**SEED LAB: MD5 Collision Attack**

Contents

[Task 1 3](#_Toc116585479)

[Question 1 6](#_Toc116585480)

[Question 2 9](#_Toc116585481)

[Question 3 11](#_Toc116585482)

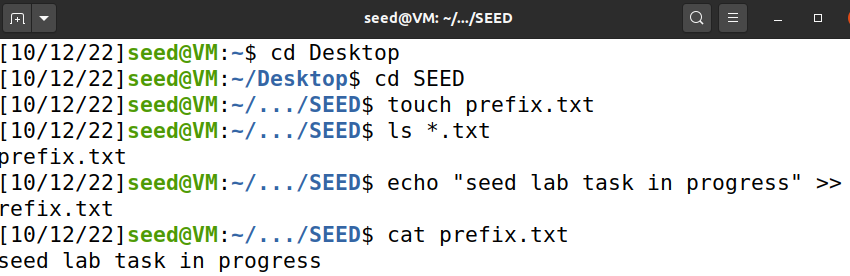
[Task 2 11](#_Toc116585483)

[Task 3 13](#_Toc116585484)

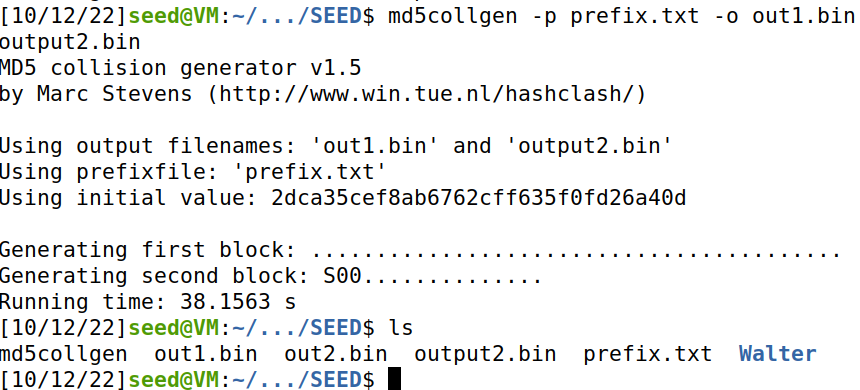
[Task 4 18](#_Toc116585485)

# Task 1

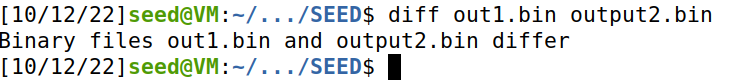
* I created a file named prefix in the folder where I placed md5collgen to generate two output files with 2 different hashes. Moreover, I have added string “seed lab task in progress” in the *prefix.txt* file to begin.

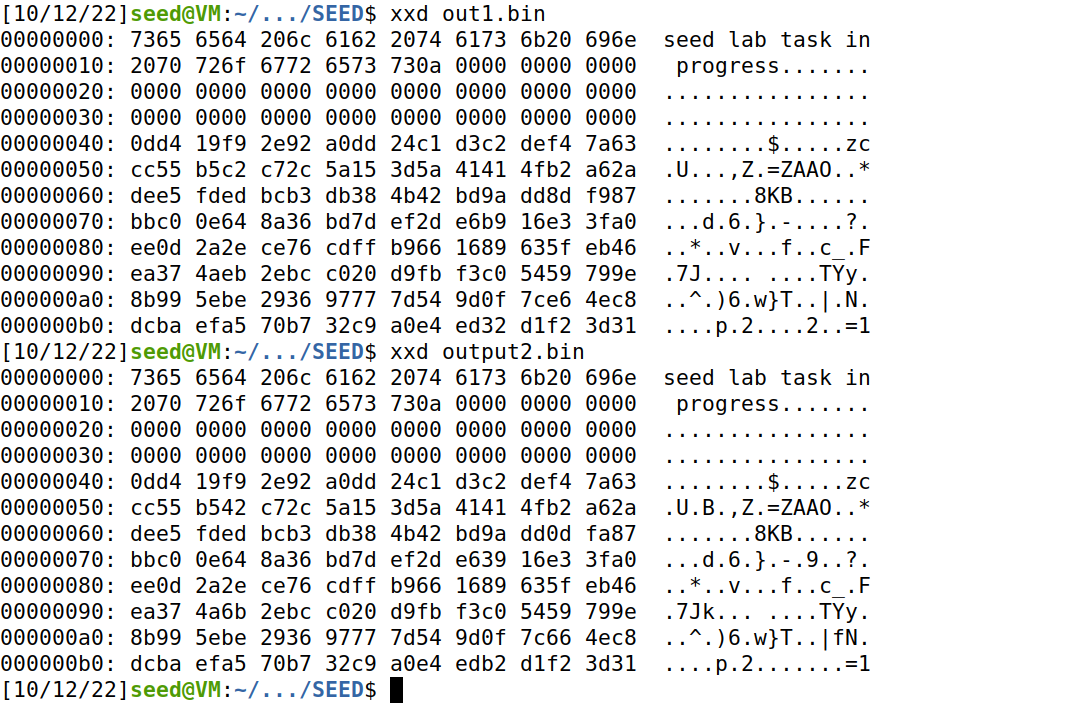


* I used md5collgen to generate two different outputs.

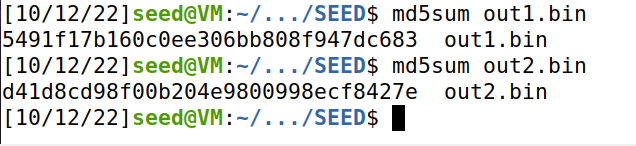


* Here, I used the command given in the manual to check if the binaries of the output files differ and yes they do.

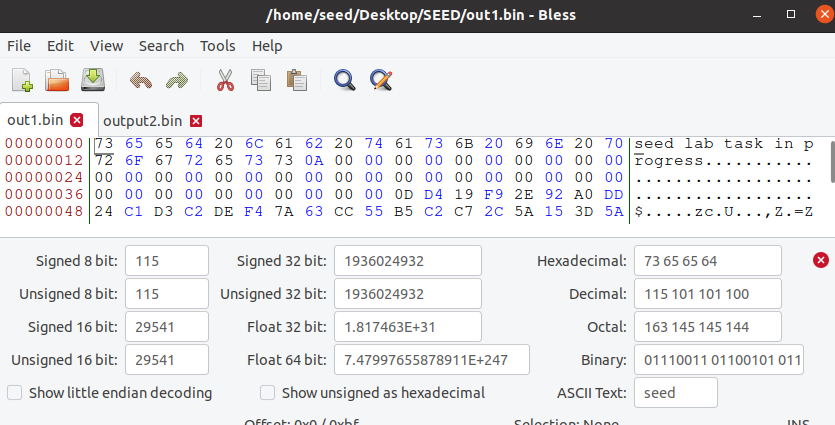


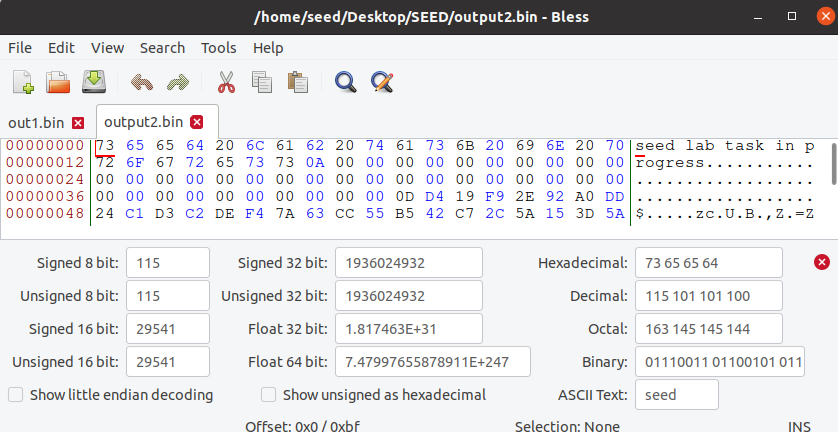


* Checking the Hash values of both output files to confirm the difference in files to view the output files which they evidently differ as shown in the screenshot below.



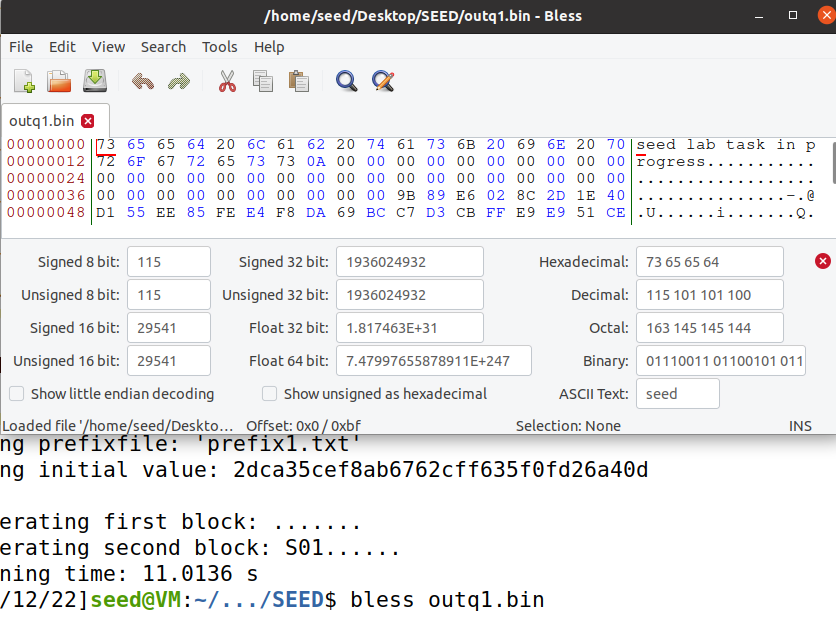
* Now I opened both the files *out1.bin* and *ouput2.bin* to check their hex values to verify the difference.

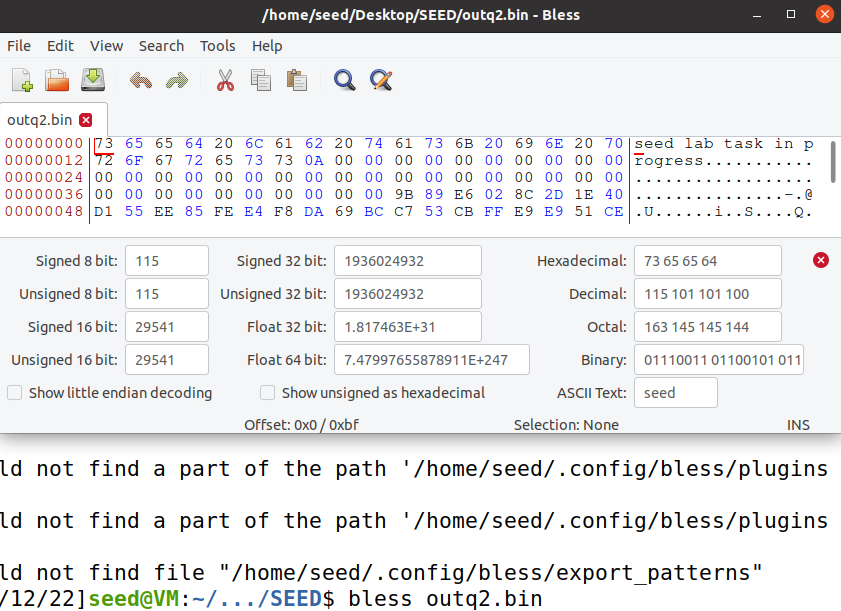




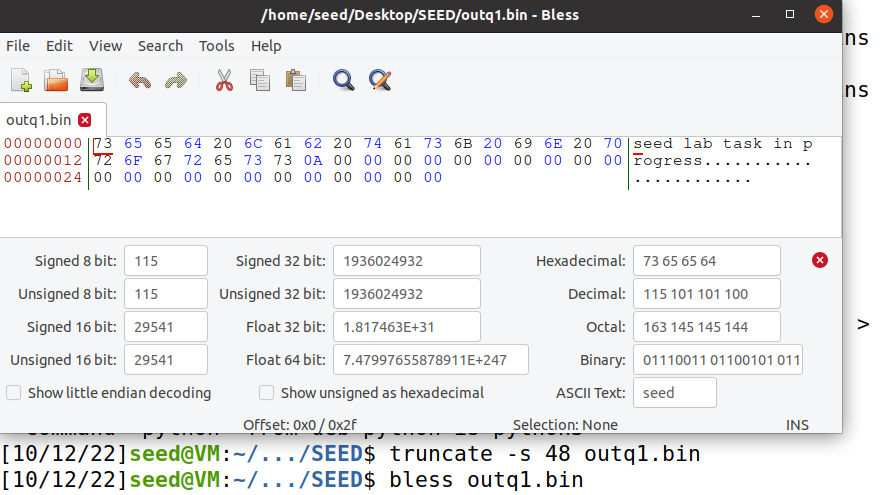
## Question 1

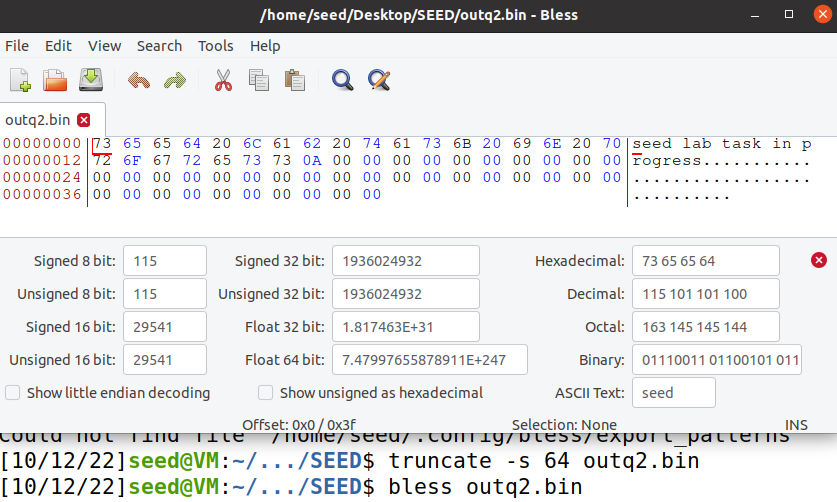
* I created a new file with string “seed lab task in progress” named as prefix1.txt. I used md5collgen to generate different output files in order to answer the questions in task1.





* Below are the outputs which have been truncated to 48 bits and 64 bits of the files named *outq1.bin* and *outq2.bin*, respectively.

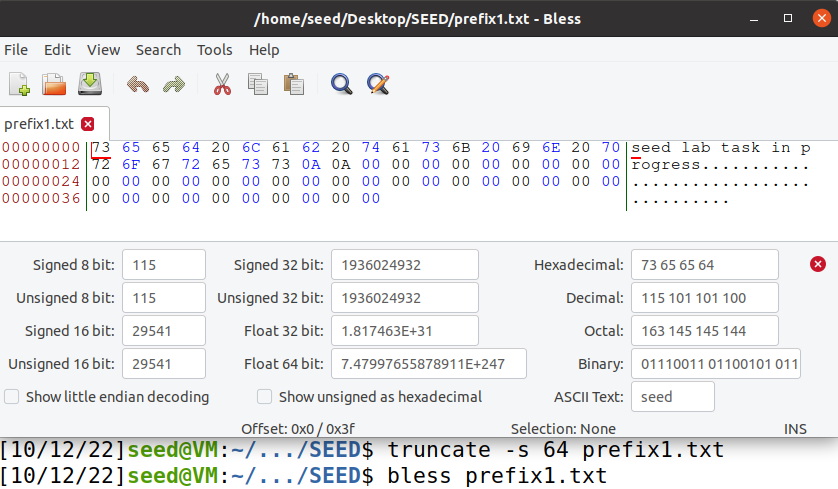




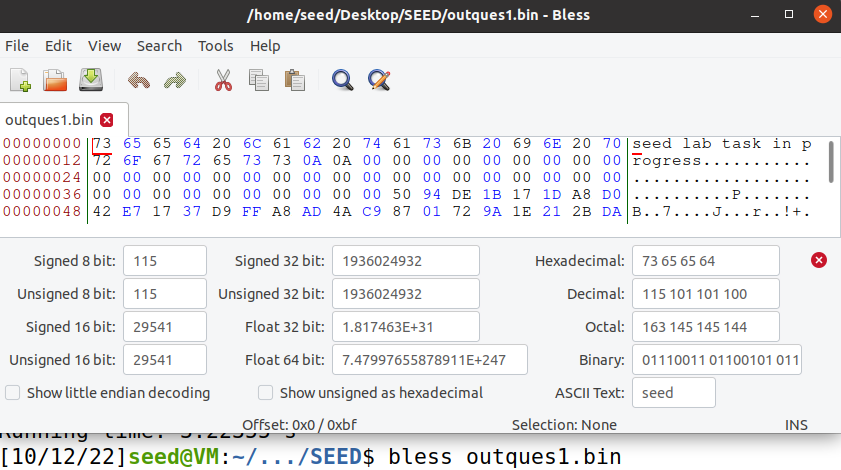
As visible from the differences and truncating the files above, we can notice that to the length will be padded to the provided bits used to truncate.

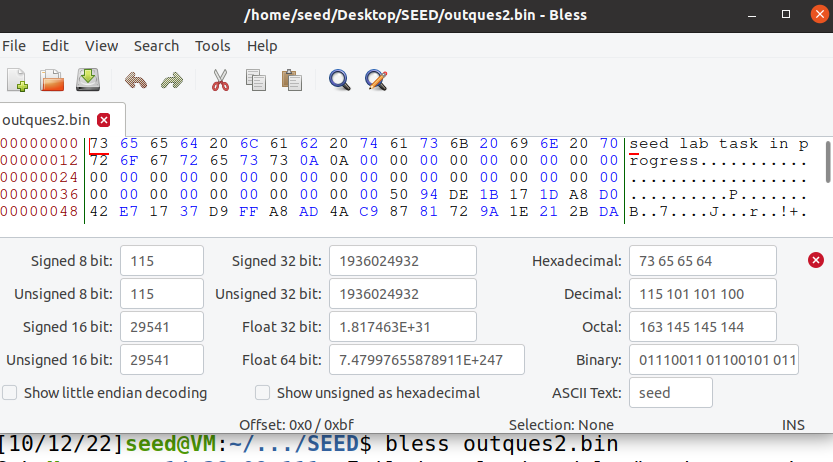
## Question 2

* Here, I have truncated *prefix1.txt* with 64 bits as size value.



* I used md5collgen to generate two output files named *outques1.bin* and *outques2.bin*. In order to observe if zero padding still exists.

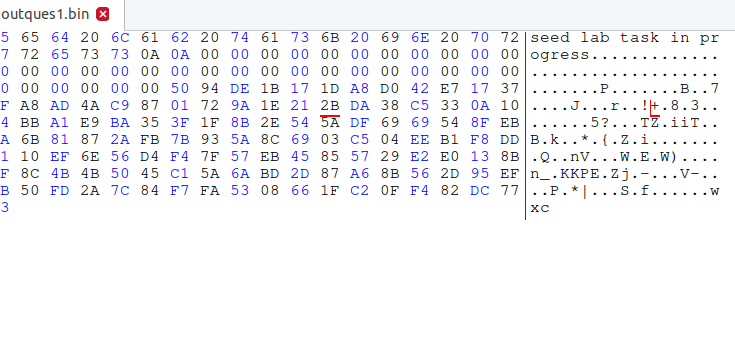


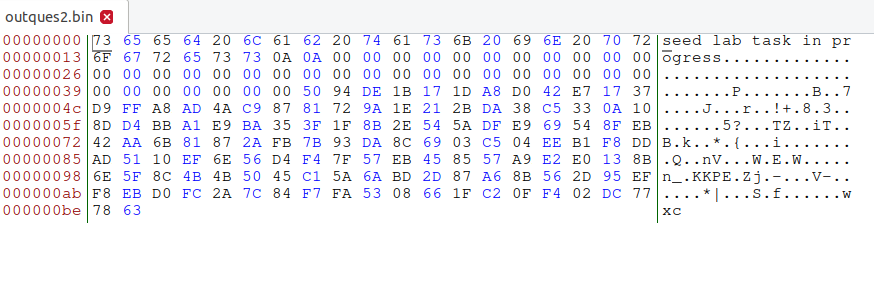


As evident from above screenshots that there is no padding effect.

## Question 3

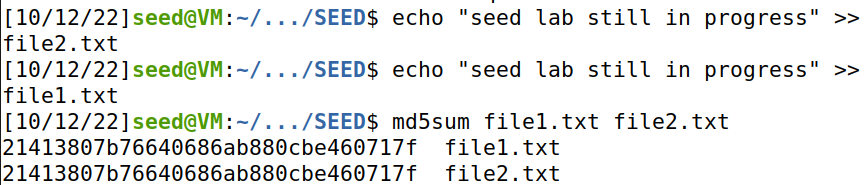
Not all bytes are different because the bytes only differ at the certain positions but these positions are not constant. Which is evident from the hex editor bless used on both output files.



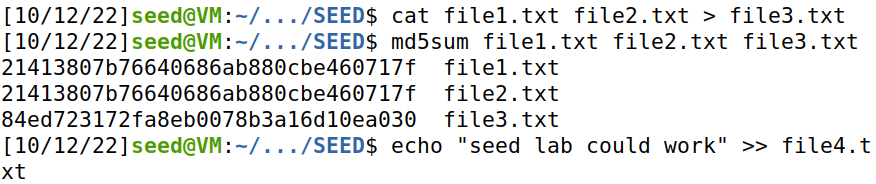


# Task 2

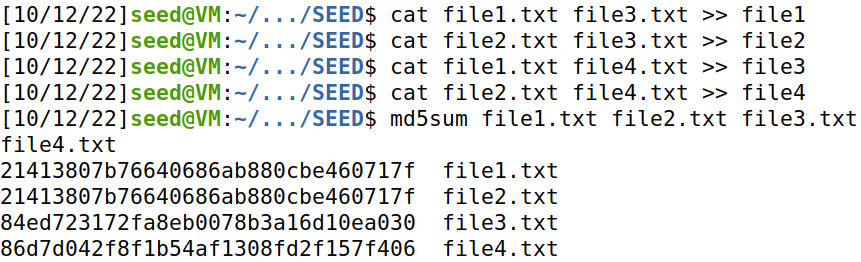
* I created two files named file1.txt and file2.txt and inserted the string “seed lab still in progress” in both of them. Followed by md5sum to compare hash values.



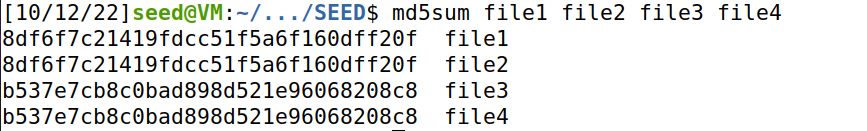
* Now I concatenated file1.txt and file2.txt and stored the output in file3.txt while comparing md5sum of files. Moreover, I created a new fie named file4.txt with “seed lab could work” string stored in it.



* Now I concatenated the files as shown below and compared the md5sum to see the md5sum values of files used as input in concatenation process.



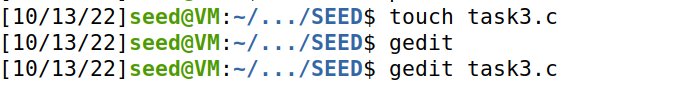
* Finally I checked the md5sum of output files which obviously shows that for different inputs while adding a suffix (which *is file3.txt* for *file1* and *file2*, and *file4.txt* for *file3* and *file4*) will lead to same md5sum as does for *file1* and *file2* with each other, and *file3* and *file4* for each other.



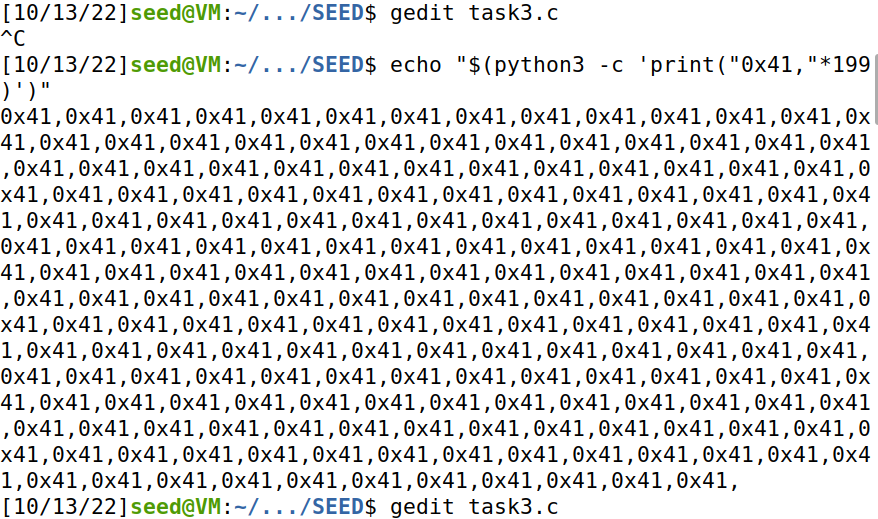
Hence, the property is proven to be right.

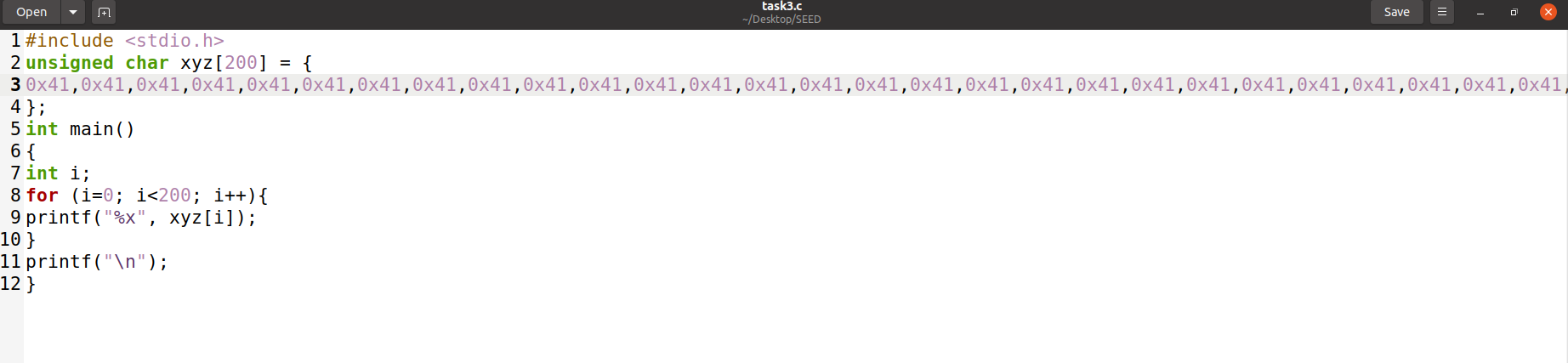
# Task 3

* I created a *program.c* file where I stored the provided code in the manual to perform the task.

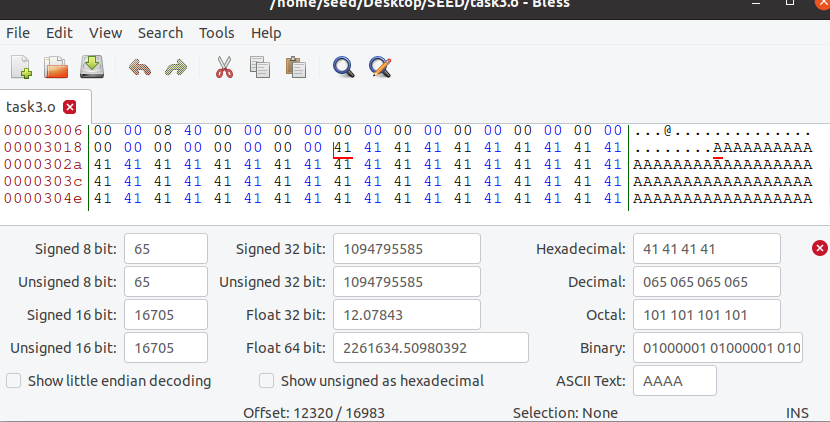


* Moreover, I added the Array Values as provided in the manual by using echo and pasting the values in the *task3.c* file.

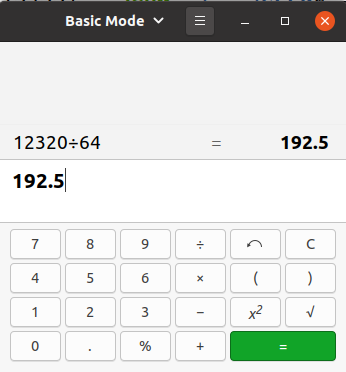


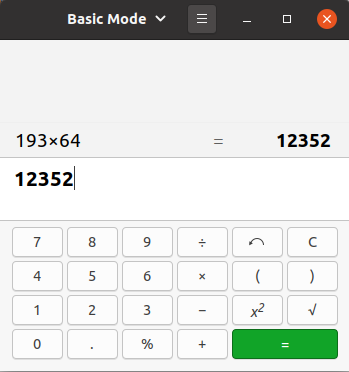


* Here, I compiled the program and used the output file to see the offset using bless hex editor from where the array values starts. Which seen in decimal is 12320 bytes away from the start of the binary file.

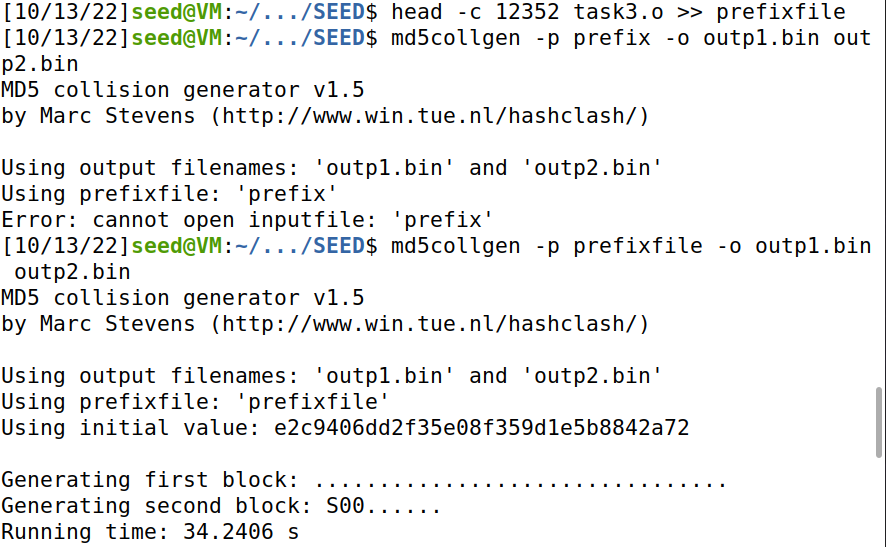


* As per the requirements the length of the prefix should be multiple of 64 so I calculated and found put it was 192. So I created the prefix file named *prefixfile* with byte size 12352 as it is a multiple of 64 about 193 times as we can’t use 192.5 or 192 due to the byte reading problems which won’t fulfill the requirements.

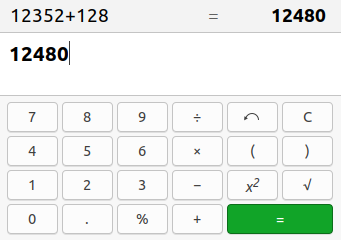




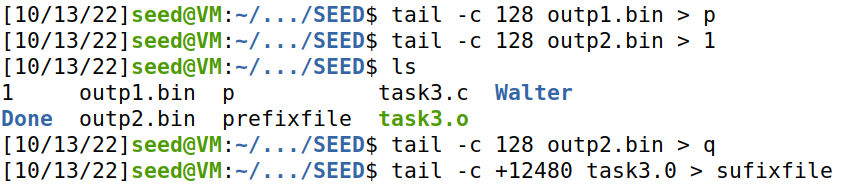
* Then I used the md5collgen to create two output files named *outp1.bin* and *outp2.bin* to compare if the md5sum is the same.



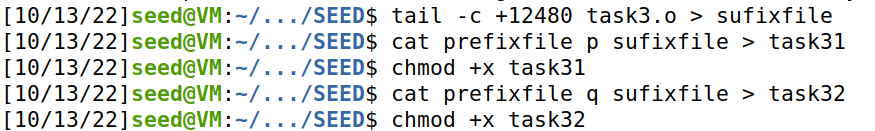
* Now to add the suffix we will add additional 128 bytes to the prefix size of 12480.



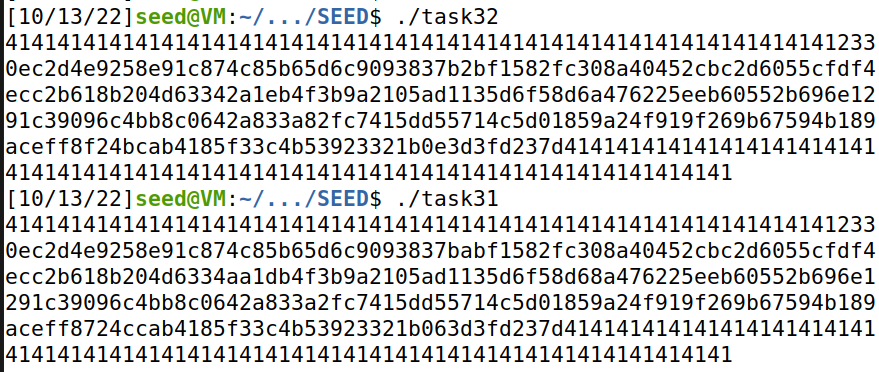
* Here I separated the values of p and q in files named *p* and *q*. Moreover, I created the *sufixfile* containing the suffix according to the above calculations.



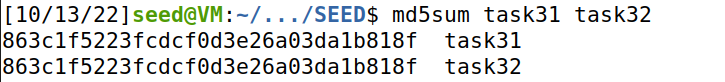
* After concatenating files as per the formula of “suffix p prefix” in document, I built their executables.



* When executing these files we get the same output



* Finally I moved to check md5sum which also was the same. Hence, proving the point of two files generating same md5sum.

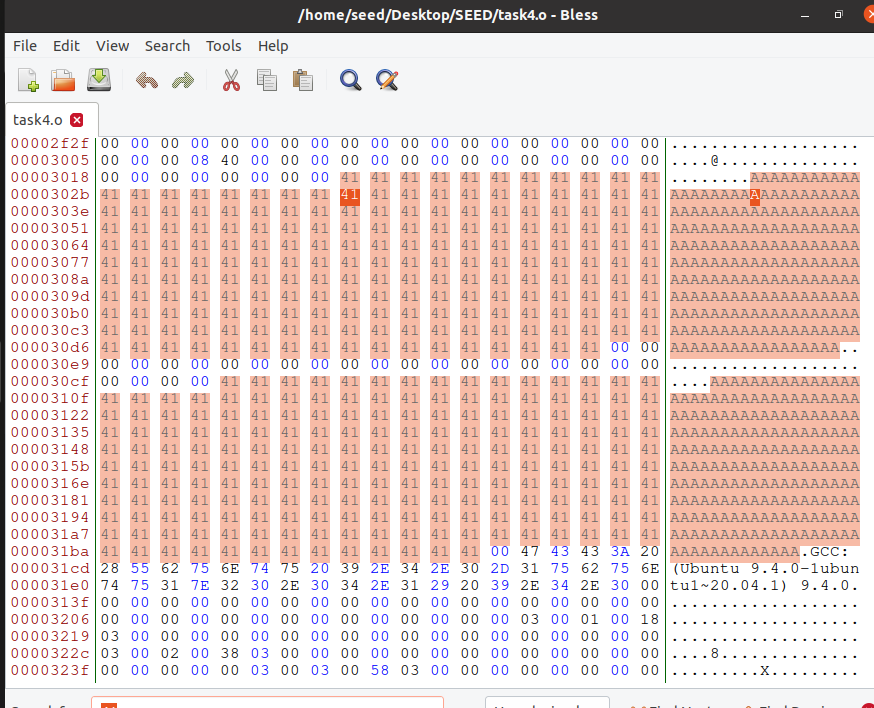


# Task 4

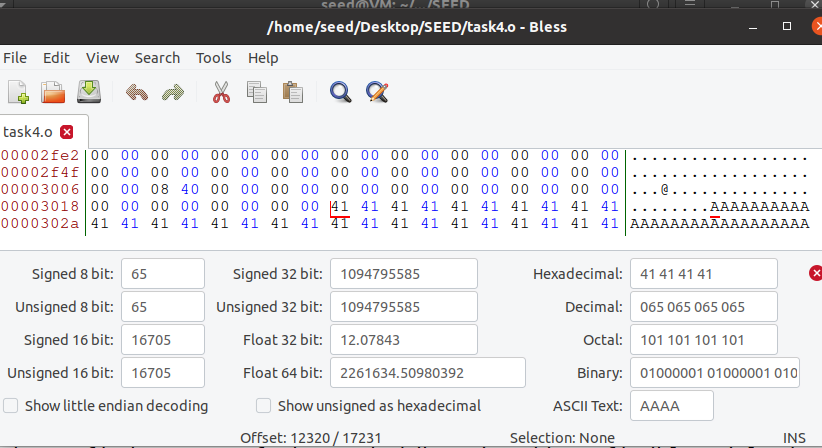
* I made a file named *task4.c* and stored the provided code with modifications left to us.



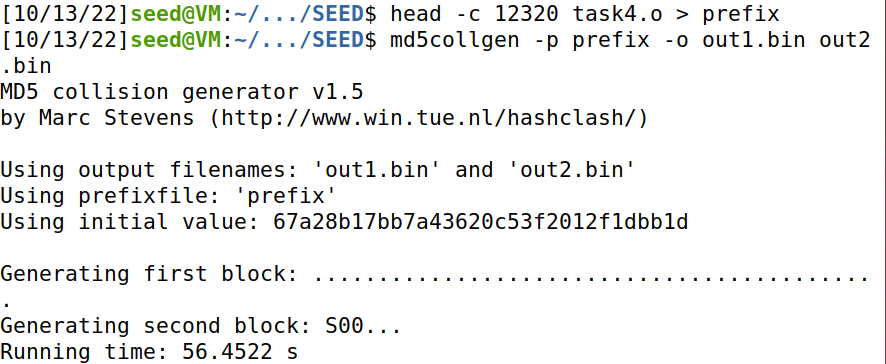
* After compiling *task4.c* in output file task4.o, I checked the hex in bless hex editor of *task4.o*. And we can observe two arrays in the screenshot below.



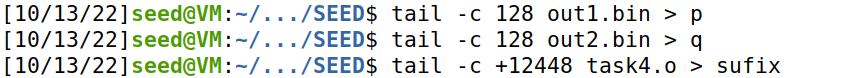
* Given that the Array starts at offset 12320 we will use it here as it is.



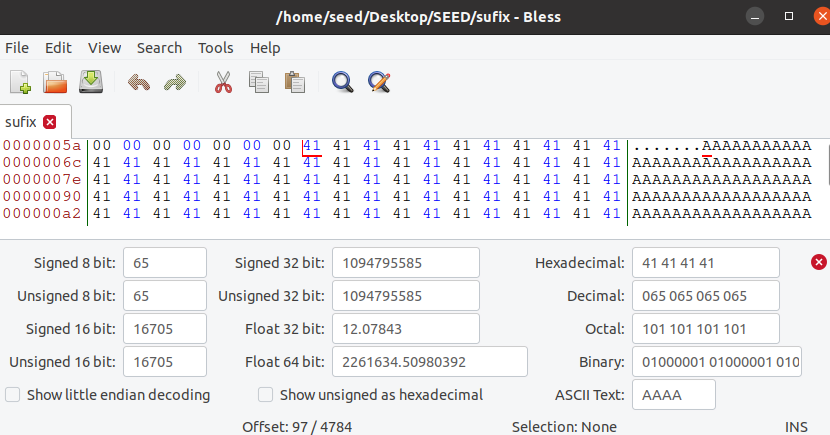
* Now creating the prefix and using md5collgen to create two output files.



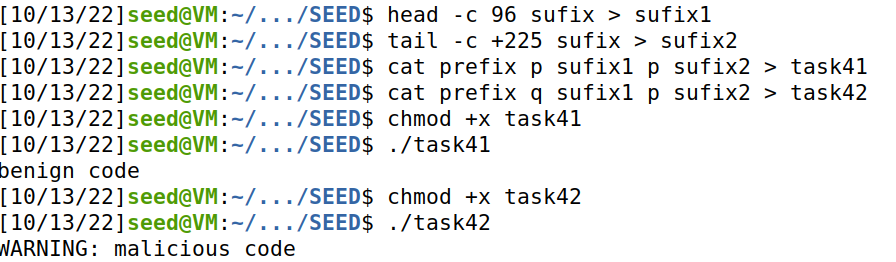
* Creating p,q and suffix files.



* Now checking the hex value of *sufix* file which is 97 in decimals.



* Taking offset value 97 bytes. Then I performed head and trail operations, and concatenated in order to perform the operation as provided in the document. By doing this the file named *task41* is declared benign and the *task42* is declared malicious.



* We achieved the results as the md5sum is the same as desired even though difficult to achieve.

