with the new version. Since MD5 is vulnerable to collision attacks, the developer update a malicious version of the program that produces the same MD5 hash as the legitimate version. When users check the MD5 hash against the one listed on the website, it will match the original hash, tricking the user into thinking that this version is also legit, which its not.

3. Which hash algorithms are vulnerable to collision attacks?

Hash algorithms vulnerable to collision attacks include MD5 and SHA-1. These algorithms suffer from weaknesses that allow attackers to find two different inputs that produce the same hash value, known as a collision. This vulnerability undermines the integrity and security of systems that rely on hash functions for data integrity and authentication.

4. What hash algorithms are safe to use?

Hash algorithms that are currently safe to use include SHA-256, SHA-384, and SHA-512. These algorithms belong to the SHA-2 family and are designed to provide strong collision resistance and cryptographic security.