Lab 3: MD5 Collision

Sahil Sheikh A20518693

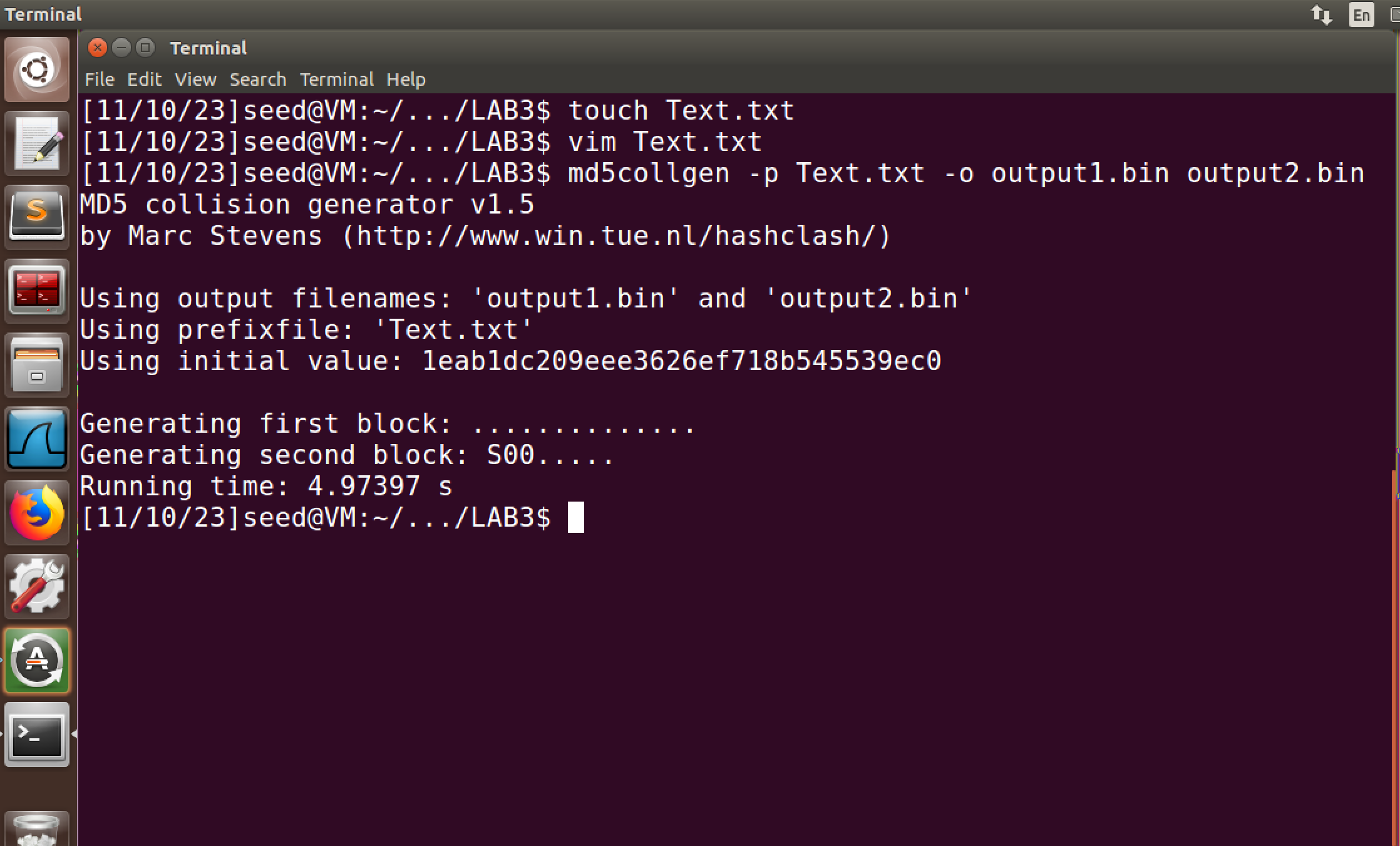
Introduction:

A collision in the context of cryptographic hash function MD5 (Message Digest Algorithm 5), occurs when two different inputs produce the same hash output. Ideally, a cryptographic hash function should produce a unique hash value for each unique input, making it computationally infeasible for two different inputs to generate the same hash value.

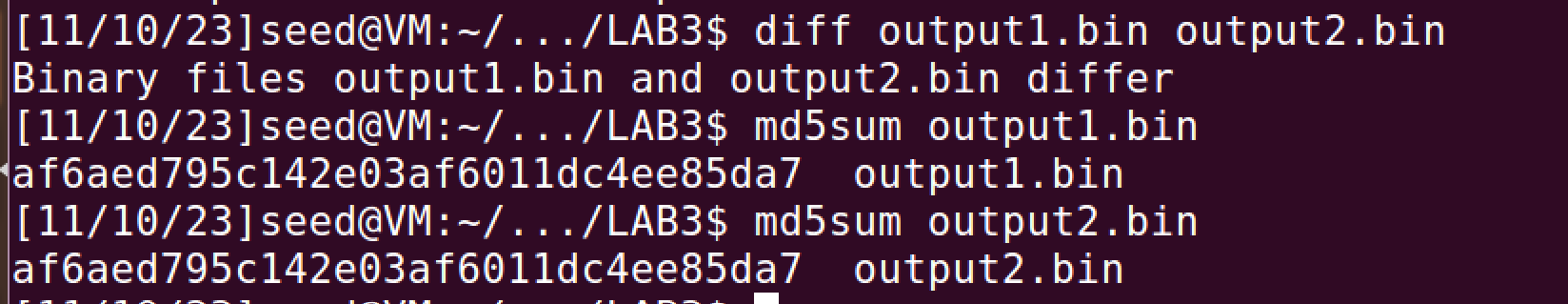
However, MD5 is considered broken for cryptographic purposes because researchers have demonstrated the ability to find collisions efficiently. Because of such vulnerabilities, MD5 is no longer considered secure for cryptographic purposes, and more secure hash functions like SHA-256 are recommended for applications requiring collision resistance and cryptographic strength. It's important to note that cryptographic hash functions should ideally be resistant to collisions, meaning that it should be computationally infeasible to find two different inputs that produce the same hash value.

##### Task I: Generating two files with same hash

Creating a file “Text.txt” then creating two md5 file with same hash values



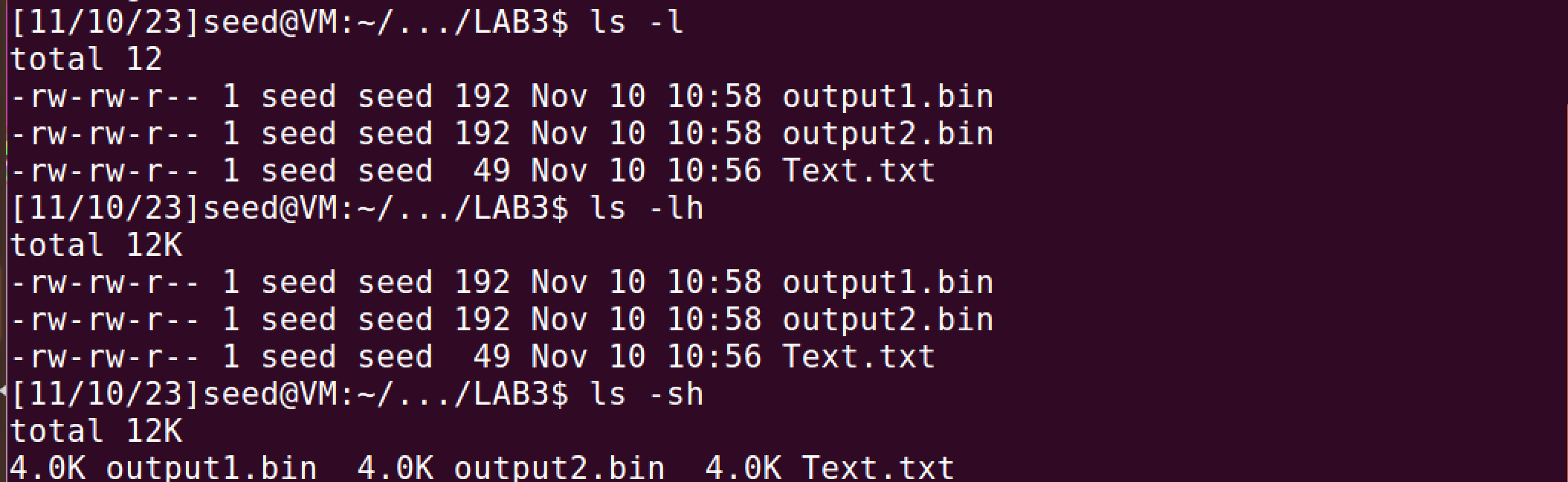
Using diff and md5sum command to check if the files are unique or not



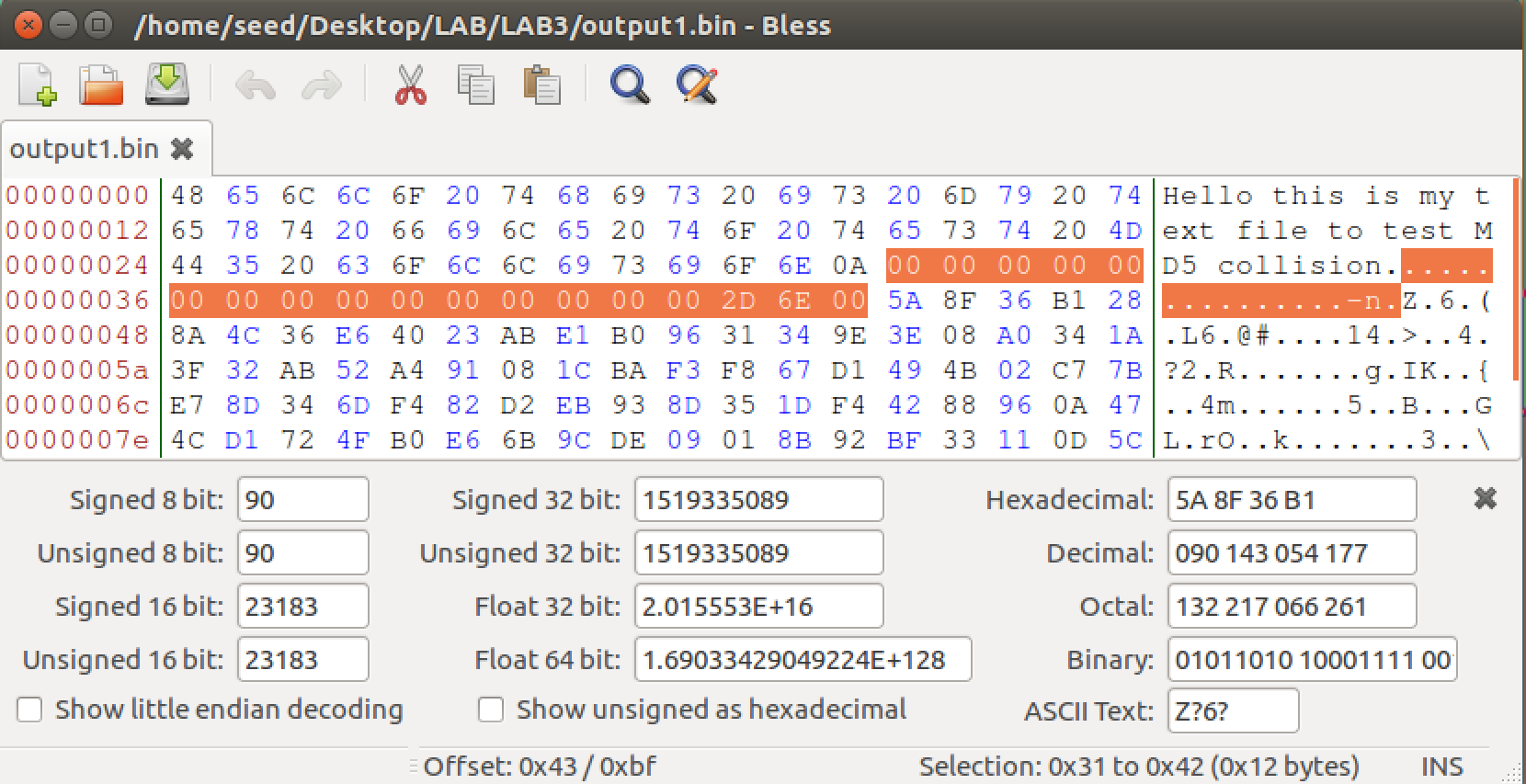
Q1.

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

In the above case the size of my file, “Text.txt” was 49 bytes(refer below for size related information)



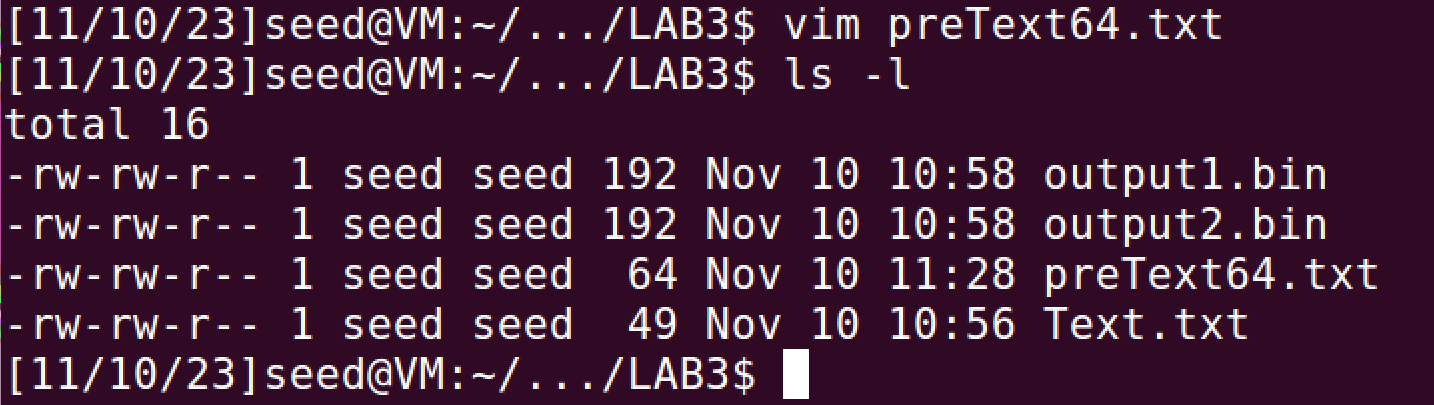
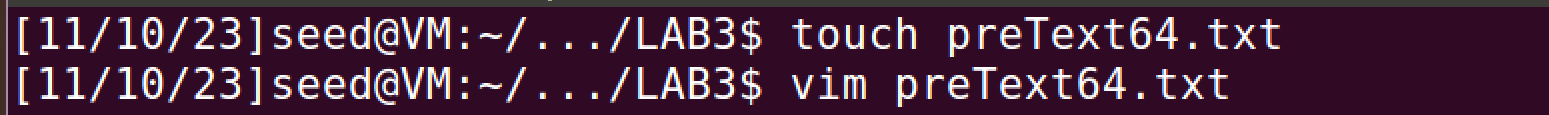
The file will be padded with zeros . I’ll view the output1.bin file in hex editor to verify this

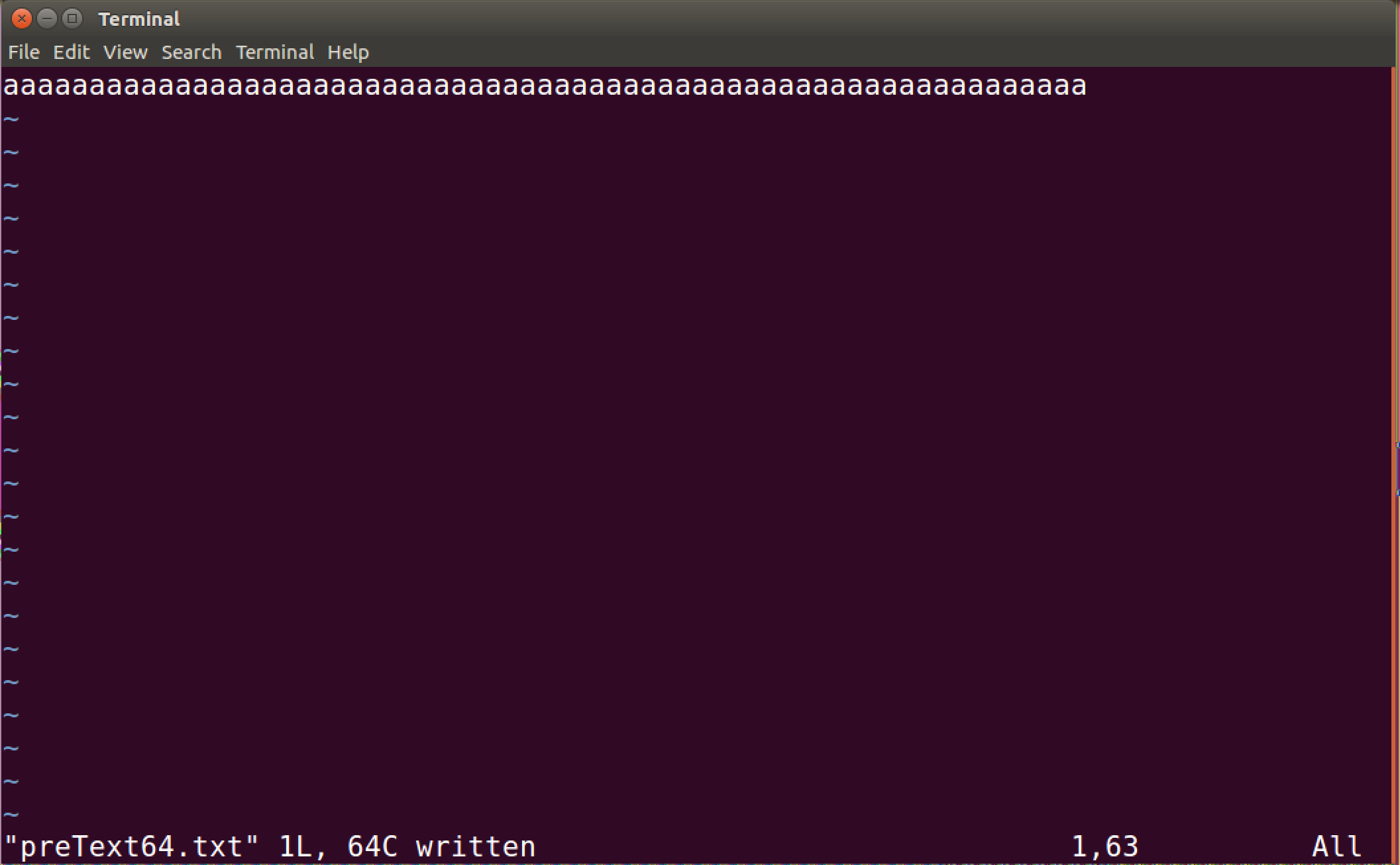


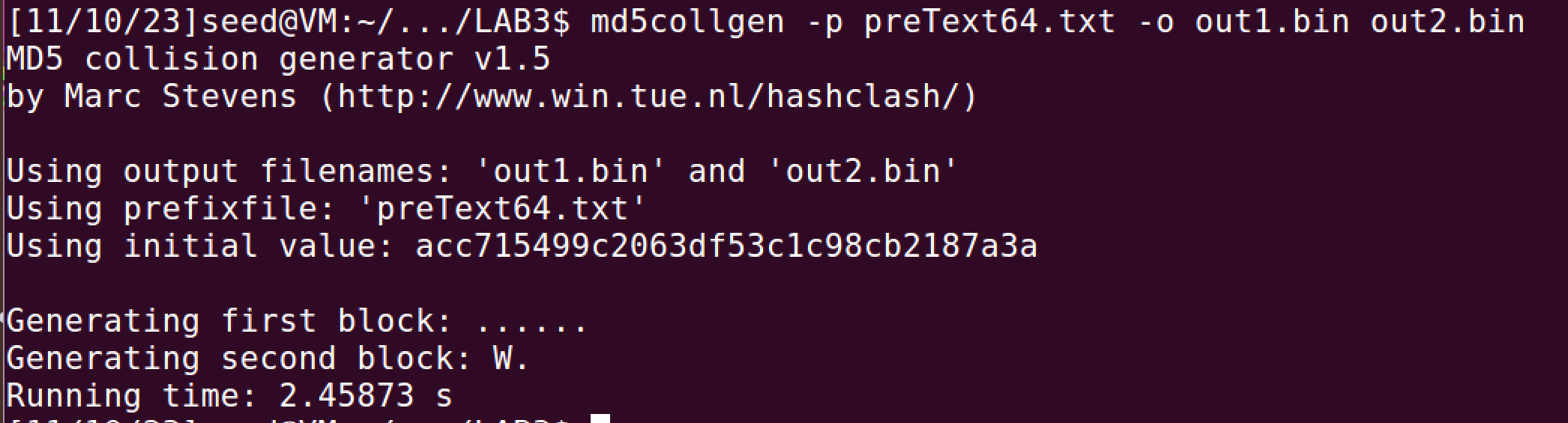
Q2.

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

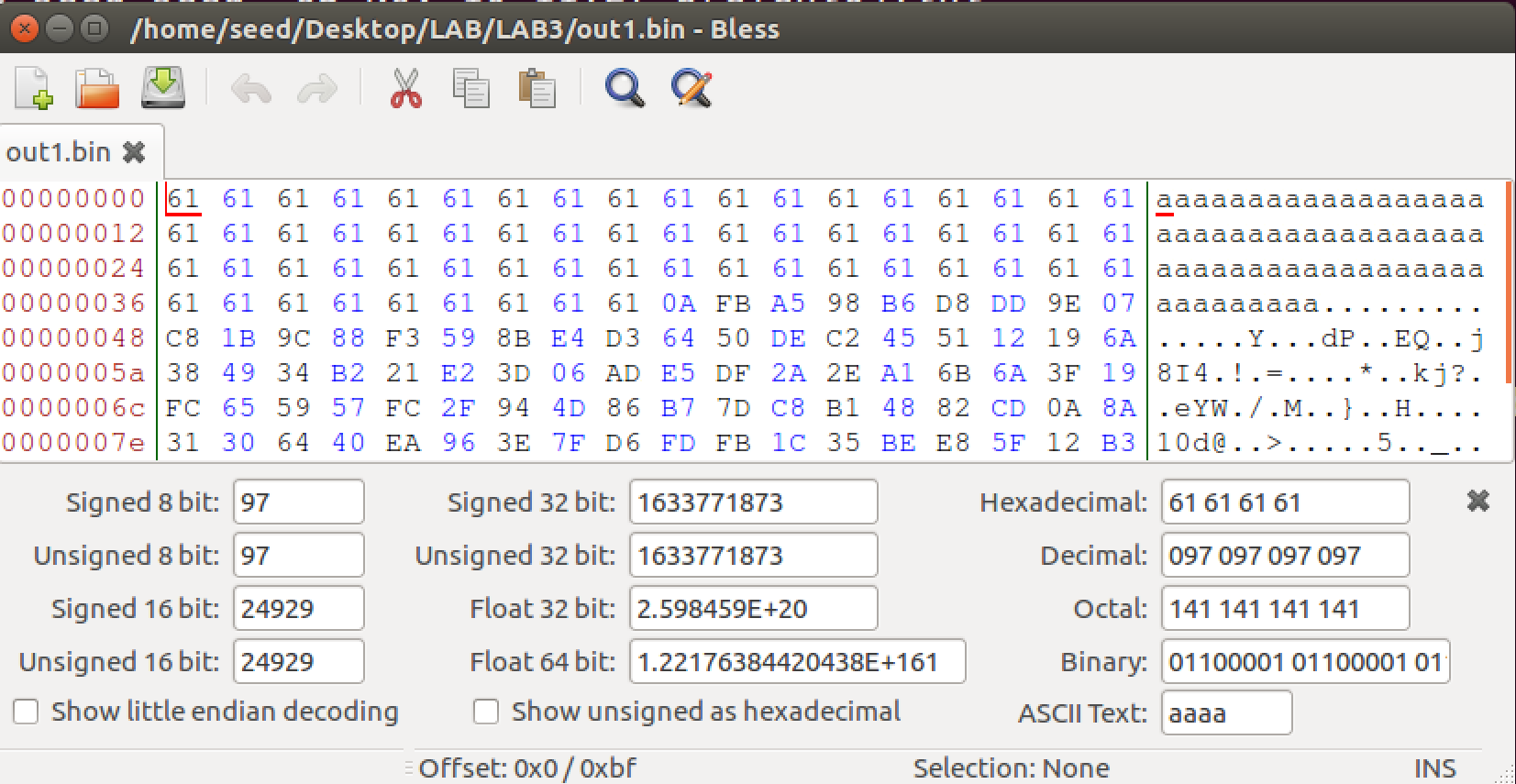
Creating a new pretext file “preText.txt” of 64 bytes in size then creating two md5 file with same hash values. Then using diff and md5sum command to check if the files are unique or not





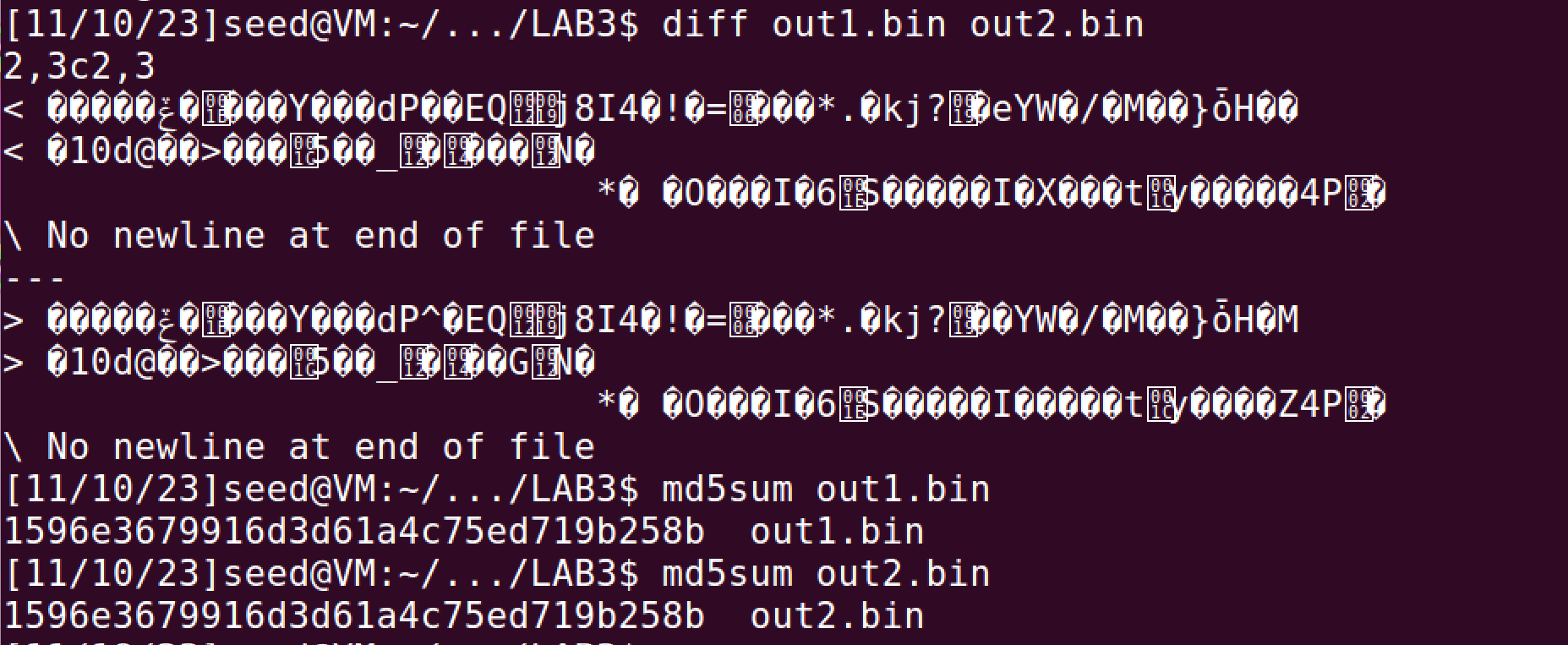


I can observe this file of size 64 bytes has no padding added to when the hash file is generated



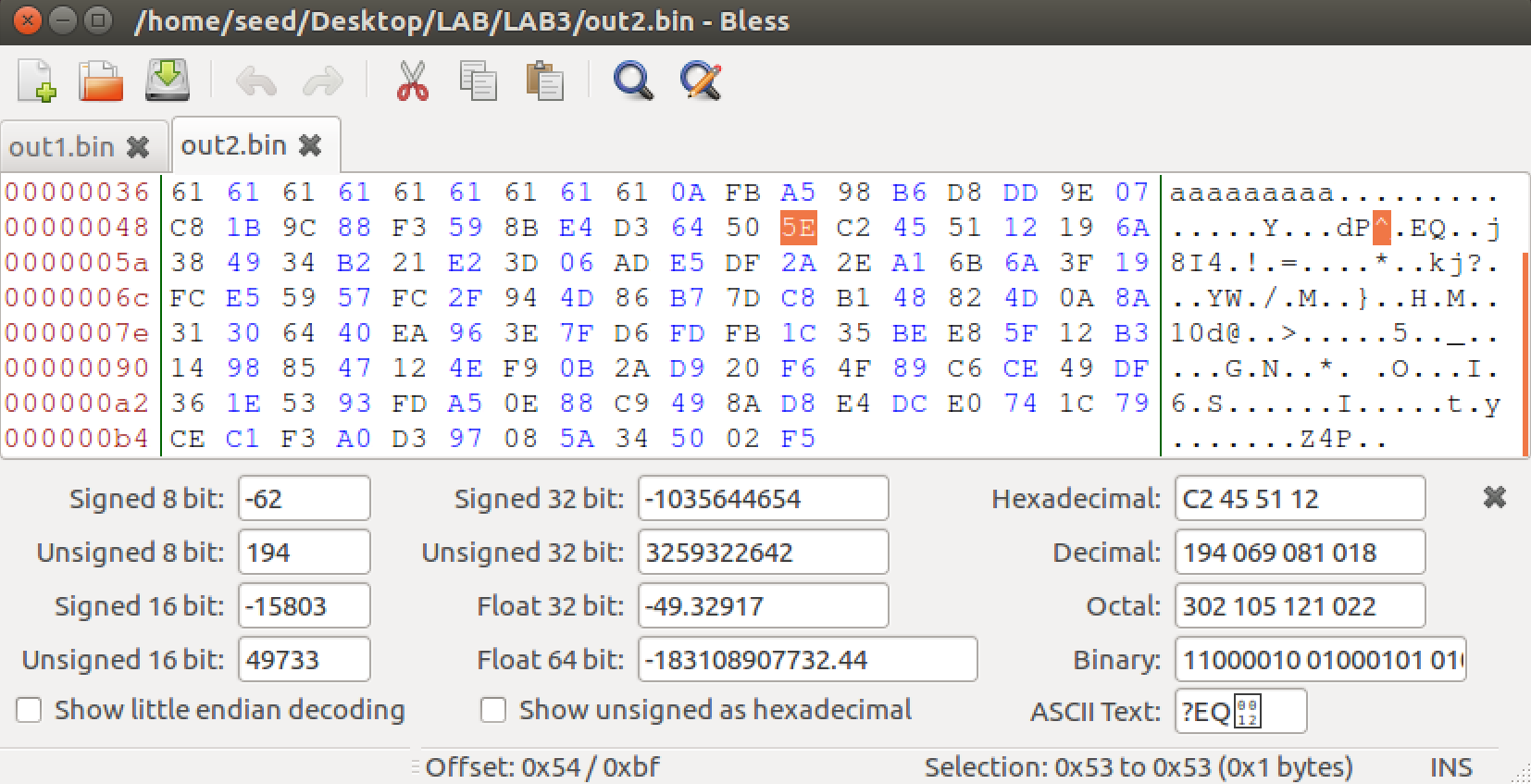
Q3.

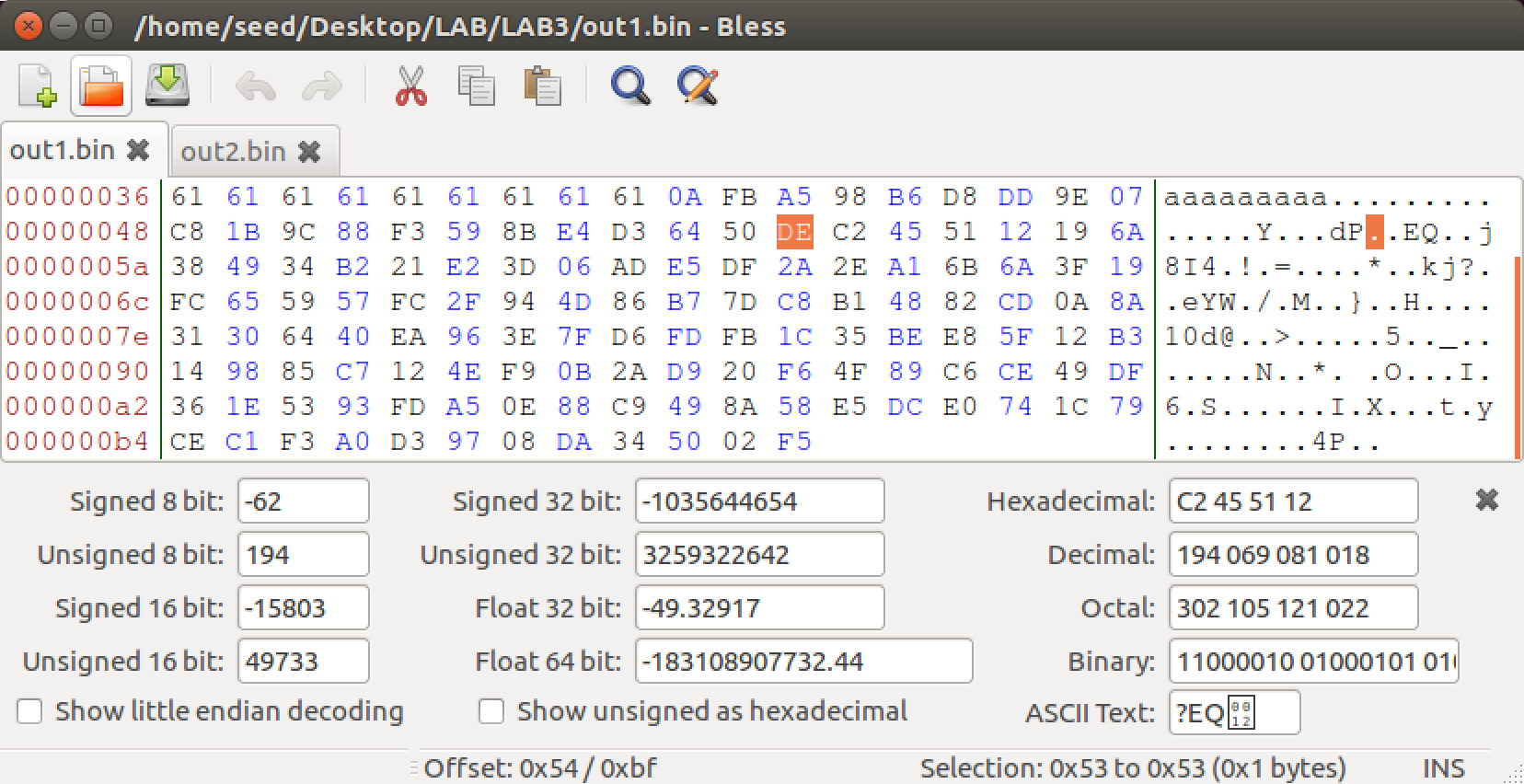
*Are the data ( 128 bytes) generated by md5collgen completely different for the two output files? Please identify all the bytes that are different.*

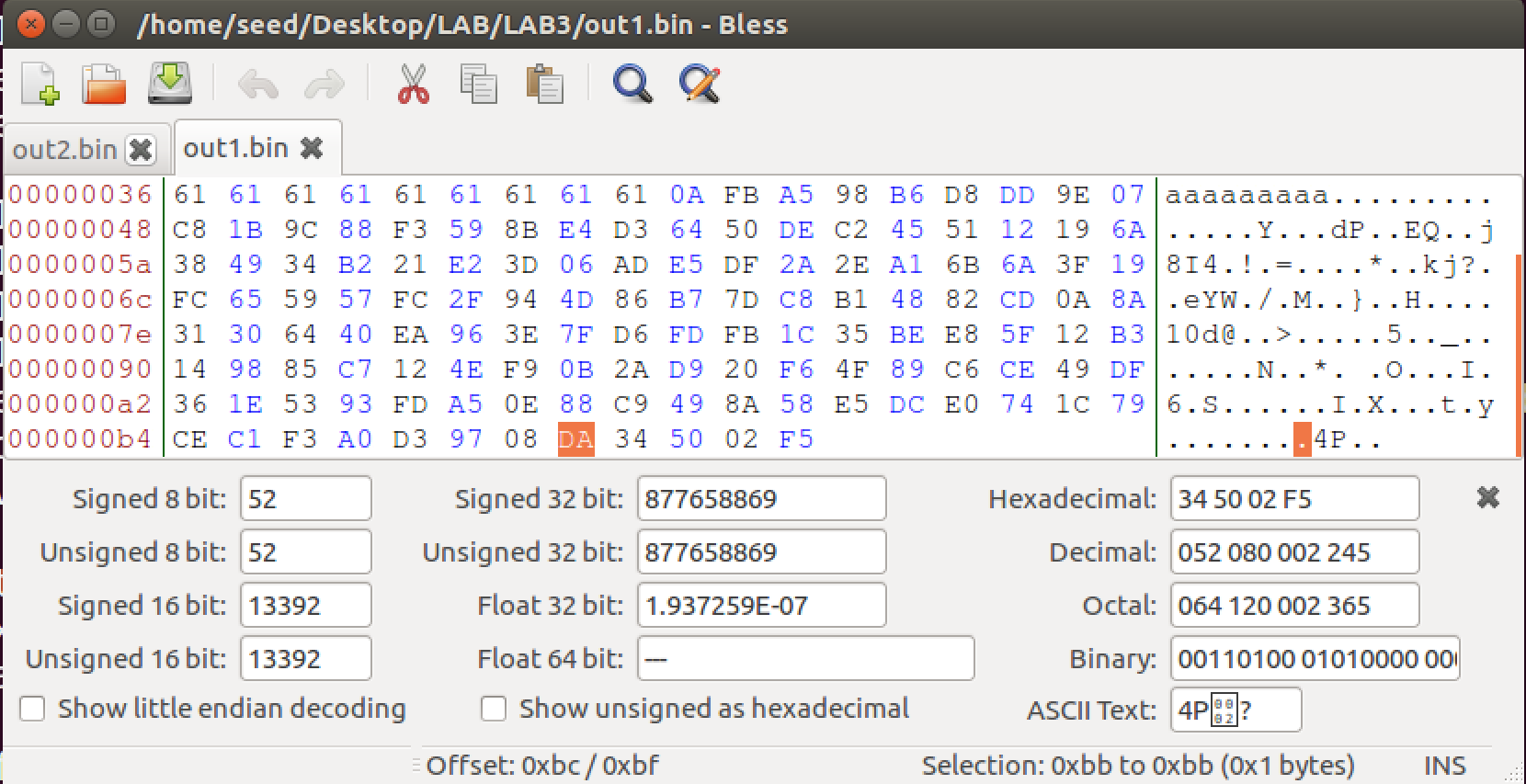


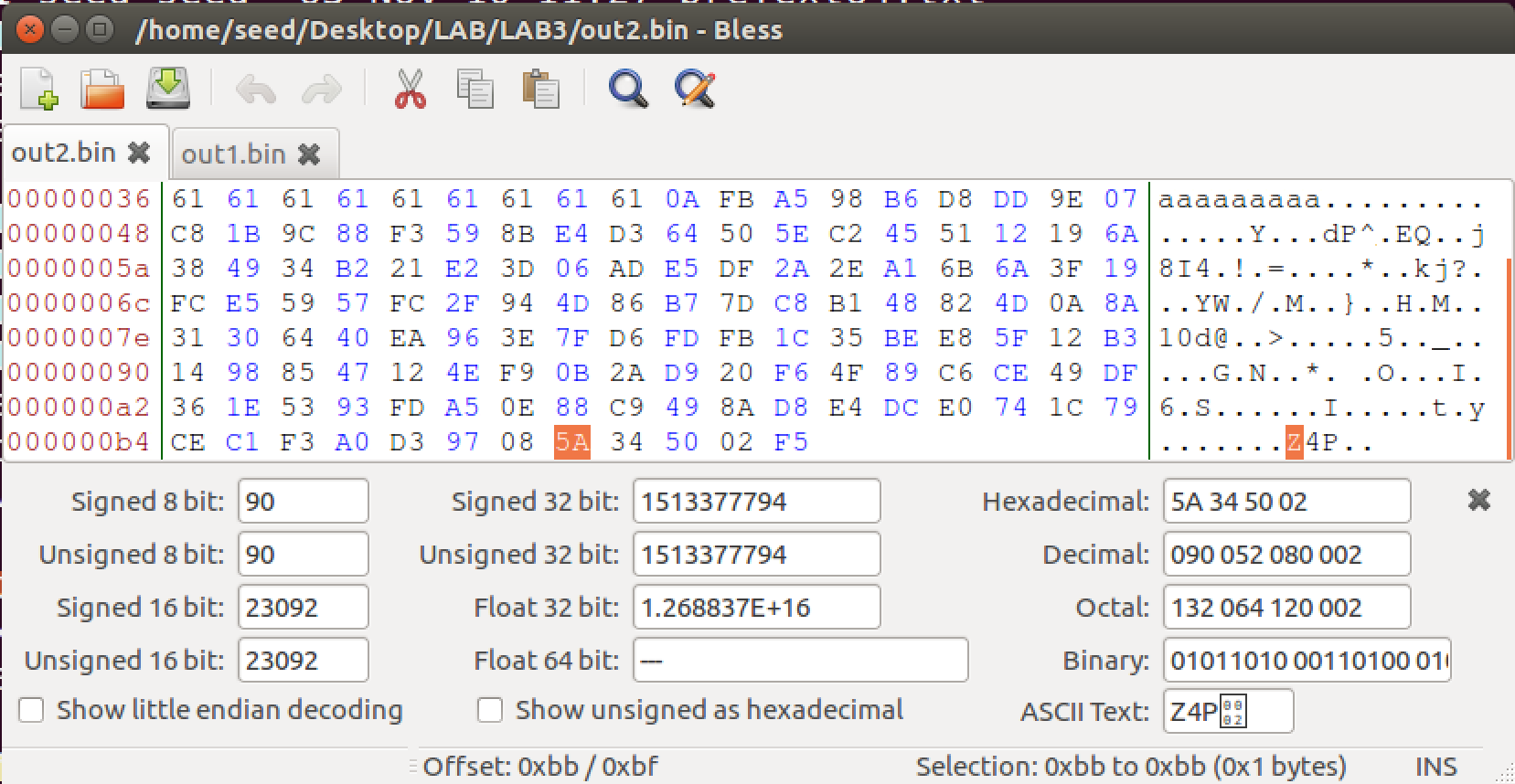
Both the output files out1.bin and out2.bin have the same hash values. (refer above pic)

Out of all the 128 bytes generated by the md5collgen few of the bytes are different. Here are those bits:

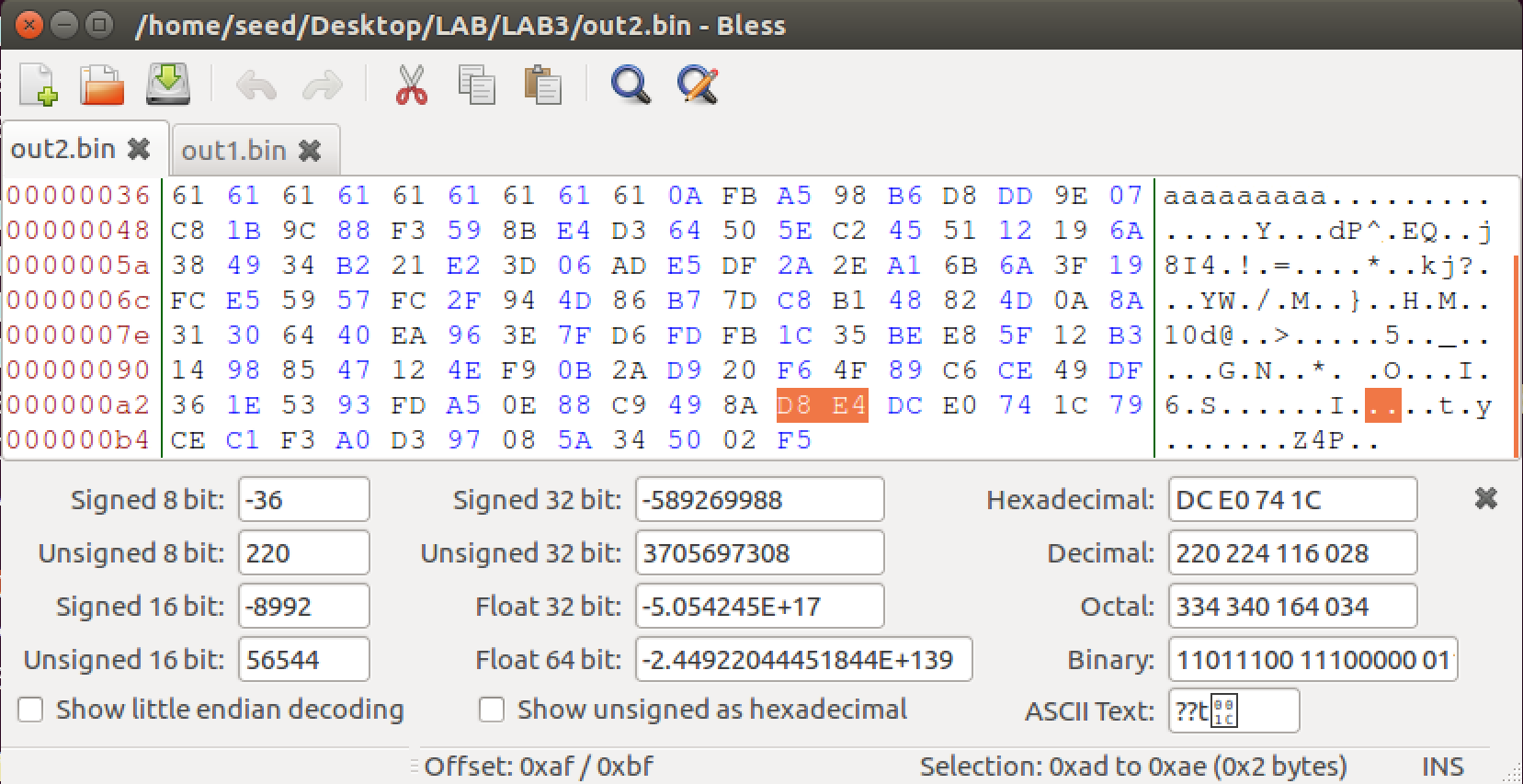


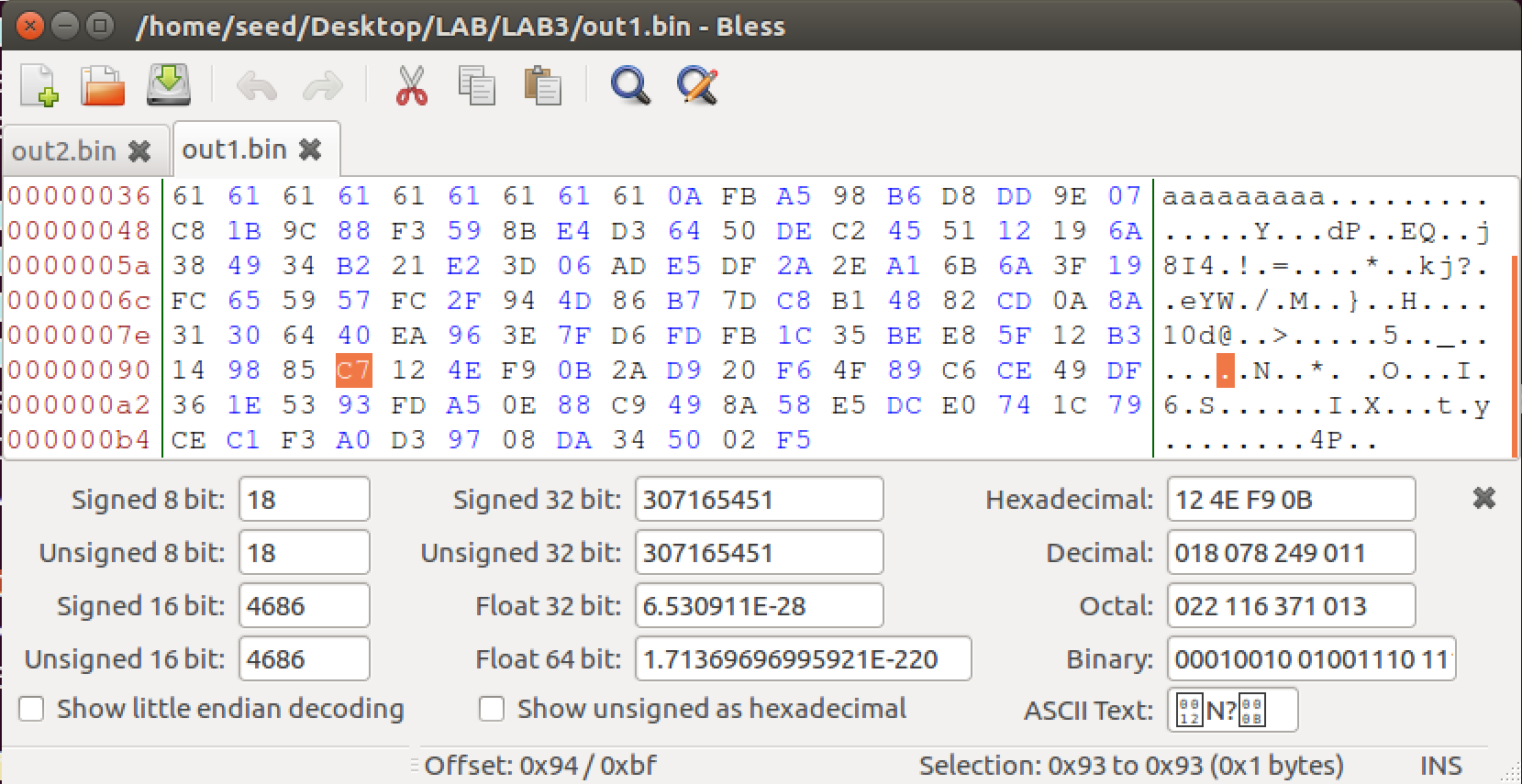


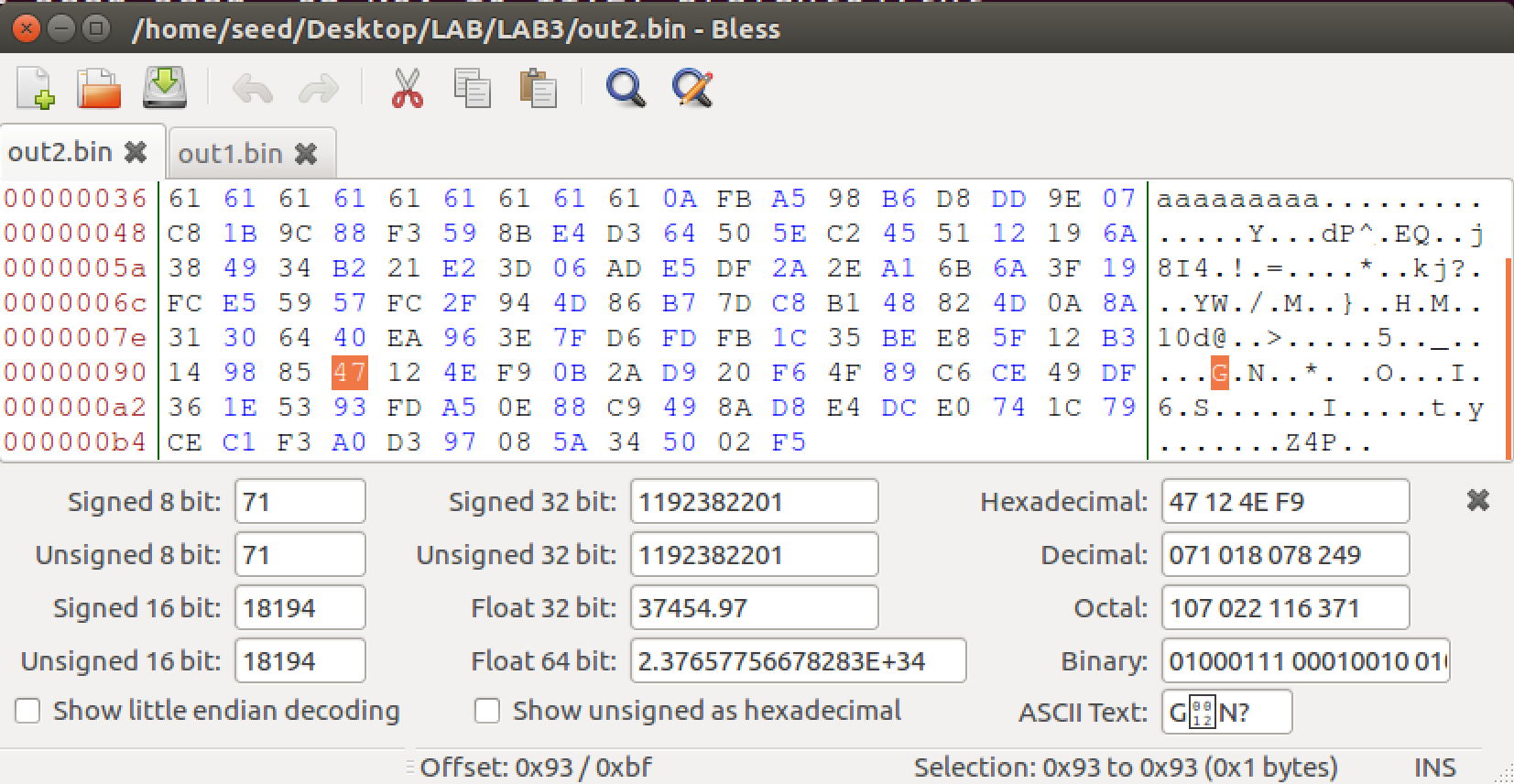


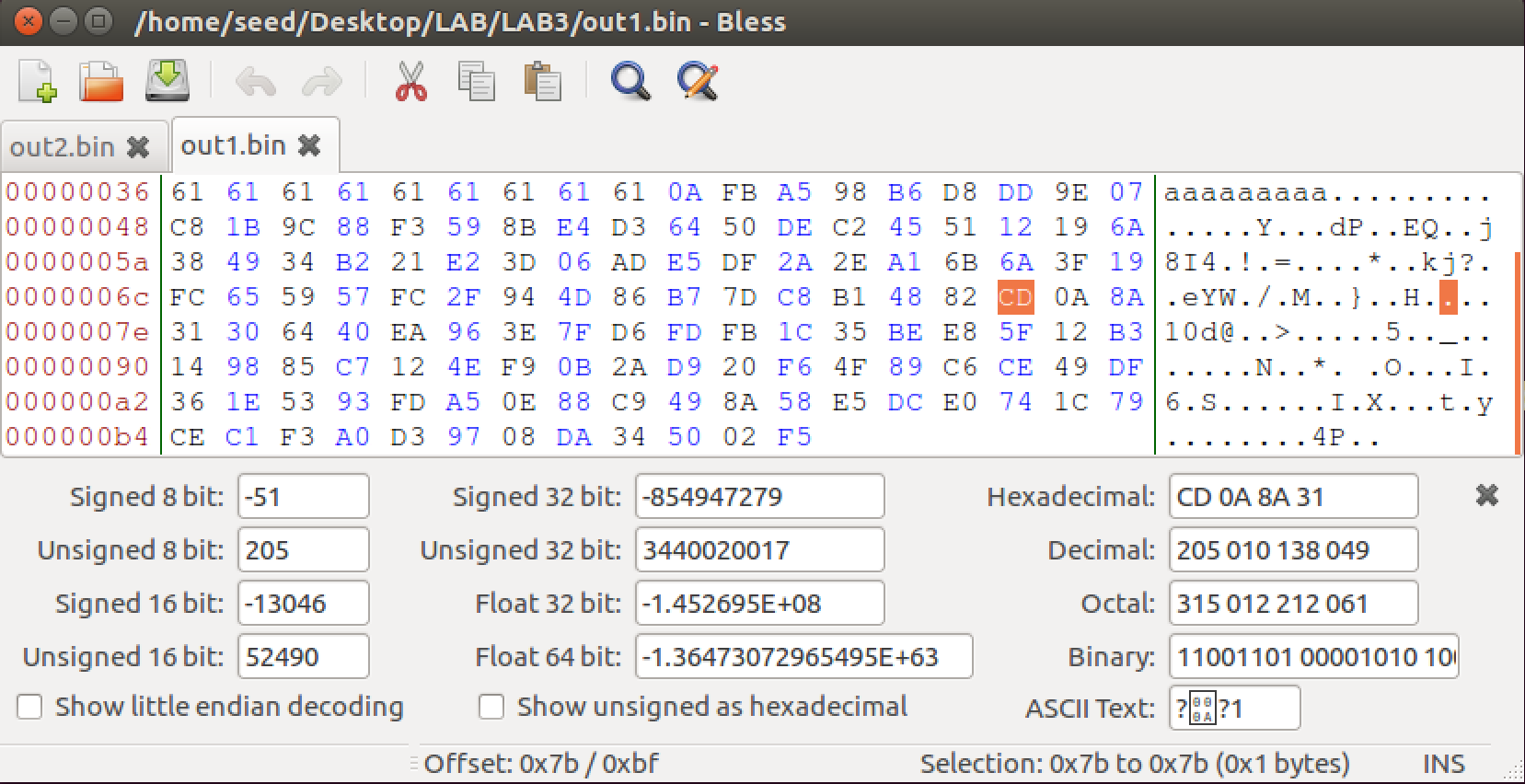


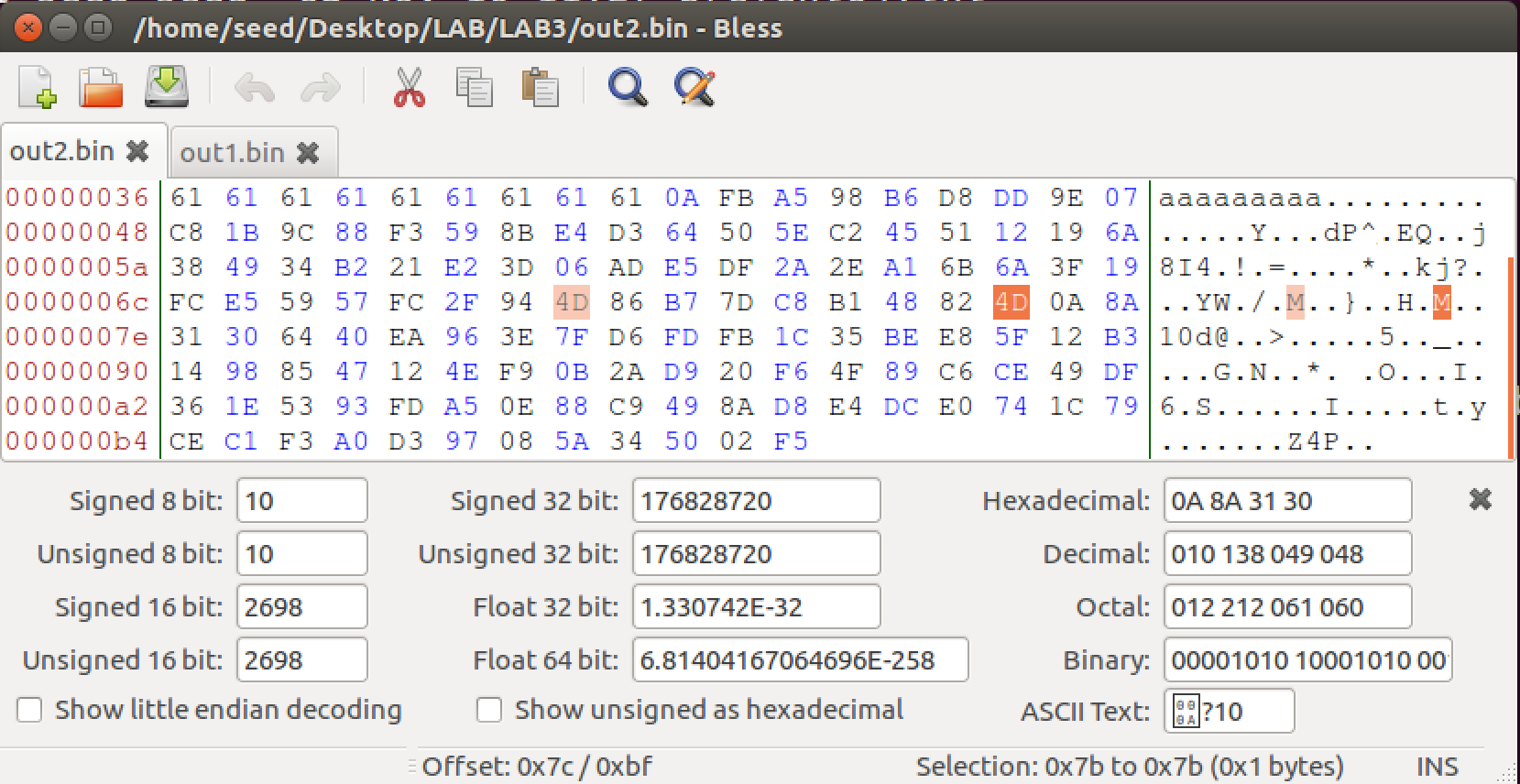


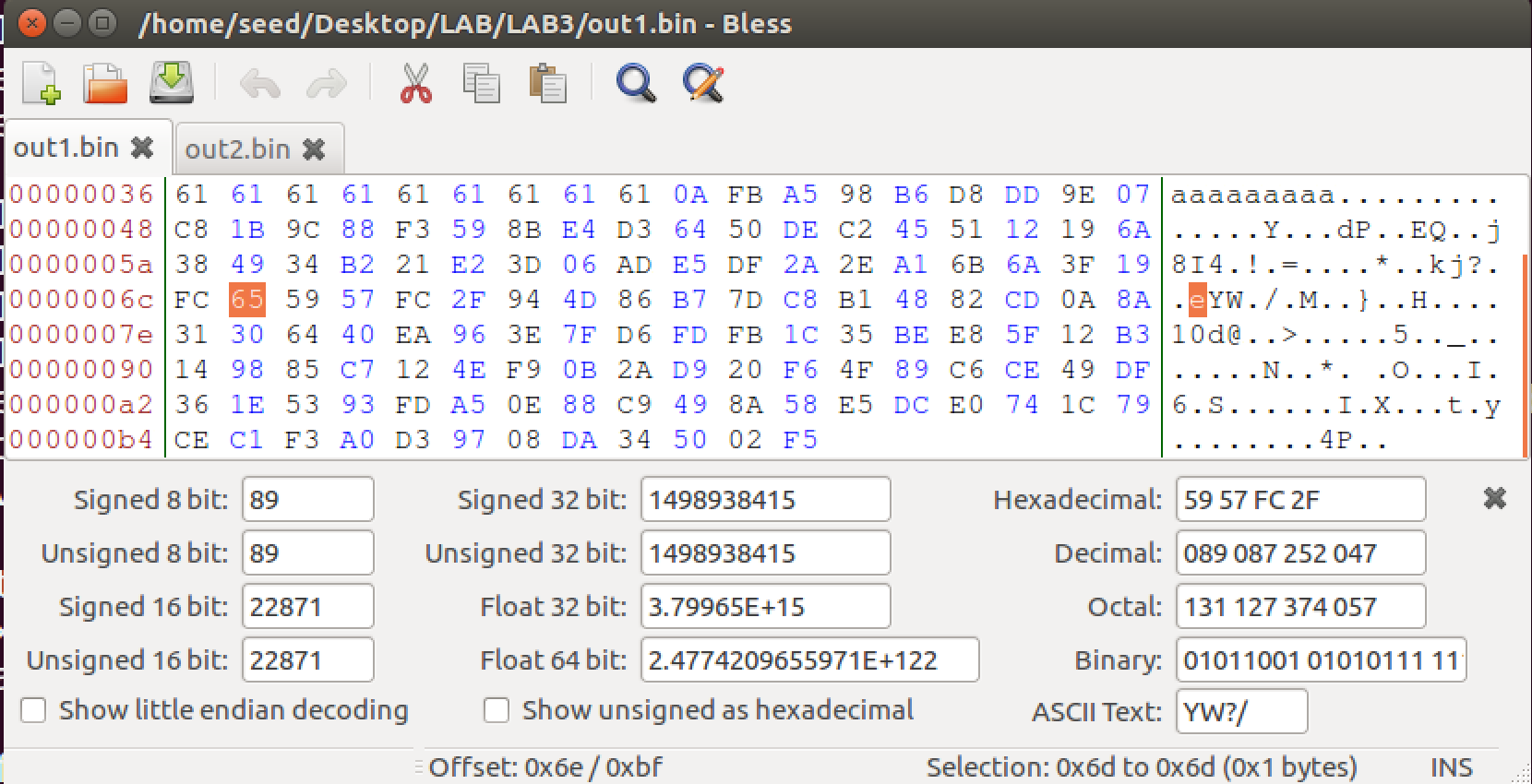


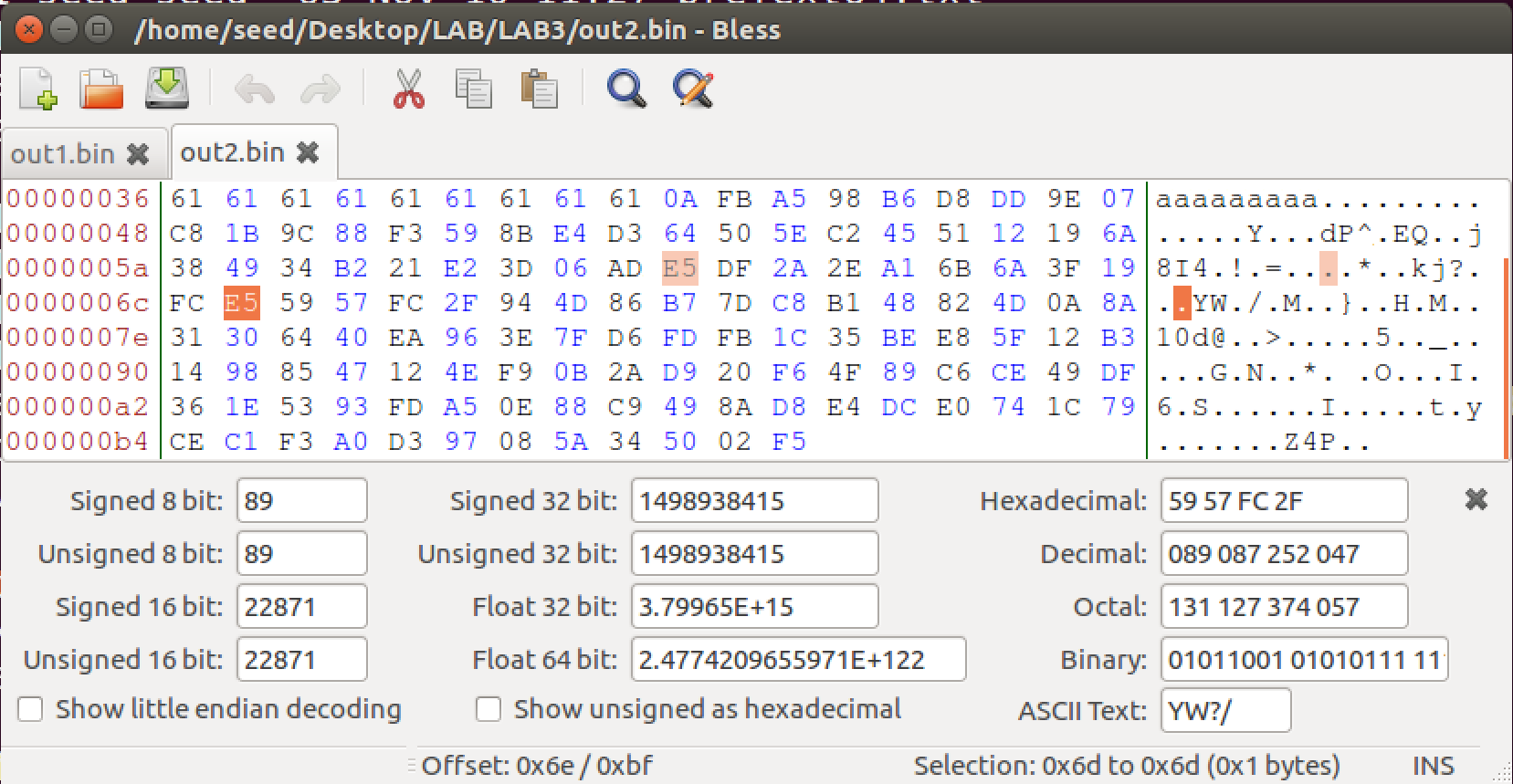












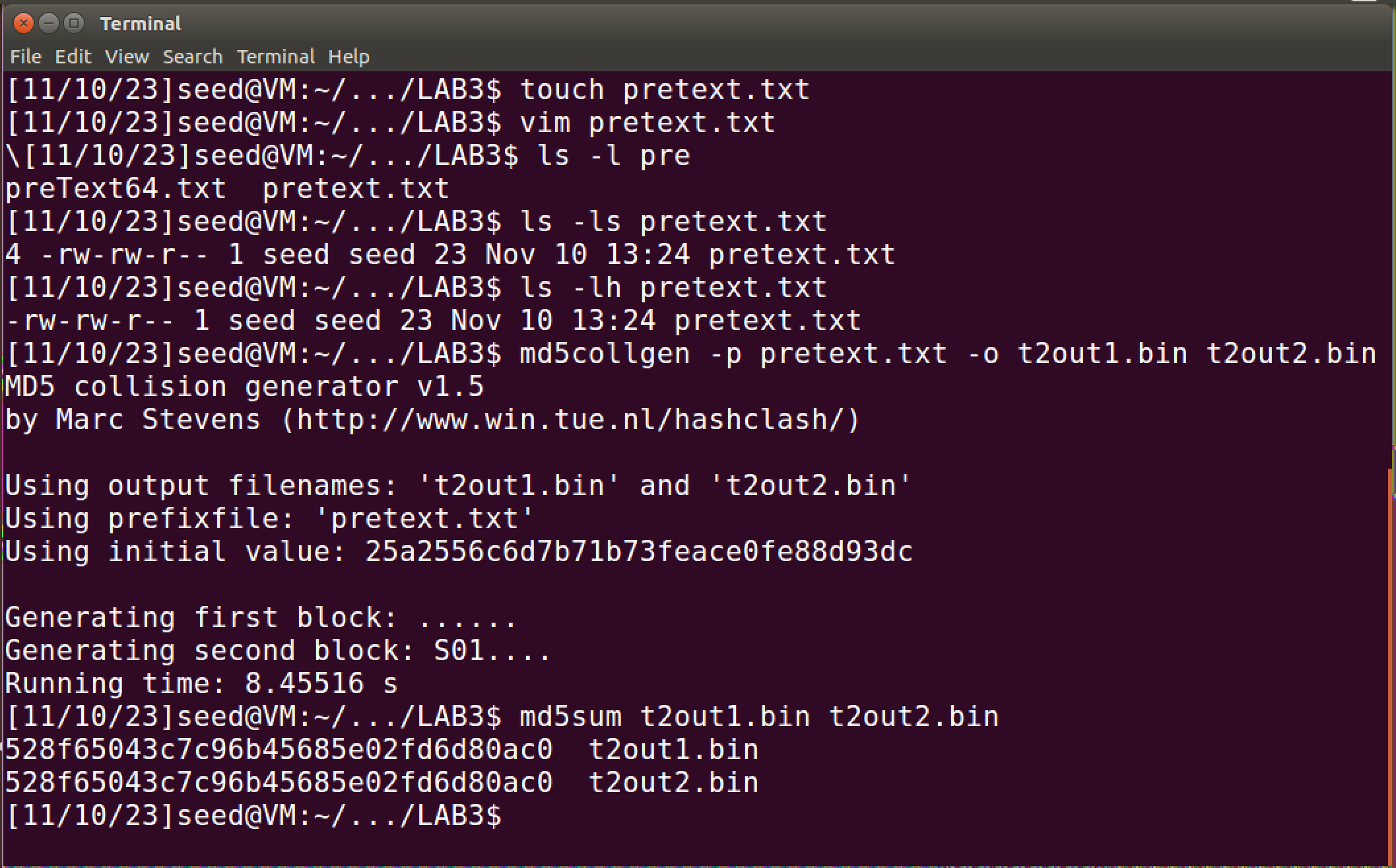
Task II: Understanding MD5 Properties

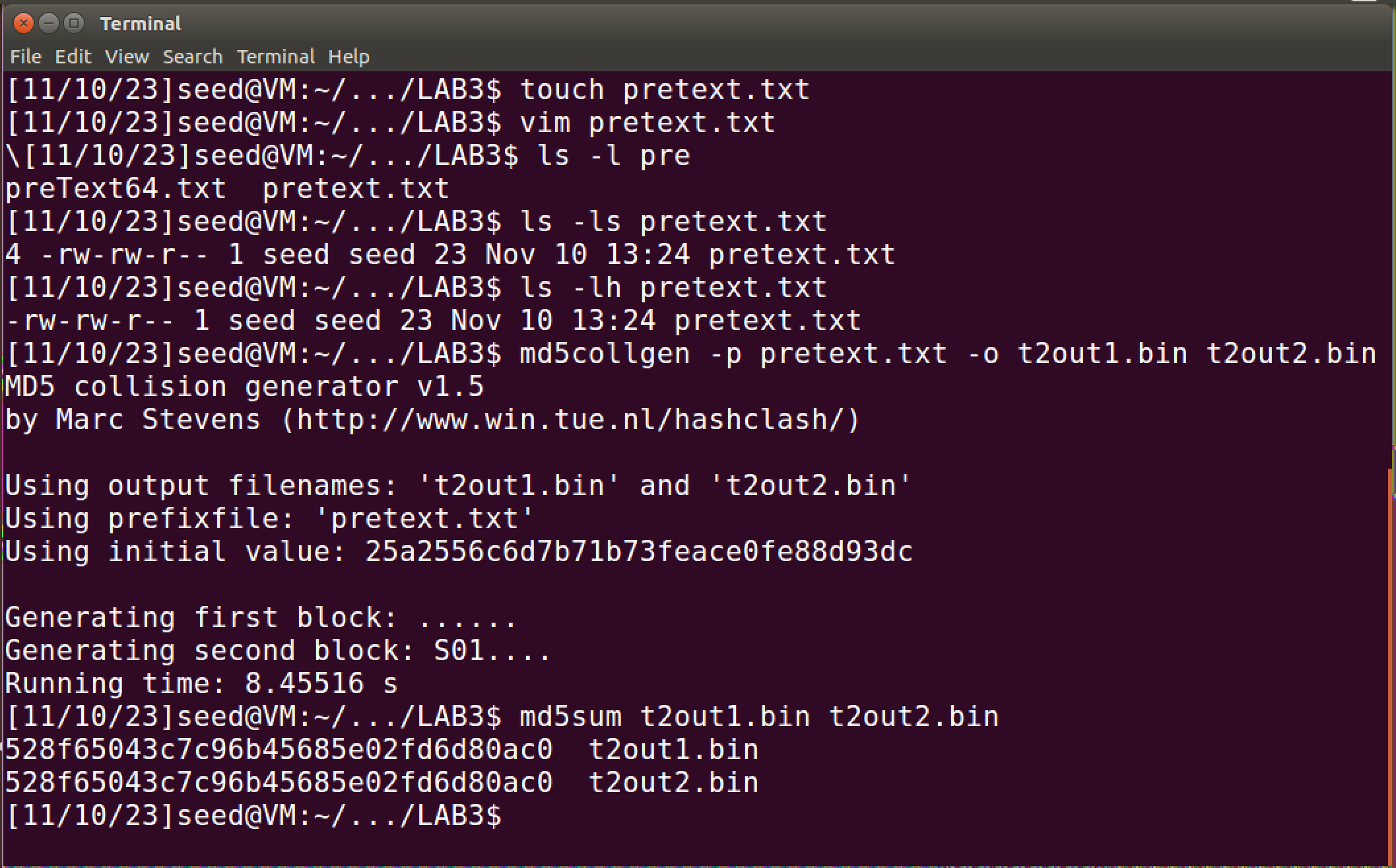
Given two inputs M and N , if MD5(M) = MD5(N) , i.e., the MD5 hashes of M and N are the same, then for any input T , MD5(M || T) = MD5(N || T) , where || represents concatenation.

That is, if inputs M and N have the same hash, adding the same suffix T to them will result in two outputs that have the same hash value.

To demonstrate the above property, I performed this experiment –

Creating a new pretext file “pretext.txt” then creating two md5 file with same hash values. Then using **md5sum** command to check the hash values of the two outputfiles.

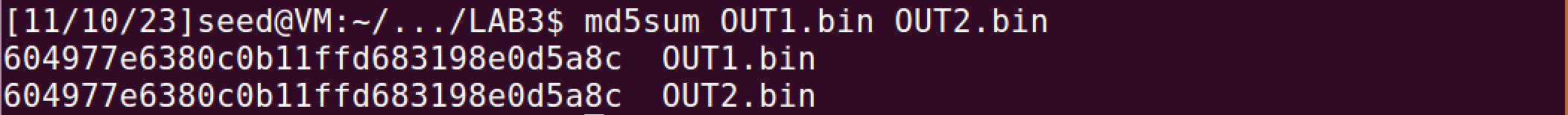




Now I’ll create a new .bin file, namely “newData.bin” with help of **echo** command and concatenate this new file to previous output files using **cat** command.



Now I’ll check the hash values of the modified files

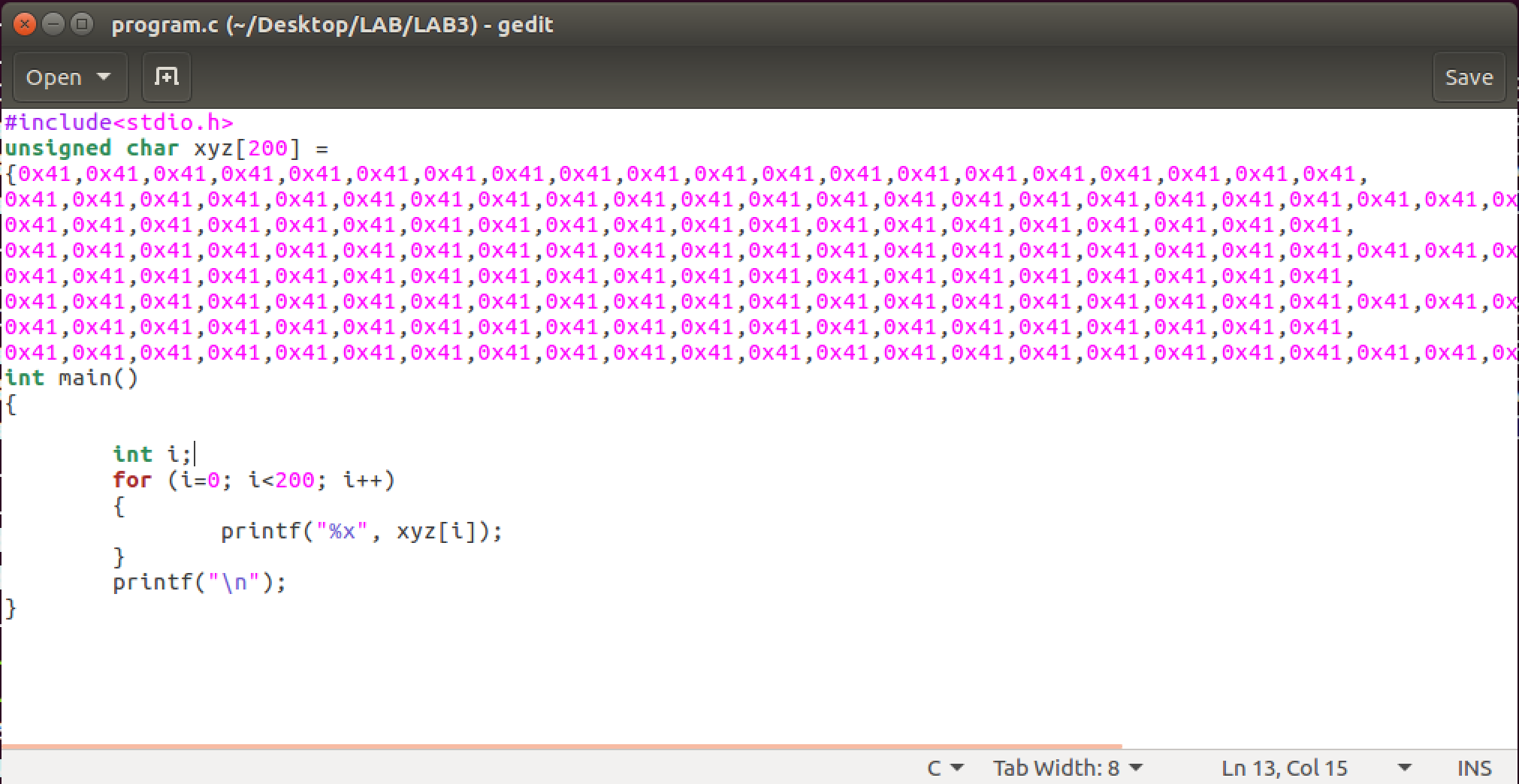


I concur that the there was a slight change in hash value due to the modification but these changes are consistent and can be seen in both the files. In the end the hash vales between two files end up being same.

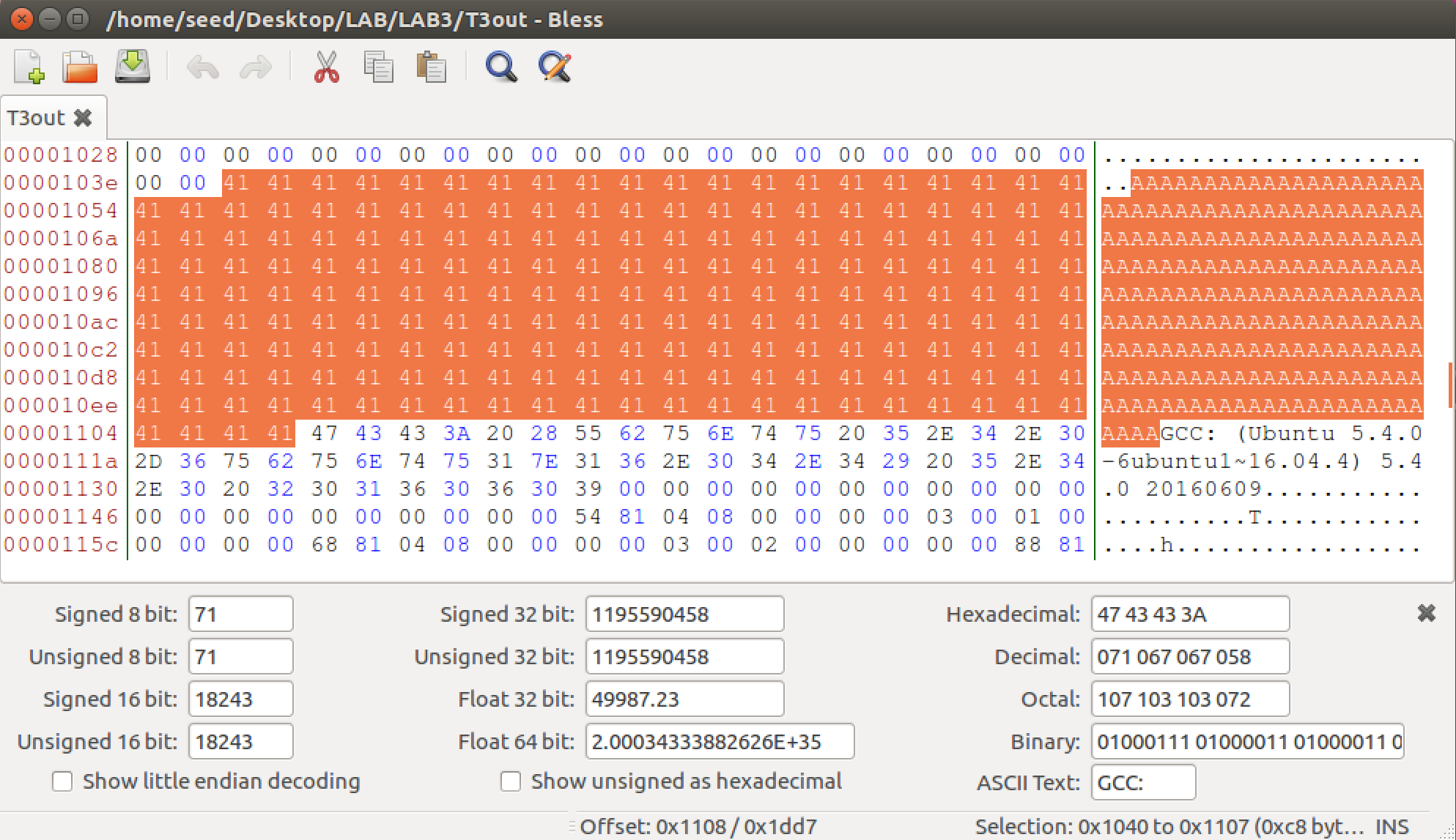
I have successfully demonstrated said property of MD5.

Task III: Generating Two executable Files with the Same MD5 Hash

Here’s the code which I prepared



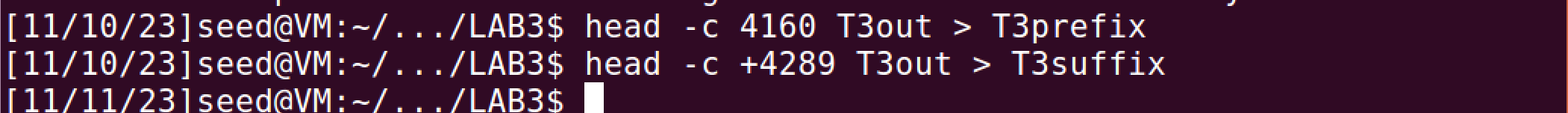




The A,A,A.. starts from 1040 location I the file, which equates to 4160th byte.

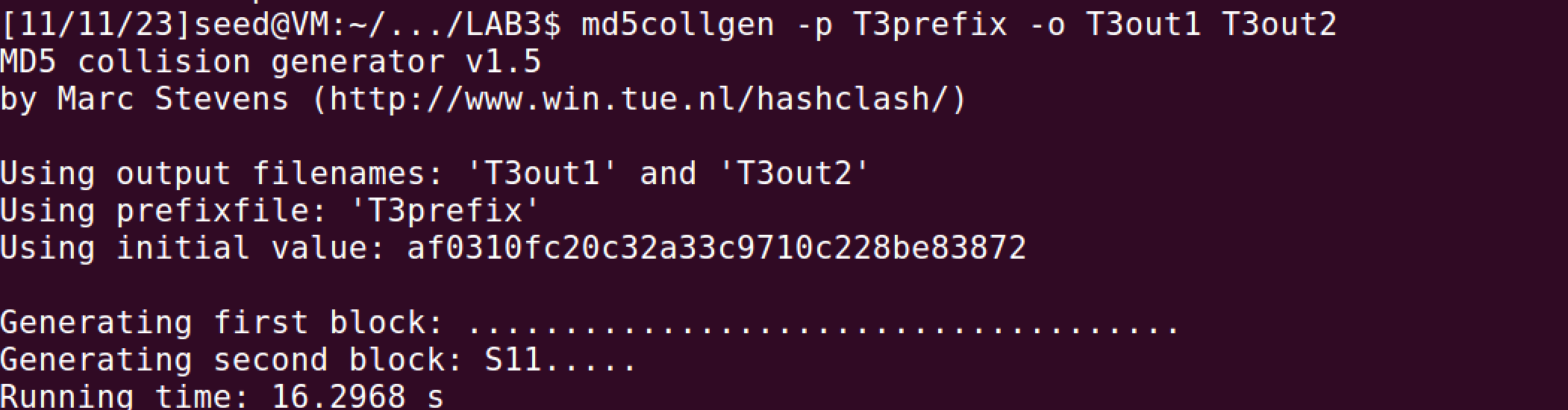
All the bytes before byte will be in prefix text

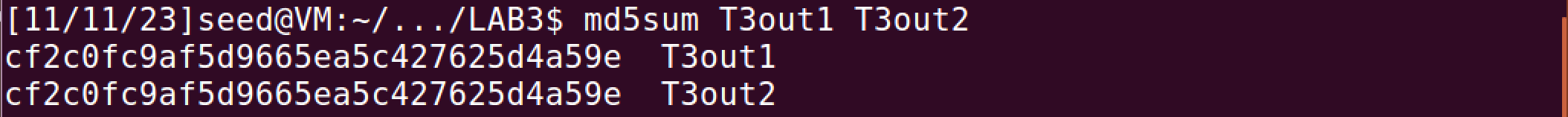
For suffix, so it will contain contents after 4160 + 128 = 4288 byte till end.





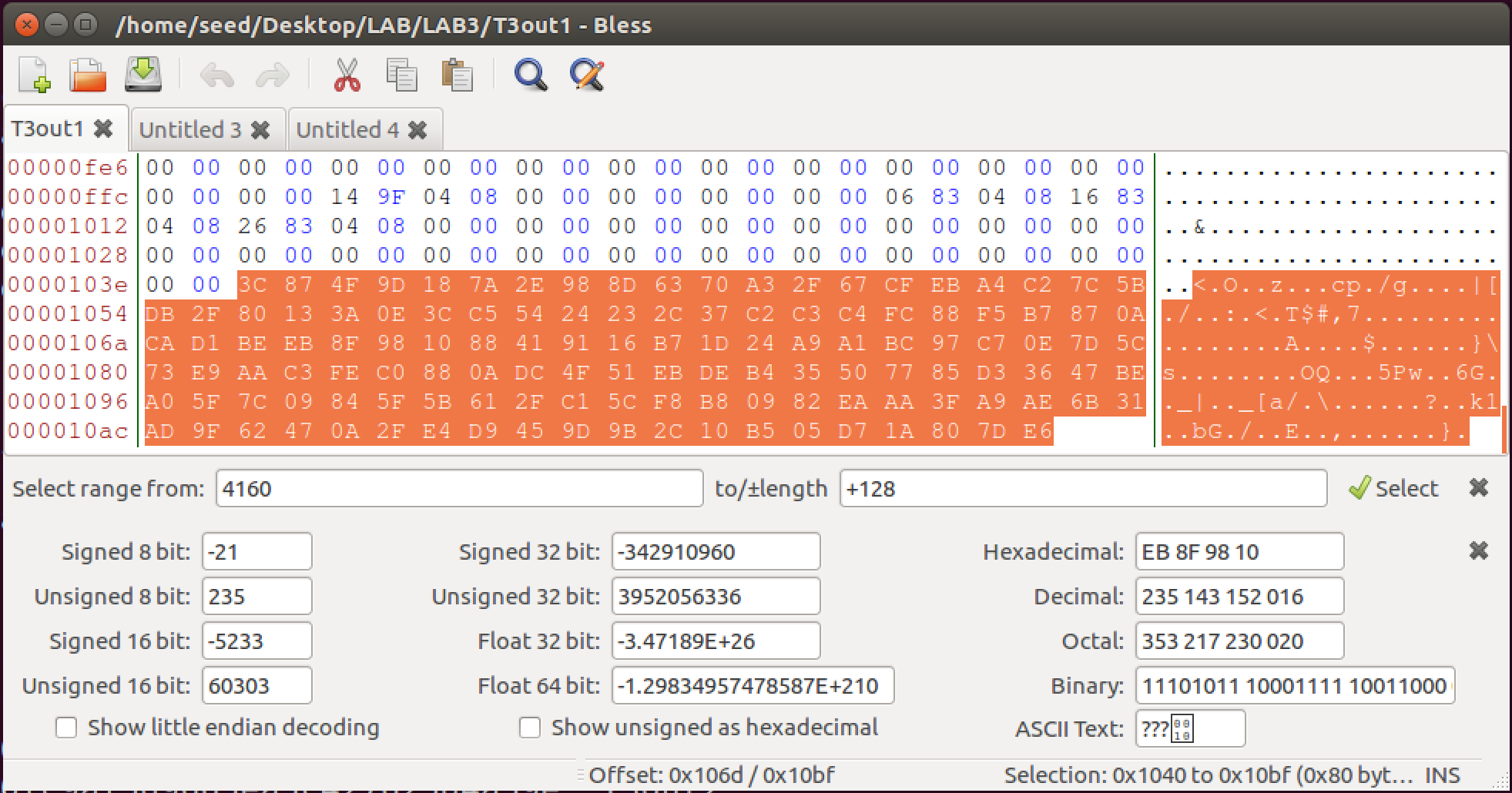
Now I’ll apply md5collgen on this T3prefix file to get two files with same hash values



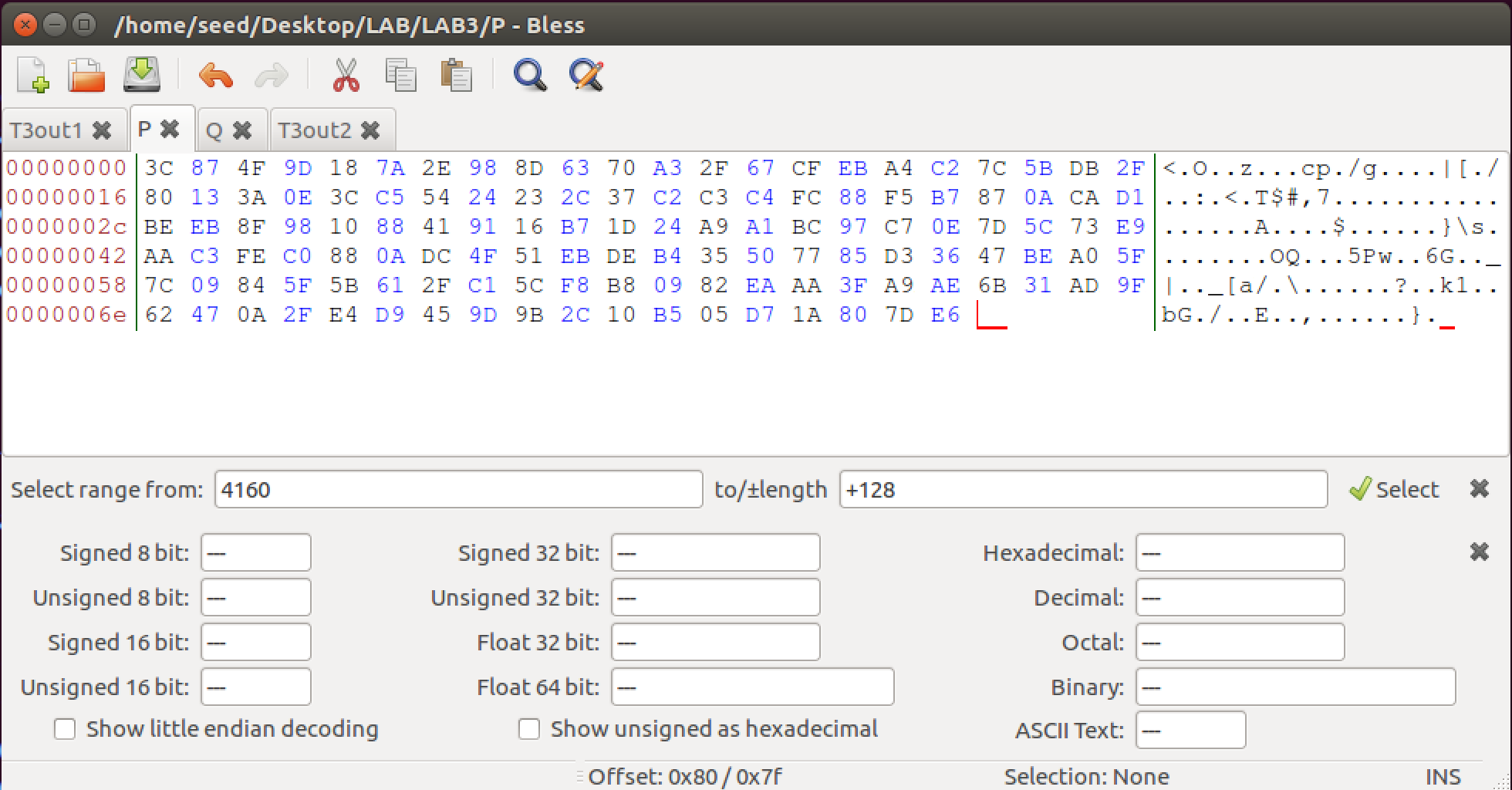


After getting these 2 outputs, the 128 bytes after the prefix’s length will be our P and Q for both outputs.

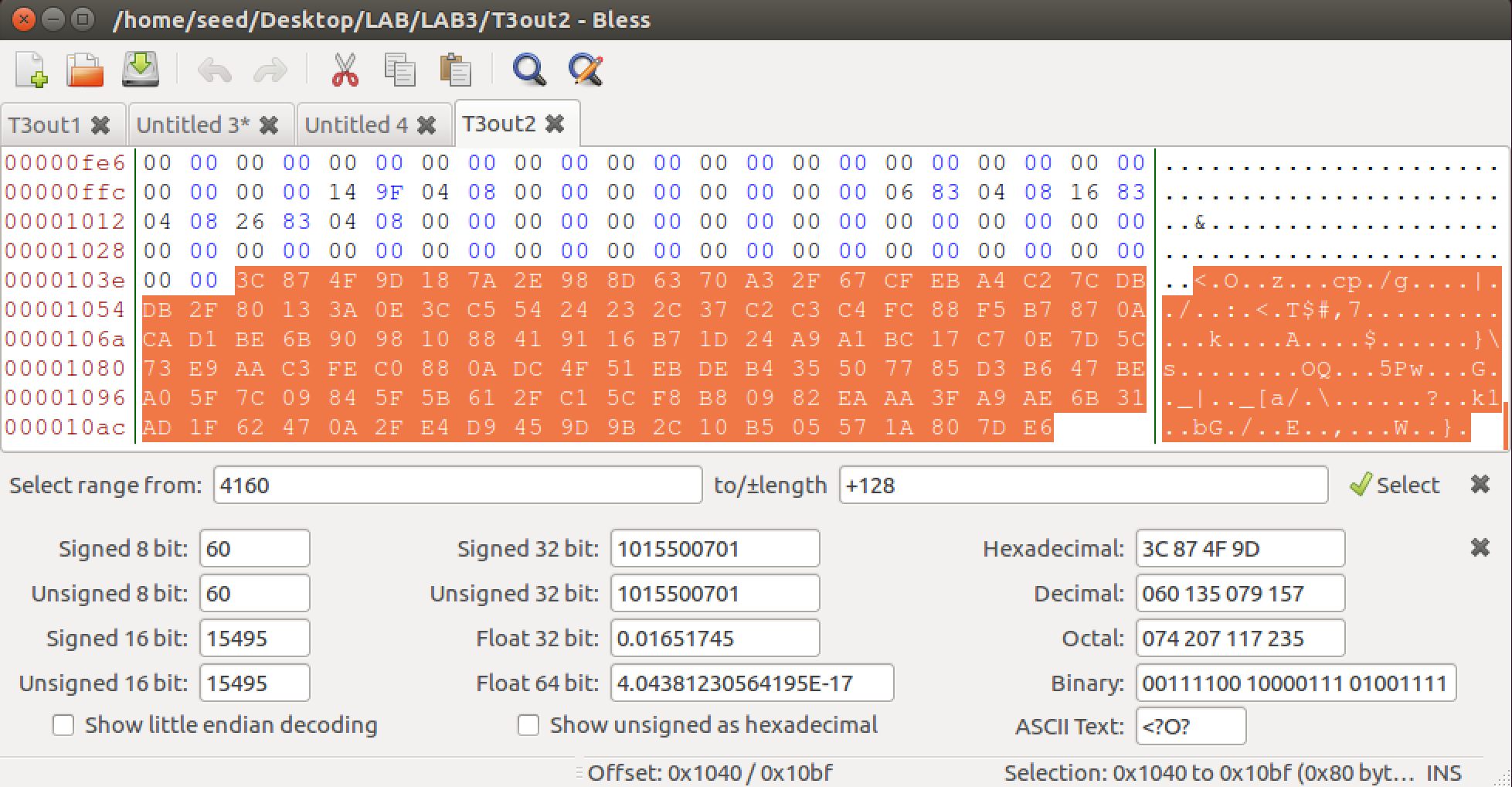
Making P file:



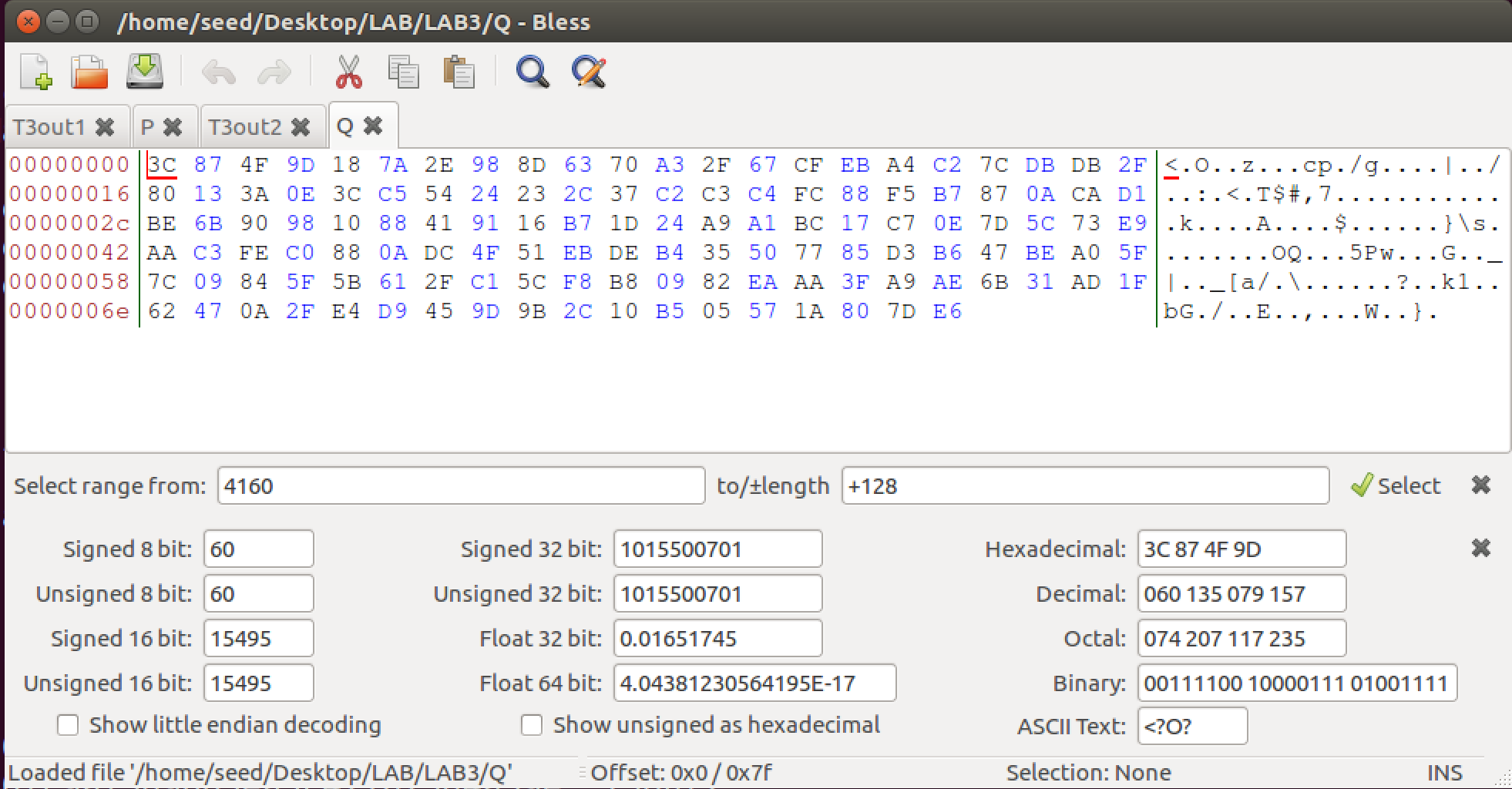
The above bytes are copies from T3out1 and pasted in P



Making Q file:

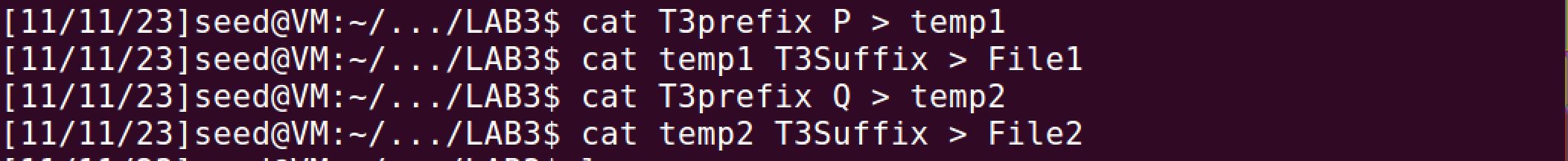


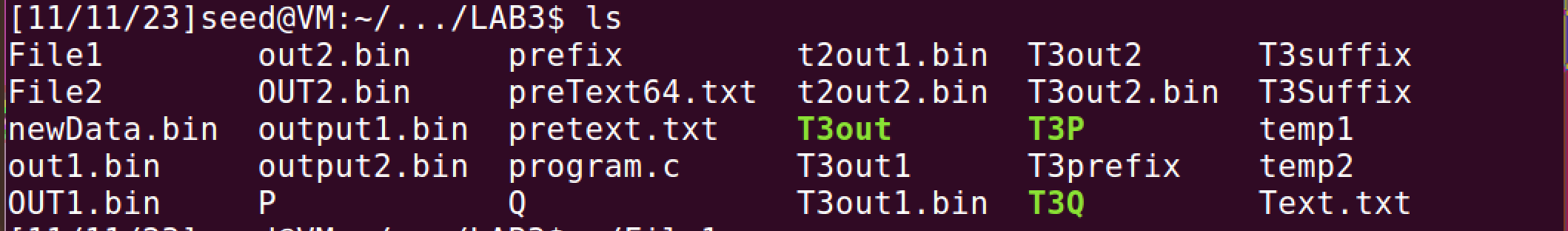
The above bytes are copies from T3out2 and pasted in Q



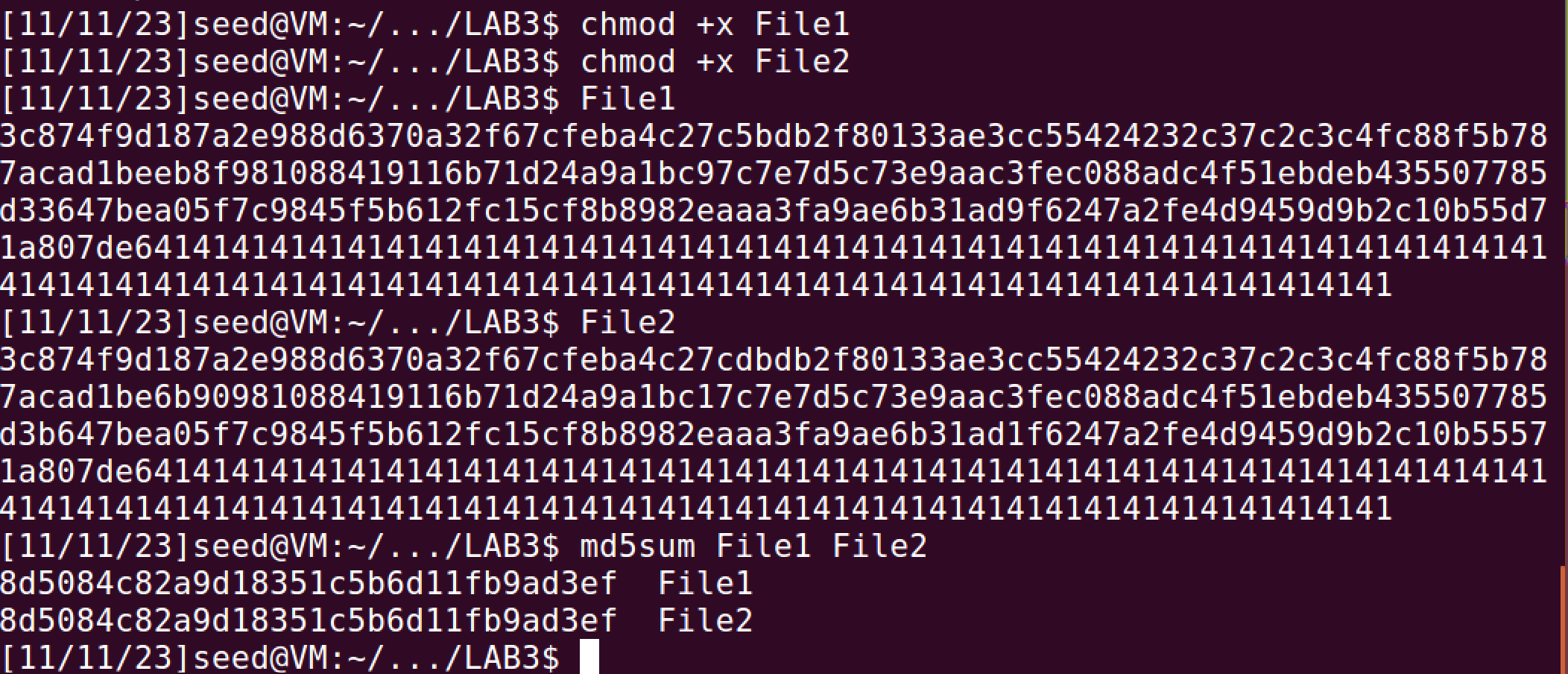
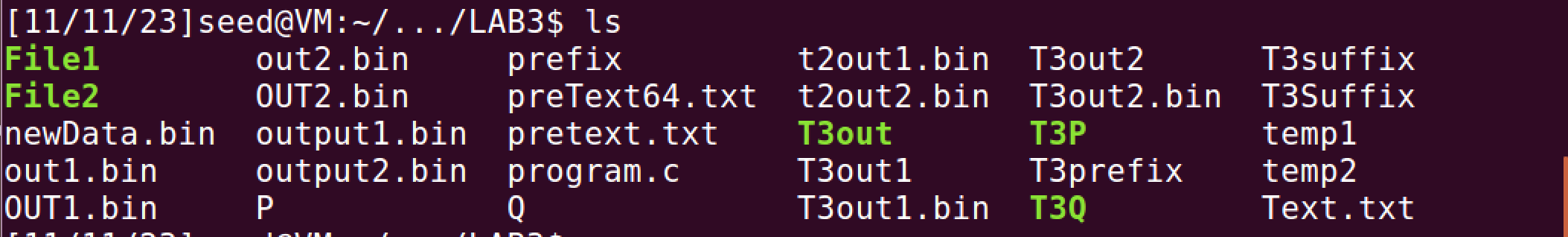
Now I’ll concatenate crate 2 file such as File1 :prefix||P||suffix

and File2: prefix||Q||suffix

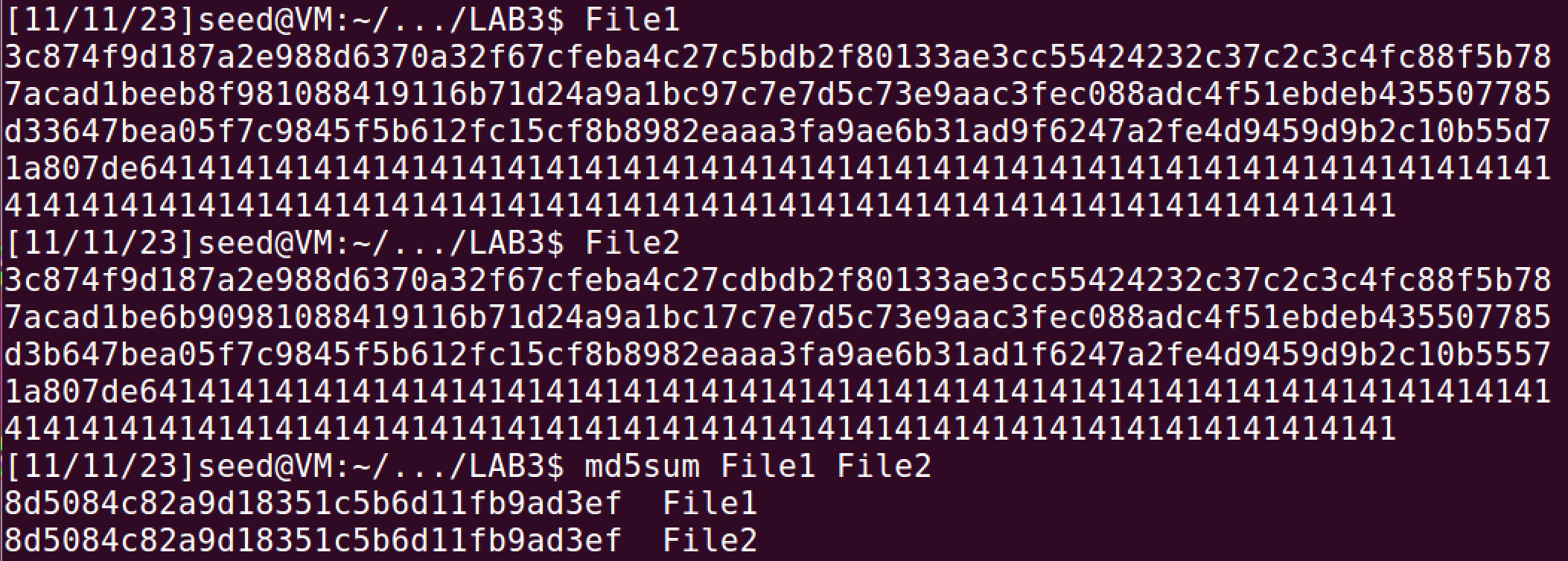




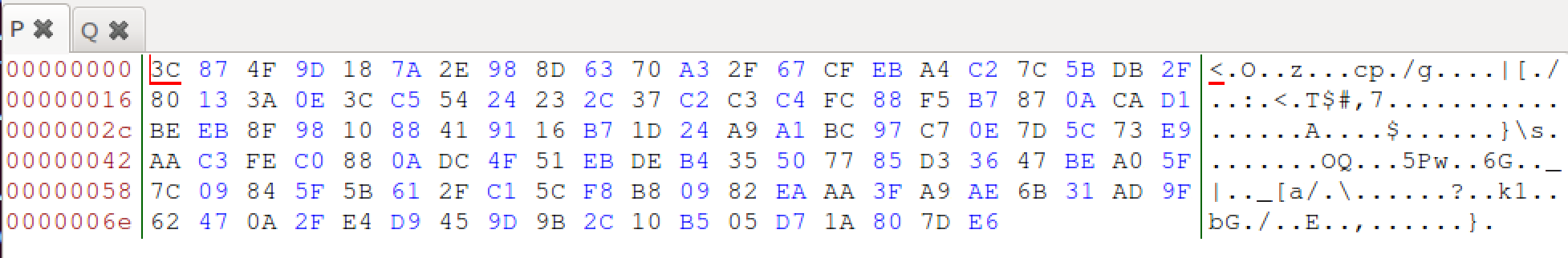
Now I can see these files are not executable, I’ll make these executable by using **chmod +x** command.

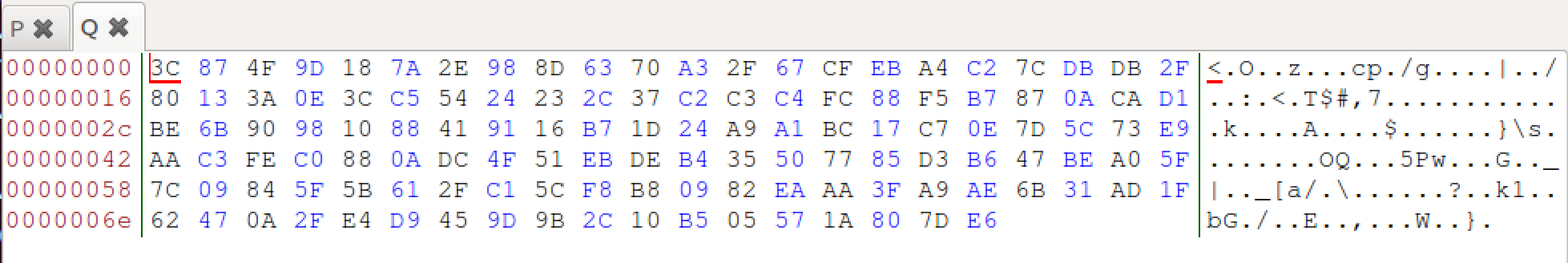
 

I’ll execute these file and comapre there harsh values



I can see both the files have same hash values but different data values.



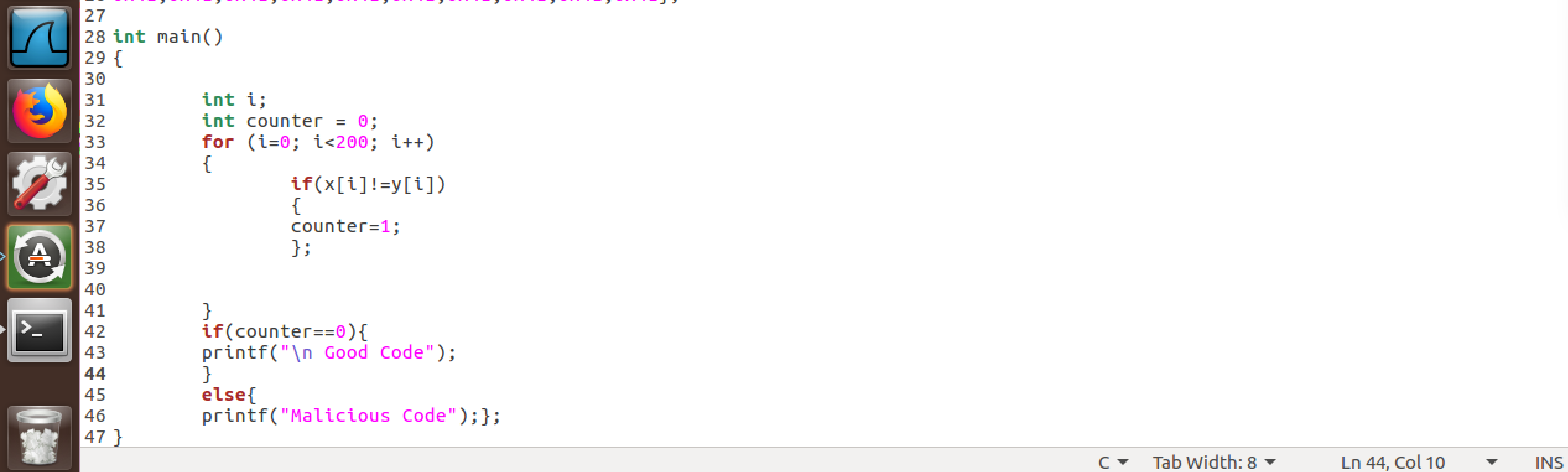


P and Q are derived from T3out1 and T3out2 respectively, which were obtained after performing md5collgen on T3prefix file. Even though T3out1 and T3out2 had same hash value but difference in their data sections. This was also observed in the first task and is consistent to MD5 properties.

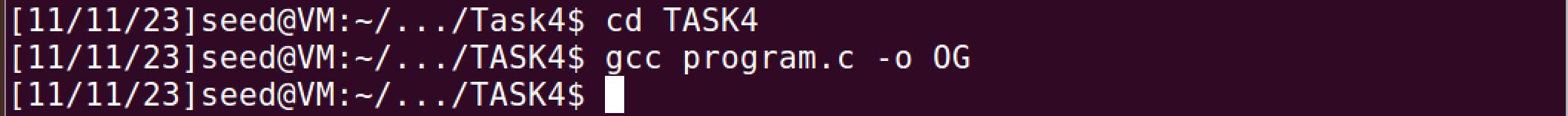
I was successfully able to create two executable files with different data and having the same hash values.

Task IV: Making the Two Programs Behave Differently

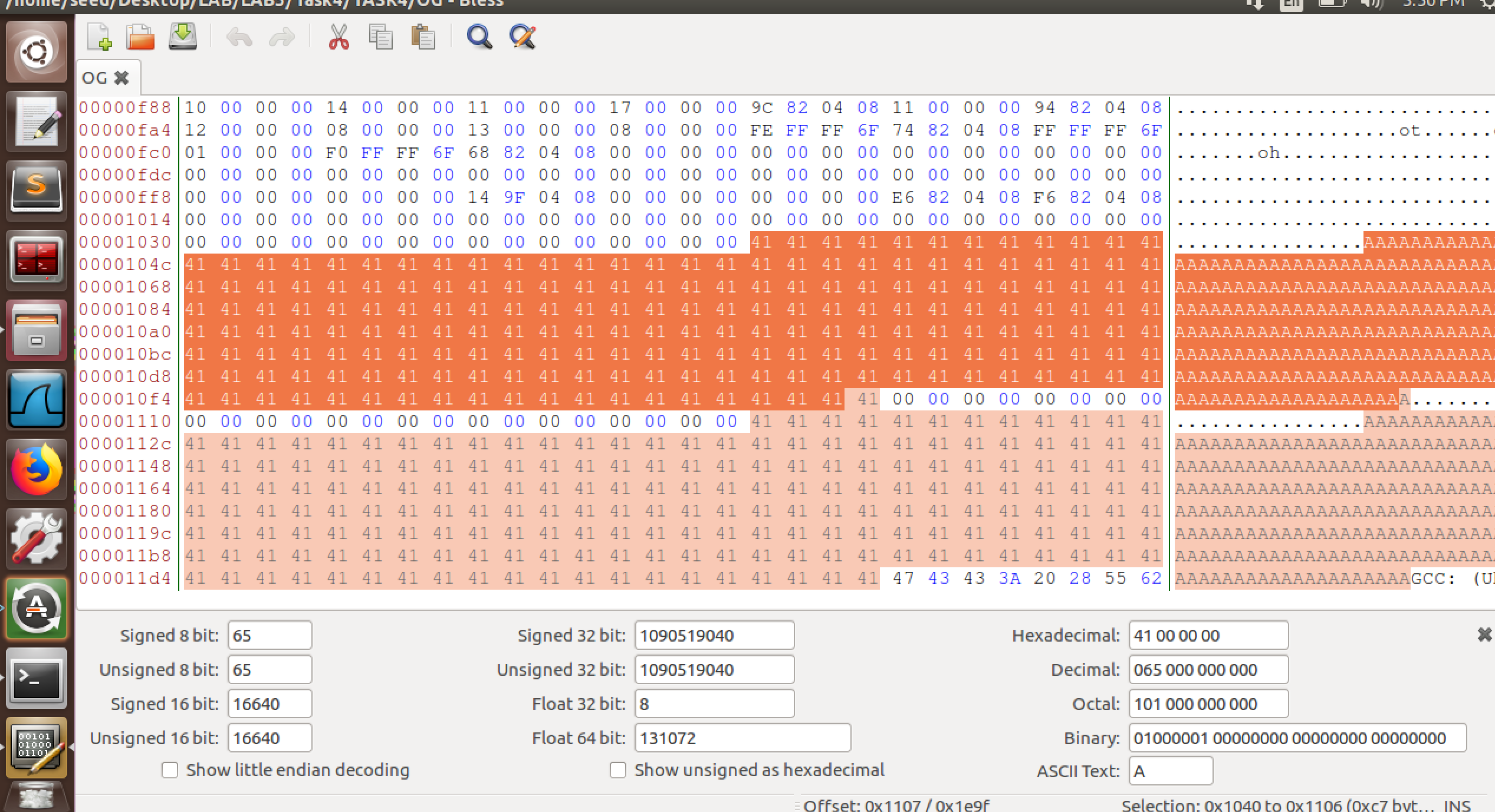
Making the code :

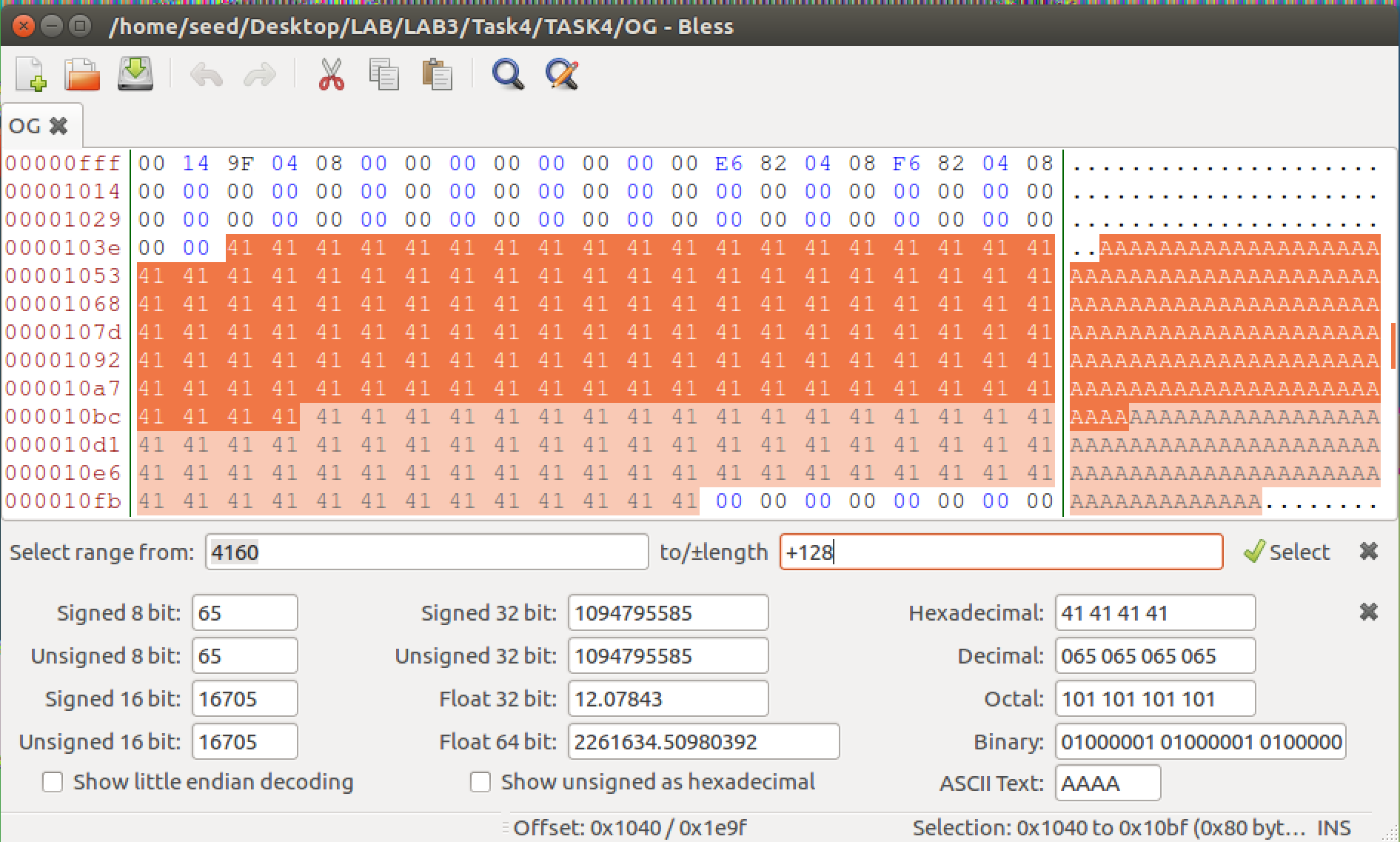
Initial output after running the code and saving it as an executable file



Now viewing this OG file on hex



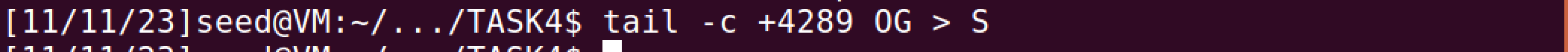
Now I find the location from where the Array X starts



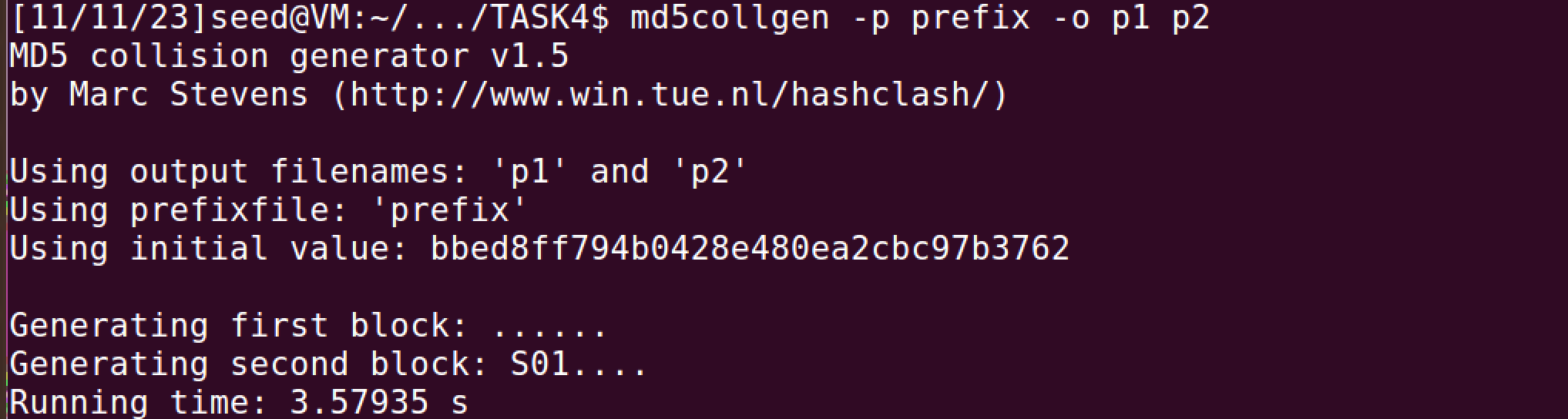
So all the bytes before the highlighted region( Array X) will be in our prefix.



And this highlighted region will the 128-byte part. And the remaining part will be S



Performing MD5 hash on prefix



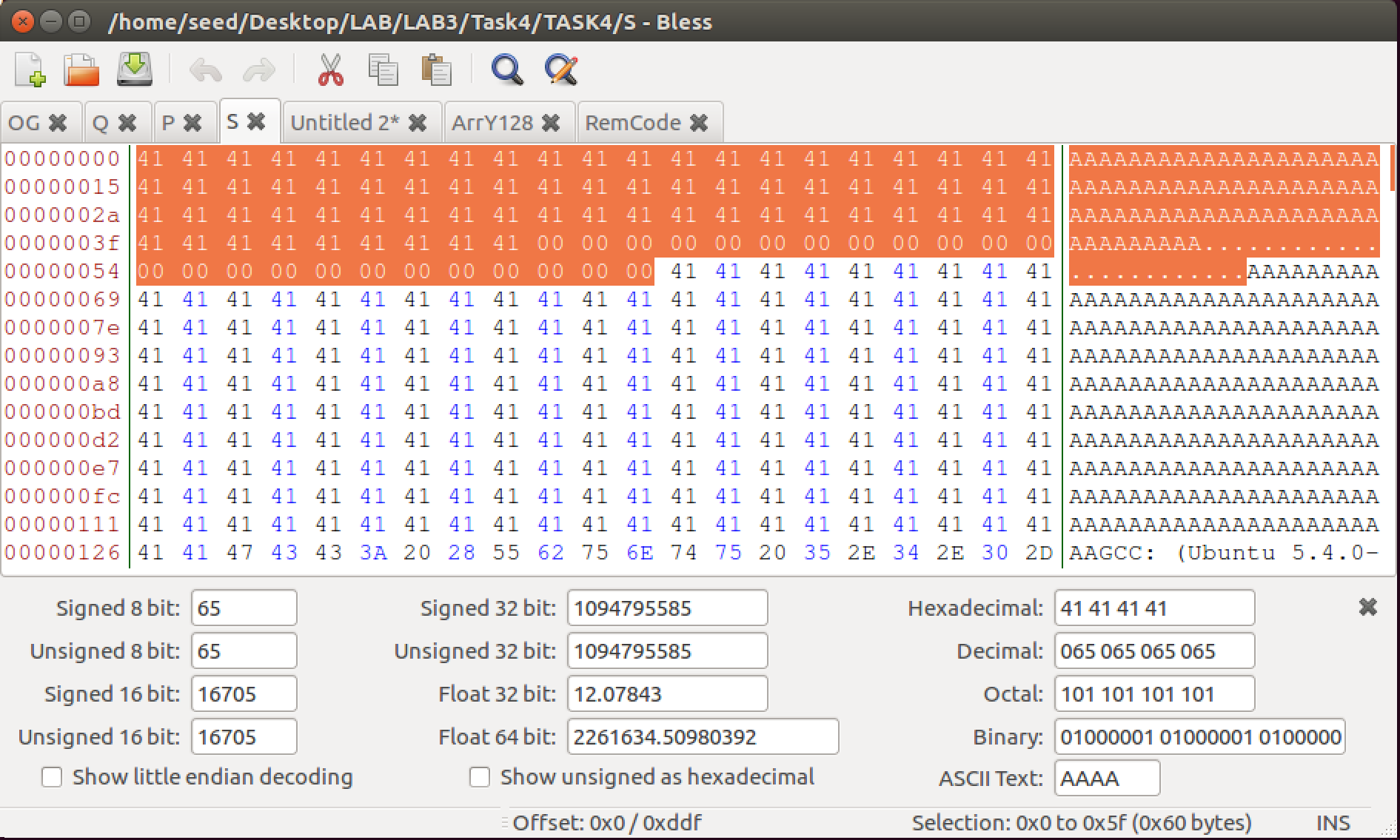
Now I save the last 128 byte from P1 and P2 as P and Q.

These 128 bytes consist of first 128 byte of array X.

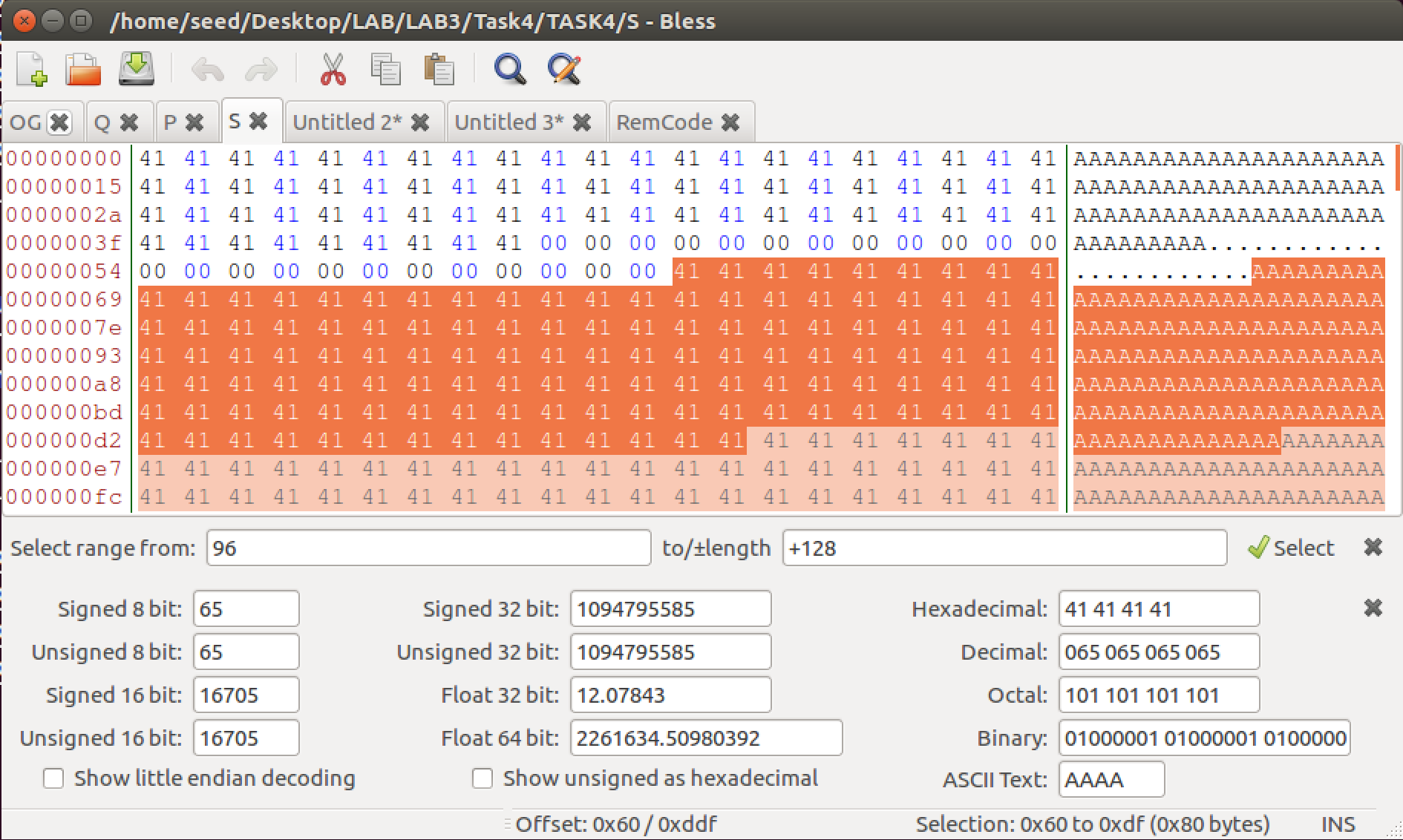
Now I’ll divide the suffix into the following parts

Remaining arrX | first 128 bytes of arry | reaming code

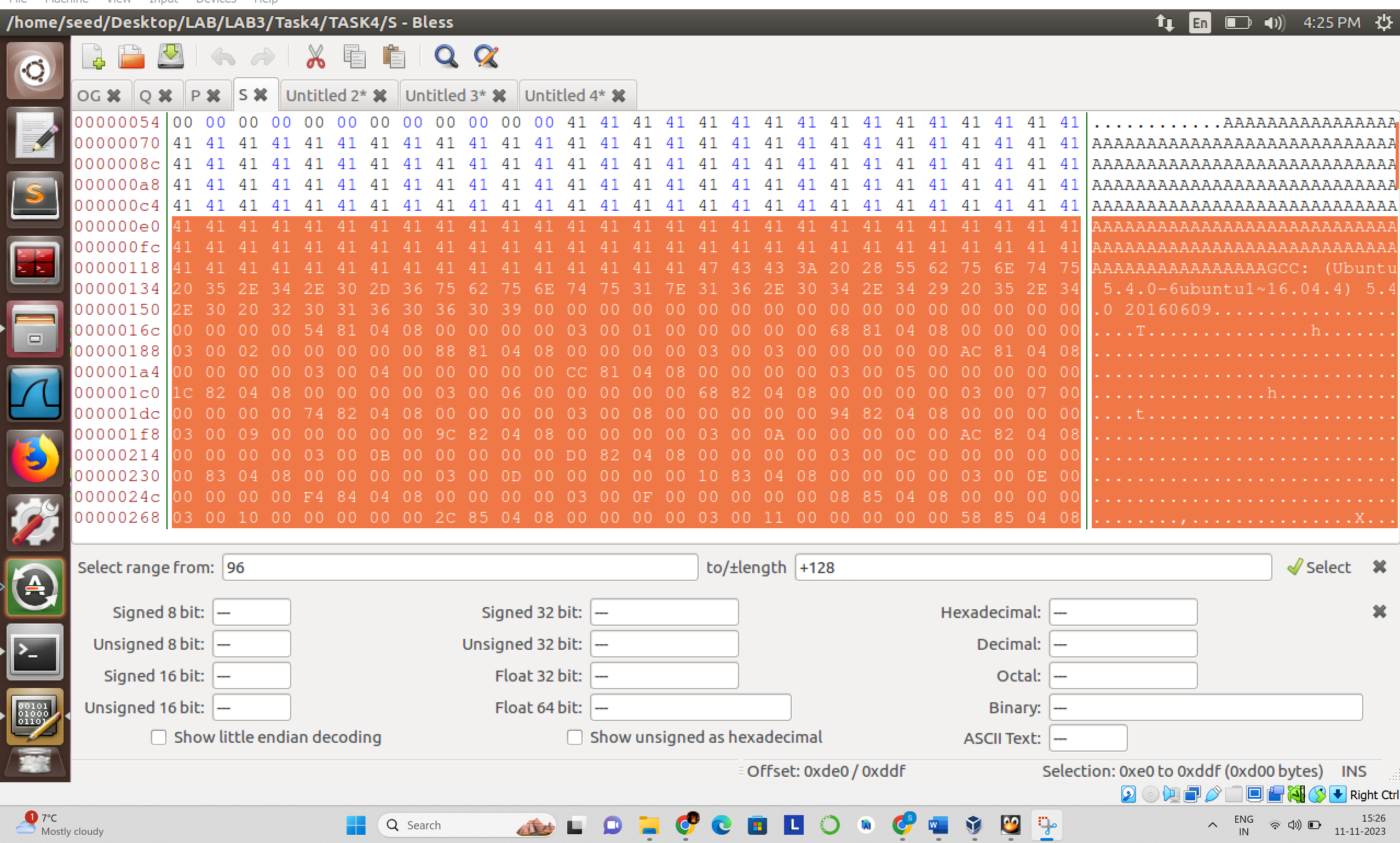
Remaining part of arr X

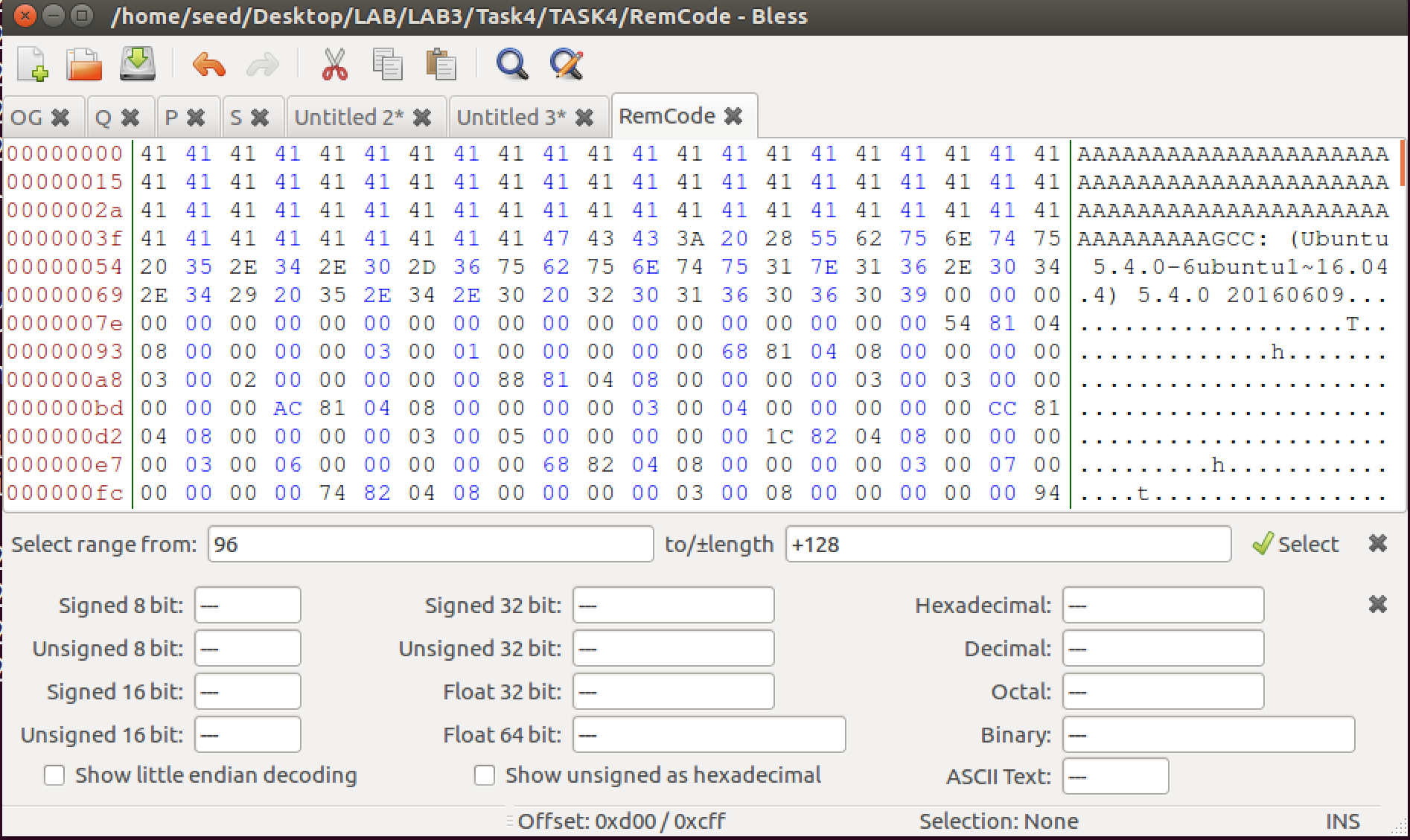


First 128 bytes of array Y



Remaining code part:





Now the Good code will consist of:

Prefix || P || Arr x Remaining || P || Remaining Code

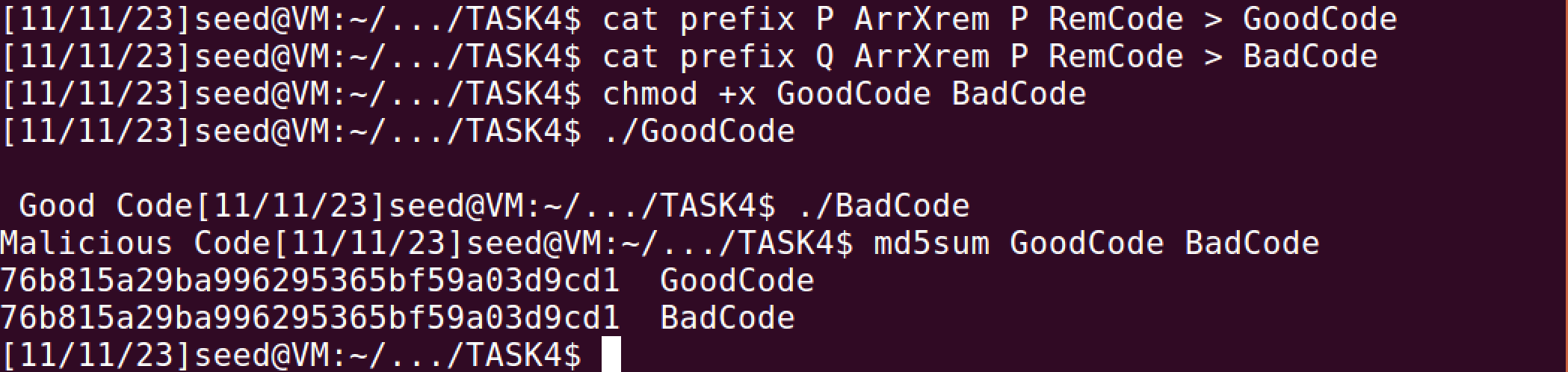
Bad Code:

Prefix || Q || Arr x Remaining ||

P || Remaining Code

After creating the Good Code and Bad Code file, I’ll make them executable

After doing so, I’ll execute them and check their hash values



As I can see, Both Good and Bad codes have the same hash values but execute different codes.

I was successfully able to create two programs with same hash values while they perform different functions. The good code performs the intended/ benign code where as the Bad code performs malicious code.

Conclusion:

After performing the various experiments on MD5 hashing function, I can concur that MD5 hash is not effective at handling collisions. Because of this MD5 is considered insecure for cryptographic purposes. Even though MD5 is a fast-hashing function but due to its vulnerability to attacks it is no longer acceptable for digital certificates.

SHA-256 and SHA3 are considered to be its modern counterparts as they are more secure