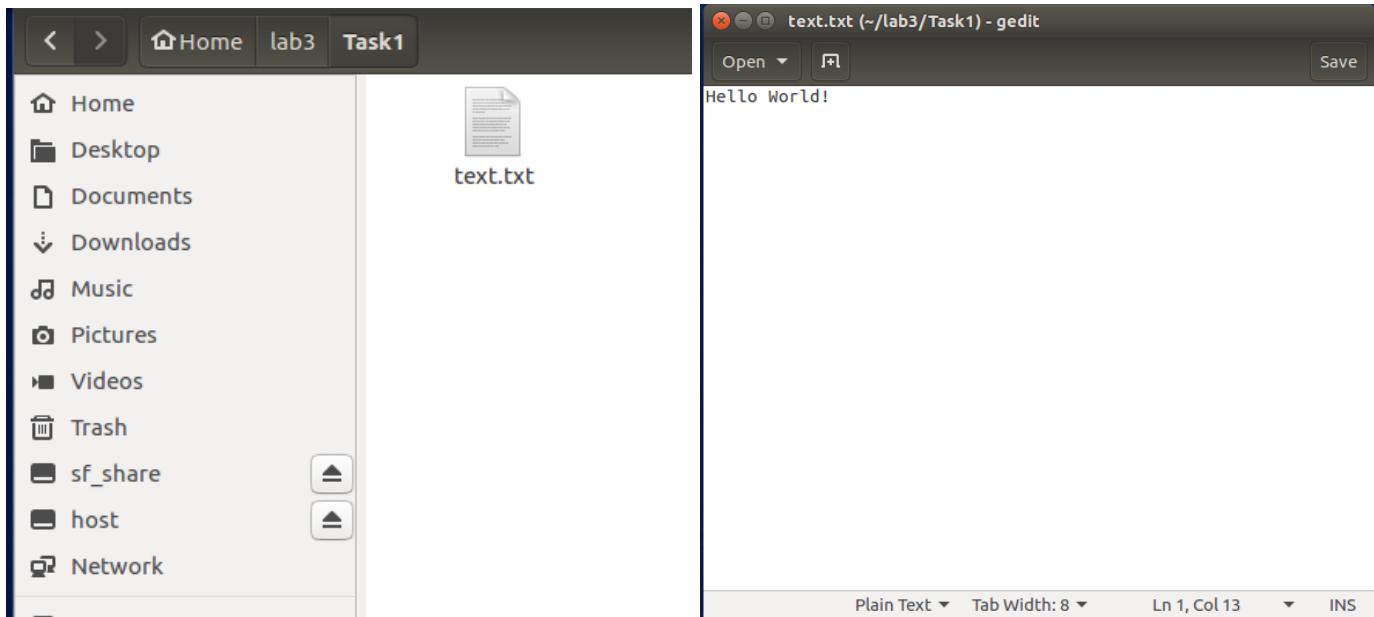


Task 1: Generating Two Different Files with the Same MD5 Hash

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



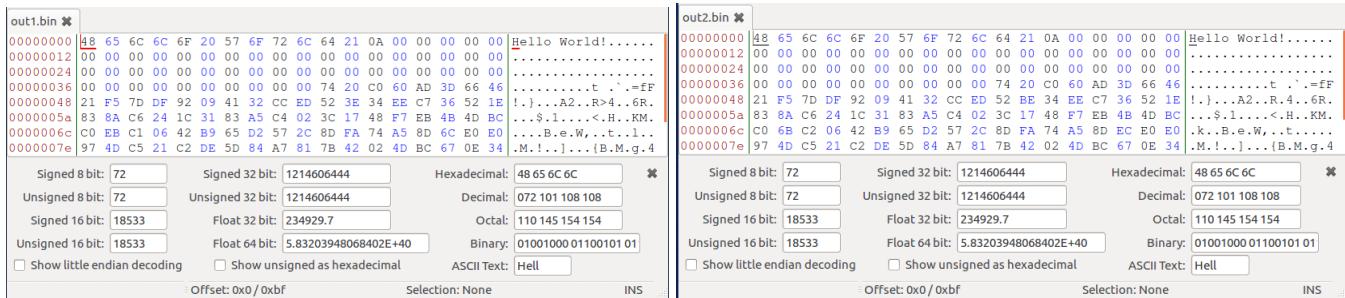
Next, we perform "md5collgen" command which allows us to provide a prefix file.

```
[06/20/21]seed@VM:~/.../Task1$ 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$
```

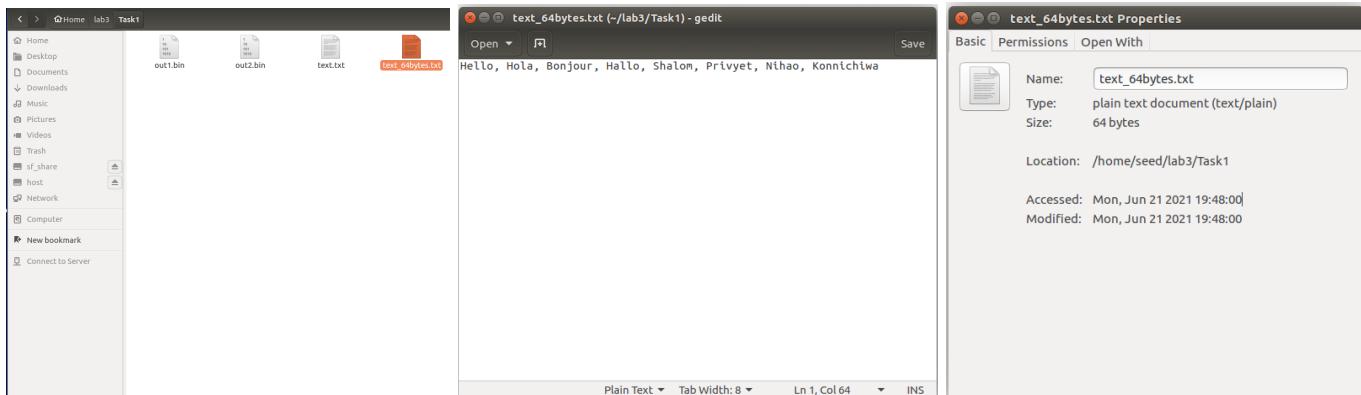
Using bless, we can check the content of “out1.bin” and “out2.bin”. As it turns out, the values are the same for both files



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/)

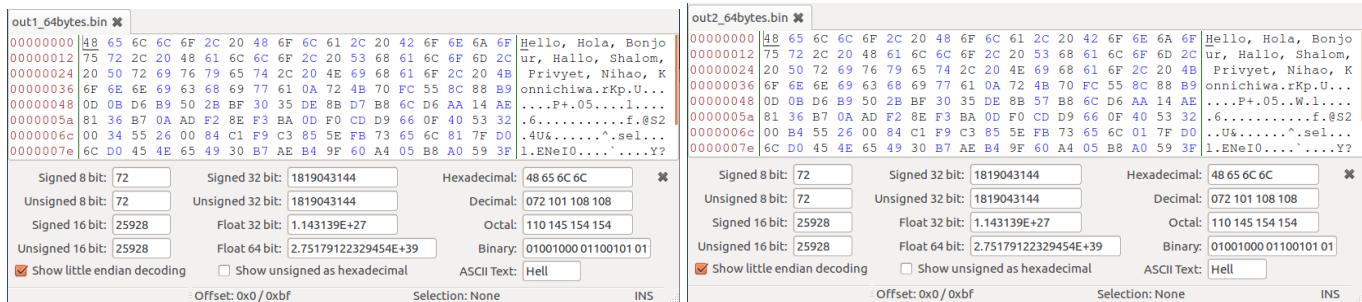
Using output filenames: 'out1_64bytes.bin' and 'out2_64bytes.bin'
Using prefixfile: 'text_64bytes.txt'
Using initial value: d26c7c12c989fb19c5149e69de930016

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$
```

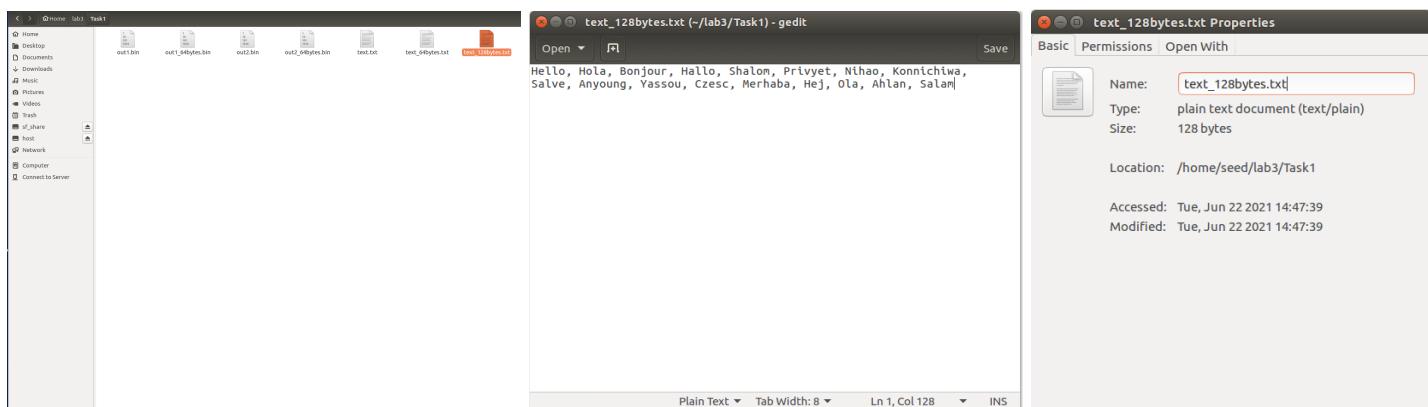
Using bless, we can check the content of “out1_64bytes.bin” and “out2_64bytes.bin”. As it turns out, the values are the same for both files. Moreover, as we can see there is no padding added after the end of the file content.



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.

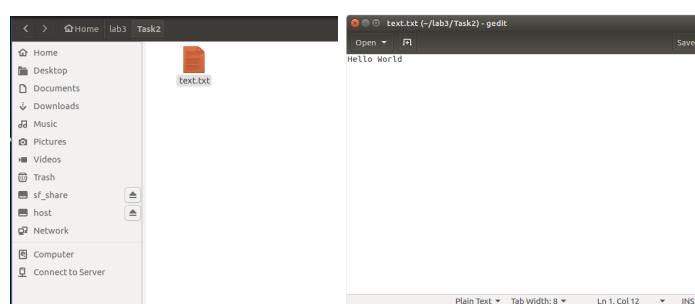
out1_128bytes.bin										out2_128bytes.bin										
<input type="checkbox"/> Show little endian decoding					<input type="checkbox"/> Show unsigned as hexadecimal					<input type="checkbox"/> Show little endian decoding					<input type="checkbox"/> Show unsigned as hexadecimal					
00000000	48	65	6C	6C	6F	2C	20	48	6F	6C	61	2C	20	42	6F	6E	6A	6F	Hello, Hola, Bonjo	
00000012	75	72	2C	20	48	61	6C	6C	6F	2C	20	53	68	61	6C	6F	6D	2C	ur, Hallo, Shalom,	
00000024	20	50	72	69	76	79	65	74	2C	20	4E	69	68	61	2C	20	4B	Privyet, Nihao, K		
00000036	6F	6E	6B	69	63	68	69	77	61	2C	20	53	61	6F	76	65	2C	20	onnichiwa, Salve,	
00000048	41	79	6F	75	66	72	67	2C	20	59	61	73	73	6F	7C	20	43	Anyoung, Yassou, C		
0000005A	7A	65	73	63	2C	20	4D	65	72	68	61	62	61	2C	20	48	65	6A	zesc, Merhaba, Hej	
0000006C	20	24	4F	6C	61	2C	20	41	68	6C	61	6E	2C	20	53	61	6C	61	, Ola, Ahian, Sala	
0000007E	6D	0A	C3	E0	D3	F5	B6	C6	D8	23	FB	FE	59	D4	42	7A	DE	E6	m.....#..Y.Bz..	
00000090	FA	B8	6D	94	8F	89	EA	17	D6	83	9E	6C	F7	2C	0A	93	F4d.....l.....		
000000A2	F1	CA	6E	66	AA	C9	EB	DF	4A	3B	DF	51	5C	6E	3A	96	68	28	.nf....J;.\n:h(h	
000000B4	49	25	2C	5D	8A	84	4C	4F	8E	E2	40	09	06	4C	0C	79	DE	K.%...LO...@.L.y.		
000000C6	40	F7	B1	2E	64	65	A1	11	75	3B	D3	BD	8C	50	78	52	20	@...de..u...P!X"		
000000D8	2B	C3	10	BF	9B	5F	48	D7	CB	AD	60	38	97	44	B1	53	76	45	+....H...`8.D.SvE	
000000EA	48	7B	AB	19	6B	E3	A9	A8	EA	5B	C7	B7	AC	23	4E	0B	C2	H{.s.k.{[...#N..		
000000FC	2F	86	00	1E															/...	
Signed 8 bit:	72	Signed 32 bit:	1819043144	Hexadecimal:	48 65 6C 6C	X	Signed 8 bit:	72	Signed 32 bit:	1819043144	Hexadecimal:	48 65 6C 6C	X	Signed 8 bit:	72	Signed 32 bit:	1819043144	Hexadecimal:	48 65 6C 6C	X
Unsigned 8 bit:	72	Unsigned 32 bit:	1819043144	Decimal:	072 101 108 108		Unsigned 8 bit:	72	Unsigned 32 bit:	1819043144	Decimal:	072 101 108 108		Unsigned 8 bit:	72	Unsigned 32 bit:	1819043144	Decimal:	072 101 108 108	
Signed 16 bit:	25928	Float 32 bit:	1.143139E+27	Octal:	110 145 154 154		Signed 16 bit:	25928	Float 32 bit:	1.143139E+27	Octal:	110 145 154 154		Signed 16 bit:	25928	Float 32 bit:	1.143139E+27	Octal:	110 145 154 154	
Unsigned 16 bit:	25928	Float 64 bit:	2.75179122329454E+39	Binary:	01001000 01100101 01		Unsigned 16 bit:	25928	Float 64 bit:	2.75179122329454E+39	Binary:	01001000 01100101 01		Unsigned 16 bit:	25928	Float 64 bit:	2.75179122329454E+39	Binary:	01001000 01100101 01	
<input type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal		ASCII Text:	Hell		<input type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal		ASCII Text:	Hell		<input type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal		ASCII Text:	Hell	
Offset: 0x0 / 0xff		Selection: None		INS			Offset: 0x0 / 0xff		Selection: None		INS			Offset: 0x0 / 0xff		Selection: None		INS		
out1_128bytes.bin										out2_128bytes.bin										
00000000	6D	0A	C3	E0	D3	F5	BB	C6	D8	23	FB	FE	59	D4	42	7A	DE	E6	m.....#..Y.Bz..	
00000090	FA	B8	BD	64	9C	8F	89	EA	17	D6	83	9E	6C	F7	2C	0A	93	F4d.....l.....	
000000A2	F1	CA	6E	66	AA	C9	EB	DF	4A	3B	DF	51	5C	6E	3A	96	68	28	.nf....J;.\n:h(h	
000000B4	49	25	2C	5D	8A	84	4C	4F	8E	E2	40	09	06	4C	0C	79	DE	K.%...LO...@.L.y.		
000000C6	40	F7	B1	2E	64	65	A1	11	75	3B	D3	BD	8C	50	78	52	20	@...de..u...P!X"		
000000D8	2B	C3	10	BF	9B	5F	48	D7	CB	AD	60	38	97	44	B1	53	76	45	+....H...`8.D.SvE	
000000EA	48	7B	AB	19	6B	E3	A9	A8	EA	5B	C7	B7	AC	23	4E	0B	C2	H{.s.k.{[...#N..		
000000FC	2F	86	00	1E															/...	
Signed 8 bit:	72	Signed 32 bit:	1819043144	Hexadecimal:	48 65 6C 6C	X	Signed 8 bit:	72	Signed 32 bit:	1819043144	Hexadecimal:	48 65 6C 6C	X	Signed 8 bit:	72	Signed 32 bit:	1819043144	Hexadecimal:	48 65 6C 6C	X
Unsigned 8 bit:	72	Unsigned 32 bit:	1819043144	Decimal:	072 101 108 108		Unsigned 8 bit:	72	Unsigned 32 bit:	1819043144	Decimal:	072 101 108 108		Unsigned 8 bit:	72	Unsigned 32 bit:	1819043144	Decimal:	072 101 108 108	
Signed 16 bit:	25928	Float 32 bit:	1.143139E+27	Octal:	110 145 154 154		Signed 16 bit:	25928	Float 32 bit:	1.143139E+27	Octal:	110 145 154 154		Signed 16 bit:	25928	Float 32 bit:	1.143139E+27	Octal:	110 145 154 154	
Unsigned 16 bit:	25928	Float 64 bit:	2.75179122329454E+39	Binary:	01001000 01100101 01		Unsigned 16 bit:	25928	Float 64 bit:	2.75179122329454E+39	Binary:	01001000 01100101 01		Unsigned 16 bit:	25928	Float 64 bit:	2.75179122329454E+39	Binary:	01001000 01100101 01	
<input type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal		ASCII Text:	Hell		<input type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal		ASCII Text:	Hell		<input type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal		ASCII Text:	Hell	
Offset: 0x0 / 0xff		Selection: None		INS			Offset: 0x0 / 0xff		Selection: None		INS			Offset: 0x0 / 0xff		Selection: None		INS		
out1_128bytes.bin										out2_128bytes.bin										
00000000	6D	0A	C3	E0	D3	F5	BB	C6	D8	23	FB	FE	59	D4	42	7A	DE	E6	m.....#..Y.Bz..	
00000090	FA	B8	BD	64	9C	8F	89	EA	17	D6	83	9E	6C	F7	2C	0A	93	F4d.....l.....	
000000A2	F1	CA	6E	66	AA	C9	EB	DF	4A	3B	DF	51	5C	6E	3A	96	68	28	.nf....J;.\n:h(h	
000000B4	49	25	2C	5D	8A	84	4C	4F	8E	E2	40	09	06	4C	0C	79	DE	K.%...LO...@.L.y.		
000000C6	40	F7	B1	2E	64	65	A1	11	75	3B	D3	BD	8C	50	78	52	20	@...de..u...P!X"		
000000D8	2B	C3	10	BF	9B	5F	48	D7	CB	AD	60	38	97	44	B1	53	76	45	+....H...`8.D.SvE	
000000EA	48	7B	AB	19	6B	E3	A9	A8	EA	5B	C7	B7	AC	23	4E	0B	C2	H{.s.k.{[...#N..		
000000FC	2F	86	00	1E															/...	
Signed 8 bit:	-62	Signed 32 bit:	8794050	Hexadecimal:	C2 F8 60	X	Signed 8 bit:	66	Signed 32 bit:	8793922	Hexadecimal:	42 F8 60	X	Signed 8 bit:	66	Signed 32 bit:	8793922	Hexadecimal:	42 F8 60	X
Unsigned 8 bit:	194	Unsigned 32 bit:	8794050	Decimal:	194 047 134 000		Unsigned 8 bit:	66	Unsigned 32 bit:	8793922	Decimal:	066 047 134 000		Unsigned 8 bit:	66	Unsigned 32 bit:	8793922	Decimal:	066 047 134 000	
Signed 16 bit:	12226	Float 32 bit:	1.232309E-38	Octal:	302 057 206 000		Signed 16 bit:	12098	Float 32 bit:	1.232291E-38	Octal:	102 057 206 000		Signed 16 bit:	12098	Float 32 bit:	1.232291E-38	Octal:	102 057 206 000	
Unsigned 16 bit:	12226	Float 64 bit:	—	Binary:	11000010 00101111 101		Unsigned 16 bit:	12098	Float 64 bit:	—	Binary:	01000010 00101111 101		Unsigned 16 bit:	12098	Float 64 bit:	—	Binary:	01000010 00101111 101	
<input type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal		ASCII Text:	?/		<input type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal		ASCII Text:	B/?		<input type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal		ASCII Text:	B/?	
Offset: 251 / 255		Selection: None		INS			Offset: 251 / 255		Selection: None		INS			Offset: 251 / 255		Selection: None		INS		

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  
c70590ffb8678e3f895ad43d6aae8e0  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

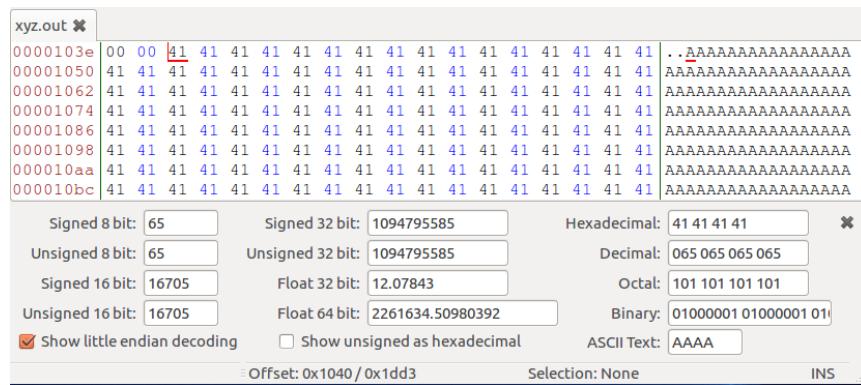
Let's create file xyz.c with the following content:



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



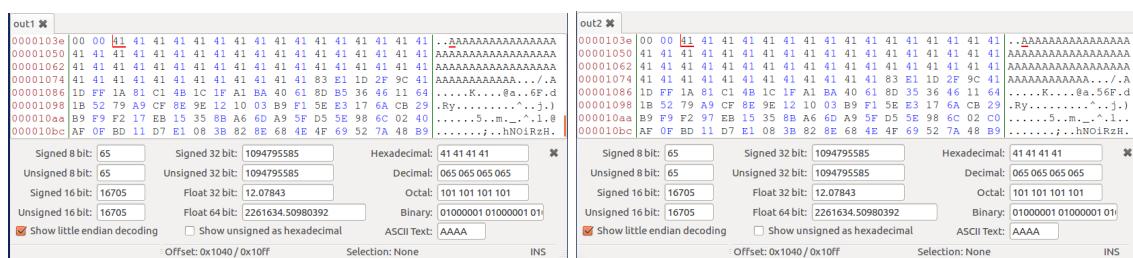
First, let’s create a prefix for this program. Since the first occurrence of ‘A’ happens at offset 0x1040, this means that 4160 bytes should be used as a prefix. Let’s add another 64 bytes so that the size of our prefix becomes 4224 bytes (or 0x1080 in hex). We will allocate the result in file “prefix”

```
[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: $10...
Running time: 1.18682 s
```

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



out1 #		out2 #			
00001074	41 41 41 41 41 41 41 41 41 41 83 E1 1D 2F 9C 41	AAAAAAAAAAA.../.A	00001074	41 41 41 41 41 41 41 41 41 41 83 E1 1D 2F 9C 41	AAAAAAAAAAA.../.A
00001086	1D FF 1A 81 C1 4B 1C 1F A1 BA 40 61 8D B5 36 46 11 64K....@.6F.d	00001086	1D FF 1A 81 C1 4B 1C 1F A1 BA 40 61 8D 35 36 46 11 64K....@.56F.d
00001098	1B 52 79 A9 CF 8E 9E 12 10 03 B9 F1 E3 17 6A CB 29	Ry.....^..j.	00001098	1B 52 79 A9 CF 8E 9E 12 10 03 B9 F1 E3 17 6A CB 29	Ry.....^..j.
000010aa	B9 F9 F2 17 EB 15 35 BB A6 6D A9 F5 D9 SE 9E 62 02 C05.m_^.^..1.0	000010aa	B9 F9 F2 97 EB 15 35 BB A6 6D A9 F5 D9 9E 62 02 C05.m_^.^..1.
000010bc	AF 0F BD 11 DT 01 08 3B 82 8E 68 4F 69 52 7A 48 B9,hNO1RzH.	000010bc	AF 0F BD 11 DT 01 08 3B 82 8E 68 4F 69 52 7A 48 B9,hNO1RzH.
000010ce	D4 D8 97 C3 13 2B 3C 9D 01 F1 8B 85 BD E6 08 46 40 A7	<....<....F'..H.	000010ce	D4 D8 97 C3 13 AB 3C 9D 01 F1 BD B5 E6 08 60 48 A7	<....<....F'..H.
000010e0	CF 9B 18 48 AF 05 42 5C F9 47 4C 98 BC AF F4 DE DO 30H..B\GL.../..0	000010e0	CF 9B 18 48 AF 05 42 5C F9 47 4C 98 BC 2F F4 DE DO 30H..B\GL.../..0
000010f2	64 69 28 B3 39 EF 47 90 F0 BA CF 80 DB 9B	di(.9.G....)	000010f2	64 69 28 B3 39 EF 47 90 F0 BA CF 80 DB 9B	di(.9.G....)
Signed 8 bit:	—	Signed 32 bit:	—	Hexadecimal:	—
Unsigned 8 bit:	—	Unsigned 32 bit:	—	Decimal:	—
Signed 16 bit:	—	Float 32 bit:	—	Octal:	—
Unsigned 16 bit:	—	Float 64 bit:	—	Binary:	—
<input checked="" type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal	ASCII Text:	—	
Offset: 4352 / 4351	Selection: None	INS	Offset: 4352 / 4351	Selection: None	INS

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

./out2

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"

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

Signed 8 bit: 65	Signed 32 bit: 1094795585	Hexadecimal: 41 41 41 41
Unsigned 8 bit: 65	Unsigned 32 bit: 1094795585	Decimal: 065 065 065 065
Signed 16 bit: 16705	Float 32 bit: 12.07843	Octal: 101 101 101 101
Unsigned 16 bit: 16705	Float 64 bit: 2261634.50980392	Binary: 01000001 01000001 01
<input type="checkbox"/> Show little endian decoding	<input type="checkbox"/> Show unsigned as hexadecimal	ASCII Text: AAAA
Offset: 0x1040 / 0x1ed3 Selection: None INS		

Signed 8 bit: 65	Signed 32 bit: 1094795585	Hexadecimal: 41 41 41 41
Unsigned 8 bit: 65	Unsigned 32 bit: 1094795585	Decimal: 065 065 065 065
Signed 16 bit: 16705	Float 32 bit: 12.07843	Octal: 101 101 101 101
Unsigned 16 bit: 16705	Float 64 bit: 2261634.50980392	Binary: 01000001 01000001 01
<input type="checkbox"/> Show little endian decoding	<input type="checkbox"/> Show unsigned as hexadecimal	ASCII Text: AAAA
Offset: 0x1120 / 0x1ed3 Selection: None INS		

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/)

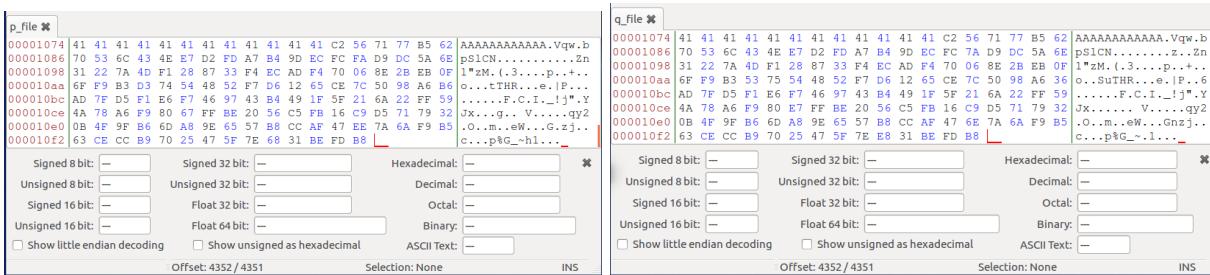
Using output filenames: 'p_file' and 'q_file'
Using prefixfile: 'prefix'
Using initial value: 7bd7e793fdcd3ee32046b97e7bcff755

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 benign 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 > benign_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 benign_program  
50e4830d648237ff8eb4bcf5051c4f23 benign_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 benign_program  
[06/26/21]seed@VM:~/.../Task4$ chmod +x malicious_program
```

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

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