Information Network Security

Lab Report

Lab 3: Symmetric encryption hashing

by

Md.Rasel Mahmud
2019831061

Submitted to

Partha Protim Paul
lecturer,IICT
Shahjalal University of Science and Technology, Sylhet

Date (6 May,2024)

task1:AES encryption using different modes

Algorith:AES-128-CBC

Encryption Command:

```
openssl enc -aes-128-cbc -e -in task1.txt -out aes128cbc.bin -K 00112233445
```

Decryption Command:

```
openssl enc -aes-128-cbc -d -in aes128cbc.bin -out aes128cbcdecrypted.txt -
```

Algorith:AES-128-CFB

Encryption Command:

```
openssl enc -aes-128-cfb -e -in task1.txt -out aes128cfb.bin -K 00112233445
```

Decryption Command:

```
openssl enc -aes-128-cfb -d -in aes128cfb.bin -out aes128cfbdycrypted.txt -
```

Algorith:AES-128-ECB

Encryption Command:

```
openssl enc -aes-128-ecb -e -in task1.txt -out aes128ecb.bin -K 00112233445
```

Decryption Command:

```
openssl enc -aes-128-ecb -d -in aes128ecb.bin -out aes128ecbdecrypted.txt -
```

Algorith:AES-256-CBC

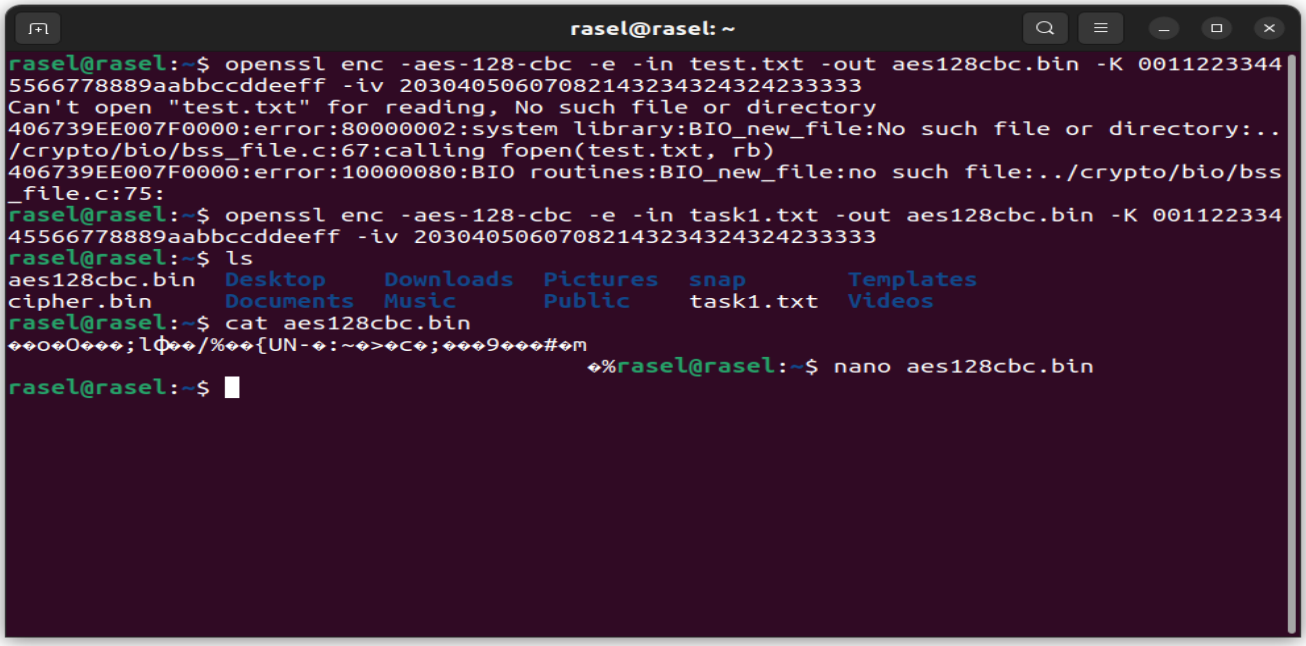
Encryption Command:

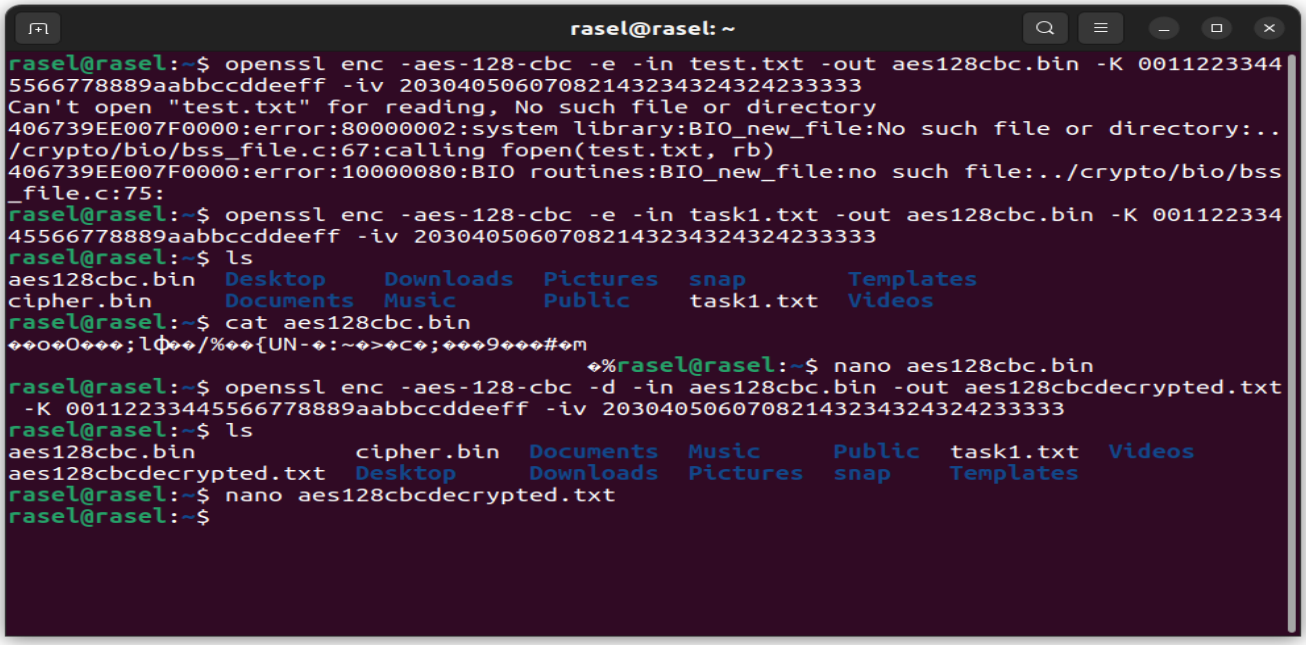
```
openssl enc -aes-256-cbc -e -in task1.txt -out aes256cbc.bin -K 2222233332
```

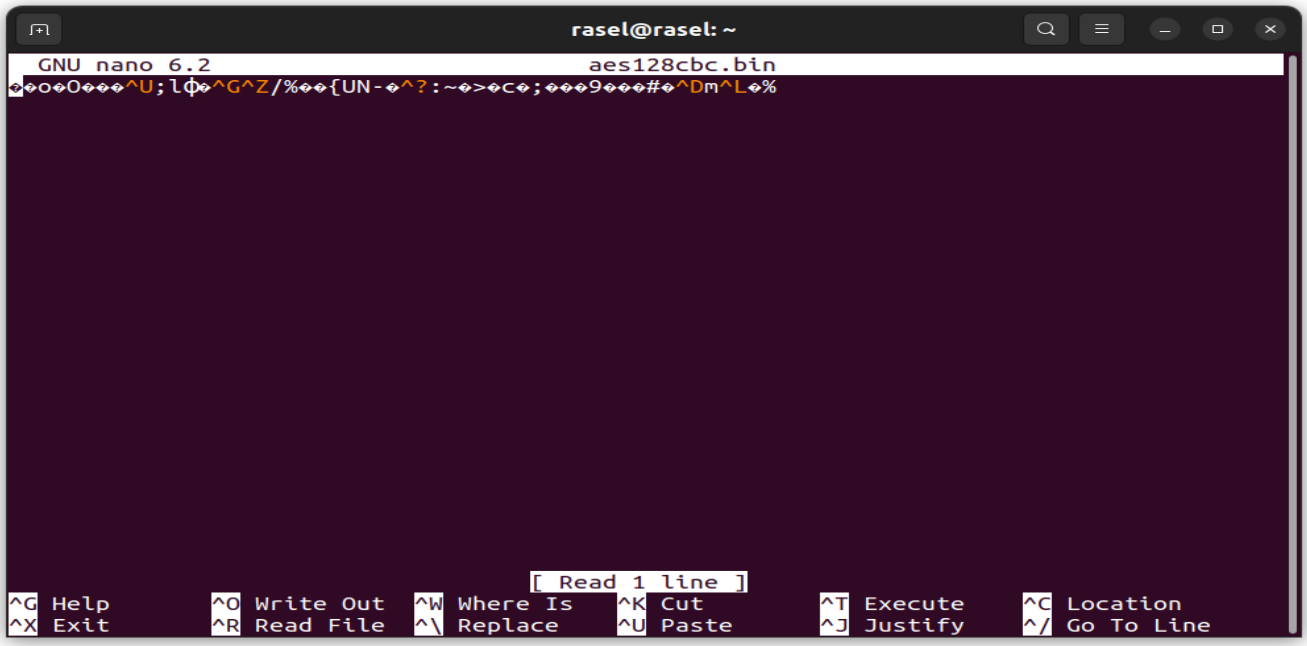
Decryption Command:

```
openssl enc -aes-256-cbc -d -in aes256cbc.bin -out aes256cbcdecrypted.txt -
```

Command ScreenShot



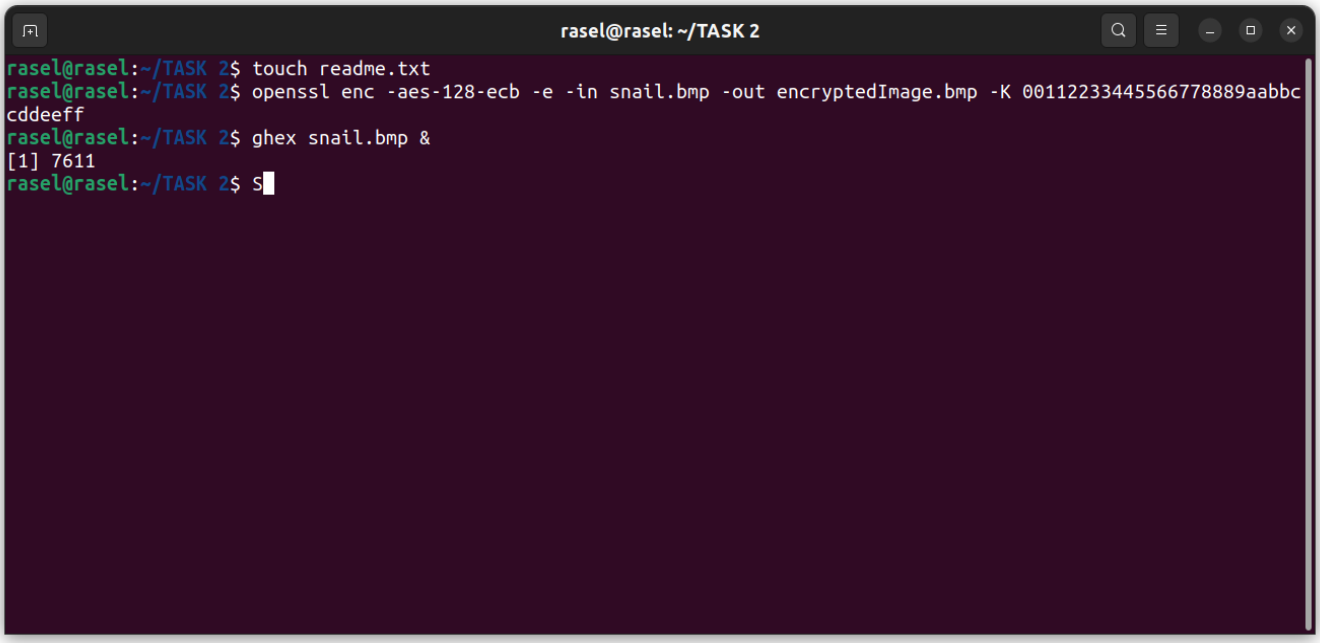


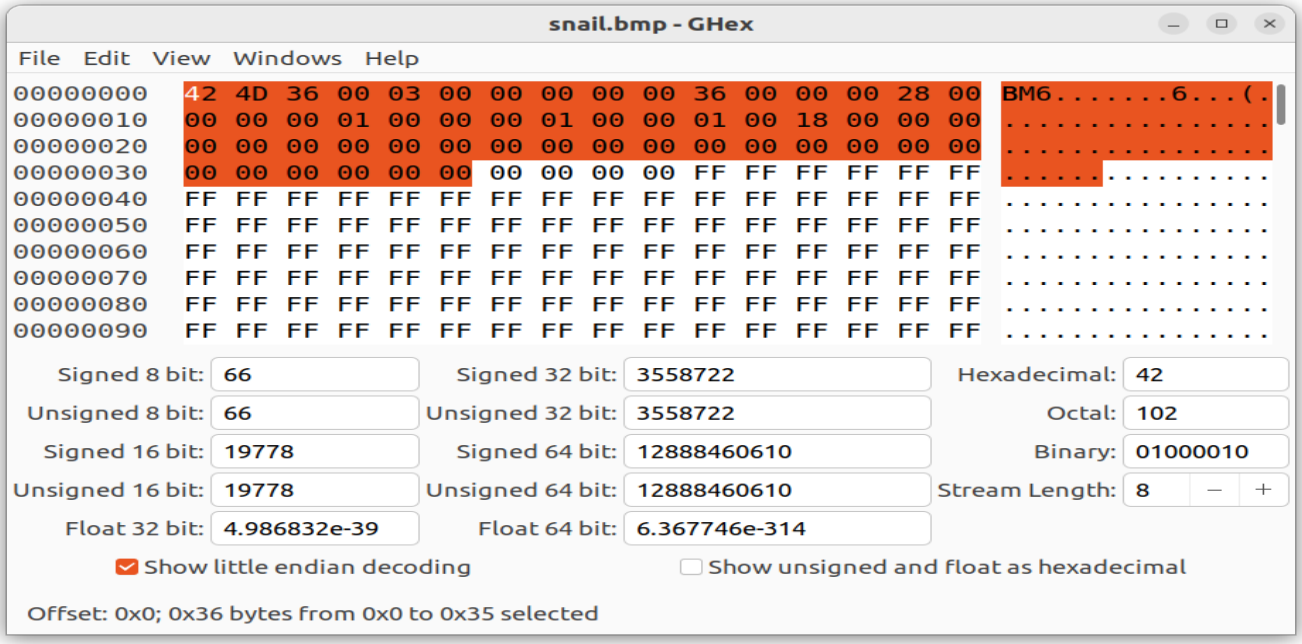


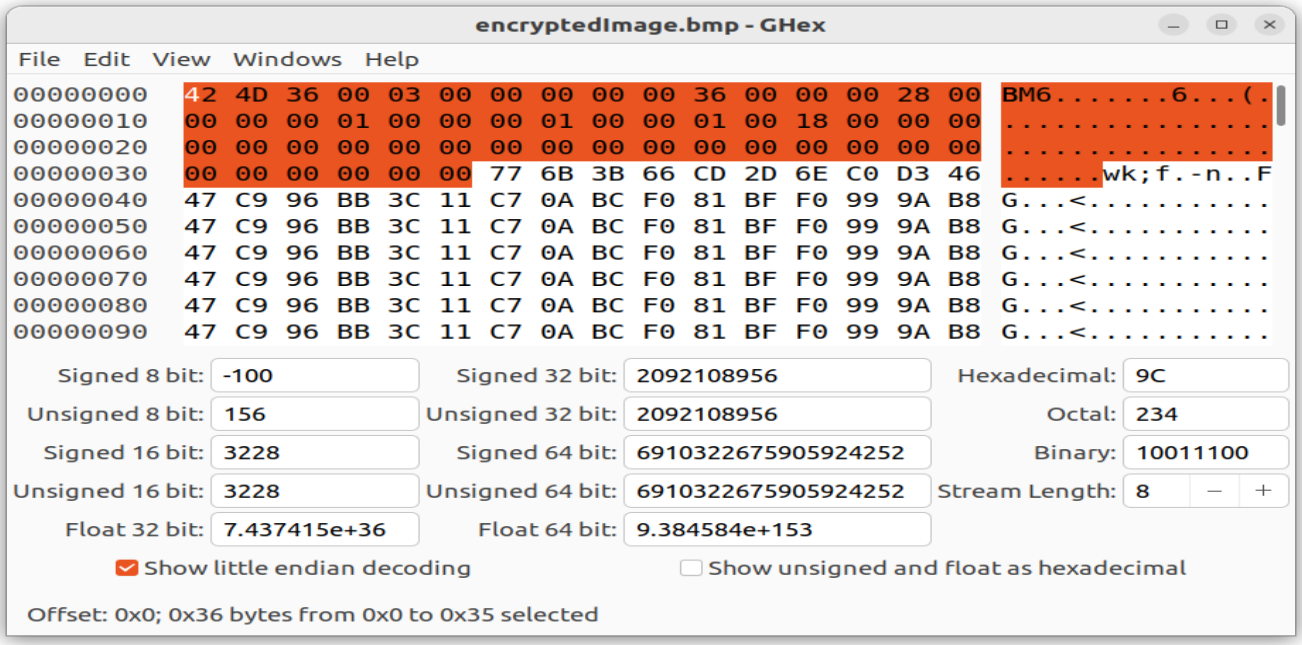
Task2: Encryption mode - ECB vs CBC

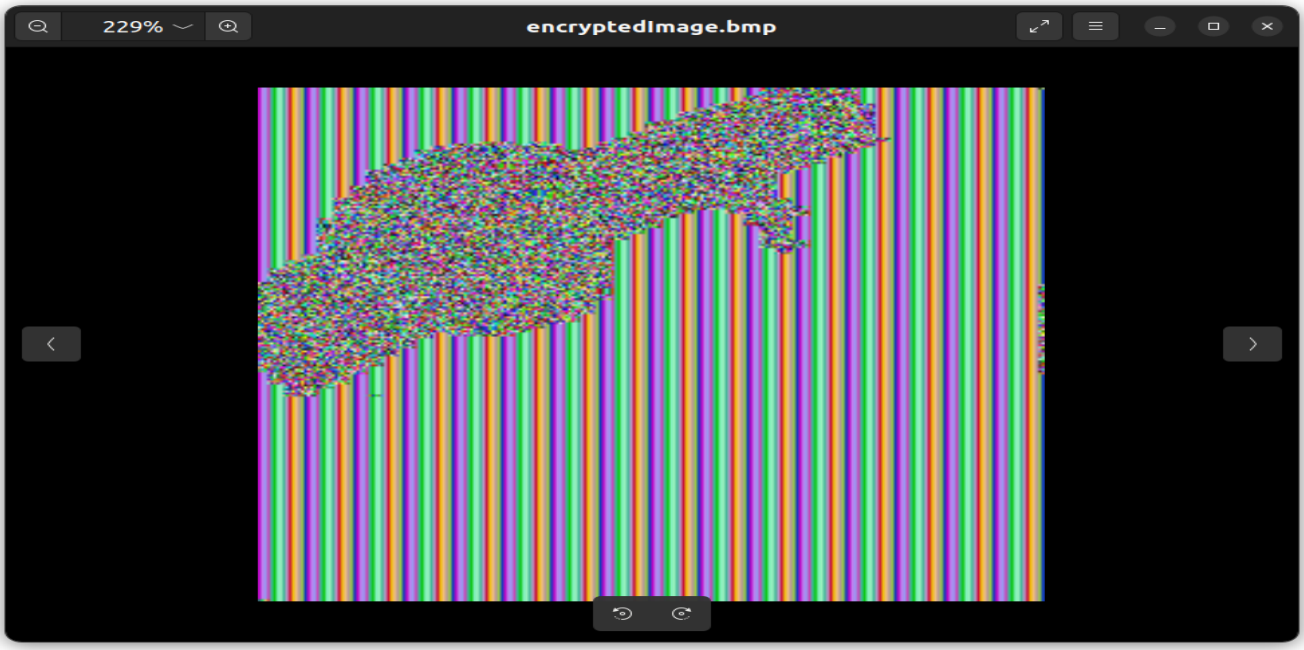
Command:

openssl enc -aes-128-ecb -e -**in** snail.bmp -out encryptedImage.bmp -K 001122
ghex snail.bmp &




snail.bmp - GHex
File Edit View Windows Help
00000000 42 4D 36 00 03 00 00 00 00 00 36 00 00 00 28 00 BM6.....6...(.
00000010 00 00 00 01 00 00 00 01 00 00 01 00 18 00 00 00 ....
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00000030 00 00 00 00 00 00 00 00 00 00 FF FF FF FF FF FF ....
00000040 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ....
00000050 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ....
00000060 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ....
00000070 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ....
00000080 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ....
00000090 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ....
Signed 8 bit: 66
Signed 32 bit: 3558722
Hexadecimal: 42
Unsigned 8 bit: 66
Unsigned 32 bit: 3558722
Octal: 102
Signed 16 bit: 19778
Signed 64 bit: 12888460610
Binary: 01000010
Unsigned 16 bit: 19778
Unsigned 64 bit: 12888460610
Stream Length: 8
Float 32 bit: 4.986832e-39
Float 64 bit: 6.367746e-314
Show little endian decoding
Show unsigned and float as hexadecimal
Offset: 0x0; 0x36 bytes from 0x0 to 0x35 selected


encryptedImage.bmp - GHex
File Edit View Windows Help
00000000 42 4D 36 00 03 00 00 00 00 00 36 00 00 00 28 00 BM6.....6...(.
00000010 00 00 00 01 00 00 00 01 00 00 01 00 18 00 00 00 ....
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00000030 00 00 00 00 00 00 77 6B 3B 66 CD 2D 6E C0 D3 46 ......wk;f.-n..F
00000040 47 C9 96 BB 3C 11 C7 0A BC F0 81 BF F0 99 9A B8 G...<.....
00000050 47 C9 96 BB 3C 11 C7 0A BC F0 81 BF F0 99 9A B8 G...<.....
00000060 47 C9 96 BB 3C 11 C7 0A BC F0 81 BF F0 99 9A B8 G...<.....
00000070 47 C9 96 BB 3C 11 C7 0A BC F0 81 BF F0 99 9A B8 G...<.....
00000080 47 C9 96 BB 3C 11 C7 0A BC F0 81 BF F0 99 9A B8 G...<.....
00000090 47 C9 96 BB 3C 11 C7 0A BC F0 81 BF F0 99 9A B8 G...<.....
Signed 8 bit: -100
Signed 32 bit: 2092108956
Hexadecimal: 9C
Unsigned 8 bit: 156
Unsigned 32 bit: 2092108956
Octal: 234
Signed 16 bit: 3228
Signed 64 bit: 6910322675905924252
Binary: 10011100
Unsigned 16 bit: 3228
Unsigned 64 bit: 6910322675905924252
Stream Length: 8
Float 32 bit: 7.437415e+36
Float 64 bit: 9.384584e+153
Show little endian decoding
Show unsigned and float as hexadecimal
Offset: 0x0; 0x36 bytes from 0x0 to 0x35 selected



Command:

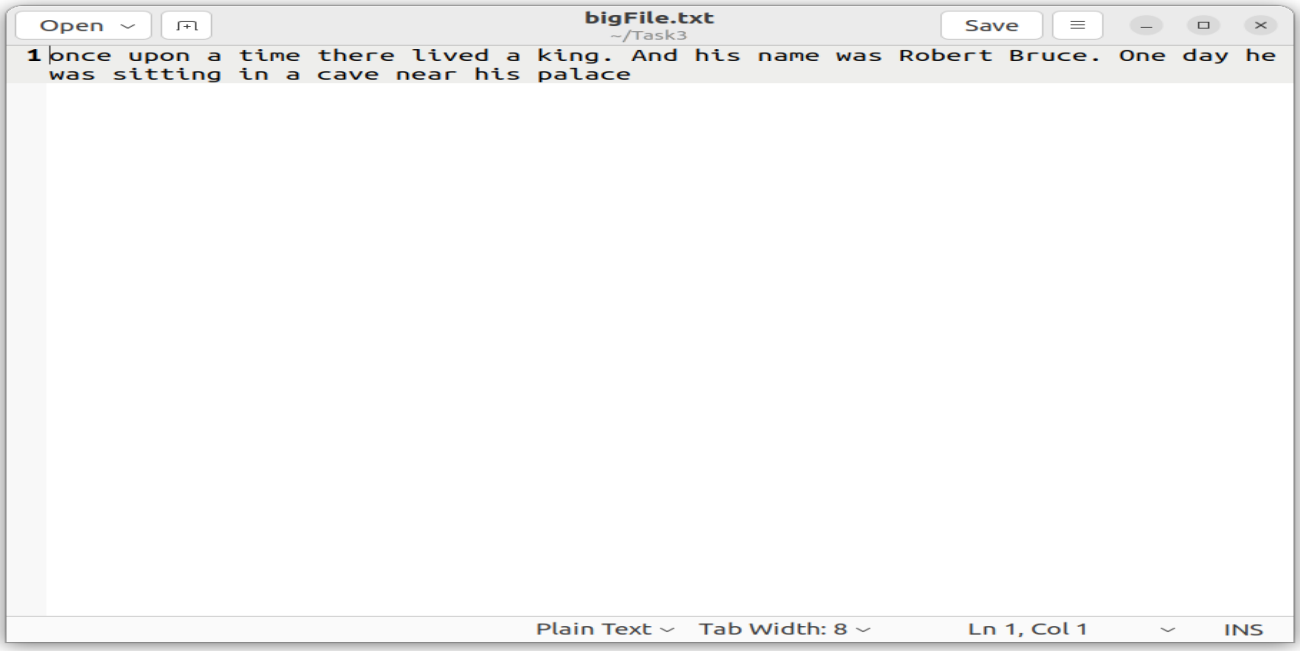
```
openssl enc -aes-128-cbc -e  -in snail.bmp -out encrypted-cbc.bmp \
            -K  00112233445566778889aabbccddeeff \
            -iv 01020304050607083241231213124f23
```

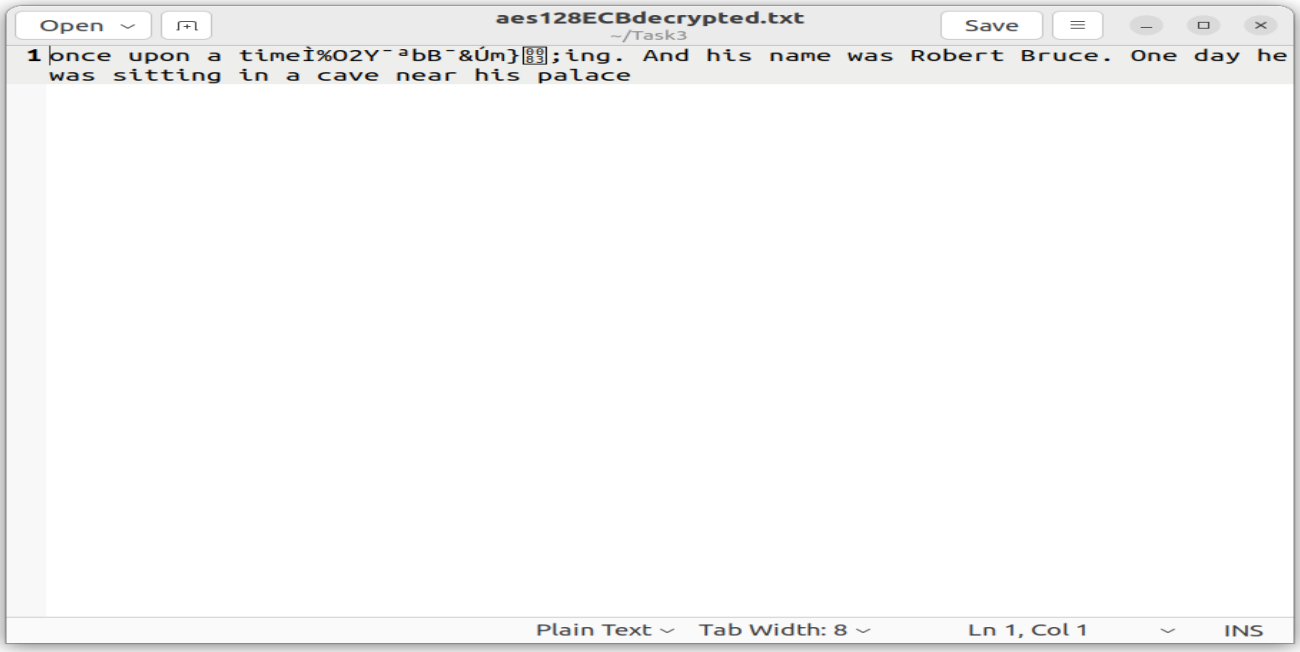
Analysis Comparison between ECB CBC Mode

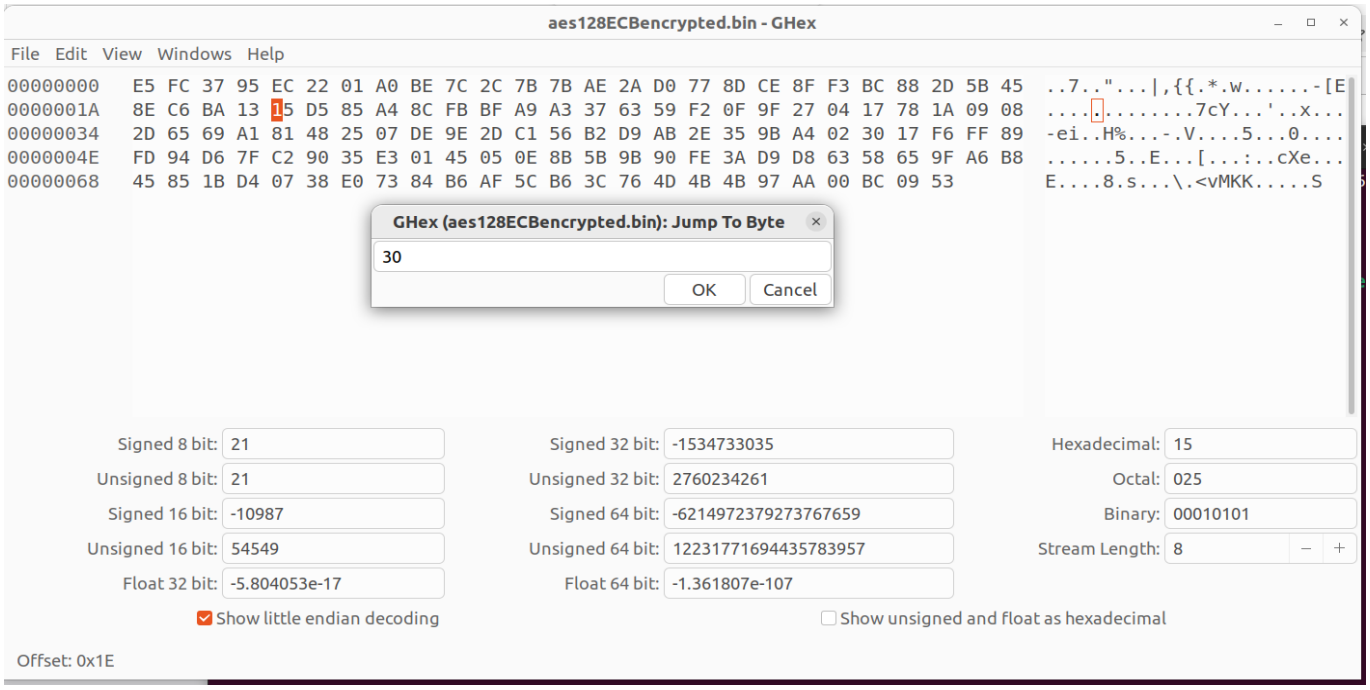
Comparison:

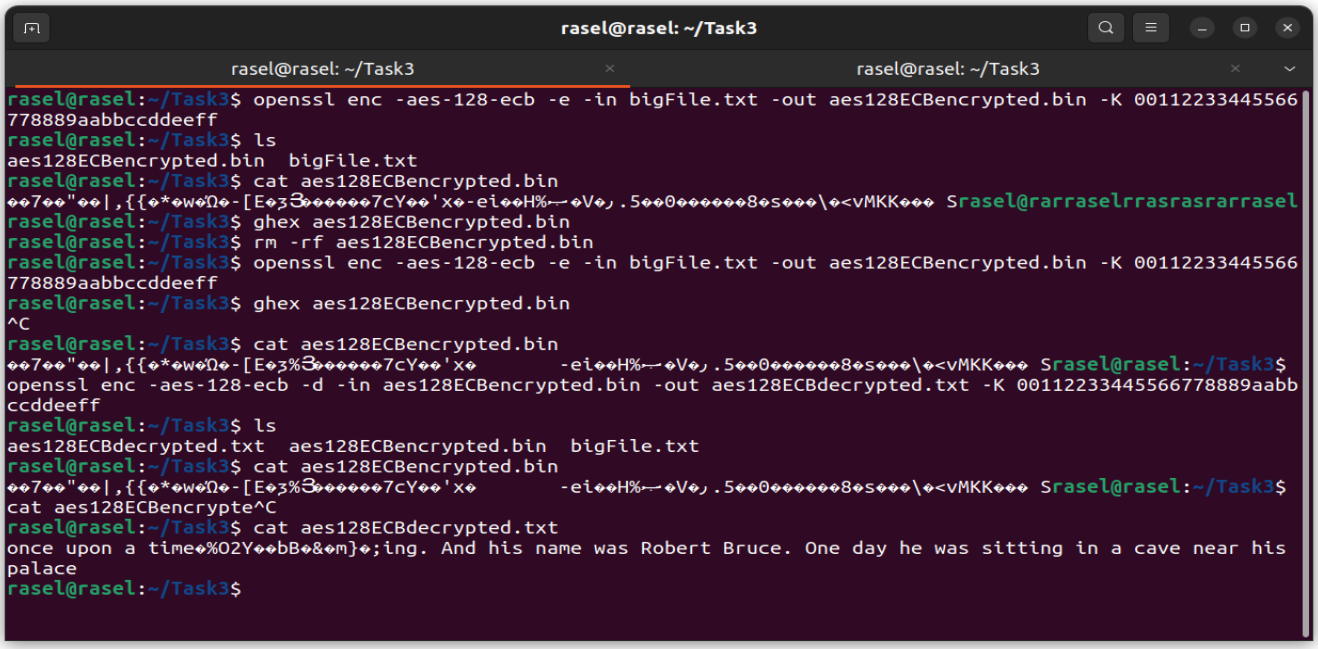
ECB is simpler and faster, but less secure, especially for images. CBC is more secure due to its chaining mechanism, but slightly slower and requires an IV. For image encryption, CBC is generally preferred over ECB due to its better resistance to pattern preservation and higher security.

Task – 3: : Encryption mode – corrupted cipher text (3 Marks)









0.0.1 Observation

ECB mode offers the least security and diffusion, as evident from the limited recoverable information.

CBC mode provides better security compared to ECB but still suffers from partial corruption propagation.

CFB and OFB modes offer improved diffusion, allowing for more recoverable information compared to CBC mode, although they are still susceptible to partial corruption propagation.

Task – 4:Padding

CBC

Encryption Command:

```
openssl enc -aes-128-cbc -e  -in task4.txt -out encrypted-cbc.bin \
            -K  00112233445566778889aabbccddeeff \
            -iv 01020304050607083241231213124f23
```

ECB

Encryption Command:

```
openssl enc -aes-128-ecb -e  -in task4.txt -out encrypted-ecb.bin -K 00112
```

CFB

Encryption Command:

```
openssl enc -aes-128-cfb -e  -in task4.txt -out encrypted-cfb.bin \
            -K  00112233445566778889aabbccddeeff \
            -iv 01020304050607083241231213124f23
```

Observation

Here the plain text size was 24 bytes. And CFB OFB encrypted files are also 24 bytes. Which means no padding is needed for these algorithms

But however, ECB CBC algorithm made the size of encrypted file 32 Bytes which is a multiple of 16 (Block size of AES-128). So here padding is needed incase of these 2 algorithms.

Task – 5: Generating message digest

Generate a text file : task5.txt

JavaScript is the most powerful and versatile programming language used in the web. It is a lightweight, cross-platform, single-threaded and interpreted programming language. It is a commonly used programming language to create dynamic and interactive elements in web applications. It is easy to learn.

Hash Algorithm-SHA-1

Command:

```
openssl dgst -sha1 task5.txt
```

Output

```
output: 9ff0f9f3abe8612d5ecc43ee02fef4b2338a5259
```

Hash Algorithm-MD-5

Command:

```
openssl dgst -md5 task5.txt
```

Output

```
output: e8ff2e9a54b8f4798b98a9c439447319
```

Hash Algorithm-SHA-256

Command:

```
openssl dgst -sha256 task5.txt
```

Output

```
output: c6886c744b2b86a0bbccbe486aa11e137bf168074cd85d0ae0445e4eaf29af4f
```

Observation

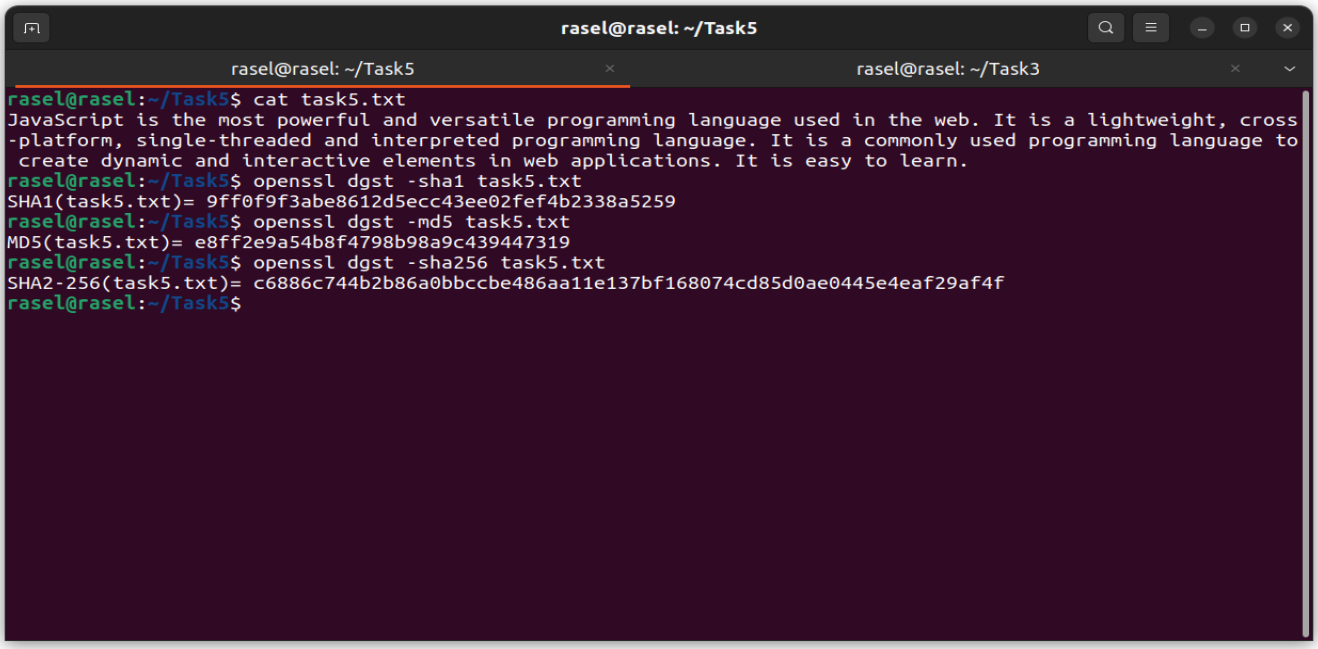
SHA-1 SHA-1 produces a 160-bit hash value.

MD-5 MD-5 produces a 128-bit hash value.

SHA-256 SHA-1 produces a 256-bit hash value.

Comprarison

SHA-256 provides the longest hash value and is the most secure among the three. MD5, while fast, is considered insecure and should be avoided for security purposes. SHA-1 is stronger than MD5 but is also vulnerable to collision attacks, making it less secure than SHA-256. It's being phased out in favor of more secure hashing algorithms.



openssl dgst -sha1 task5.txt
output: 9ff0f9f3abe8612d5ecc43ee02fef4b2338a5259
openssl dgst -md5 task5.txt
output: e8ff2e9a54b8f4798b98a9c439447319
openssl dgst -sha256 task5.txt
output: c6886c744b2b86a0bbccbe486aa11e137bf168074cd85d0ae0445e4eaf29af4f

Task – 6: Keyed hash and HMAC

Generate a text file : task6.txt

JavaScript is a versatile, lightweight, client-side scripting language used in web development. It can be used for both Client-side as well as Server-side developments. JavaScript is also known as a scripting language for web pages, It supports variables, data types, operators, conditional statements, loops, functions, arrays, and objects. With JavaScript, you can create dynamic, interactive, and engaging websites.

Hash Algorithm-HMAC-SHA1

Command:

```
openssl dgst -sha1 -hmac "I-am-rasel" task6.txt
```

Output

```
output: 5511b1276e404e28ce9bf63474ec04dc787f46a5
```

Hash Algorithm-HMAC-MD5

Command:

```
openssl dgst -md5 -hmac "I-am-rasel" task6.txt
```

Output

```
output: 52c0f73e58de367ae413c63b99cc394a
```

Hash Algorithm-HMAC-SHA256

Command:

```
openssl dgst -sha256 -hmac "I-am-rasel" task6.txt
```

Output

```
output: 738e8e4b31b2621358bd43cf753d2ad0245cfc79395004a5043163a51846f0b6
```

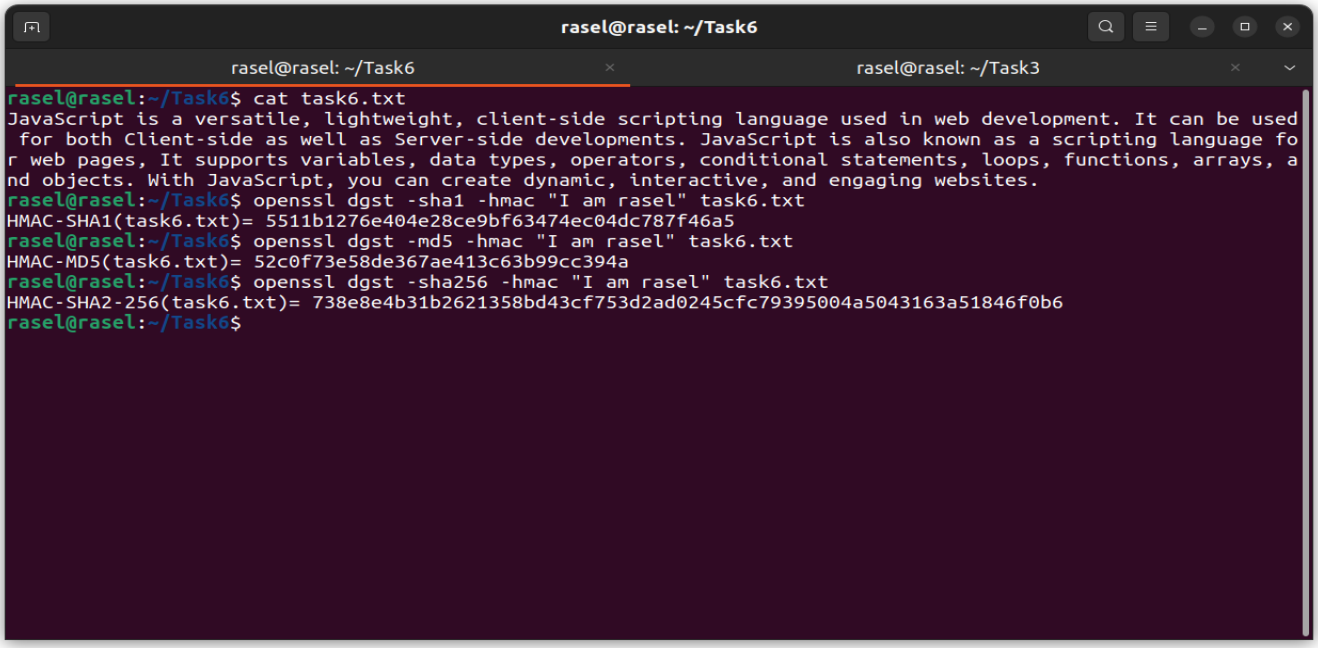
Observation

Do we have to use a key with a fixed size in HMAC?

ANS: No, HMAC does not mandate a fixed key size. However, the key size should be appropriate for the hash function being used.

What is the key size?

The key size depends on the hash function. For example: For HMAC-MD5, the recommended key size is at least 128 bits. For HMAC-SHA256, the recommended key size is 256 bits. For HMAC-SHA1, the recommended key size is also 160 bits.



openssl dgst -sha1 -hmac ”I am rasel” task6.txt
output: 5511b1276e404e28ce9bf63474ec04dc787f46a5
openssl dgst -md5 -hmac ”I am rasel” task6.txt
output: 52c0f73e58de367ae413c63b99cc394a
openssl dgst -sha256 -hmac ”I am rasel” task6.txt
output: 738e8e4b31b2621358bd43cf753d2ad0245cfc79395004a5043163a51846f0b6

Task – 7: Keyed hash and HMAC

Generate a text file : task7.txt

These are the operators that operate upon the numerical values and return a numerical value.

Hash Algorithm-MD5

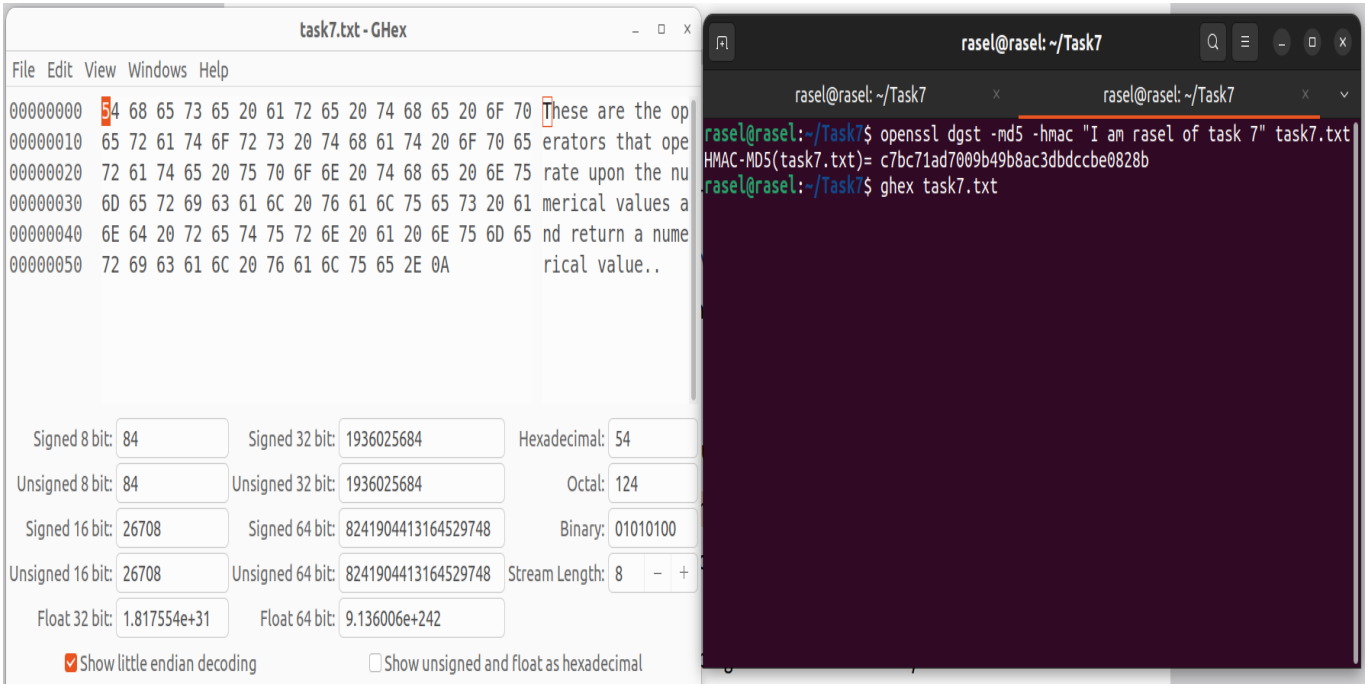
Command:

openssl dgst –md5 –hmac ”I-am-rasel-of-task7” task7.txt

Output

output:H1= c7bc71ad7009b49b8ac3dbdccbe0828

Modify:



h1 = ”c7bc71ad7009b49b8ac3dbdccbe0828” h2 = ”c6e852599b594f7e54db1da9e15c8729”
i = 0 cnt = 0
for element in range(0, len(h1)): if h1[element] == h2[element]: cnt = cnt + 1
print(cnt)

Similar Bit

:

same bit =3
These are the operators that operate upon the numerical values and return a numerical value.

Hash Algorithm-SHA-256

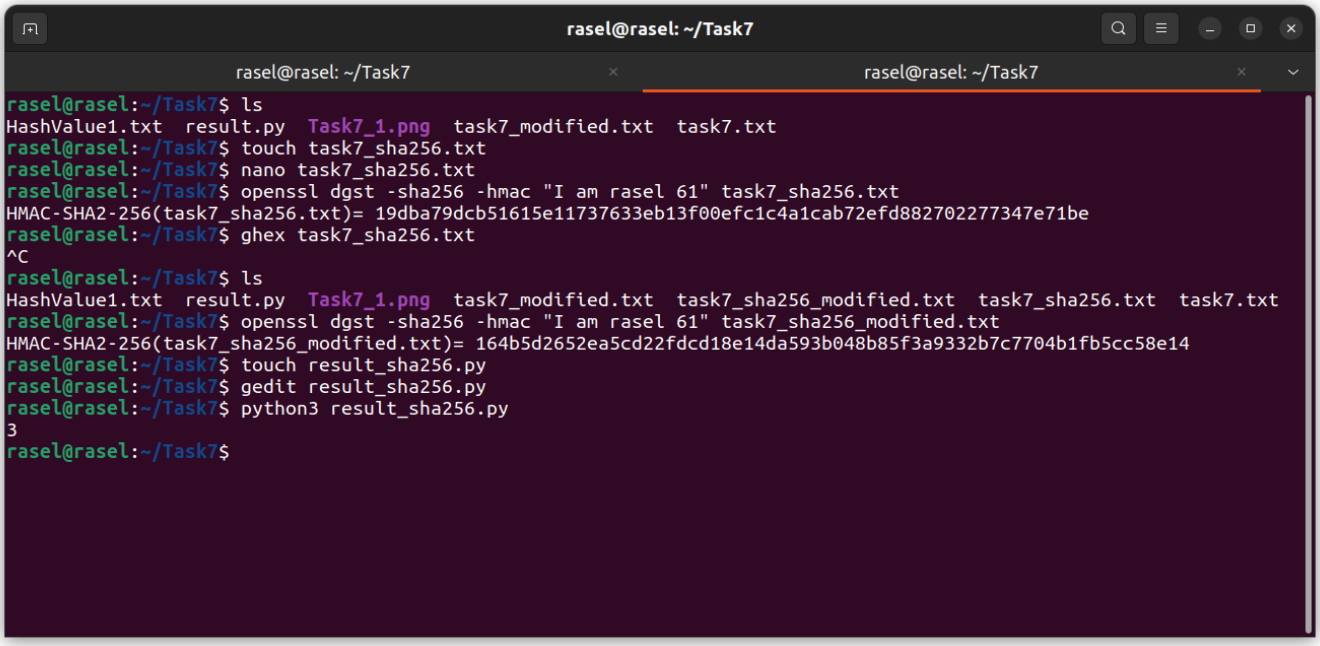
Command:

```
openssl dgst -sha256 -hmac "I am rasel 61" task7_sha256.txt
```

Output

```
output:h1 = "19dba79dcb51615e11737633eb13f00efc1c4a1cab72efd882702277347e71
```

Modify:



Similar Bit

:

same bit =3