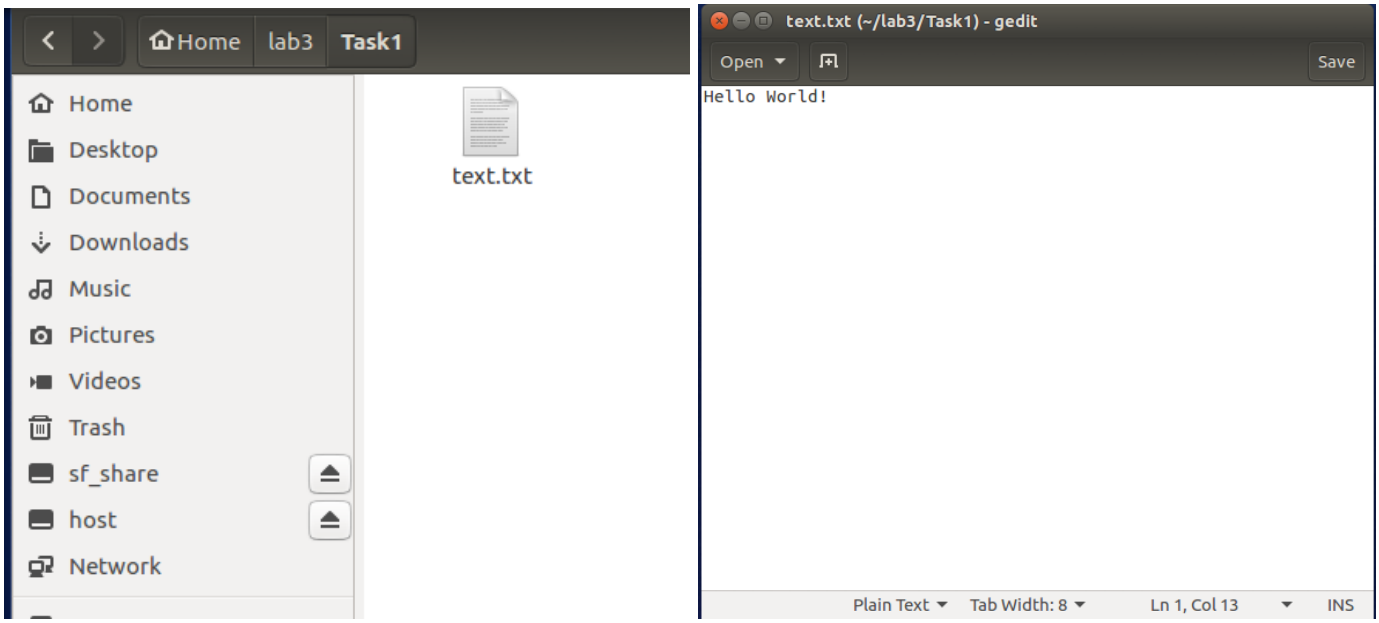
Task 1: Generating Two Different Files with the Same MD5 Hash

Let's create file called "text.txt" with the following content:



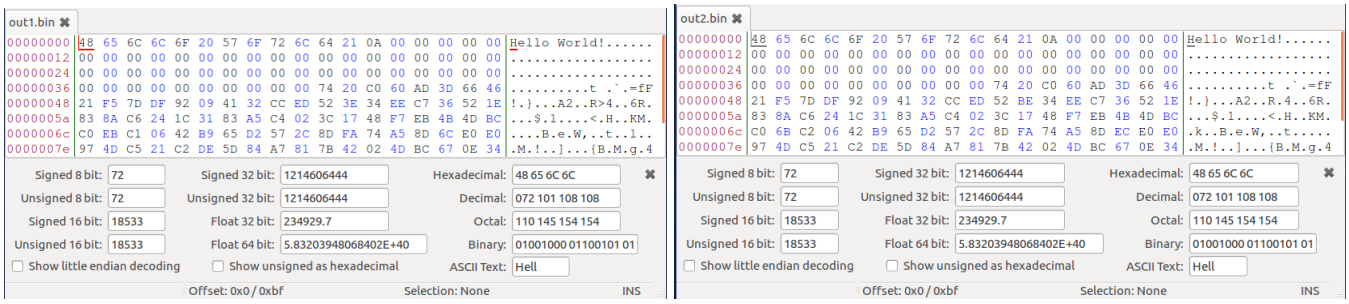
Next, we perform "md5collgen" command which allows us to provide a prefix file.

```
[06/20/21]seed@VM:~$ cd lab3/Task1
[06/20/21]seed@VM:~/.../Task1$ md5collgen -p text.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: 'text.txt'
Using initial value: bc7a2c389bfa80dae23108b4ad86f79a

Generating first block: ......................
Generating second block: S10.....................
Running time: 14.3556 s
[06/20/21]seed@VM:~/.../Task1$
```

After that, we compare files "out1.bin" and "out2.bin"

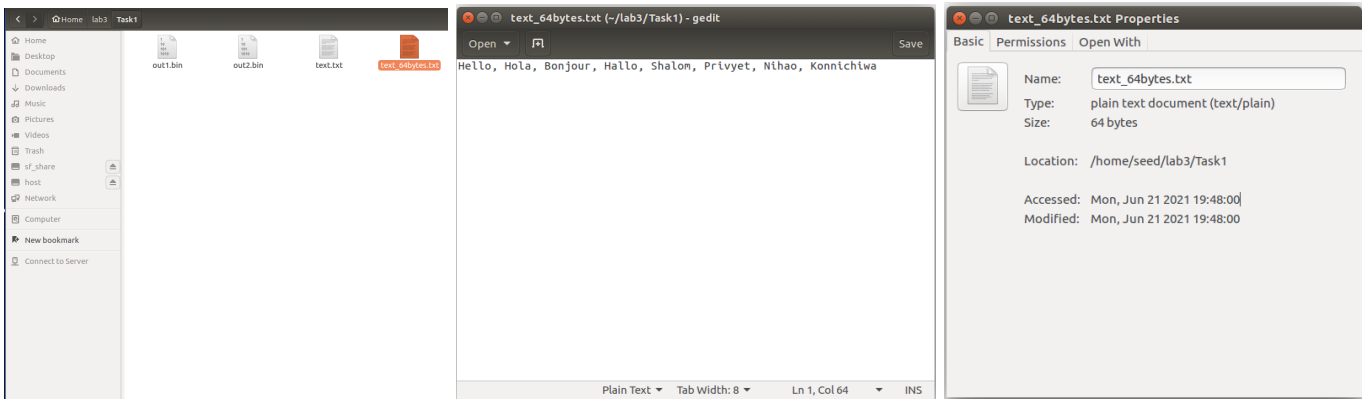
```
[06/20/21]seed@VM:~/.../Task1$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
[06/20/21]seed@VM:~/.../Task1$ md5sum out1.bin
19f1df301500f2583b055ba9bea0c697  out1.bin
[06/20/21]seed@VM:~/.../Task1$ md5sum out2.bin
19f1df301500f2583b055ba9bea0c697  out2.bin
[06/20/21]seed@VM:~/.../Task1$ ls
out1.bin  out2.bin  text.txt
[06/20/21]seed@VM:~/.../Task1$
```

Using bless, we can check the content of “out1.bin” and “out2.bin”. As it turns out, the values are the same for both files



Question 1. If the length of your prefix file is not multiple of 64 , what is going to happen?
Answer: After the last byte of the file content, the files will be padded with 0s

Let’s create a new file that will contain exactly 64 bytes. Let’s call it a “text_64bytes.txt”



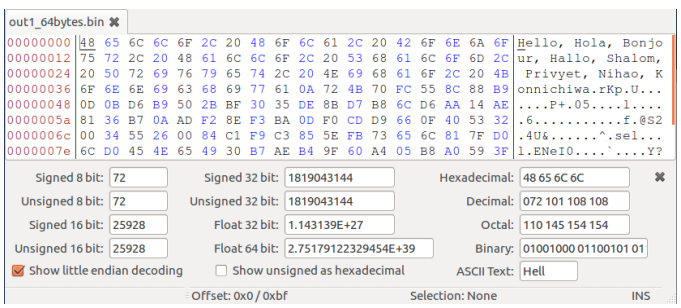
Then, we perform command “md5collgen”on file “text_64bytes.txt” which allows us to provide a prefix file.

```
[06/21/21]seed@VM:~/.../Task1$ md5collgen -p text_64bytes.txt -o out1_64bytes.bin out2_64bytes.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1_64bytes.bin' and 'out2_64bytes.bin'
Using prefixfile: 'text_64bytes.txt'
Using initial value: d26c7c12c989fb19c5149e69de930016

Generating first block: ............
Generating second block: S11.....
Running time: 6.47366 s
[06/21/21]seed@VM:~/.../Task1$
```

After that, we compare files “out1_64bytes.bin” and “out2_64bytes.bin”

```
[06/21/21]seed@VM:~/.../Task1$ diff out1_64bytes.bin out2_64bytes.bin
Binary files out1_64bytes.bin and out2_64bytes.bin differ
[06/22/21]seed@VM:~/.../Task1$ md5sum out1_64bytes.bin
995a7d4c81368ac2a75e890dcc490d36  out1_64bytes.bin
[06/22/21]seed@VM:~/.../Task1$ md5sum out2_64bytes.bin
995a7d4c81368ac2a75e890dcc490d36  out2_64bytes.bin
[06/22/21]seed@VM:~/.../Task1$
```

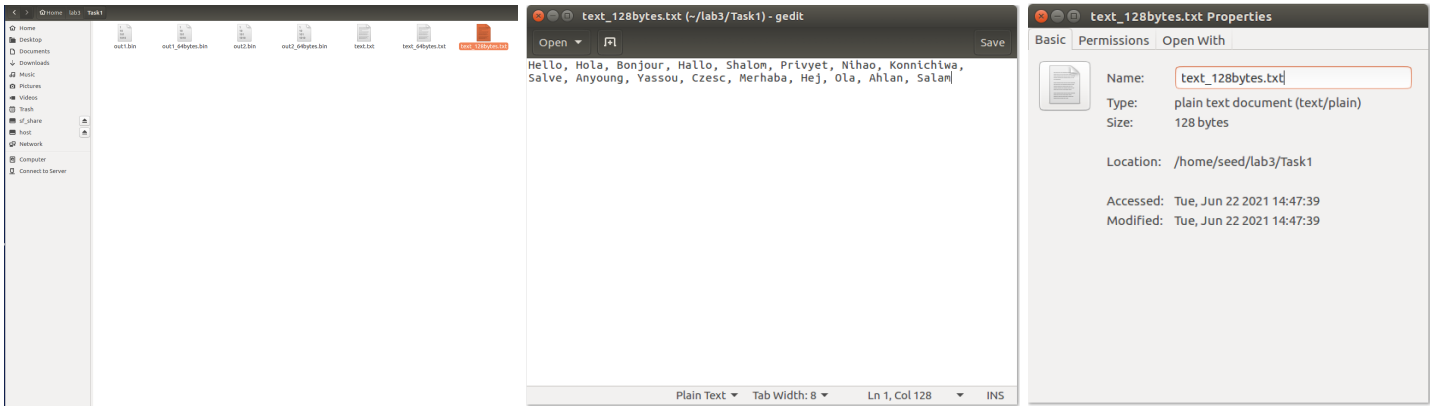
Using bless, we can check the content of “out1_64bytes.bin” and “out2_64bytes.bin”. As it turns out, the values are the same for both files. Moreover, as we can see there is no padding added after the end of the file content.



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

Answer: There are no paddings added after the end of the file (EOF)

Let’s create a new file that will contain exactly 128 bytes. Let’s call it a “text_128bytes.txt”



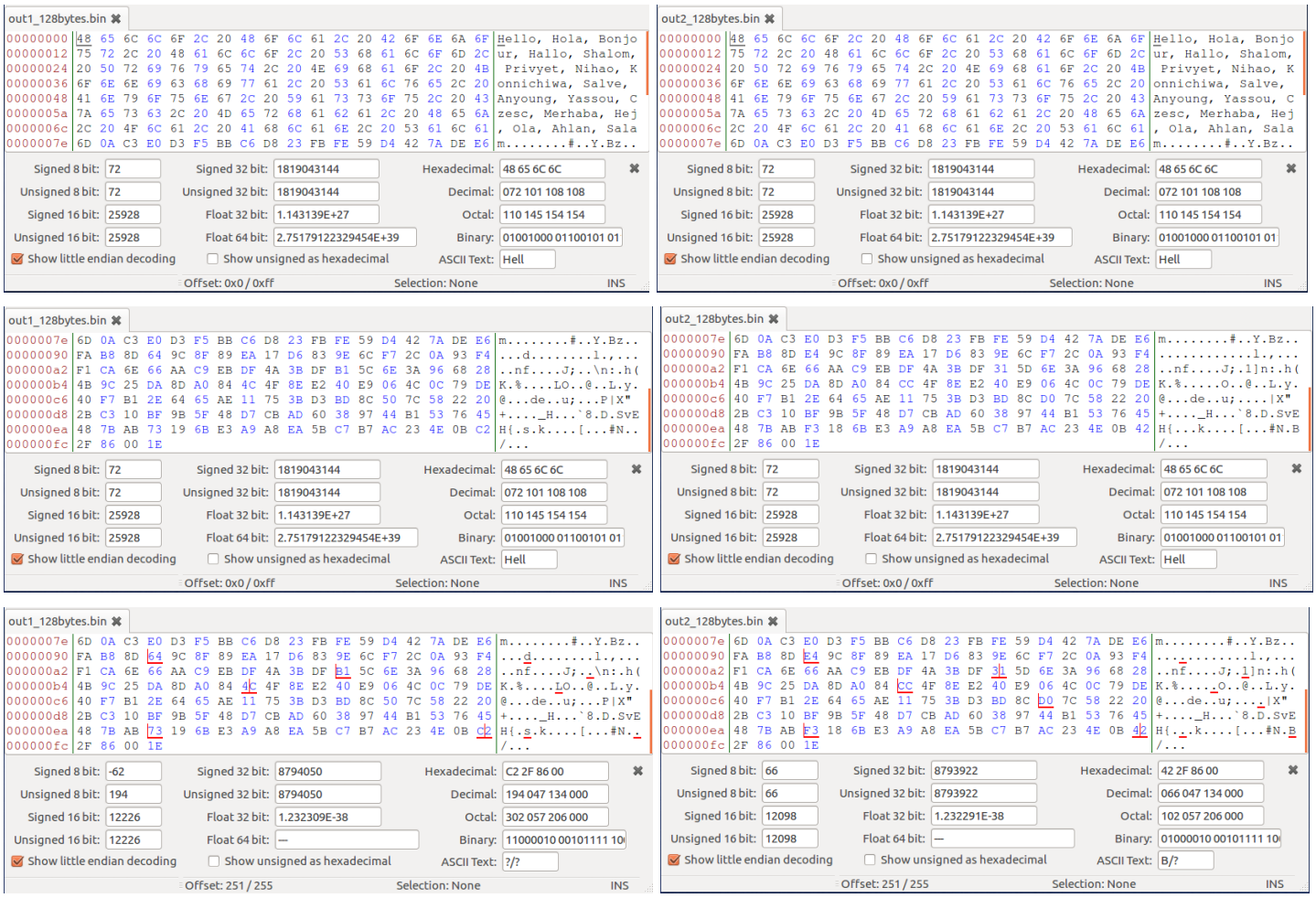
Then, we perform command “md5collgen”on file “text_128bytes.txt” which allows us to provide a prefix file.

```
[06/22/21]seed@VM:~$ cd lab3/Task1
[06/22/21]seed@VM:~/.../Task1$ md5collgen -p text_128bytes.txt -o out1_128bytes.bin out2_128bytes.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1_128bytes.bin' and 'out2_128bytes.bin'
Using prefixfile: 'text_128bytes.txt'
Using initial value: 951e717f31c1c7ac130d94112d6c8ed9

Generating first block: .......
Generating second block: S10.....................................
Running time: 14.9921 s
[06/22/21]seed@VM:~/.../Task1$
```

After that, we compare files “out1_128bytes.bin” and “out2_128bytes.bin”

```
[06/22/21]seed@VM:~/.../Task1$ diff out1_128bytes.bin out2_128bytes.bin
Binary files out1_128bytes.bin and out2_128bytes.bin differ
[06/22/21]seed@VM:~/.../Task1$ md5sum out1_128bytes.bin
f64cb51a4ac60c8a197a59be5cd95cf2  out1_128bytes.bin
[06/22/21]seed@VM:~/.../Task1$ md5sum out2_128bytes.bin
f64cb51a4ac60c8a197a59be5cd95cf2  out2_128bytes.bin
[06/22/21]seed@VM:~/.../Task1$
```

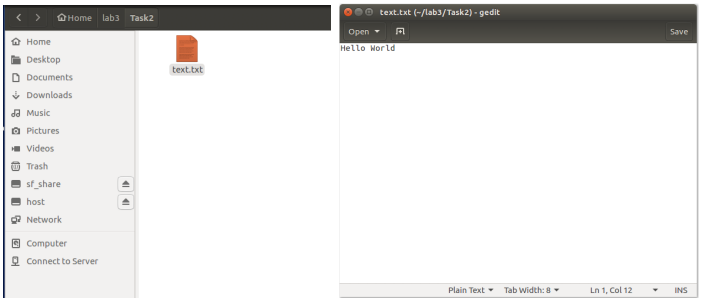
Using bless, we can check the content of “out1_128bytes.bin” and “out2_128bytes.bin”. As it turns out, the values are the same for both files. Moreover, as we can see there is no padding added after the end of the file content.



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.
Answer: No, but there are some differences that are shown in Bless between the 2 files. Those are located on bytes 147, 173, 187, 211, 237, and 251

Task 2: Understanding MD5’s Property

Let’s create a file called “text.txt” with the following content:



Next, we perform “md5collgen” command which allows us to provide a prefix file.

```
[06/22/21]seed@VM:~$ cd lab3/Task2
[06/22/21]seed@VM:~/.../Task2$ ls
text.txt
[06/22/21]seed@VM:~/.../Task2$ md5collgen -p text.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: 'text.txt'
Using initial value: bc699b8e62060bc2c27b6ab3cb5a91e8

Generating first block: .........
Generating second block: S11.............................................
...............................................
Running time: 11.6362 s
[06/22/21]seed@VM:~/.../Task2$
```

After, that we calculate the combined outputs using md5sum

```
[06/22/21]seed@VM:~/.../Task2$ md5sum out1.bin out2.bin
5dc76a6dbccd5d5c2d7790427f696b99  out1.bin
5dc76a6dbccd5d5c2d7790427f696b99  out2.bin
[06/22/21]seed@VM:~/.../Task2$
```

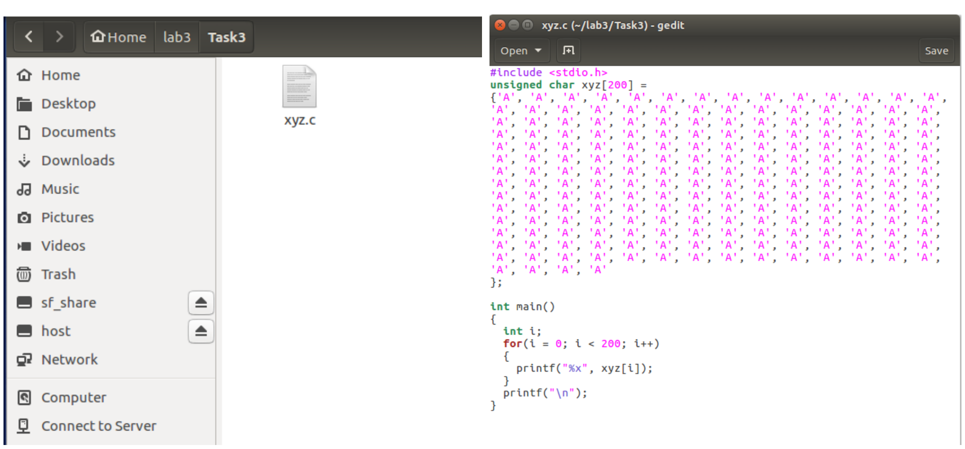
Next, we append the same message to the outputs and see the new hash values

```
[06/22/21]seed@VM:~/.../Task2$ echo "I like eating fish" > out1.bin
[06/22/21]seed@VM:~/.../Task2$ echo "I like eating fish" > out2.bin
[06/22/21]seed@VM:~/.../Task2$ md5sum out1.bin out2.bin
c70590ffbf8678e3f895ad43d6aae8e0  out1.bin
c70590ffbf8678e3f895ad43d6aae8e0  out2.bin
[06/22/21]seed@VM:~/.../Task2$
```

As we can see, the hash values changed from initial. However, when the new values of hash are compared to each other, they are the same.

Task 3: Generating Two Executable Files with The Same MD5 Hash

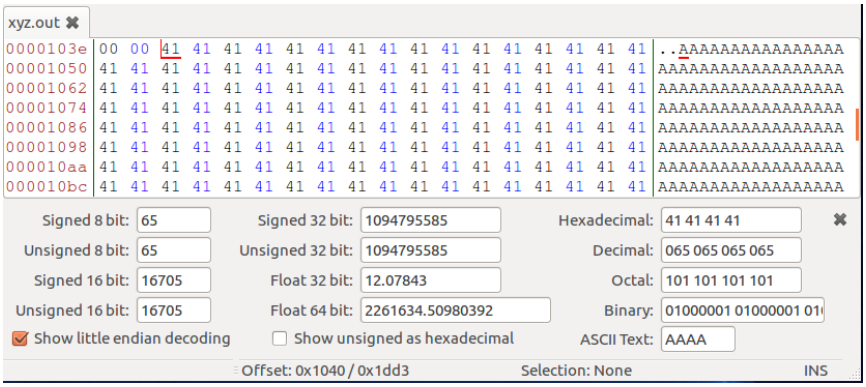
Let’s create file xyz.c with the following content:



To see the generated hex values of file “xyz.c” file, we use create a new file called “xyz.out” which will be our the first output

```
[06/25/21]seed@VM:~/.../Task3$ gcc xyz.c -o xyz.out
[06/25/21]seed@VM:~/.../Task3$ ls
xyz.c  xyz.out
```

Next, open up Bless to see the ASCII value for ‘A’. We notice that the first occurrence of ‘A’ is located at offset 0x1040, which is a hex value



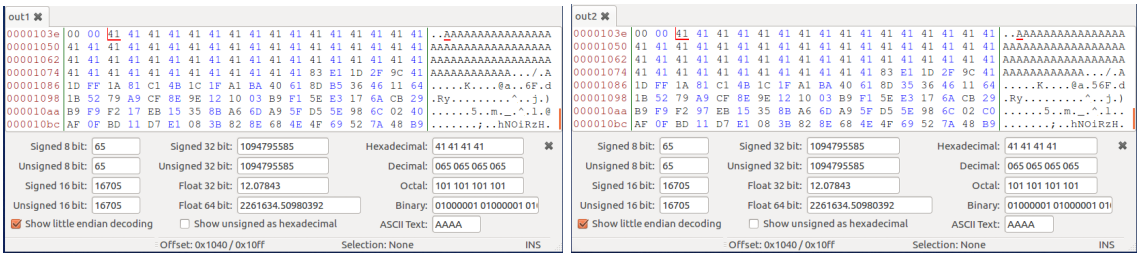
First, let’s create a prefix for this program. Since the first occurrence of ‘A’ happens at offset 0x1040, this means that 4160 bytes should be used as a prefix. Let’s add another 64 bytes so that the size of our prefix becomes 4224 bytes (or 0x1080 in hex). We will allocate the result in file “prefix”

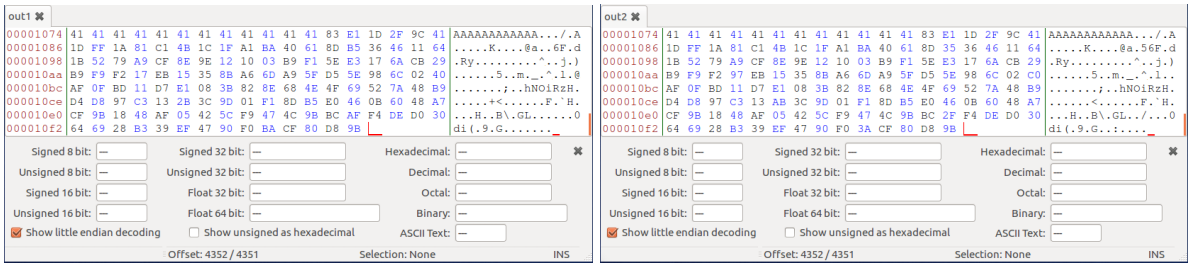
```
[06/25/21]seed@VM:~/.../Task3$ head -c 4224 xyz.out > prefix
```

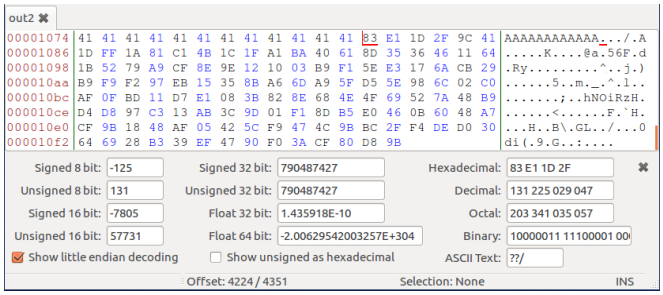
Second, we perform “md5collgen” command which allows us to provide a prefix file to new generated files. Let these generated files be called “out1” and “out2”.

```
[06/25/21]seed@VM:~/.../Task3$ md5collgen -p prefix -o out1 out2
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1' and 'out2'
Using prefixfile: 'prefix'
Using initial value: d00b2da83dd7fdd75e5f7d523d32732f

Generating first block: .
Generating second block: S10...
Running time: 1.18682 s
```

Let’s check the file content of both “out1” and “out2” files using bless







From the pictures above, we notice that the end of the file (EOF) occurs at offset 4352. Let's append the suffix (from bytes 4224 to 4352) to the file called "suffix"

```
[06/25/21]seed@VM:~/.../Task3$ tail -c 4352 xyz.out > suffix
```

After this, we append the same suffix our generated files "out1" and "out2"

```
[06/25/21]seed@VM:~/.../Task3$ cat suffix >> out1
[06/25/21]seed@VM:~/.../Task3$ cat suffix >> out2
```

Next, we need to give permission to execute files (i.e. to be able to write in them)

```
[06/25/21]seed@VM:~/.../Task3$ chmod +x out1
[06/25/21]seed@VM:~/.../Task3$ chmod +x out2
```

./out1

```
[06/25/21]seed@VM:~/.../Task3$ ./out1
414141414141414141414141414141414141414141414141414141414141414141414141
414141414141414141414141414141414141414141414141414183e11d2f9c411dff1a81c14b1c1fa1ba
40618db5364611641b5279a9cf8e9e12103b9f15ee3176acb29b9f9f217eb15358ba66da95fd55e9
86c240affbd11d7e183b828e684e4f69527a48b9d4d897c3132b3c9d1f18db5e046b6048a7cf9b18
48af5425cf9474c9bbcaff4ded030646928b339ef4790f0bacf80d89b00000000
```

./out2

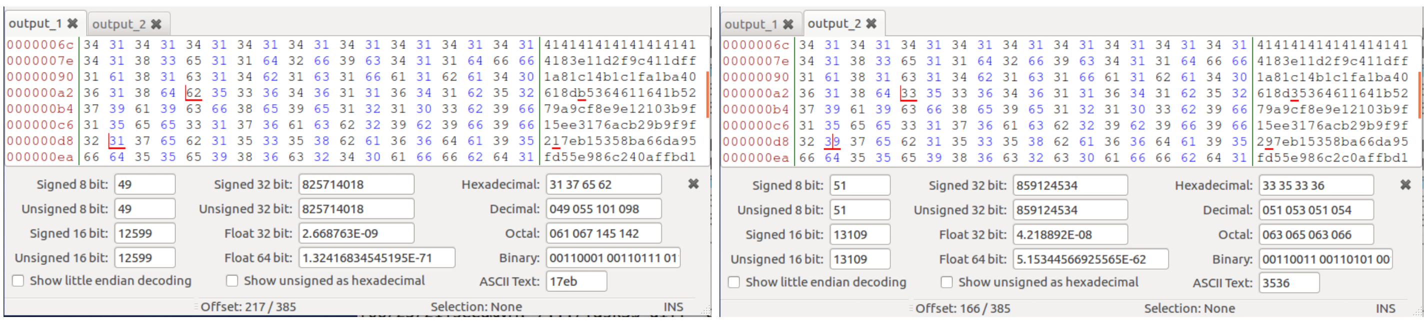
```
[06/25/21]seed@VM:~/.../Task3$ ./out2
414141414141414141414141414141414141414141414141414141414141414141414141
414141414141414141414141414141414141414141414141414183e11d2f9c411dff1a81c14b1c1fa1ba
40618d35364611641b5279a9cf8e9e12103b9f15ee3176acb29b9f9f297eb15358ba66da95fd55e9
86c2c0affbd11d7e183b828e684e4f69527a48b9d4d897c313ab3c9d1f18db5e046b6048a7cf9b18
48af5425cf9474c9bbc2ff4ded030646928b339ef4790f03acf80d89b00000000
```

Let's save these results of "out1" and "out2" into files "output_1" and "output_2" respectively

```
[06/25/21]seed@VM:~/.../Task3$ ./out1 > output_1
[06/25/21]seed@VM:~/.../Task3$ ./out2 > output_2
```

Then, we perform command "diff -q output_1 output_2"to see whether the files are different

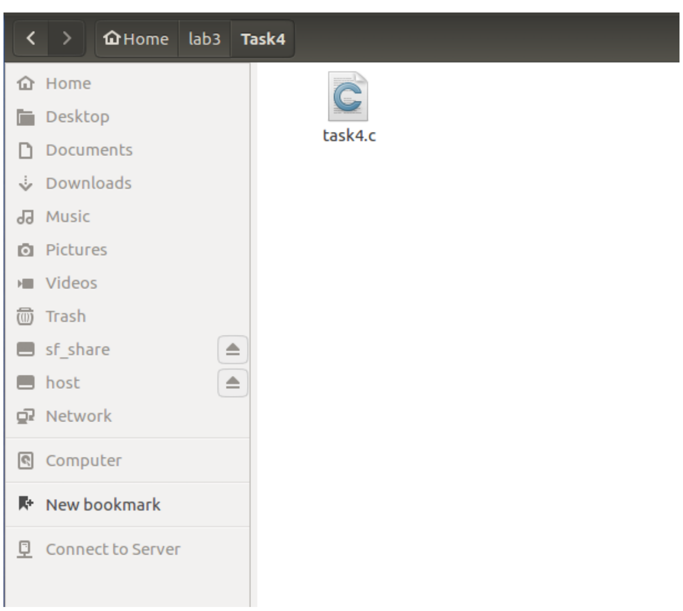
```
[06/25/21]seed@VM:~/.../Task3$ diff -q output_1 output_2
Files output_1 and output_2 differ
[06/25/21]seed@VM:~/.../Task3$ md5sum out1
3489426d131f6c8f7ac881d72636cbf2  out1
[06/25/21]seed@VM:~/.../Task3$ md5sum out2
3489426d131f6c8f7ac881d72636cbf2  out2
[06/25/21]seed@VM:~/.../Task3$
```



The top picture shows that the hash values of "out_1" and "out_2" are the same. However, the bottom two pictures of values inside "output_1" and "output_2" (which are the results of out_1 and out_2) show that there are some differentiations between each other.

Task 4: Making the Two Programs Behave Differently

Let's create a c-file called "task4.c"

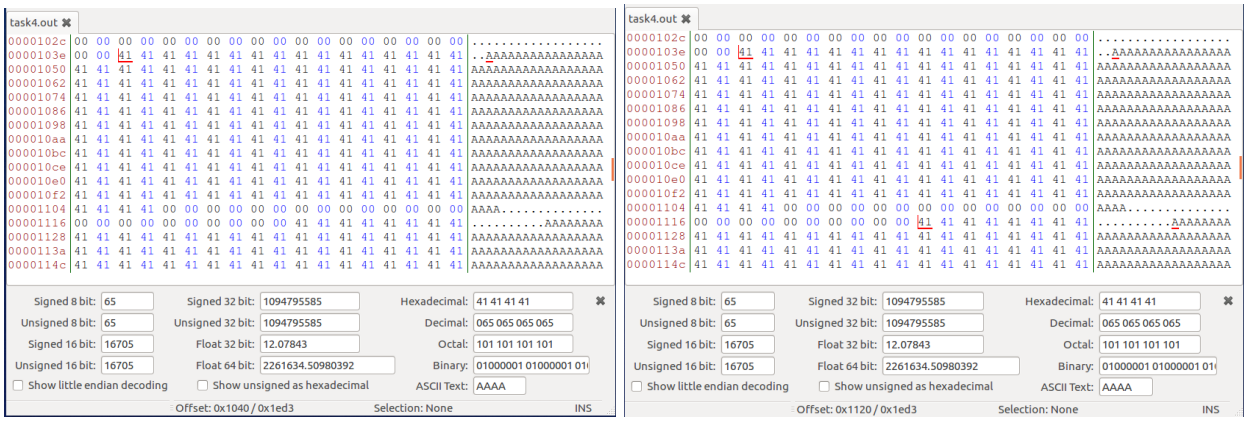



```
task4.c (~/lab3/Task4) - gedit
#include <stdio.h>
unsigned char bening[200] =
{'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A'
};
unsigned char malicious[200] =
{'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A'
};

int main()
{
  int i;
  int compared_values = 1;
  for(i = 0; i < 200; i++)
  {
    if(bening[i] == malicious[i])
    {
      compared_values = 0;
    }
  }
  if(compared_values)
  {
    printf("Run bening program...");
  }
  else
  {
    printf("Run malicious program...");
  }
  printf("\n");
}
```

To see the generated hex values of file“task4.c” file, we use create a new file called “task4.out” which will be our the first output

```
[06/26/21]seed@VM:~/.../Task4$ gcc task4.c -o task4.out
```

Next, open up Bless to see the ASCII value for ‘A’. We notice that the first occurrence of ‘A’ is located at offset 0x1040, which is a hex value. The second occurrence of ‘A’ happens at offset 0x1120, which is also a hex value



First, let’s create a prefix for the first array. Since the first occurrence of ‘A’ happens at offset 0x1040, this means that 4160 bytes should be used as a prefix. Let’s add another 64 bytes so that the size of our prefix becomes 4224 bytes (or 0x1080 in hex). We will allocate the result in file “prefix”

```
[06/26/21]seed@VM:~/.../Task4$ head -c 4224 task4.out > prefix
```

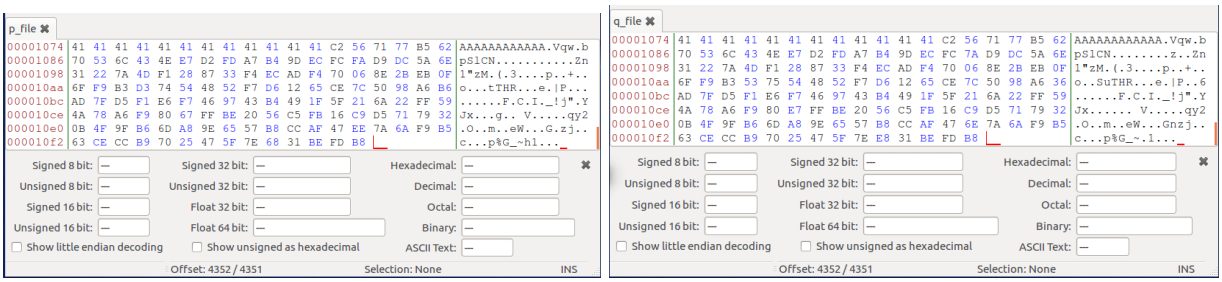
Second, we perform “md5collgen” command which allows us to provide a prefix file to new generated files. Let these generated files be called “p_file” and “q_file”.

```
[06/26/21]seed@VM:~/.../Task4$ md5collgen -p prefix -o p_file q_file
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'p_file' and 'q_file'
Using prefixfile: 'prefix'
Using initial value: 7bd7e793fdcd3ee32046b97e7bcff755

Generating first block: ..............................
Generating second block: S00........
Running time: 21.1749 s
```

We can check the MD5 hash of combined outputs for “p_file” and “q_file”. In fact, they are the same

```
[06/26/21]seed@VM:~/.../Task4$ md5sum p_file q_file
85ddd075ce101975055704d067add923  p_file
85ddd075ce101975055704d067add923  q_file
```

Let's check the file content of both "p_file" and "q_file" files using bless. As we can see, the EOF is at offset 4352.



Next, we generate 2 files using "prefix" and add all bytes after 4352 in task4.out to end

```
[06/26/21]seed@VM:~/.../Task4$ head -c +4353 task4.out > end
```

Let's add the first 8 bytes to the EOF for both "p_file" and "q_file" . This can be done by creating the temporary files for them that contain their first 8 bytes. We can create a "suffix" that will contain the remaining content after the 8th byte.

```
[06/26/21]seed@VM:~/.../Task4$ head -c 8 end > temp_arr
[06/26/21]seed@VM:~/.../Task4$ cat p_file temp_arr > temp_out1
[06/26/21]seed@VM:~/.../Task4$ cat q_file temp_arr > temp_out2
[06/26/21]seed@VM:~/.../Task4$ tail -c +9 end > suffix
```

The next step is to create a file called "**temp_suffix**". This file will include the bytes that are located between the end of the first array and the beginning of the second array. Then, we need to store the bytes beginning in the second array in "**suffix**" to "**end**". Finally, we add these bytes to "**temp_out1**" and "**temp_out2**" to new files called "**file1**" and "**file2**"

```
[06/26/21]seed@VM:~/.../Task4$ tail -c +25 suffix > end
[06/26/21]seed@VM:~/.../Task4$ head -c 24 suffix > temp_suffix
```

```
[06/26/21]seed@VM:~/.../Task4$ cat temp_out1 temp_suffix > file1
[06/26/21]seed@VM:~/.../Task4$ cat temp_out2 temp_suffix > file2
```

Now the two result files become the two separate part executables, which have the contents up to the beginning of the second array. To demonstrate the difference between the two programs, the good program prints out "Run the bening program...", and the bad program prints out "Run the malicious program...". To do this, the values of the second array need to be equal to one of the generated arrays.
In order to do this, we replace the bytes of array size (200) after the second array from "**end**" to "**suffix**". Then, we copy the first array from "**temp_out1**" (starting from EOF) to a new file called "**middle**". Finally, the file "**middle**" is appended to "**file1**" and "**file2**" along with suffix to give the final executables "**bening_program**" and "**malicioius_program**".

```
[06/26/21]seed@VM:~/.../Task4$ tail -c +201 end > suffix
[06/26/21]seed@VM:~/.../Task4$ tail -c +4161 temp_out1 > middle
[06/26/21]seed@VM:~/.../Task4$ cat file1 middle suffix > bening_program
[06/26/21]seed@VM:~/.../Task4$ cat file2 middle suffix > malicious_program
```

Next, we check for the created files hash using md5sum. As it turns out, both programs have the same hash value, which means that the malicious program can be certified.

```
[06/26/21]seed@VM:~/.../Task4$ md5sum bening_program
50e4830d648237ff8eb4bcf5051c4f23  bening_program
[06/26/21]seed@VM:~/.../Task4$ md5sum malicious_program
50e4830d648237ff8eb4bcf5051c4f23  malicious_program
```

After that, we need to give permission to execute files (i.e. to be able to write in them)

```
[06/26/21]seed@VM:~/.../Task4$ chmod +x bening_program
[06/26/21]seed@VM:~/.../Task4$ chmod +x malicious_program
```

The results: (./bening_program and ./malicious_program)

```
[06/26/21]seed@VM:~/.../Task4$ ./bening_program
Run malicious program...
[06/26/21]seed@VM:~/.../Task4$ ./malicious_program
Run malicious program...
```