Mark Gameng

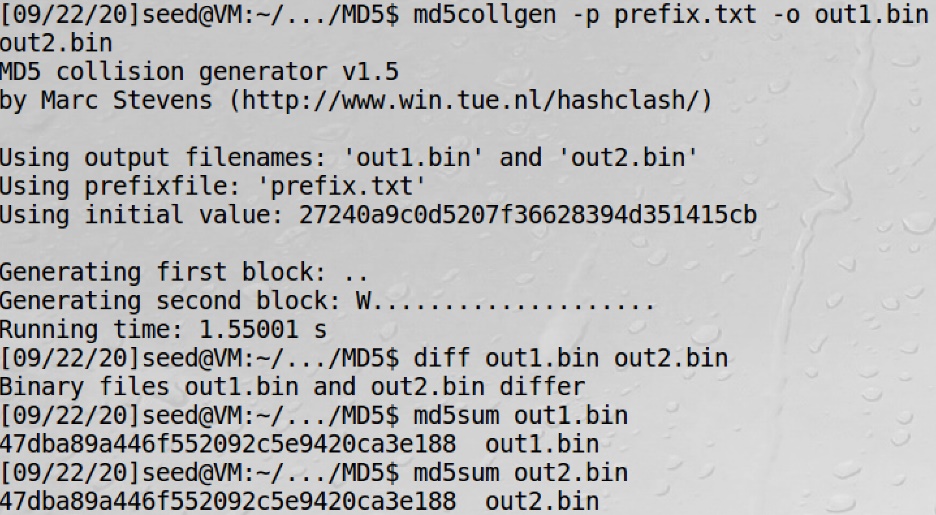
CS 458 – Dong Jin

MD5 Collision Attack Lab

## Part 2.1 – Generating two different files with the same MD5 hash

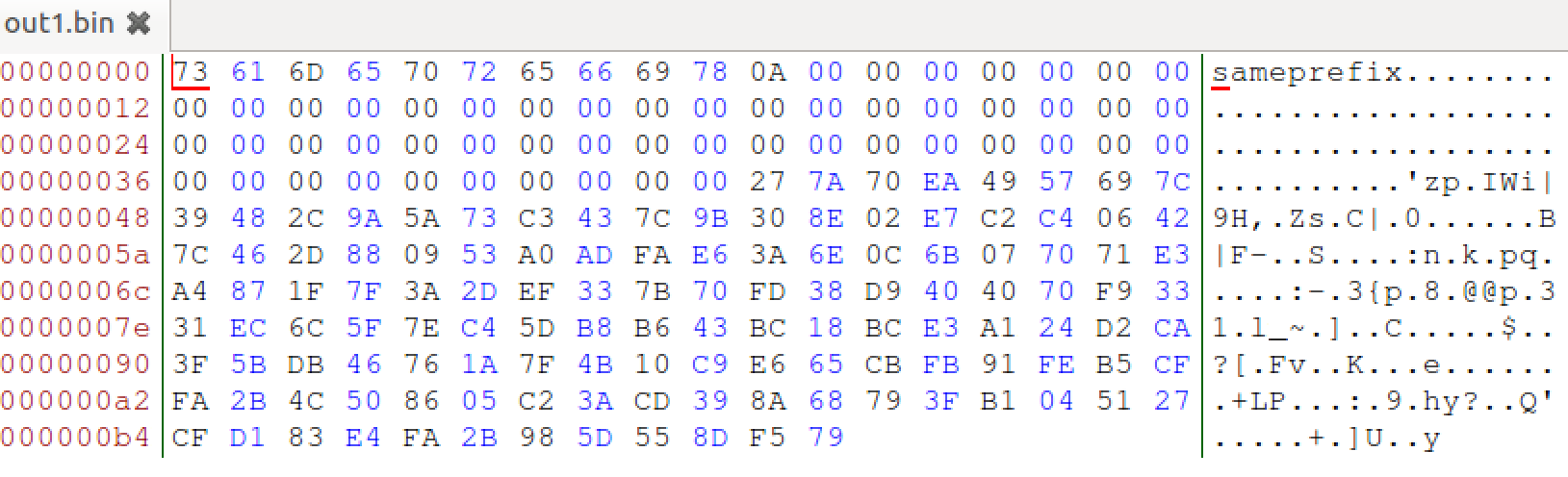
Created a text file named **prefix.txt**, having the text “sameprefix”.

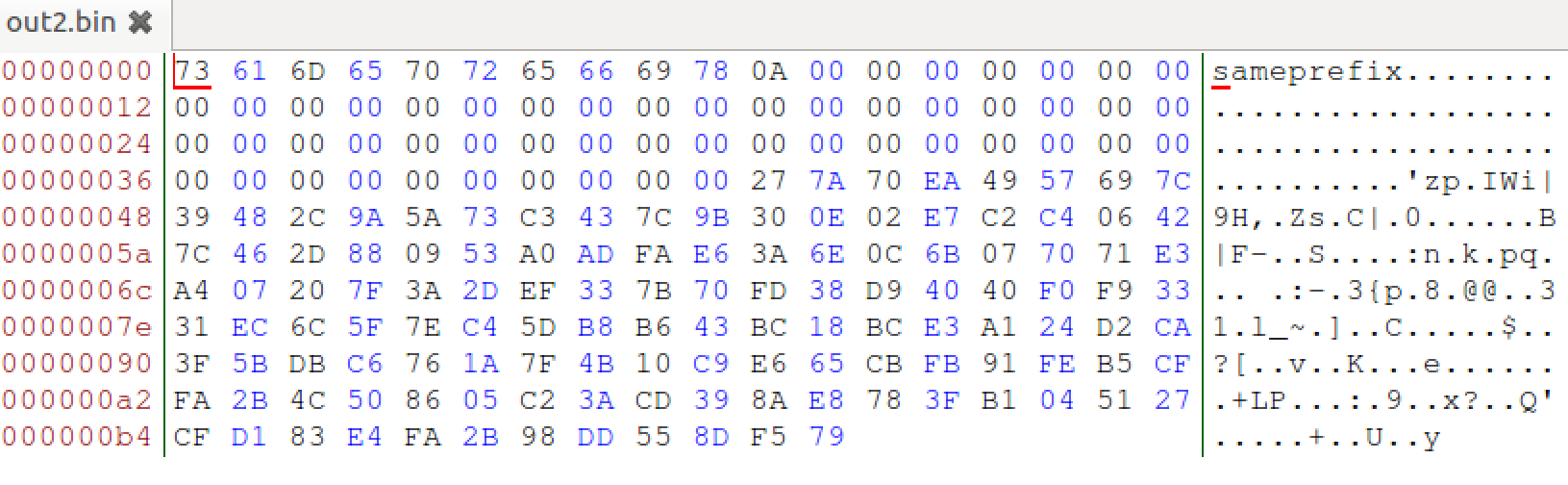
Using the following command in the terminal, **md5collgen -p prefix.txt -o out1.bin out2.bin**, it generates two output files that are different but sharing the same prefixes.



Using **diff** and **md5sum**, we can see that the two files differ but they have the same md5 hash values.

Opening the two files with a text editor, we see the following:

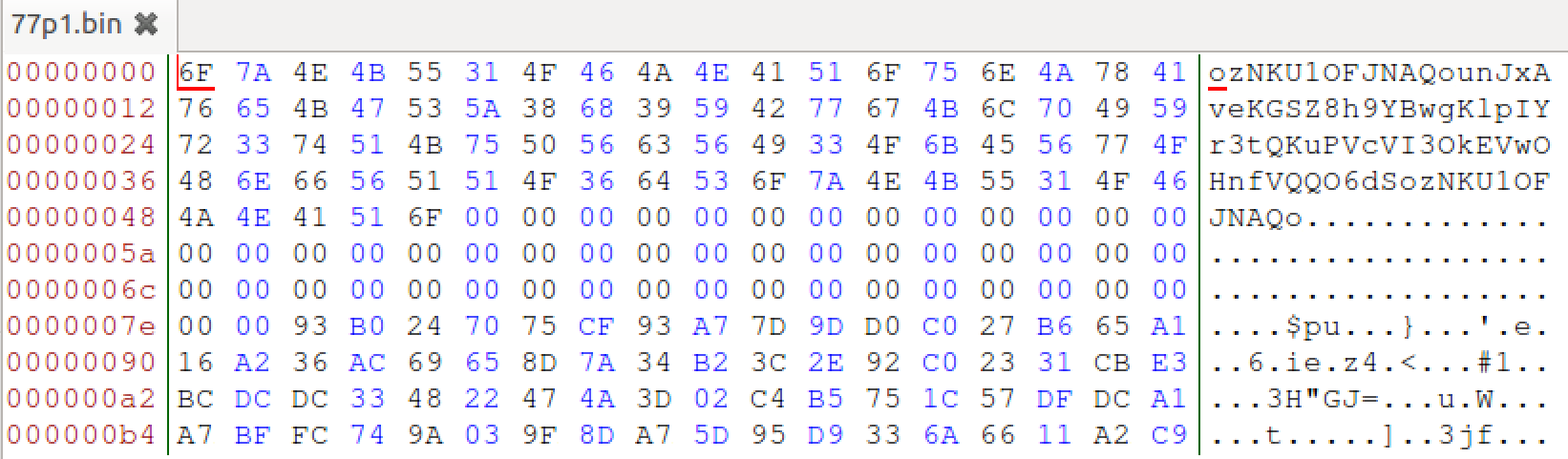




Looking at both files, there are some minor differences. The beginning is similar due to the same prefixes, and the adding of zeroes afterwards is similar, but after that there are some differences.

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

I made another file that is greater than 64 but not multiple of 64 (77 bytes), and the result is this:

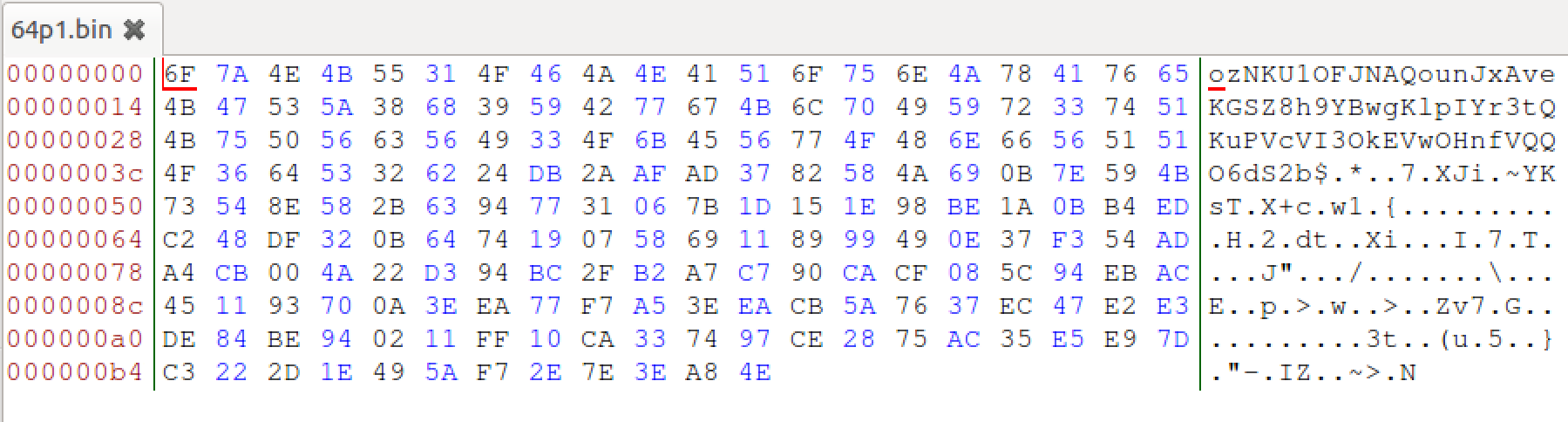


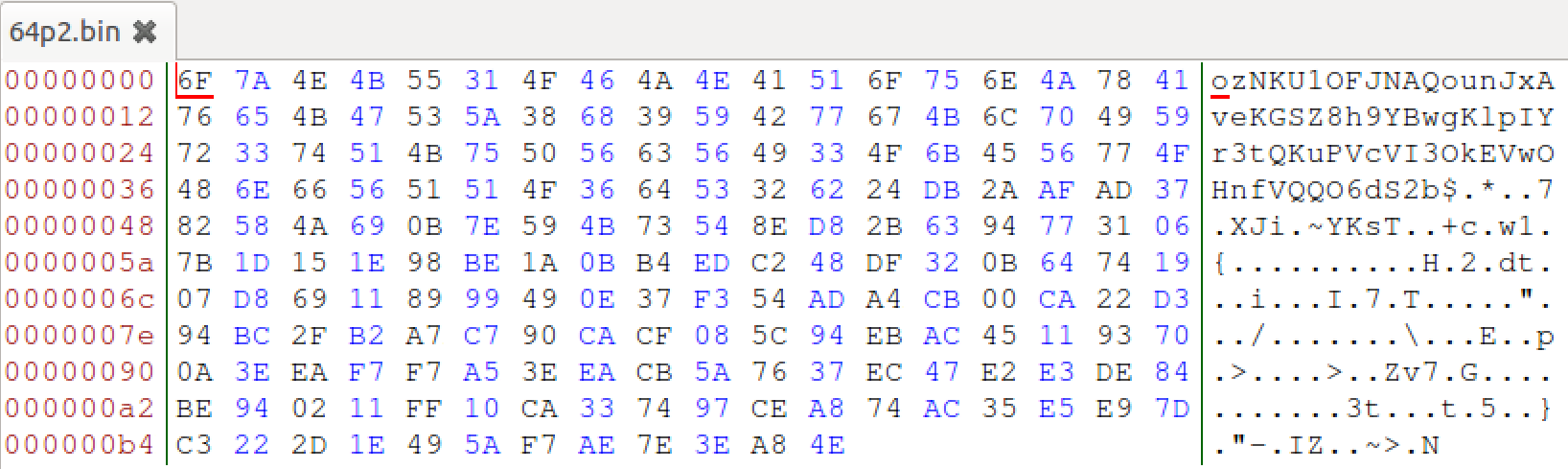
From looking at the previous prefix and this one, there will be a bunch of added zeroes if it is not a multiple of 64.

Thus, if the length of the prefix file is not a multiple of 64, there will be zero padding.

**Create a prefix file with exactly 64 bytes, and run the collision tool again, and see what happens**

Made a new prefix text file, making sure it is 64 bytes by using **truncate**. Looking at the two output files, there are no zero paddings. Thus, if the prefix file is exactly 64 bytes, there is no zero padding.





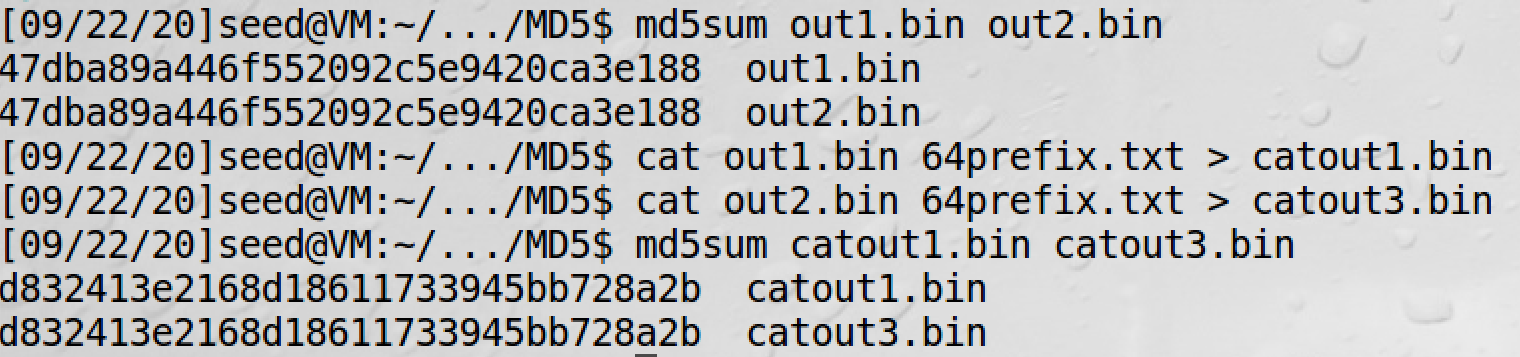
**Are the data (128 bytes) generated by md5collgen completely different for the two output files? Please identify all the bytes that are different.**

The data generated by md5collgen are not completely different for the two output files, most are the same but some bytes are different. Looking at the previous prefix file I made, the bolded are the differences:

6F 7A 4E 4B 55 31 4F 46 4A 4E 41 51 6F 75 6E 4A 78 41 76 65 4B 47 53 5A 38 68 39 59 42 77 67 4B 6C 70 49 59 72 33 74 51 4B 75 50 56 63 56 49 33 4F 6B 45 56 77 4F 48 6E 66 56 51 51 4F 36 64 53 32 62 24 DB 2A AF AD 37 82 58 4A 69 0B 7E 59 4B 73 54 8E **58** 2B 63 94 77 31 06 7B 1D 15 1E 98 BE 1A 0B B4 ED C2 48 DF 32 0B 64 74 19 07 **58** 69 11 89 99 49 0E 37 F3 54 AD A4 CB 00 **4A** 22 D3 94 BC 2F B2 A7 C7 90 CA CF 08 5C 94 EB AC 45 11 93 70 0A 3E EA **77** F7 A5 3E EA CB 5A 76 37 EC 47 E2 E3 DE 84 BE 94 02 11 FF 10 CA 33 74 97 CE **28** **75** AC 35 E5 E9 7D C3 22 2D 1E 49 5A F7 **2E** 7E 3E A8 4E

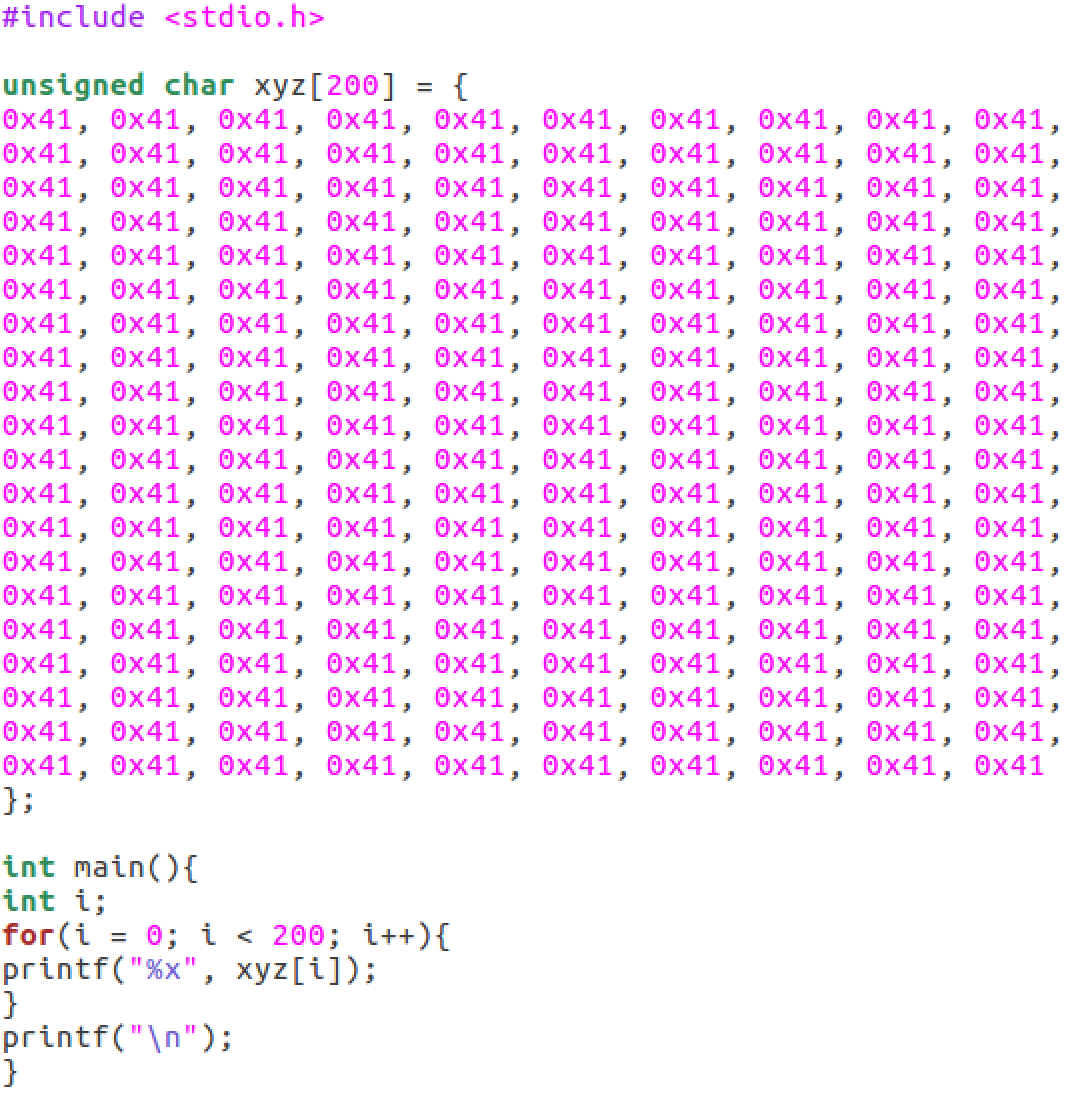
6F 7A 4E 4B 55 31 4F 46 4A 4E 41 51 6F 75 6E 4A 78 41 76 65 4B 47 53 5A 38 68 39 59 42 77 67 4B 6C 70 49 59 72 33 74 51 4B 75 50 56 63 56 49 33 4F 6B 45 56 77 4F 48 6E 66 56 51 51 4F 36 64 53 32 62 24 DB 2A AF AD 37 82 58 4A 69 0B 7E 59 4B 73 54 8E **D8** 2B 63 94 77 31 06 7B 1D 15 1E 98 BE 1A 0B B4 ED C2 48 DF 32 0B 64 74 19 07 **D8** 69 11 89 99 49 0E 37 F3 54 AD A4 CB 00 **CA** 22 D3 94 BC 2F B2 A7 C7 90 CA CF 08 5C 94 EB AC 45 11 93 70 0A 3E EA F7 **F7** A5 3E EA CB 5A 76 37 EC 47 E2 E3 DE 84 BE 94 02 11 FF 10 CA 33 74 97 CE **A8** **74** AC 35 E5 E9 7D C3 22 2D 1E 49 5A F7 **AE** 7E 3E A8 4E

## Part 2.2 – Understanding MD5’s Property

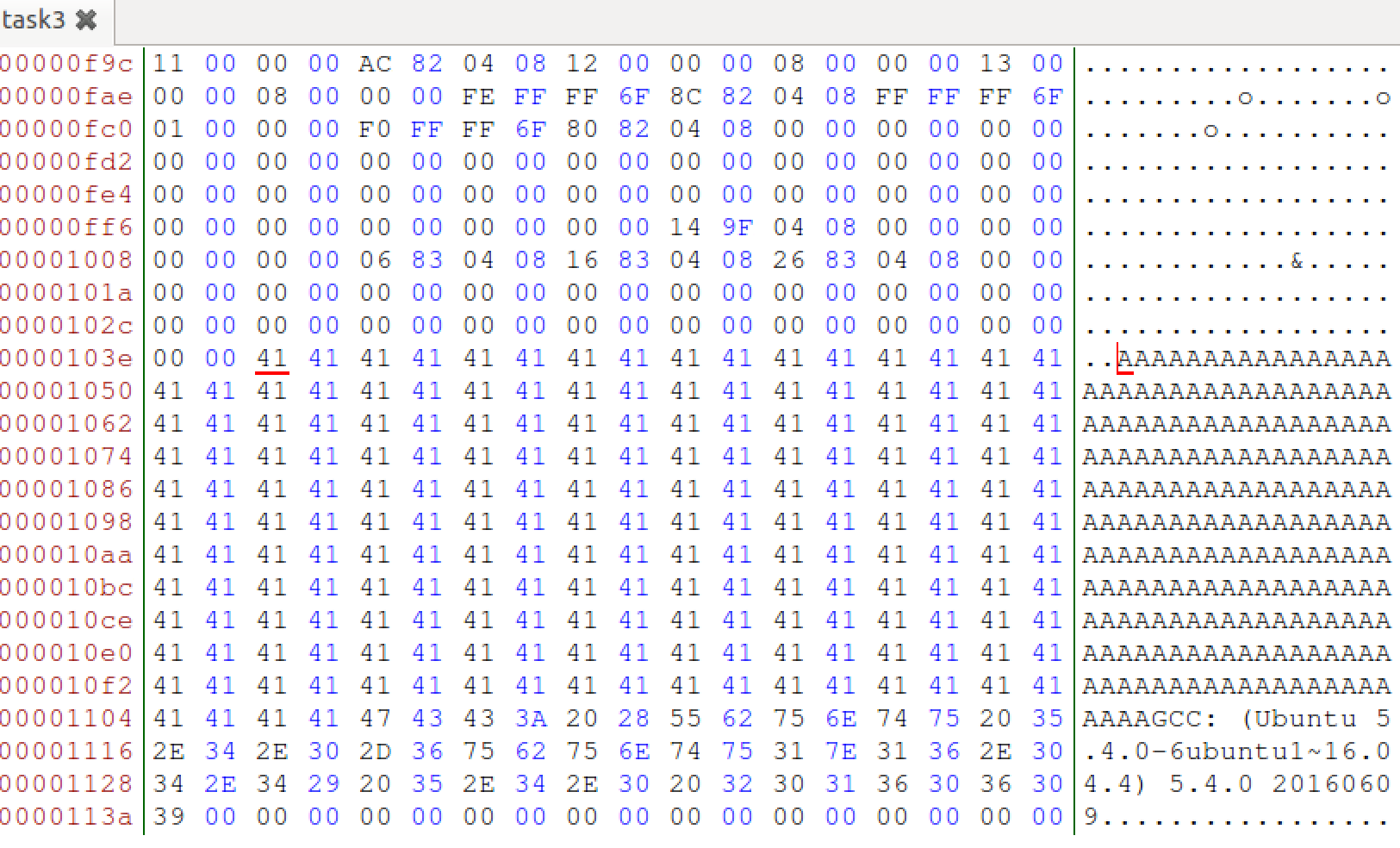
****

The screenshot above shows that, with two files having the same MD5 hashes, and adding the same suffix (concatenating both files with the same text file) to both of them will result in two outputs that have the same MD5 hash value. This value can be found using **md5sum**, and before and after the concatenation, the two files have same hash values. Thus, given two inputs M and N, if MD5(N)=MD5(N), then for any input T, MD5(M || T) = MD5(N || T), where || represents concatenation.

## Part 2.3 – Generating two executable files with the same MD5 hash



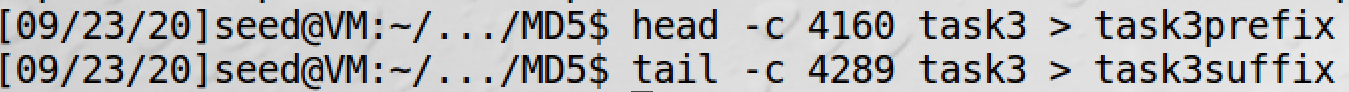
Compiling the above code and using **bless** to view the binary executable file, we get this:



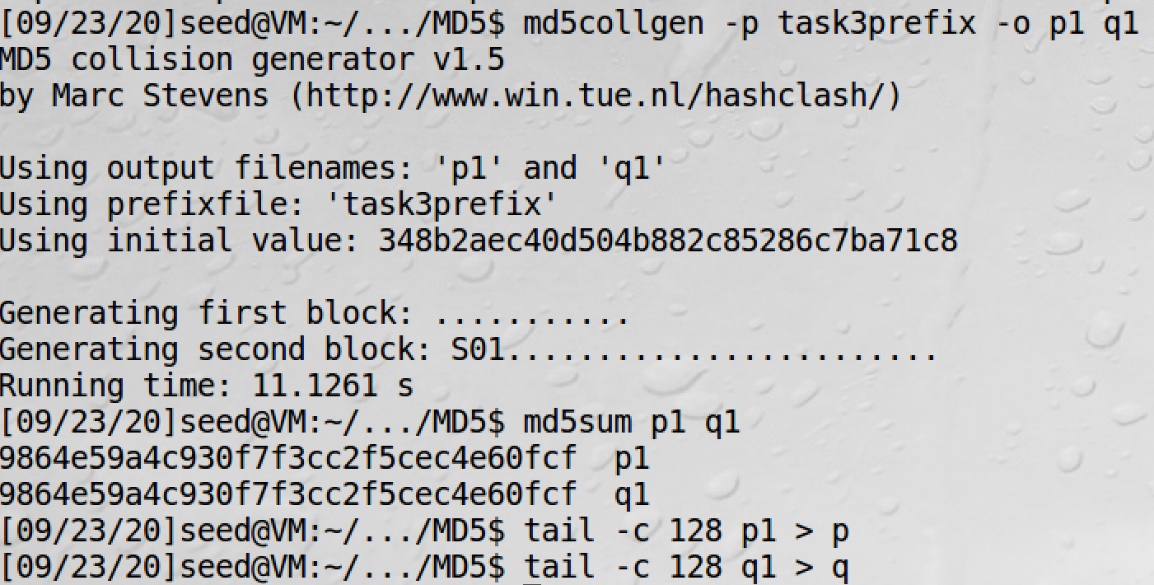
We see the “A”s from our code, and before that we can set as the prefix, and after all the “A”s, is the suffix. Due to how MD5’s property, as long as the same suffix is appended to both files, the MD5 value will still be the same for both files, even if the data in the middle are different.

In my executable file, the “A”s starts at 0x1040, or 4160, and ends at 0x1107, or 4359.

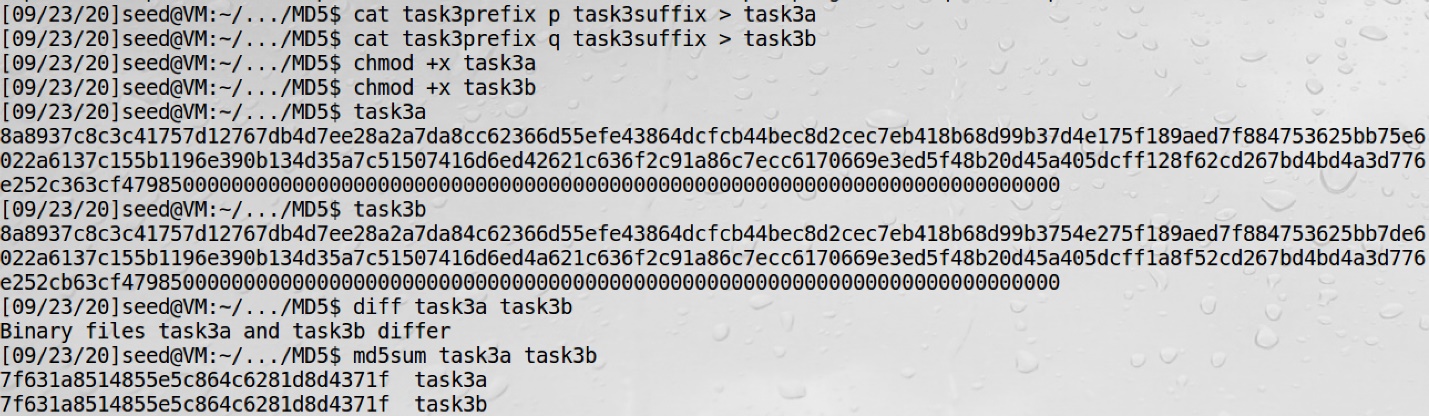
I will then divide the executable file into 3 parts, prefix, a 128-byte region, and a suffix. The prefix is the data before 4160, which is fine because it is a multiple of 64. Then 128 bytes after, and after that is the suffix.



Generating two files that have the same MD5 hash value and 128 bytes after the prefix, we get that MD5(prefix || p) = MD5(prefix || q):



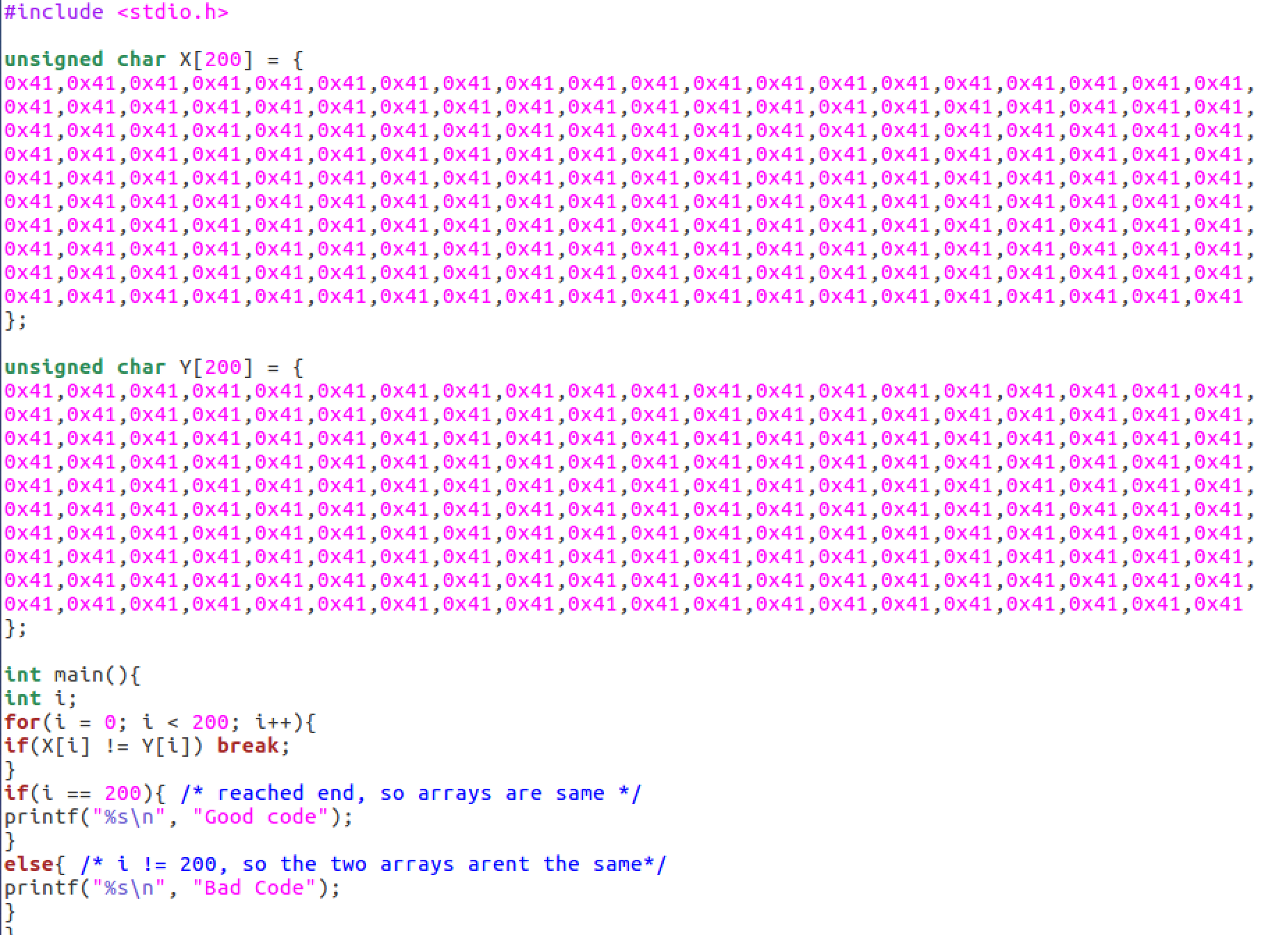
Now, we can concatenate p and q with the prefix and suffix, and get two executable files that have the same MD5 hash value but behave differently: MD5(prefix || p || suffix) = MD5(prefix || q || suffix)



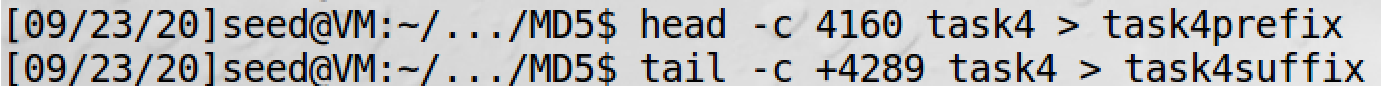
Thus, we now know and have two executable files with the same MD5 hash but what they print out are different.

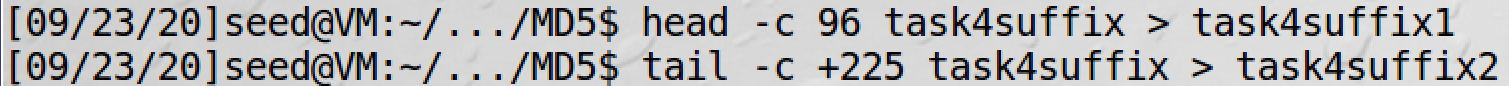
## Part 2.4 – Making the two programs behave differently

Made the following c program, that has two arrays, X and Y, that are the same. If they are the same, then it will print out “Good code”, if not, then it will print out “Bad code”.



Doing a similar process of the previous task, I get the prefix of the executable, right before the data of the first array and also the suffix afterwards. But, we also want to change the array content of the second array so we want the data between them and another suffix afterwards.

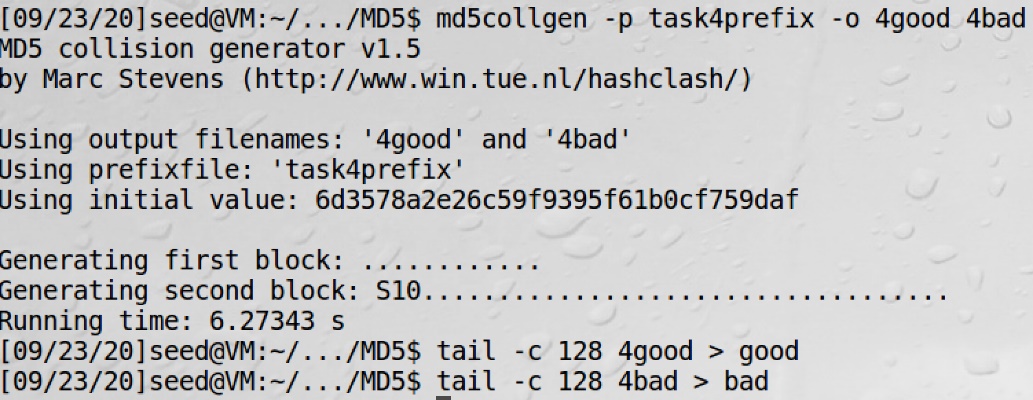




Now the following could be done:

MD5(prefix || X || suffix1 || X || suffix2) = MD5(prefix || Y || suffix1 || X || suffix2)

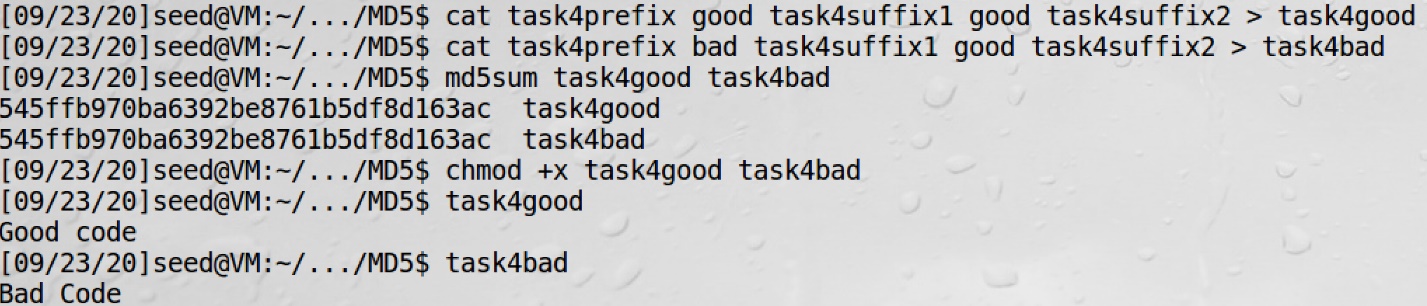
Generating two files again that have the same MD5 hash value and getting the 128 bytes after the prefix:



“good” and “bad” will be used to change the content of the arrays. Inserting “good” in both arrays, results in good code, while inserting “bad” and then “good” will result in malicious code because it sees that the arrays are not the same. Thus, concatenating them with the prefix and suffixes, we get two executable programs with the same MD5 hash but one executes malicious code and other is good code. The process can be written as an equation similar to previous tasks:

MD5(prefix || good || suffix1 || good || suffix2) = MD5(prefix || bad || suffix1 || good || suffix2)

Because of the insertions, the program sees that the arrays are no longer the same, so it executes the malicious code. However, because of MD5’s property, the MD5 hash stays the same for both programs even though the array content of the other program is different.



Thus, both programs have same MD5 hash value. The first version, “task4good”, have the contents of the X and Y array the same, so it runs good code. However, the second version, “task4bad”, the contents of the array are different which makes the program run malicious code.

## Final Thoughts

This was a very interesting lab, as I have never known about MD5 before this class, and also how easy it is to essentially to “mask” malicious code as good due to having the same MD5 hash value of a certified benign program. Also, this was my first time really using Ubuntu and VirtualBox, so it was a nice introduction to those as well.