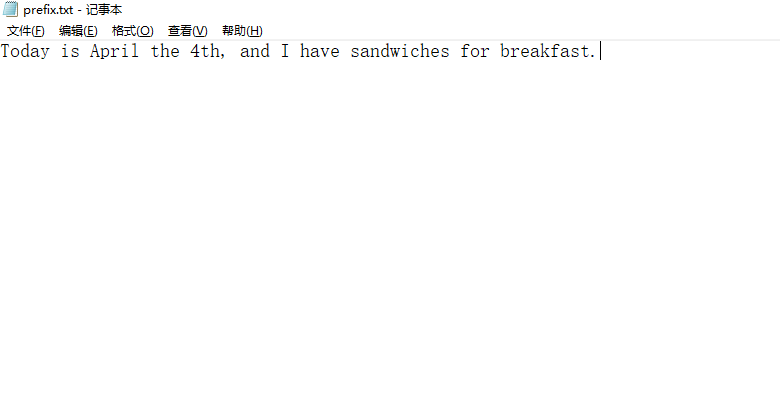
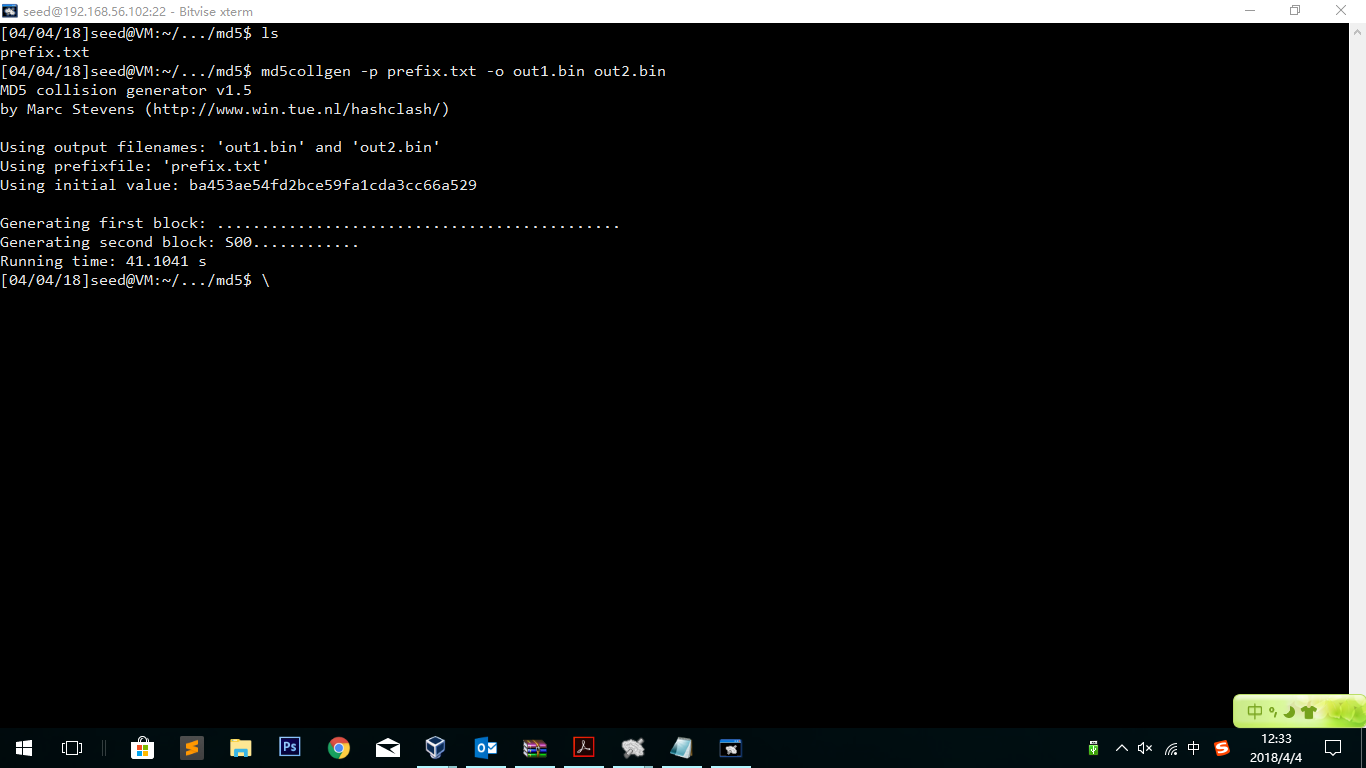
**MD5 Collision Attack Lab**

**2.1 Task 1: Generating Two Different Files with the Same MD5 Hash**

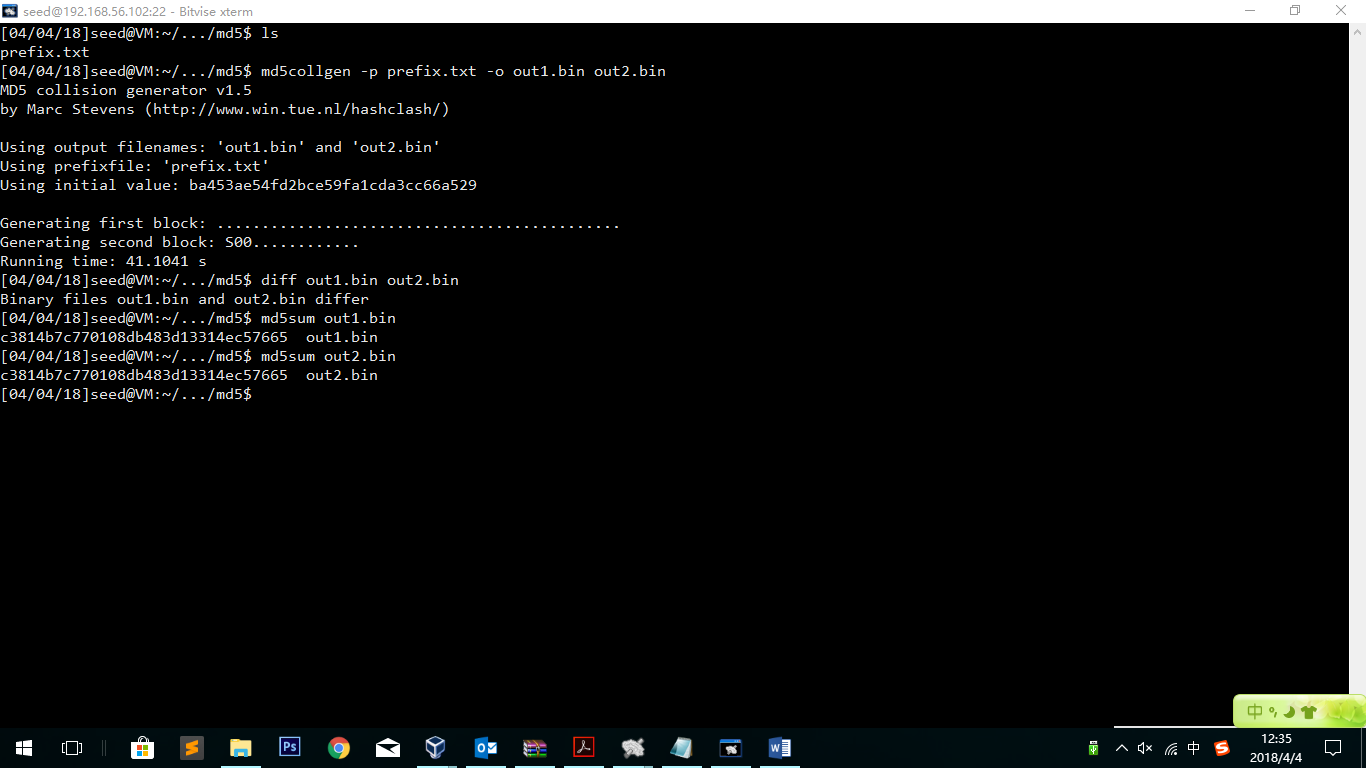
Build a file called ”prefix.txt”



*md5collgen -p prefix.txt -o out1.bin out2.bin*



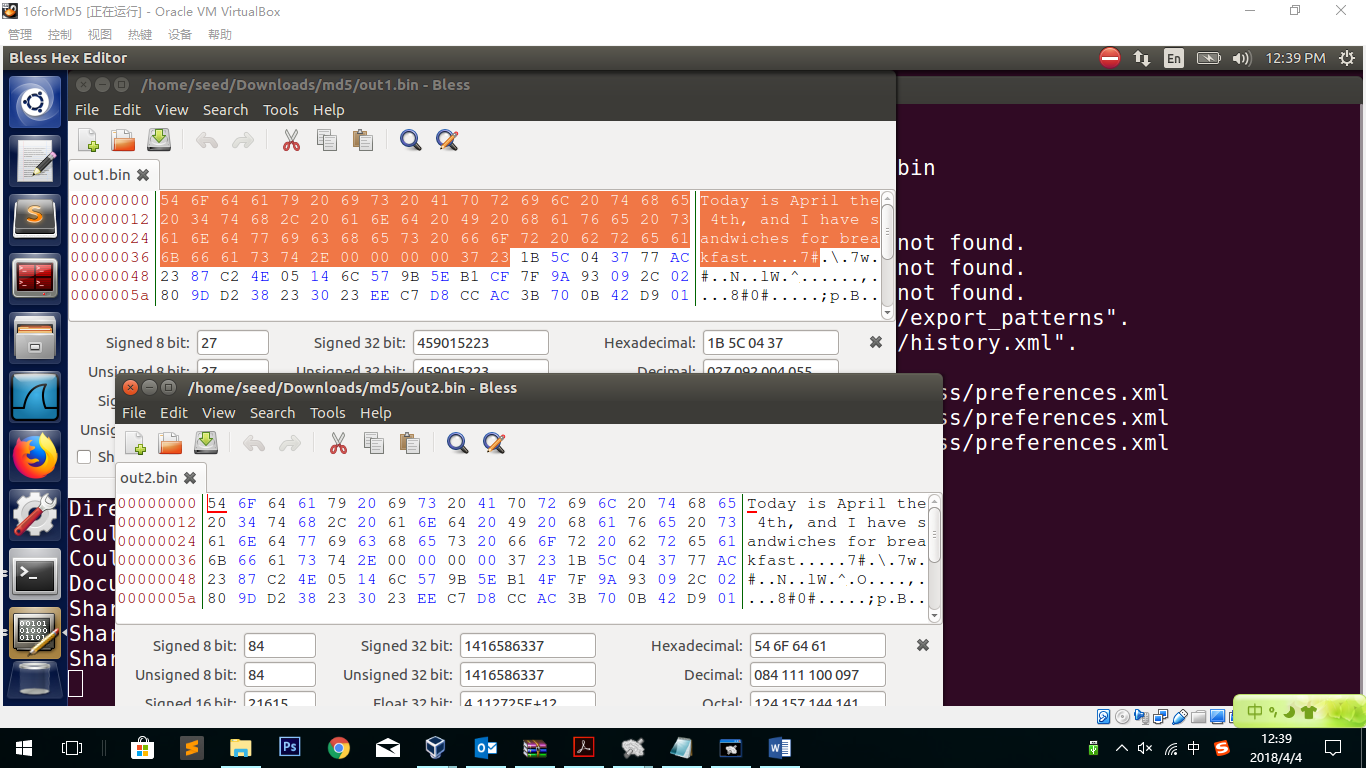
diff out1.bin out2.bin

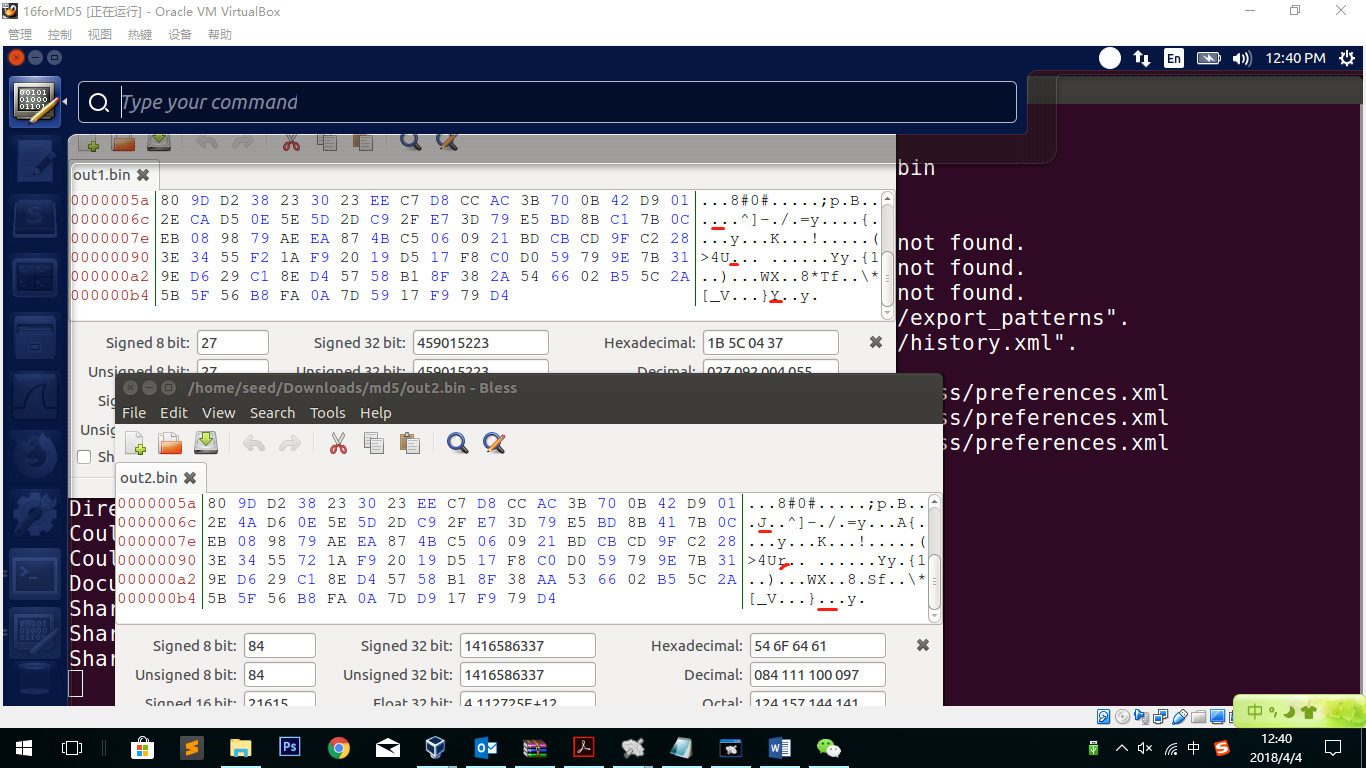


Observation:

Using the diff command we see that out1.bin and out2.bin are different. But while running md5sum, we find that out1.bin and out2.bin share same hash value.

Bless the two files to see where are the differences:





Observation:

There is no difference in the plaintext part, but in the padding part two files are different.

Explanation:

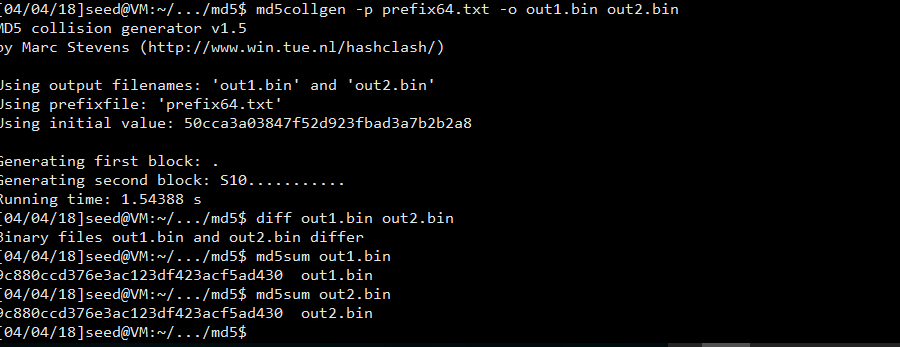
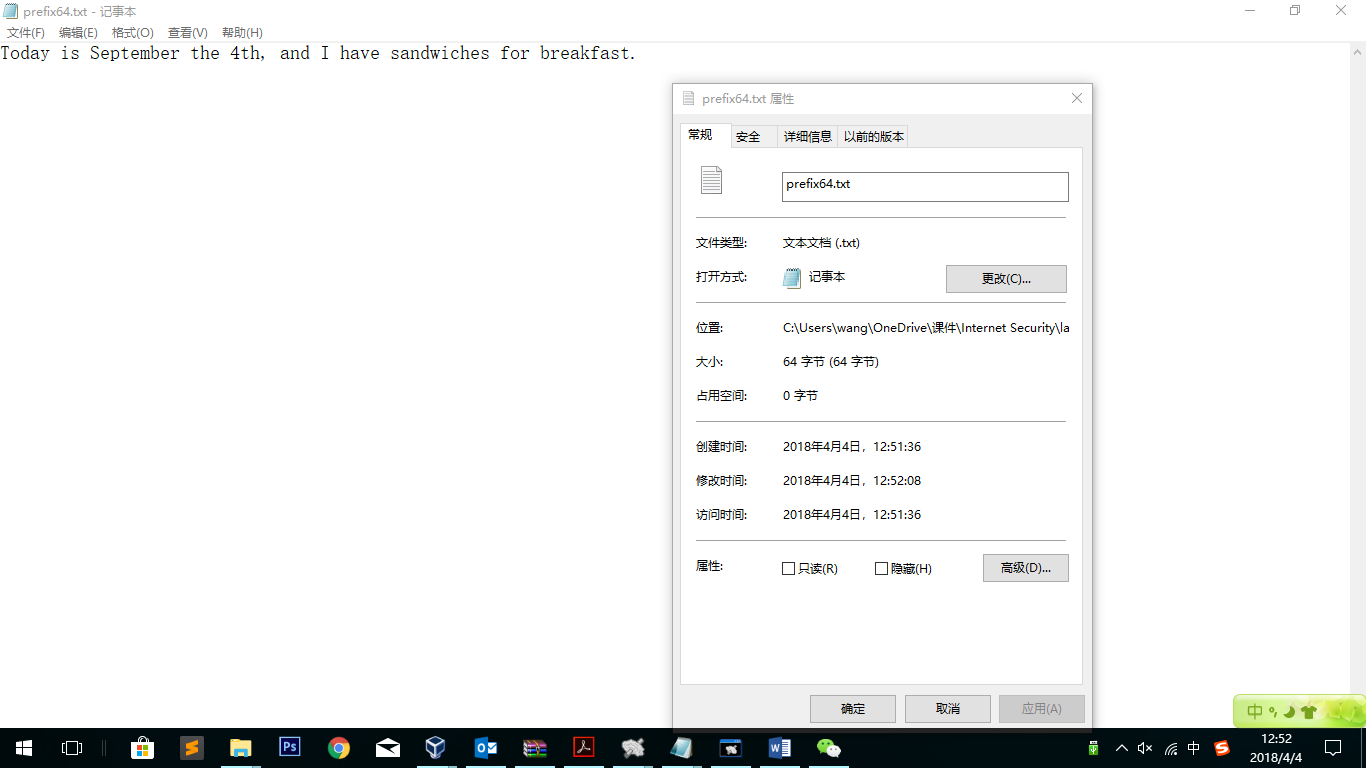
Although the two files have same hash file and are transformed from same file, but with some bytes different, they are regarded as different in the diff command.

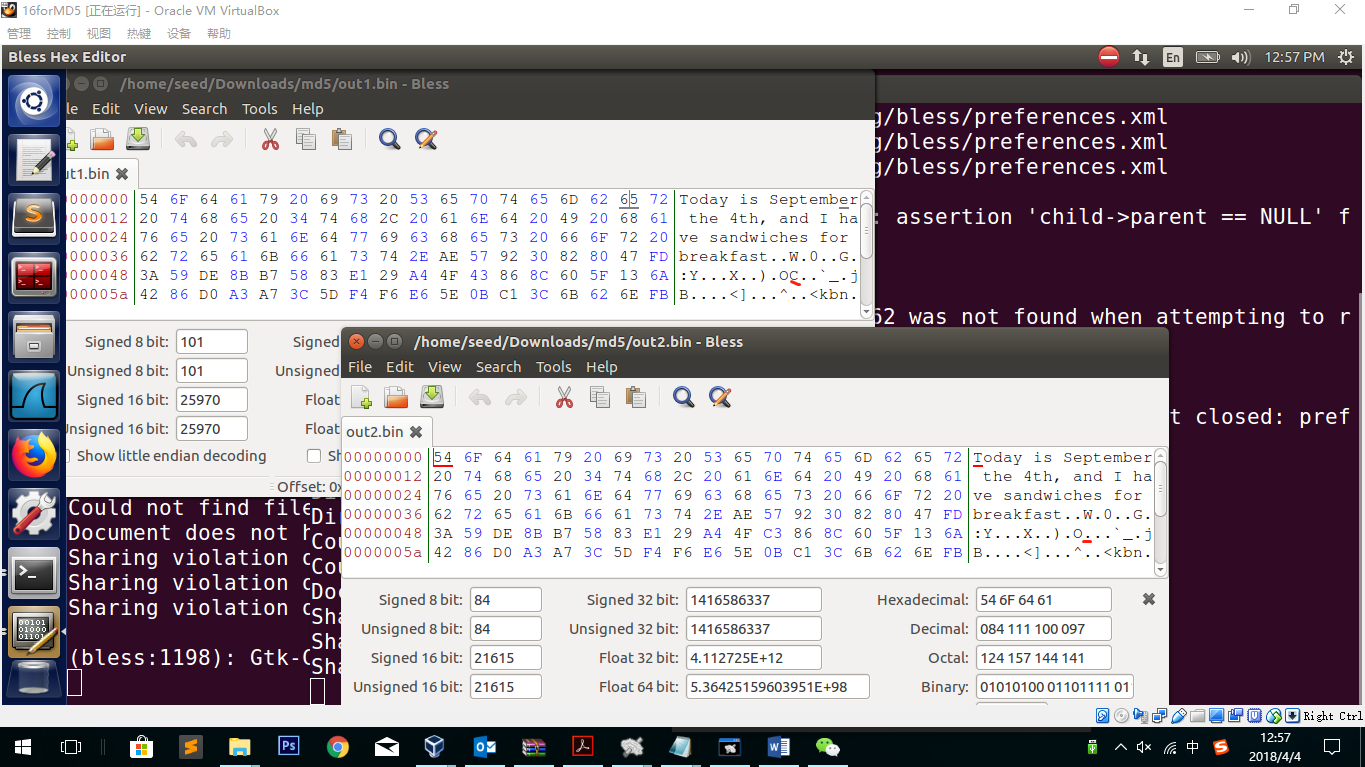
**– Question 1. If the length of your prefix file is not multiple of 64, what is going to happen?**

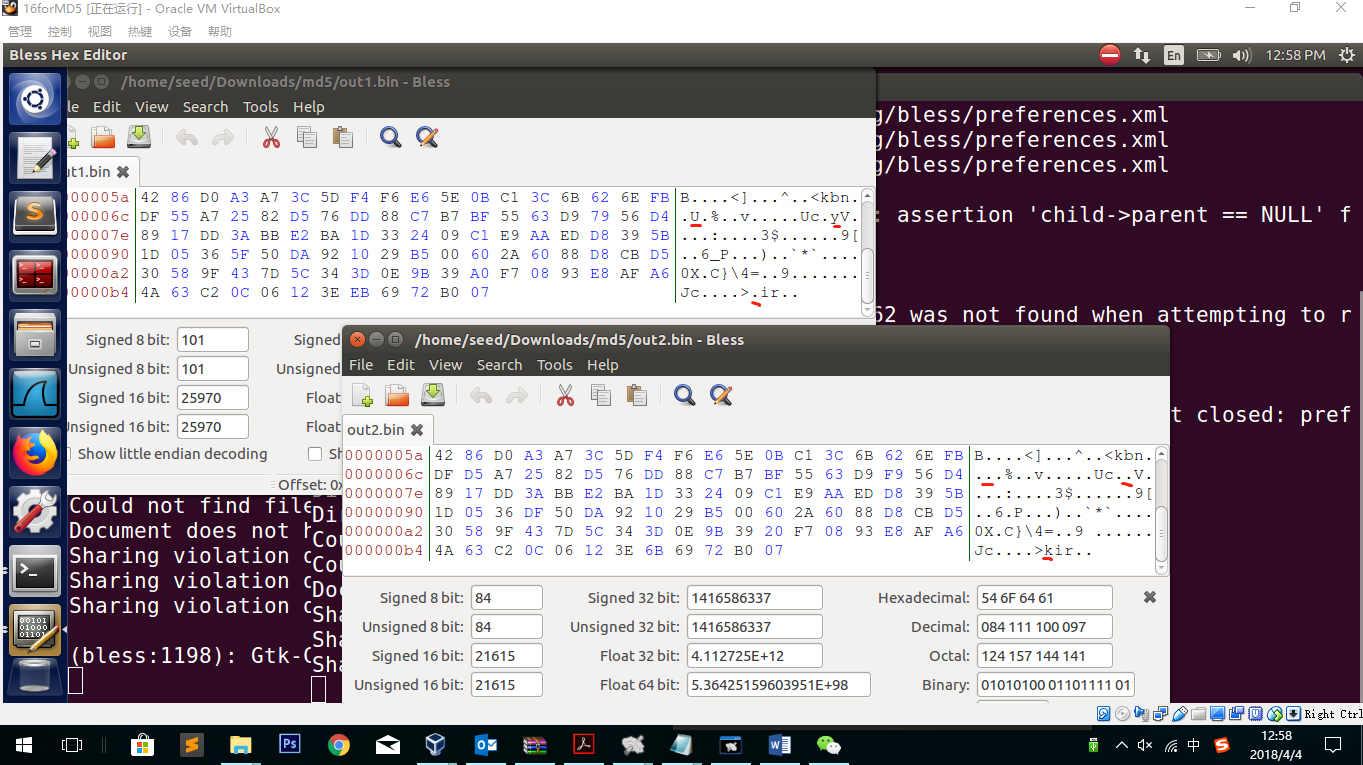
In the Lab above, the prefix file is 60 bytes not multiple of 64. In the hex editor, we can clearly see that there are 4 bytes of “00” just after the plaintext. Which means, if the prefix file is not multiple of 64, there will be some bytes padding.

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

64 bytes file:

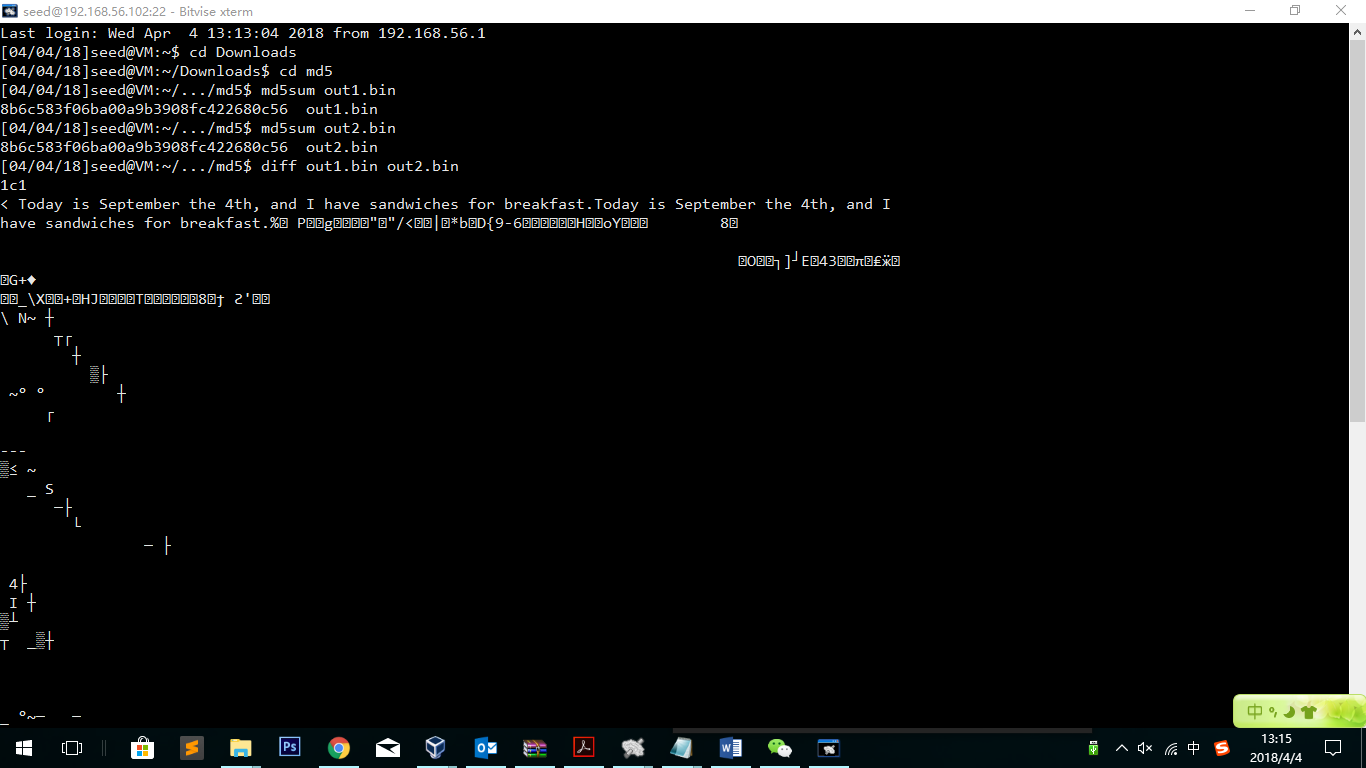






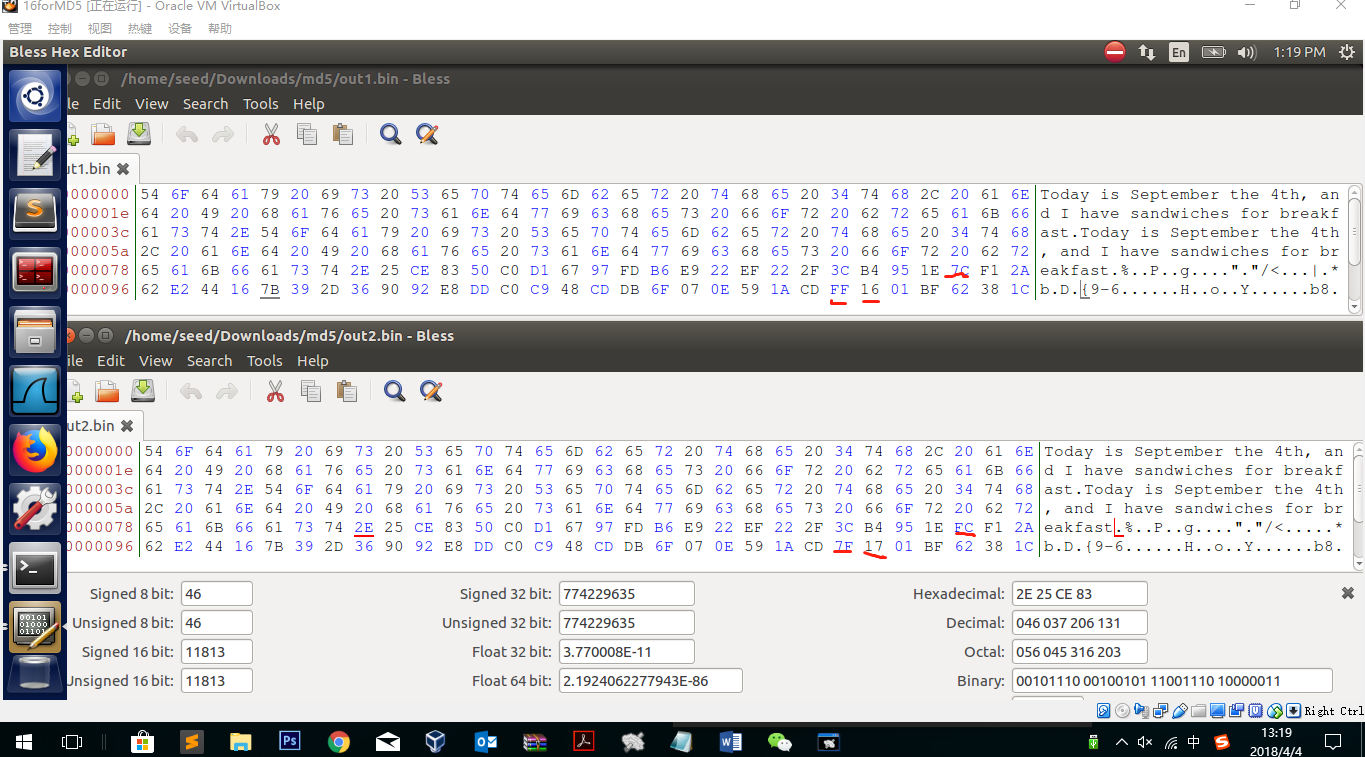
The result seems same to the previous one. This time there is no padding.

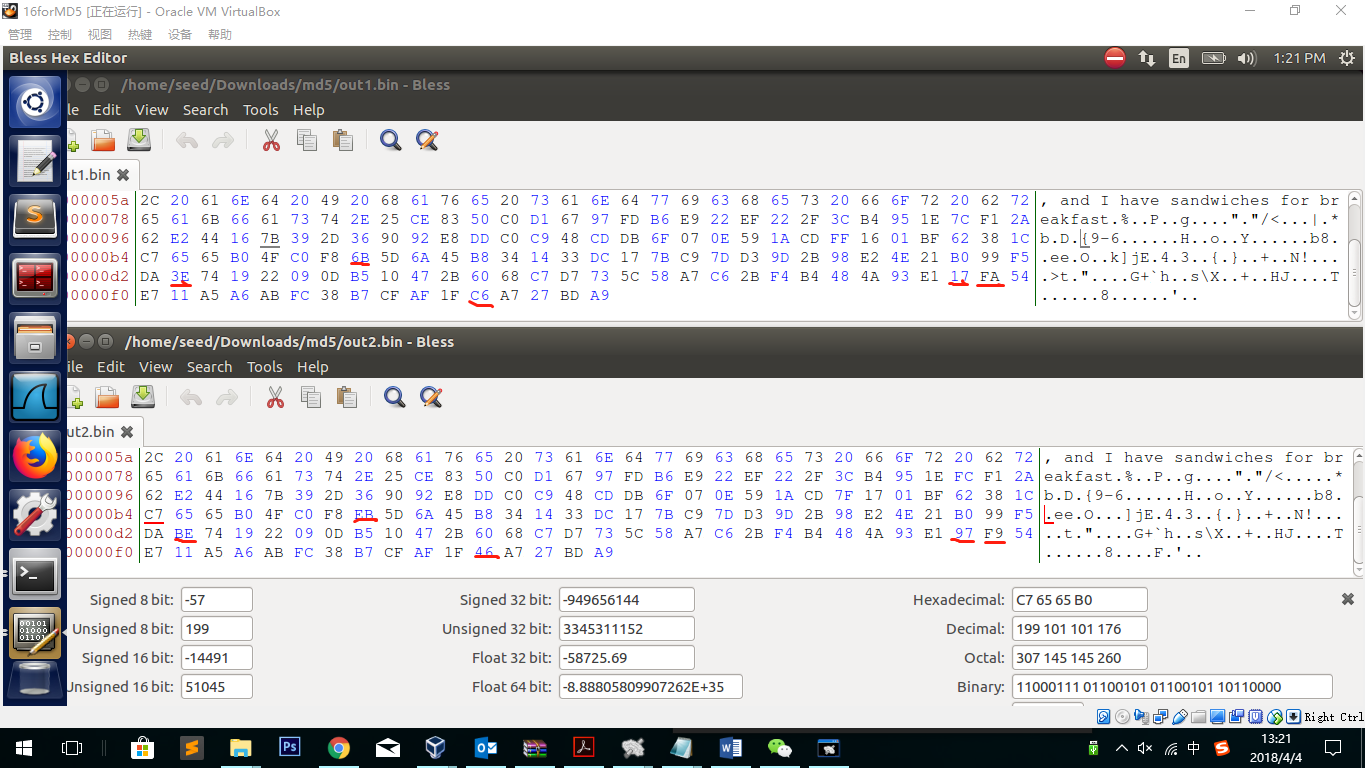
\*In the 128 bytes test, I find if I differ the files there will be many messy codes appear (Screen shots as follows). I think if there is no padding, the “end of file” flag will have something wrong and that is why the messy codes appear. However, in the 64 bytes test they doesn’t appear, which I believe is a probability event that sometimes the “end of file” flag doesn’t go wrong.

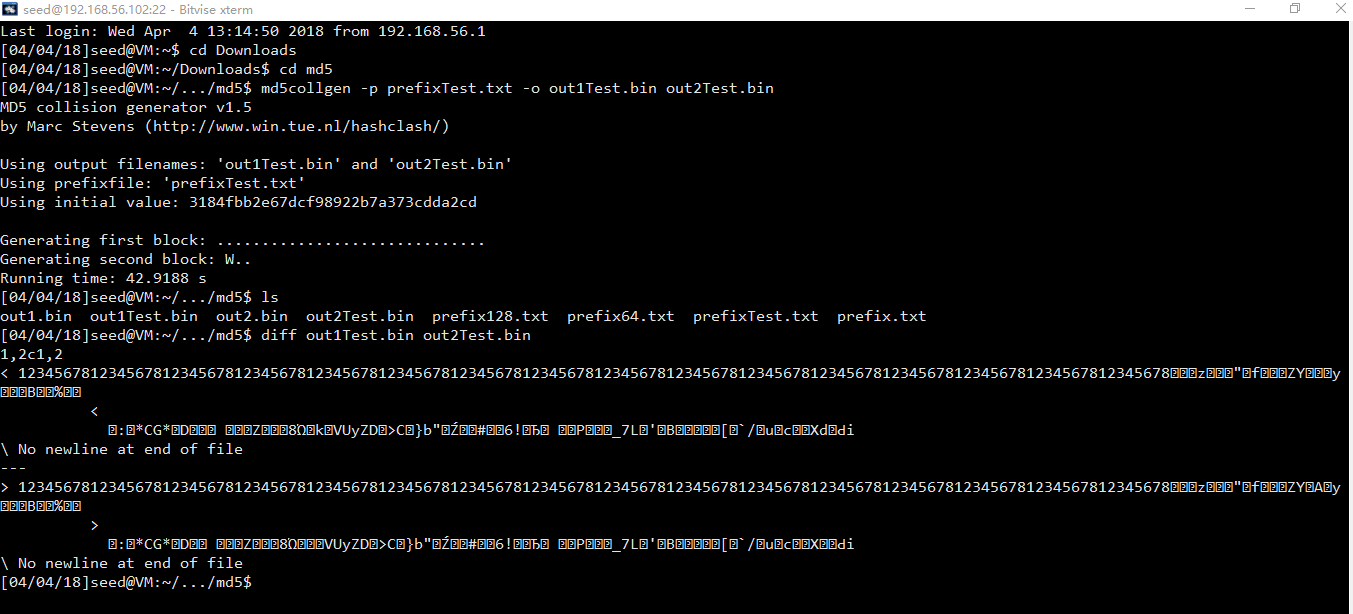


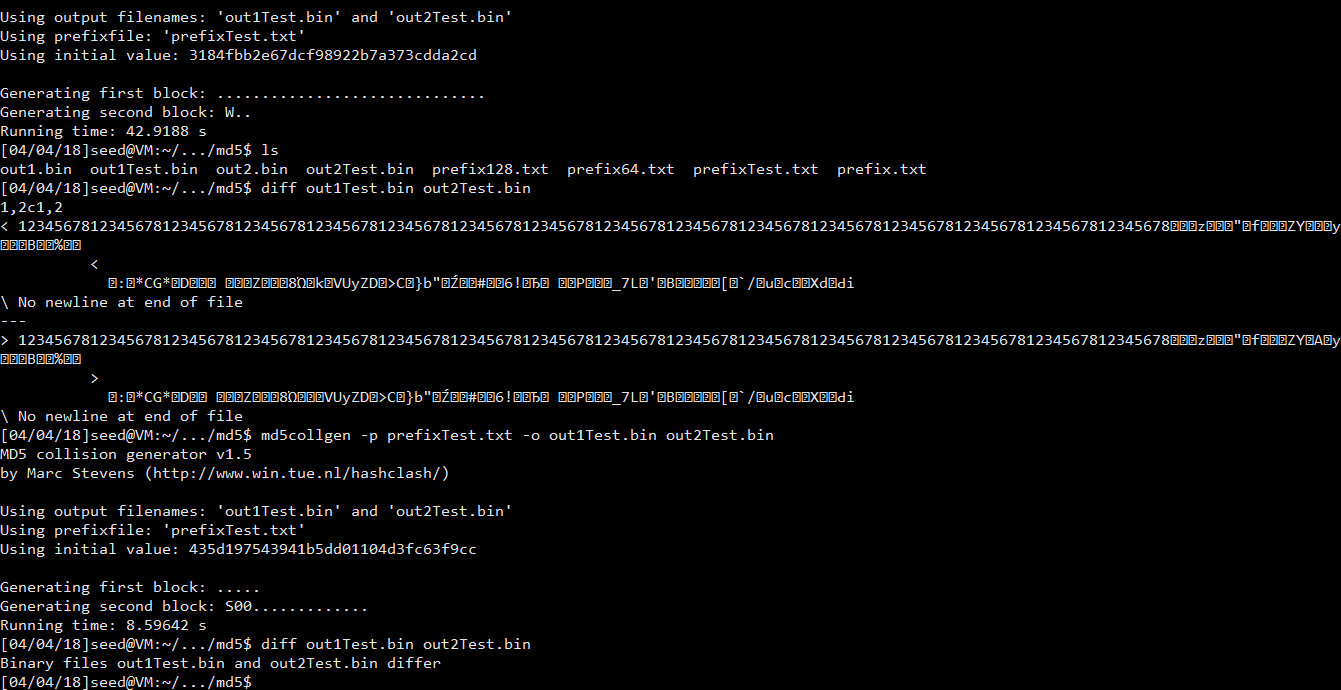
**– Question 3. Are the data (128 bytes) generated by md5collgen completely different for the two output files? Please identify all the bytes that are different.**

**Differences:**









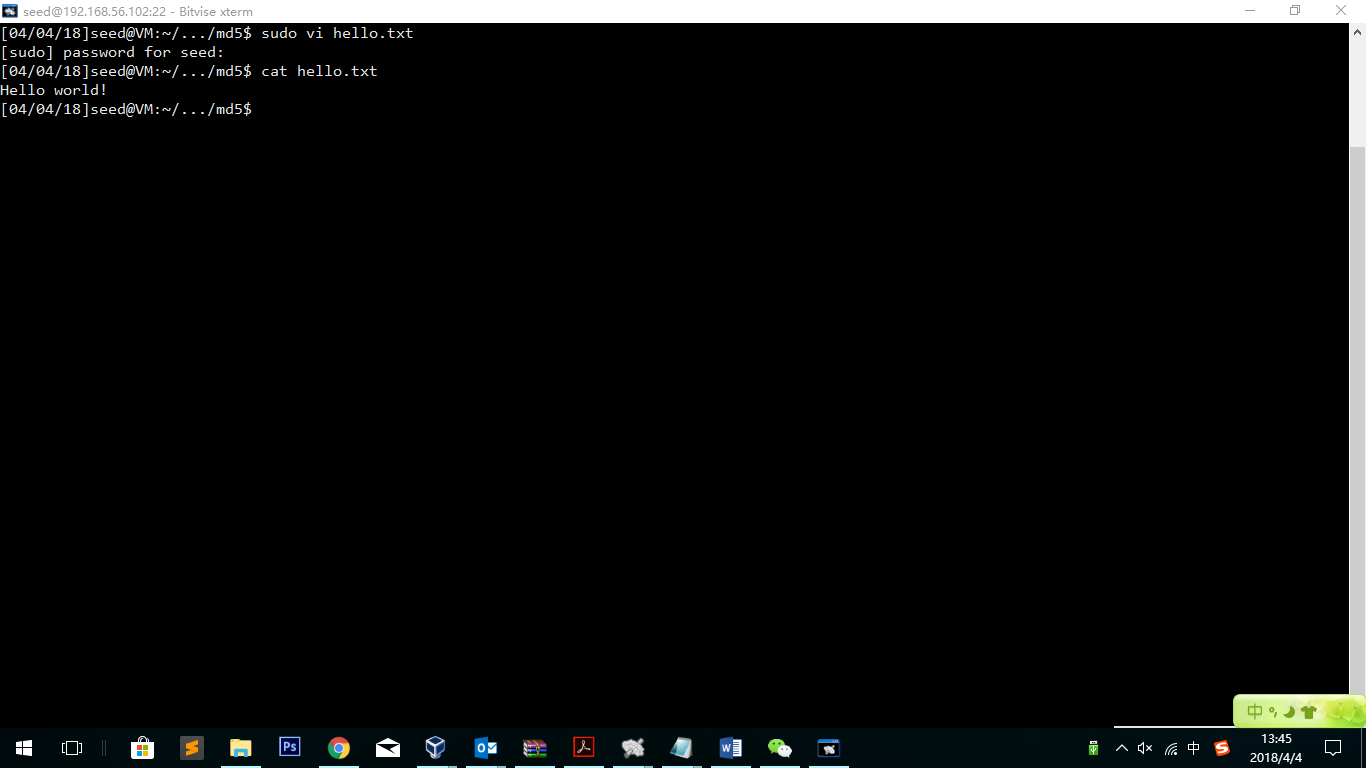
**128->127**

**2.2 Task 2: Understanding MD5’s Property**

In this task, I will use out1.bin and out2.bin. Because these two different files share the same hash value, we can suffix a same file to it.

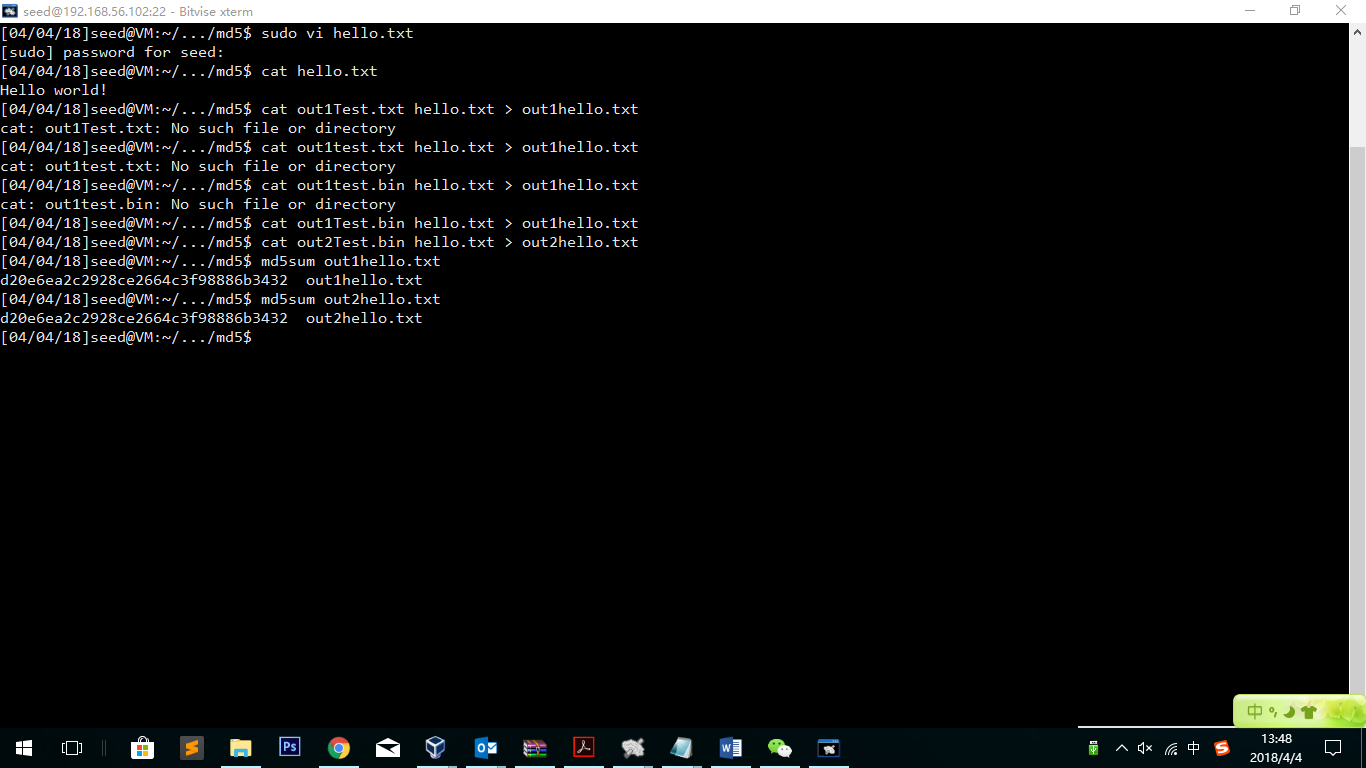
Create the suffix file:

Sudo vi hello.txt



cat out1Test.bin hello.txt > out1hello.txt

cat out2Test.bin hello.txt > out2hello.txt



md5sum out1hello.txt

md5sum out2hello.txt

Observation:

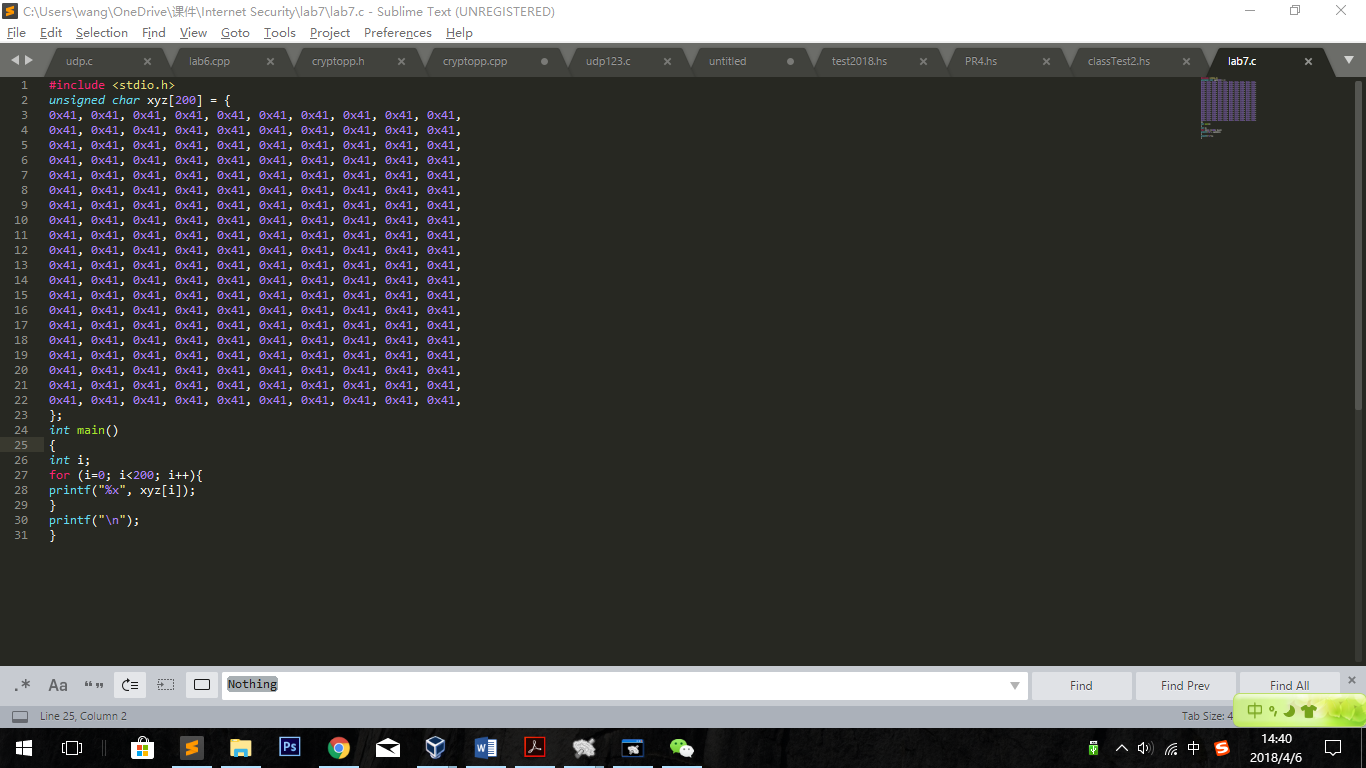
The new files share the same hash value.

Explanation:

It proves that the property that *adding the same suffix to files with same hash value will result in two outputs that have the same hash value* holds for MD5 as well.

**2.3 Task 3: Generating Two Executable Files with the Same MD5 Hash**

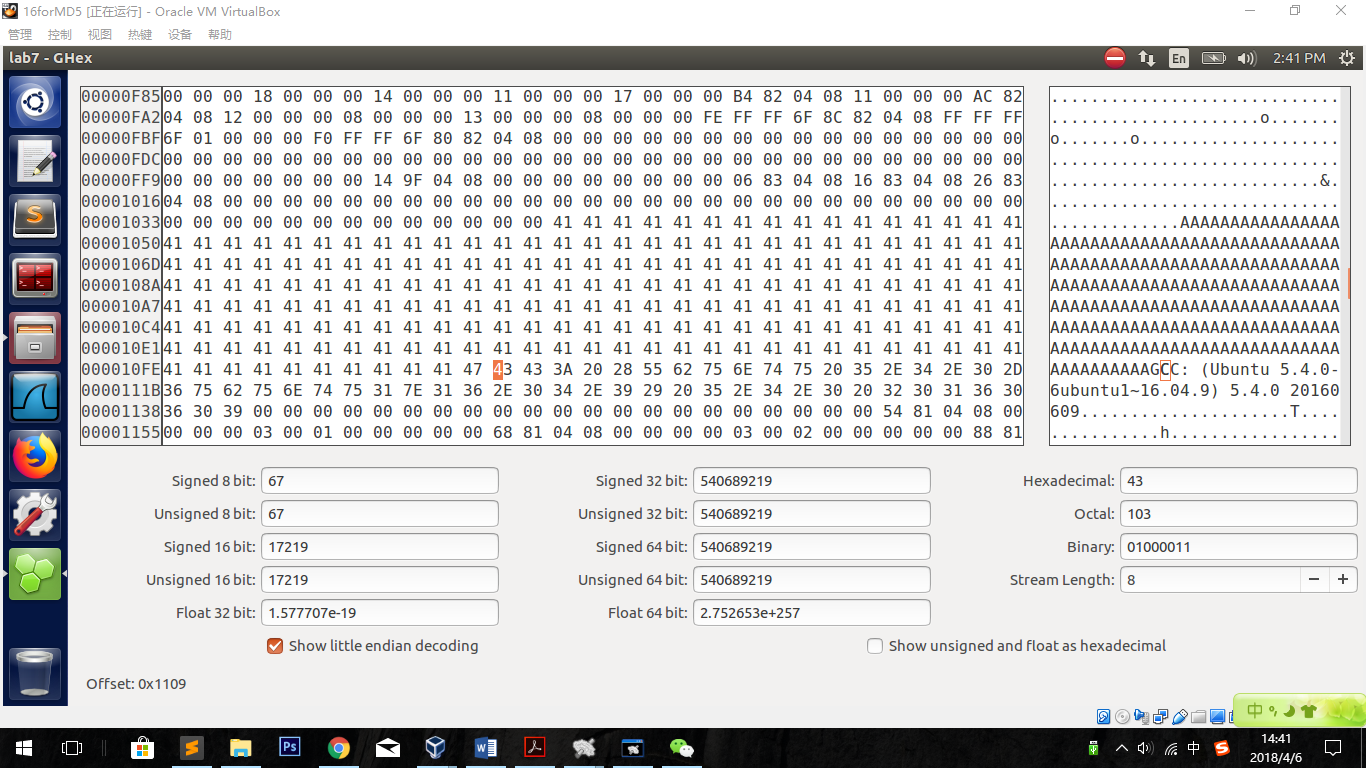
Firstly, fill the codes with A:



By doing this, we can clearly see the values directly from binary execution codes.

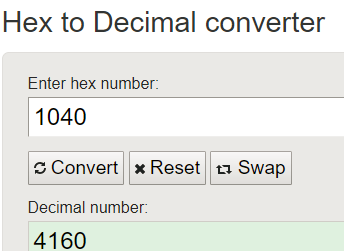
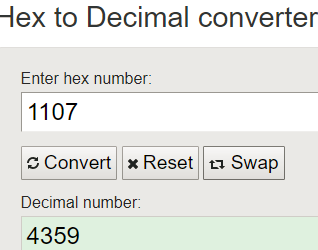
Gcc –o lab7 lab7.c

Open with hex Editor:



We can clearly see from the right side where our As are. Clicking on the first and the last A to find the byte number, which are 0X1040 and 0X1107. We need to transform the offset to decimal base.

Using the online tool: <https://www.rapidtables.com/convert/number/hex-to-decimal.html>

Clearly see that the bytes we set is in the range from 4160 to 4359.

4160 mod 64=0, and we can insert our binary codes from 4160 to 4160+128=4288

The suffix is start from 4288.

head -c 4160 lab7 > prefix

tail -c +4288 lab7 >suffix

Using the prefix, we can get two output share same hash.

*md5collgen -p prefix -o out1.bin out2.bin*

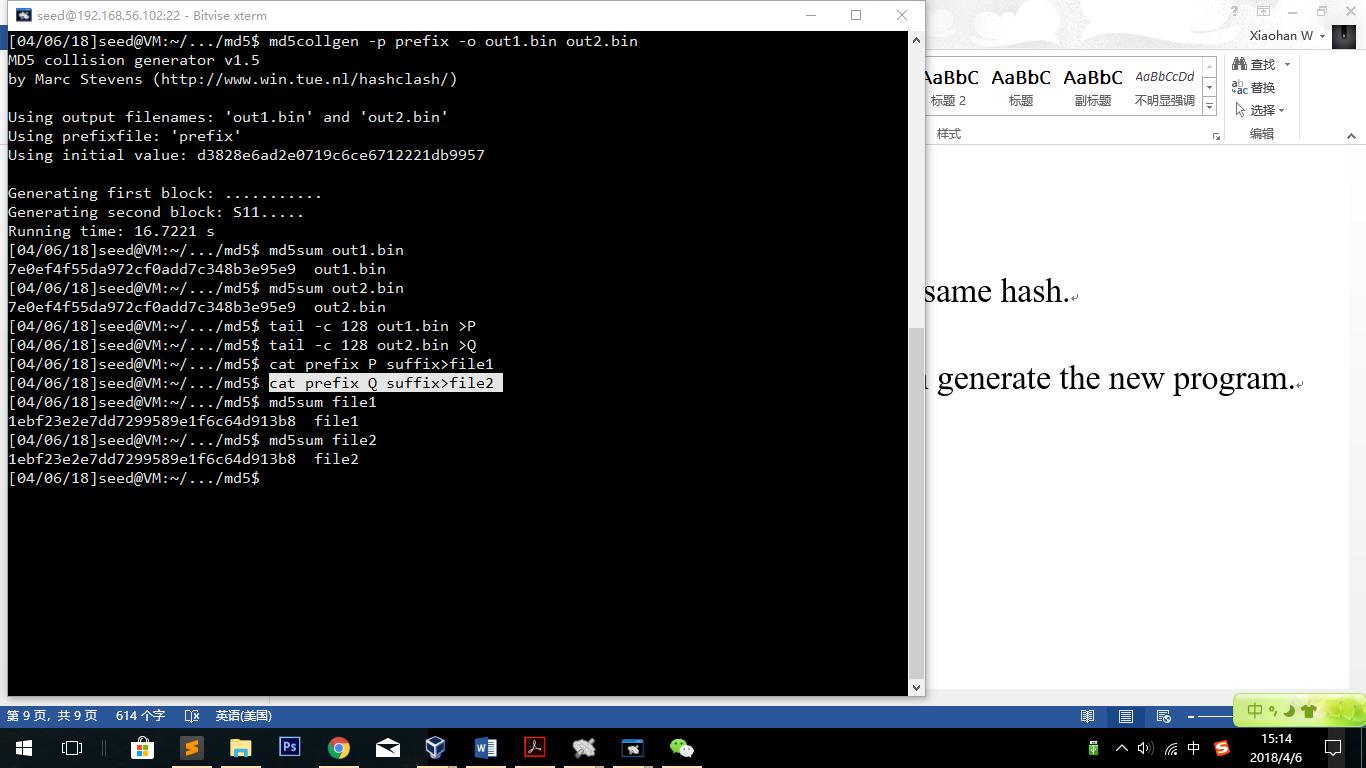
Get the last 128 bytes as the P and Q, and then generate the new program.

tail -c 128 out1.bin >P

tail -c 128 out2.bin >Q

cat prefix P suffix>file1

cat prefix Q suffix>file2



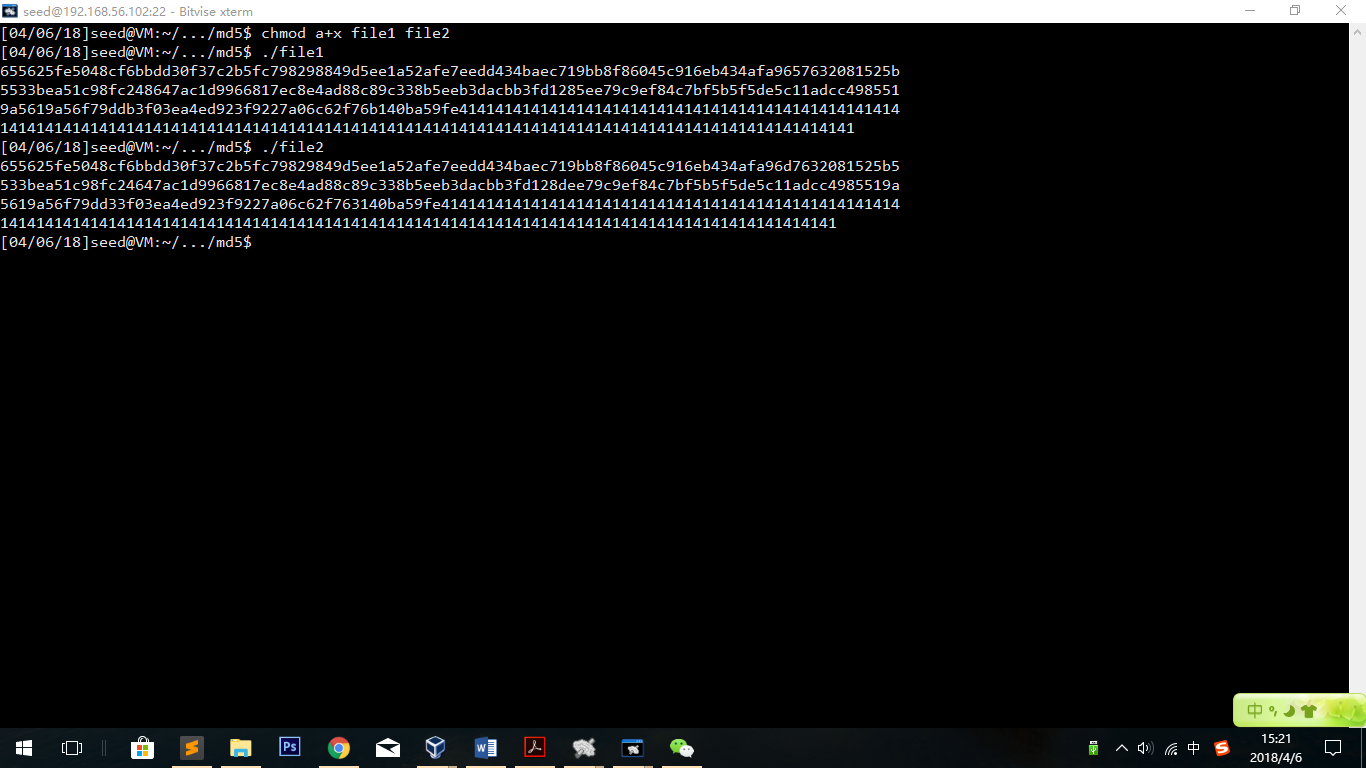
Observation:

As we can see, not only the out1.bin and out2.bin files have same md5 value, but the final programs file1 and file2 have the same md5 value.

Explanation:

We can regard prefix & P and prefix & Q are the two platforms of prefix’s md5 value, and as well we can also regard them as the 2 kinds of start of program. Suffixing the same end of the program, 2 different programs are generated.

Run the programs:



Observation:

The two programs have different results! We succeed!

**2.4 Task 4: Making the Two Programs Behave Differently**

Codes:

#include <stdio.h>

unsigned char A[200] = {

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

};

unsigned char B[200] = {

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

};

int comp(unsigned char \*m,unsigned char \*n){

int i;

for(i=0;i<200;i++){

if(m[i]!=n[i]) return 0;

}

return 1;

}

int main()

{

printf("1+1=? \n");

if(comp(A,B)){

printf("The answer is %d\n", 2);

}

else{

printf("The answer is %d\n", 3);

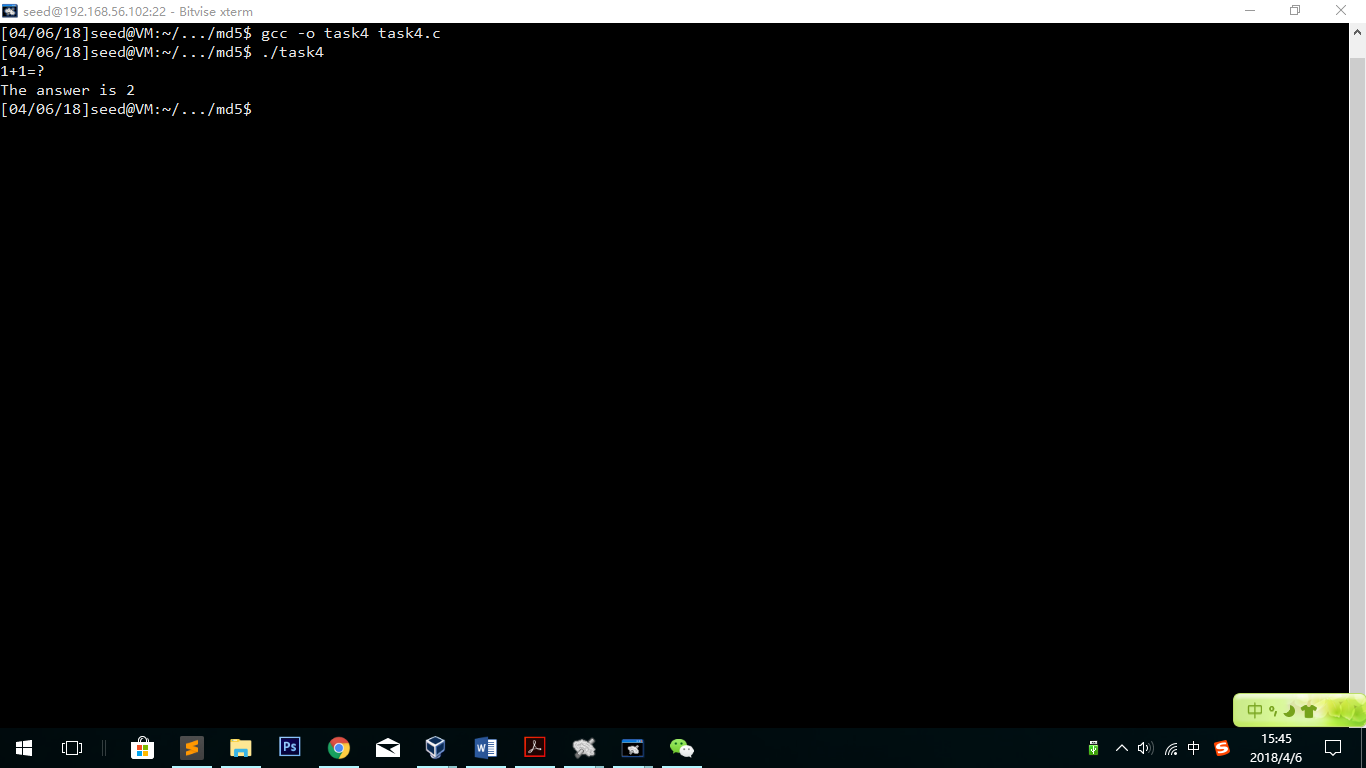
}

}

Codes explanation:

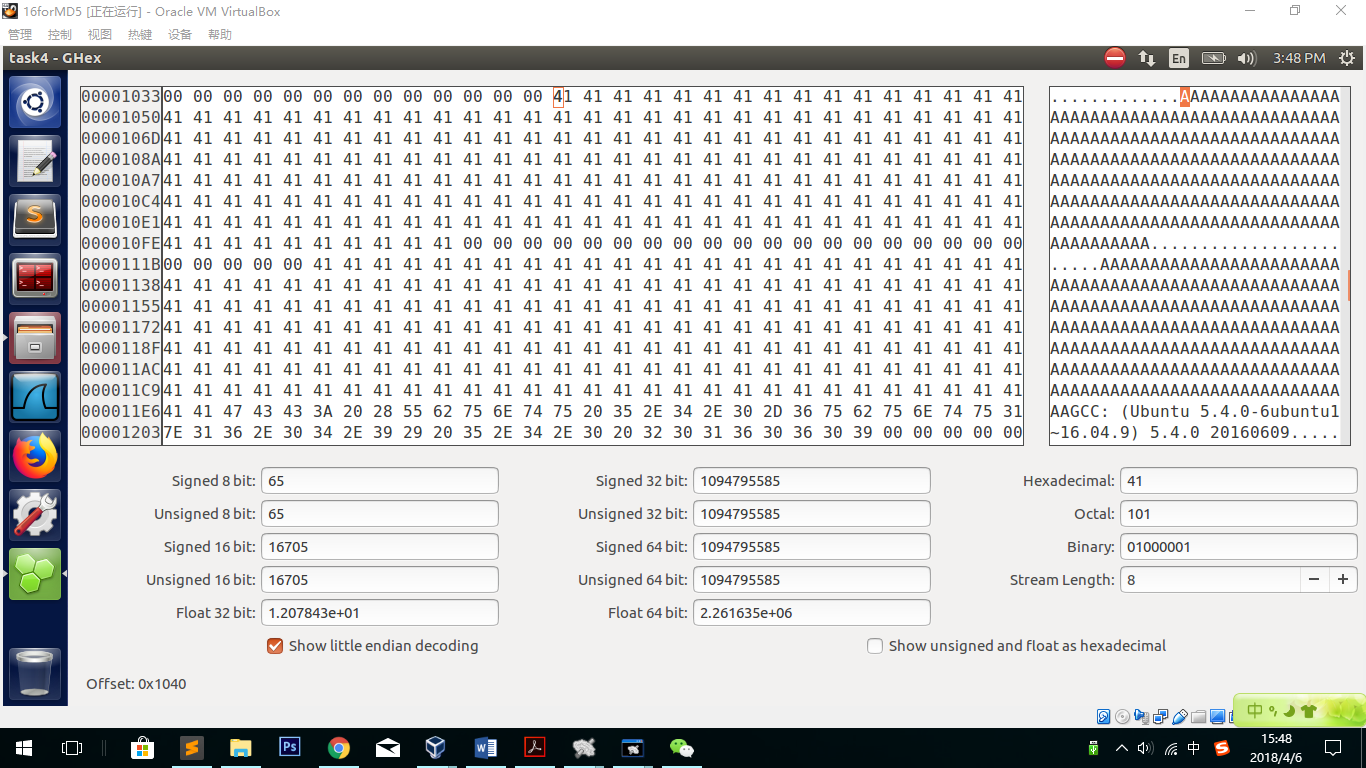
We generate 2 arrays and change them to the md5 values. Just like the previous task, both arrays are data as well as the 128-byte md5 value. So, with the 2 different values, we can change the result of the program.

The result of the codes:



It is the benign version, and the program says 1+1=2.

Look task4 file in hex editor:



Observation:

In the hex file, we can clearly see the data above. The success As are unsigned char A and B. As a result, if we change one of the array, the code will get another result.

We also see that the offset is 1040 as well, so we can use 4160 and 4288 again in this task.

However, we can’t only change A to two values. If so, A and B are not equal forever. In this case, we can use 2 different types of A and only one type of B. Between the two md5 values only one can get right answers. To achieve this, we need to change the same part of B. B’s offset is 0x1120, which is 4384.

Because 4384 mod 64=32, so we need to use 4384+32=4416 as the start.

4416+128=4544, so 4544 is the end.

Firstly, we need to get P and Q in this task:

*head -c 4160 task4 > prefix*

*md5collgen -p prefix -o out1.bin out2.bin*

*tail -c 128 out1.bin > P*

*tail -c 128 out2.bin > Q*

Change B to P:

head -c 4384 task4 >allPrefix

tail -c +4513 task4 > allSuffix

cat allPrefix P allSuffix >newTask4

generate the two programs: (in the prefix we can use task4 as well as newTask4)

head -c 4160 task4 > prefix

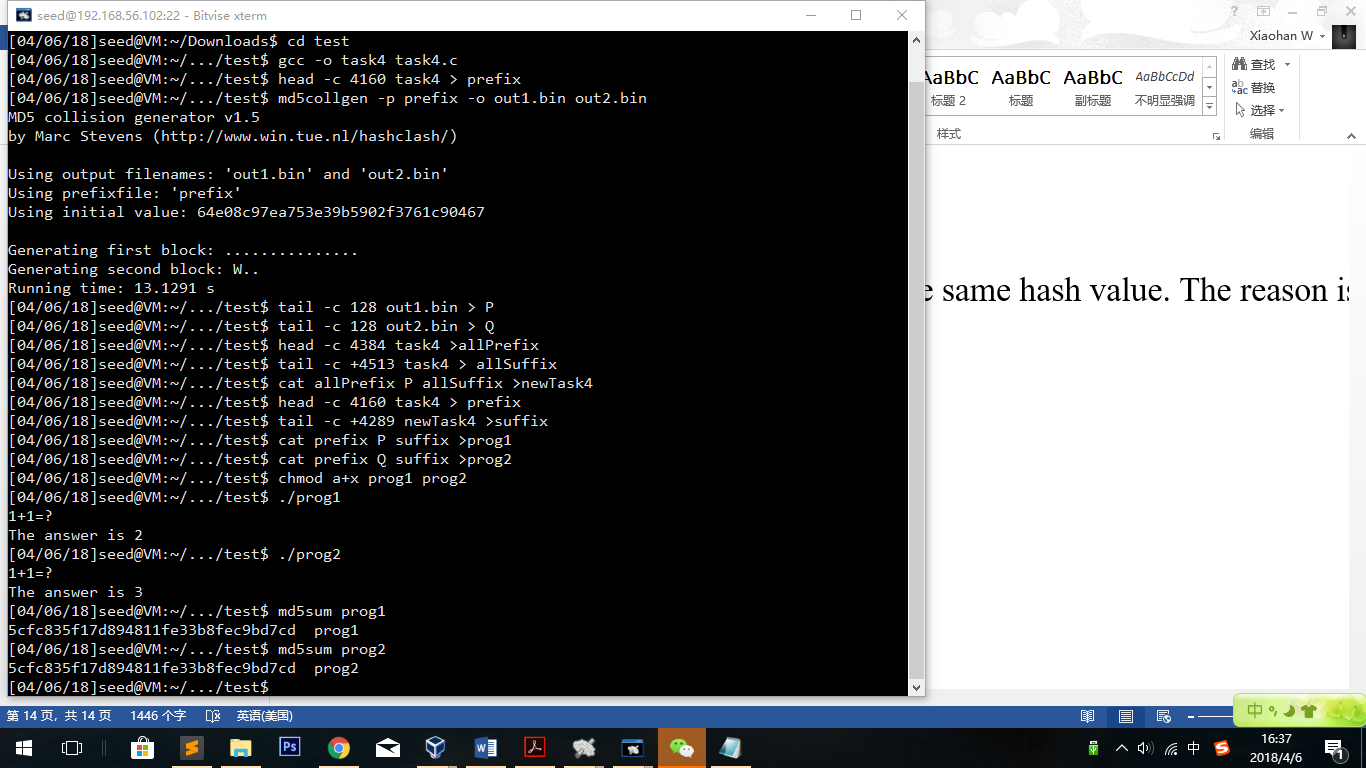
tail -c +4289 newTask4 >suffix

cat prefix P suffix >prog1

cat prefix Q suffix >prog2

Chmod a+x prog1 prog2

Run the two codes and check the md5sum:



Observation:

The two programs have the same hash value. However, the programs has different results.

Explanation:

Since we change some part of A and B to P and Q, we successfully change the condition of the Boolean in the “if structure”. A has been changed to P||A..A. B has been changed to P||A..A in program1 and Q||A..A in program2. As a matter of fact, the programs run different results.

Another observation:

At first my program goes wrong. It is because I use the start plus 128 as the number of start of suffix. In fact, I should +1 on that value because after adding 128 bytes, the next one should be (prefix+128+1)th bytes.