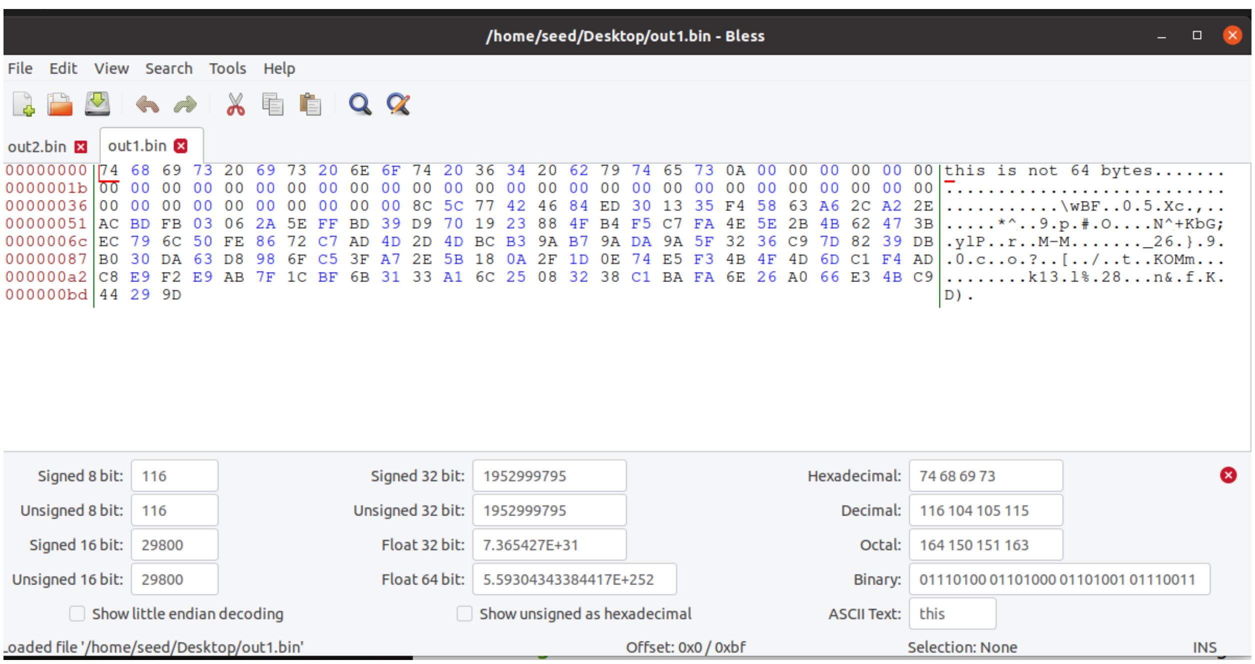
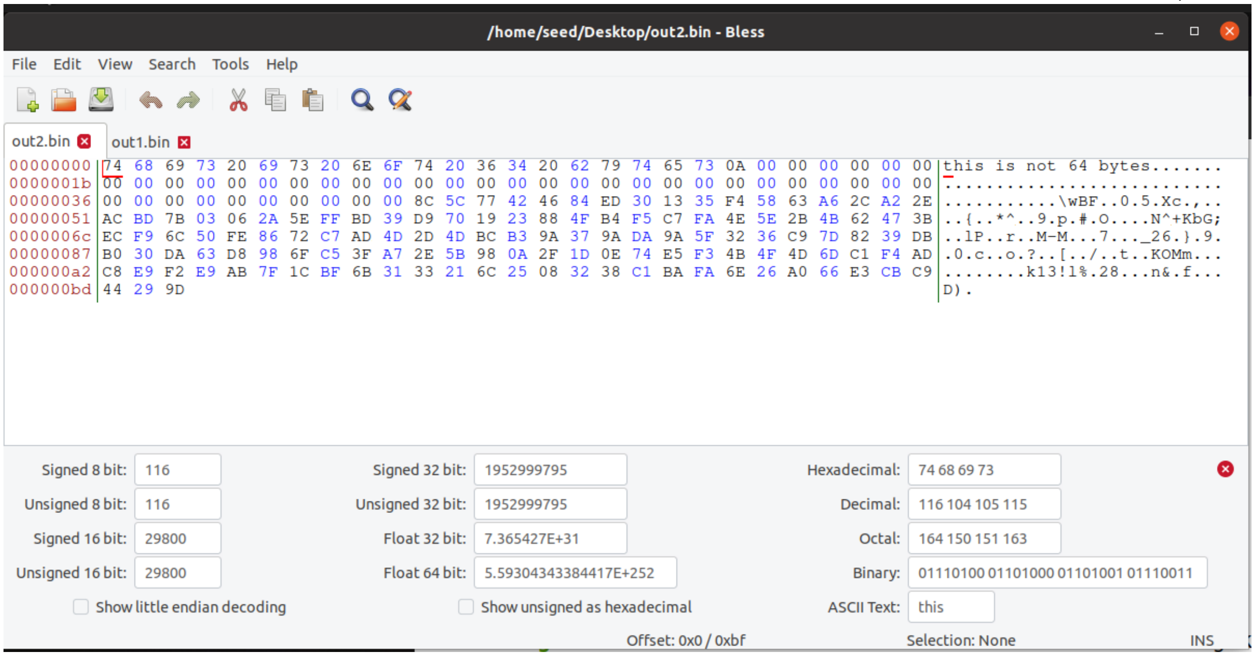
Markus Afonso
Set C
March 14, 2024

```
[03/15/24]seed@VM:~/Desktop$ cat prefix.txt
this is not 64 bytes
[03/15/24]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: 4d45bb867de8981d57be0314e720f9d6

Generating first block: ..........
Generating second block: S11...............................................
...........................................................................
...
Running time: 27.0695 s
[03/15/24]seed@VM:~/Desktop$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
[03/15/24]seed@VM:~/Desktop$ md5sum out1.bin
c44b7e795daa00d14b786bcad3367a2f  out1.bin
[03/15/24]seed@VM:~/Desktop$ md5sum out2.bin
c44b7e795daa00d14b786bcad3367a2f  out2.bin
[03/15/24]seed@VM:~/Desktop$
```






Notice the difference between the two out files, it is slight, only a couple different bytes. Also the padding of extra 0s used to make the string 64 bytes.

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

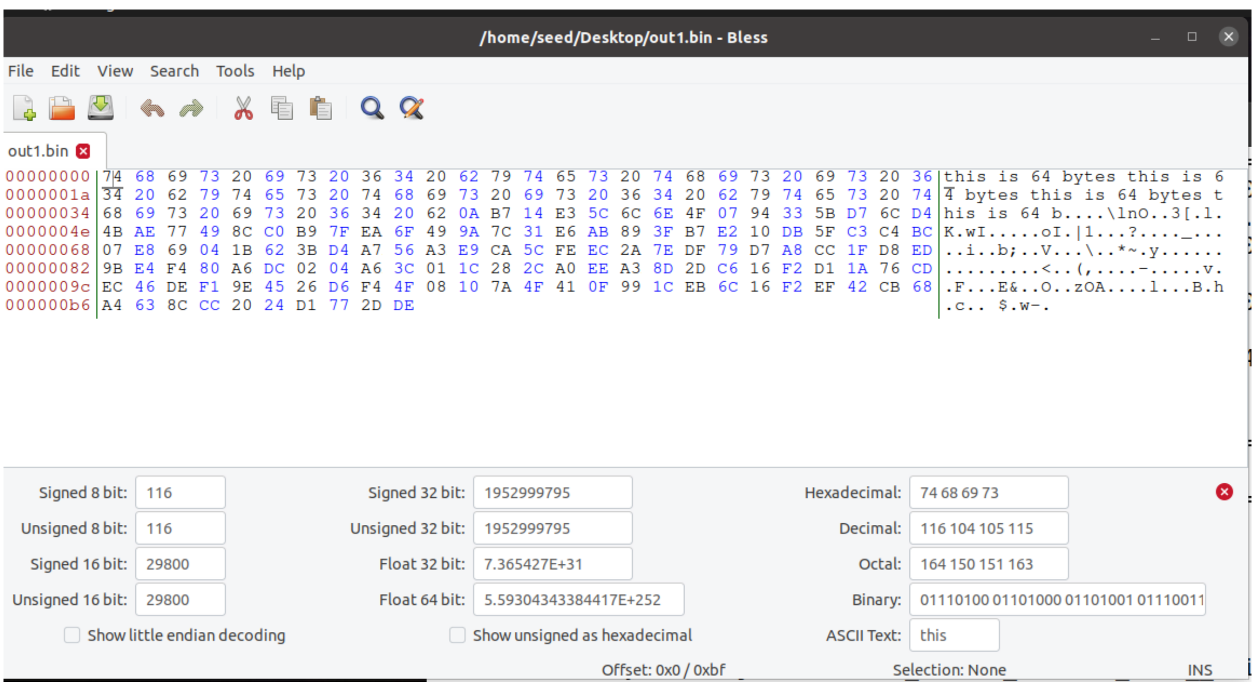
> The md5collgen program may not work properly because it expects the prefix to be padded to a multiple of 64 bytes, but will attempt to add 0s to make it a multiple of 64.

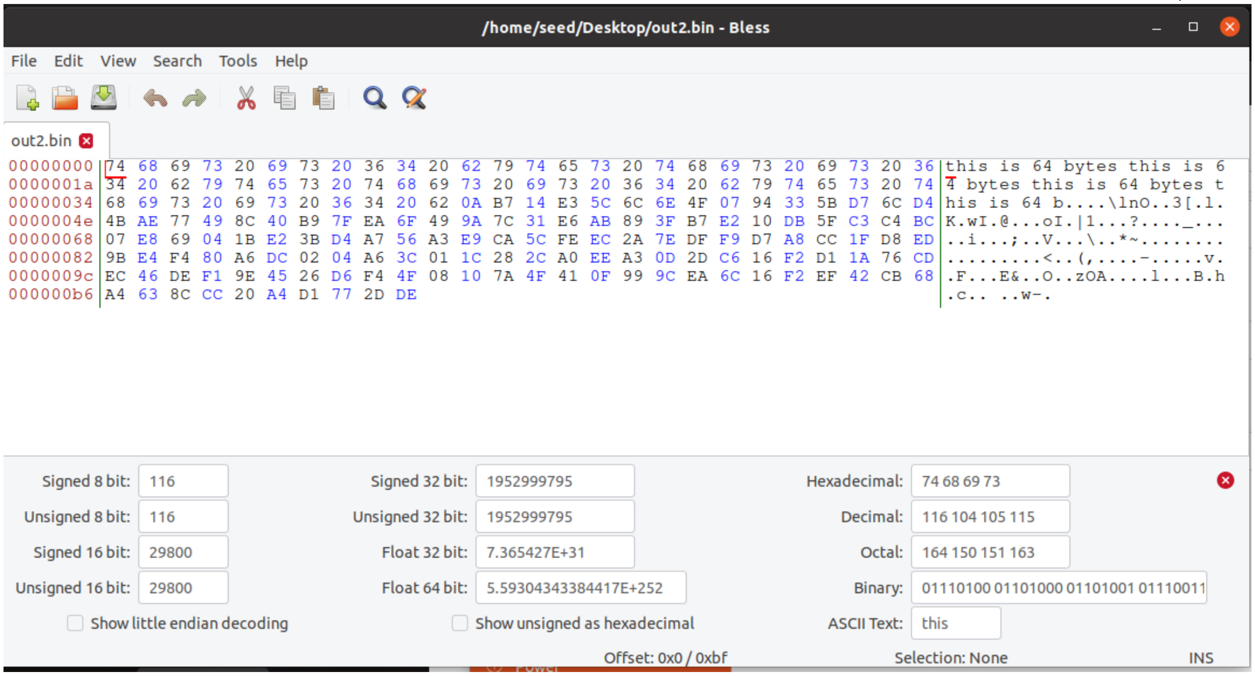
Question 2. Create a prefix file with exactly 64 bytes, and run the collision tool again, and see what happens?

```
seed@VM: ~/Desktop
[03/15/24]seed@VM:~/Desktop$ cat prefix.txt
this is 64 bytes this is 64 bytes this is 64 bytes this is 64 b
[03/15/24]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: 36d17b653e3f1862e0f0f9a8b01f6589

Generating first block: .
Generating second block: W..............
Running time: 0.562731 s
[03/15/24]seed@VM:~/Desktop$ diff out1.bin out2.bin
2,4c2,4
< ����:�d��TrT�c���Ik=bD�j'
< �lzgn`��9�D���X-@v[;�~�m�n0��.�%E���#
< |�5rS�<�L�″Ům�wCMm���M9z�l?��m�C�d���C^       �
\ No newline at end of file
---
> ����:�d��TrT�c��Ik=bD�j'
> �lzgn`���9�D���X-@v�;�~�m�n0��.�%E���#
> |��rS�<�L�″Ům�wCMm���M9z��?��m�C�d��lC^       �
\ No newline at end of file
[03/15/24]seed@VM:~/Desktop$ md5sum out1.bin
54acb0bd707d73abadd1512aeab65699  out1.bin
[03/15/24]seed@VM:~/Desktop$ md5sum out2.bin
54acb0bd707d73abadd1512aeab65699  out2.bin
[03/15/24]seed@VM:~/Desktop$ 
```






We can see that the first bytes are the exact same, which is the prefix we gave, and there is no extra padding since we provided exactly 64 bytes. Also notice the difference in the **diff** command.

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.

Yes they are different, as you can see in the screenshot above, there are a few different bytes at indexes: 74, 77, 117, 150, and 168.

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.