



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

Algorithm: AES-128-CBC

Encryption Command:

```
openssl enc -aes-128-cbc -e -in task1.txt -out aes128cbc.bin -K 001122334455
```

Decryption Command:

```
openssl enc -aes-128-cbc -d -in aes128cbc.bin -out aes128cbcdecrypted.txt -K 001122334455
```

Algorithm: AES-128-CFB

Encryption Command:

```
openssl enc -aes-128-cfb -e -in task1.txt -out aes128cfb.bin -K 001122334455
```

Decryption Command:

```
openssl enc -aes-128-cfb -d -in aes128cfb.bin -out aes128cfbdencrypted.txt -K 001122334455
```

Algorithm: AES-128-ECB

Encryption Command:

```
openssl enc -aes-128-ecb -e -in task1.txt -out aes128ecb.bin -K 001122334455
```

Decryption Command:

```
openssl enc -aes-128-ecb -d -in aes128ecb.bin -out aes128ecbdecrypted.txt -K 001122334455
```

Algorithm: AES-256-CBC

Encryption Command:

```
openssl enc -aes-256-cbc -e -in task1.txt -out aes256cbc.bin -K 222223333333
```

Decryption Command:

```
openssl enc -aes-256-cbc -d -in aes256cbc.bin -out aes256cbcdecrypted.txt -K 222223333333
```

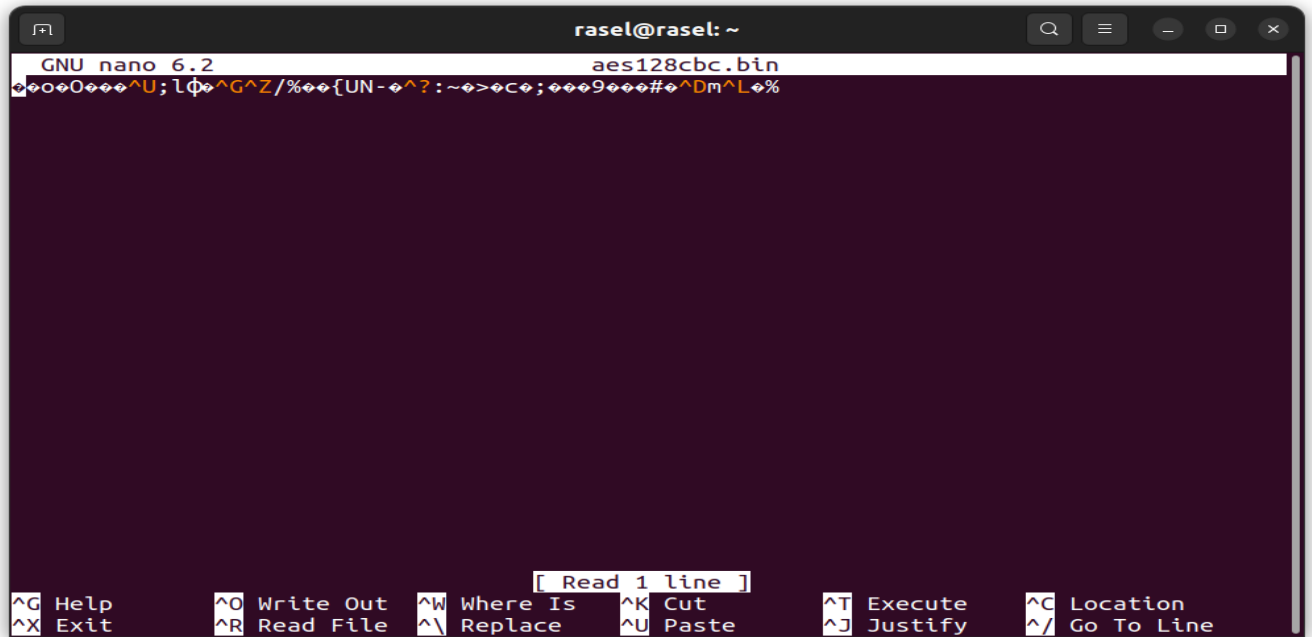
Command ScreenShot

```

rasel@rasel: ~
rasel@rasel:~$ openssl enc -aes-128-cbc -e -in test.txt -out aes128cbc.bin -K 0011223344
5566778889aabbccddeeff -iv 20304050607082143234324324233333
Can't open "test.txt" for reading, No such file or directory
406739EE007F0000:error:80000002:system library:BIIO_new_file:No such file or directory:..
/crypto/bio/bss_file.c:67:calling fopen(test.txt, rb)
406739EE007F0000:error:10000080:BIIO routines:BIIO_new_file:no such file:../crypto/bio/bss
_file.c:75:
rasel@rasel:~$ openssl enc -aes-128-cbc -e -in task1.txt -out aes128cbc.bin -K 0011223344
45566778889aabbccddeeff -iv 20304050607082143234324324233333
rasel@rasel:~$ ls
aes128cbc.bin  Desktop  Downloads  Pictures  snap  Templates
cipher.bin    Documents Music      Public   task1.txt Videos
rasel@rasel:~$ cat aes128cbc.bin
00000000;l000/%00{UN-0:~0>0c0;0009000#0m
0%rasel@rasel:~$ nano aes128cbc.bin
rasel@rasel:~$
```

```

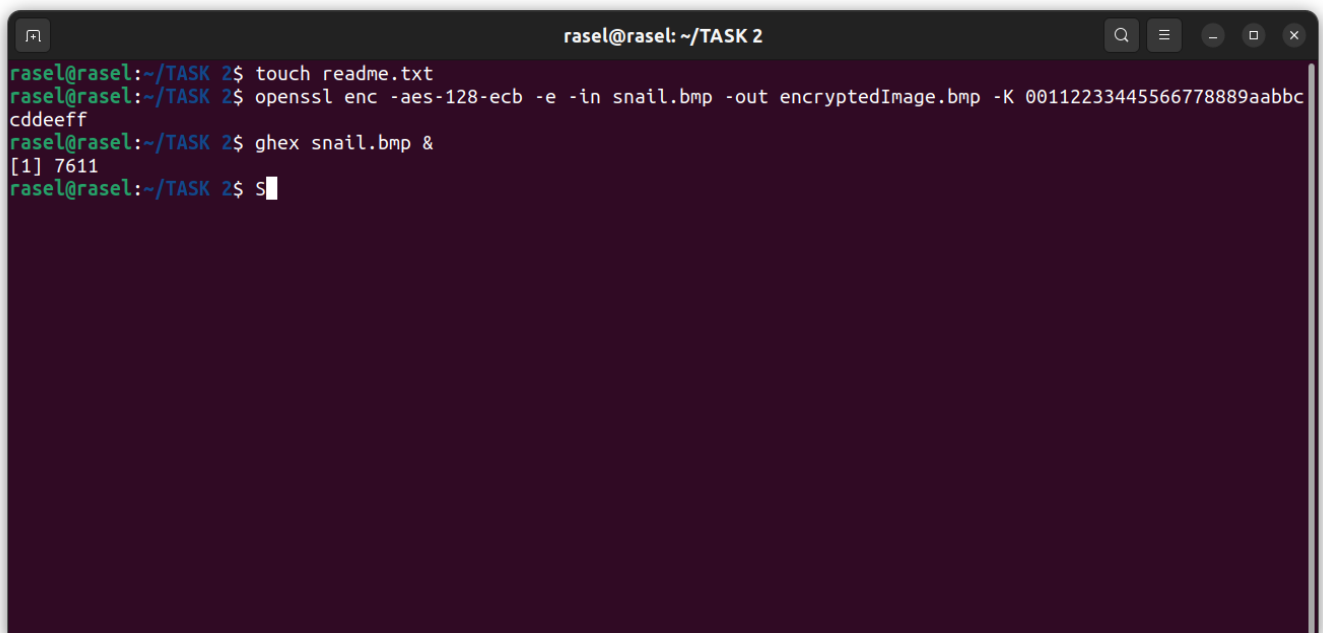
rasel@rasel: ~
rasel@rasel:~$ openssl enc -aes-128-cbc -e -in test.txt -out aes128cbc.bin -K 0011223344
5566778889aabbccddeeff -iv 20304050607082143234324324233333
Can't open "test.txt" for reading, No such file or directory
406739EE007F0000:error:80000002:system library:BIIO_new_file:No such file or directory:..
/crypto/bio/bss_file.c:67:calling fopen(test.txt, rb)
406739EE007F0000:error:10000080:BIIO routines:BIIO_new_file:no such file:../crypto/bio/bss
_file.c:75:
rasel@rasel:~$ openssl enc -aes-128-cbc -e -in task1.txt -out aes128cbc.bin -K 0011223344
45566778889aabbccddeeff -iv 20304050607082143234324324233333
rasel@rasel:~$ ls
aes128cbc.bin  Desktop  Downloads  Pictures  snap  Templates
cipher.bin    Documents Music      Public   task1.txt Videos
rasel@rasel:~$ cat aes128cbc.bin
00000000;l000/%00{UN-0:~0>0c0;0009000#0m
0%rasel@rasel:~$ nano aes128cbc.bin
rasel@rasel:~$ openssl enc -aes-128-cbc -d -in aes128cbc.bin -out aes128cbcdecrypted.txt
-K 00112233445566778889aabbccddeeff -iv 20304050607082143234324324233333
rasel@rasel:~$ ls
aes128cbc.bin  cipher.bin  Documents  Music  Public  task1.txt  Videos
aes128cbcdecrypted.txt Desktop  Downloads  Pictures  snap  Templates
rasel@rasel:~$ nano aes128cbcdecrypted.txt
rasel@rasel:~$
```

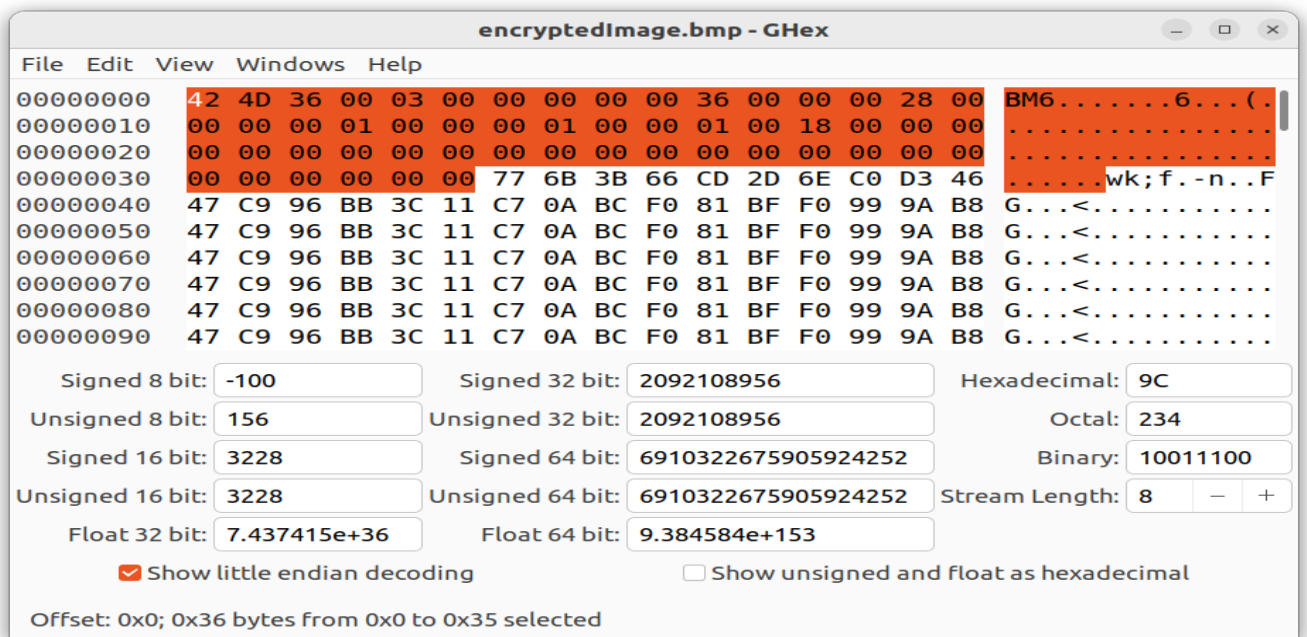
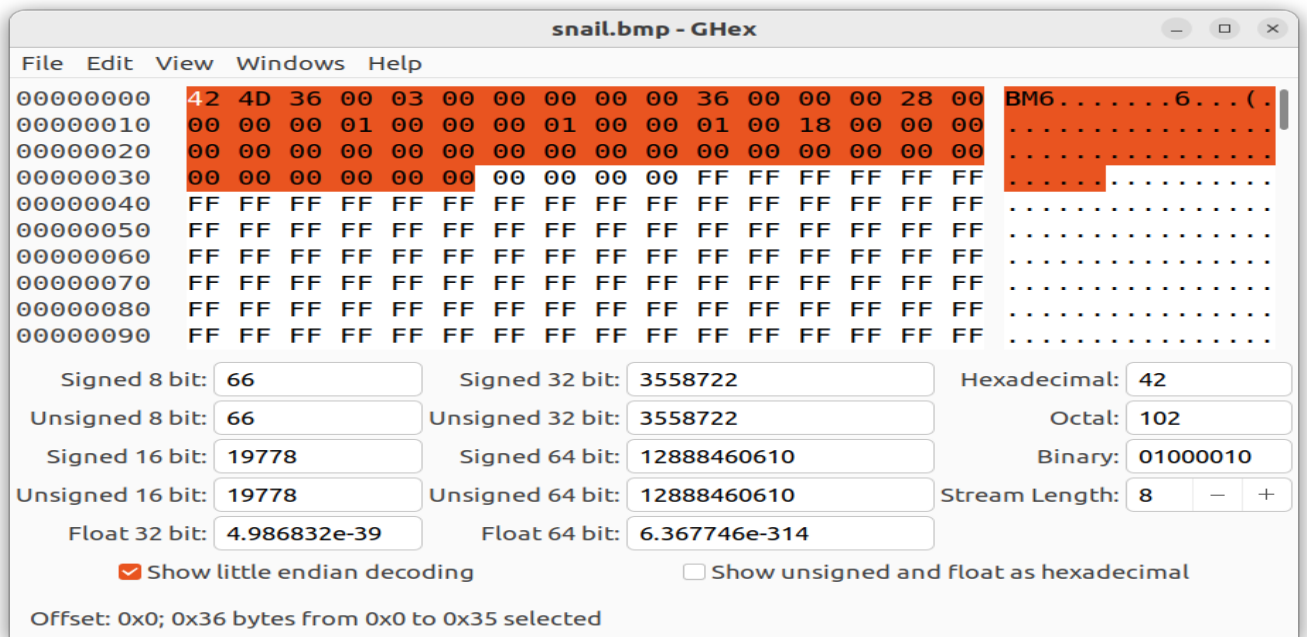


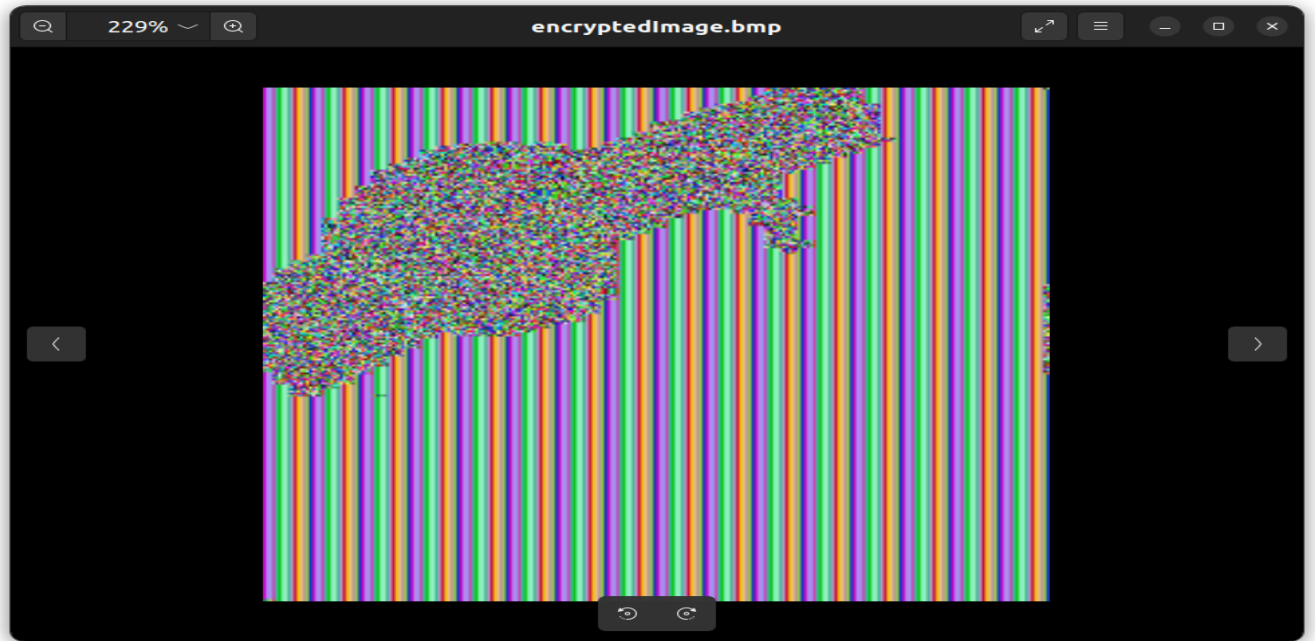
Task2: Encryption mode - ECB vs CBC

Command:

```
openssl enc -aes-128-ecb -e -in snail.bmp -out encryptedImage.bmp -K 0011223344556677889aabbccddeeff  
ghex snail.bmp &
```







Command:

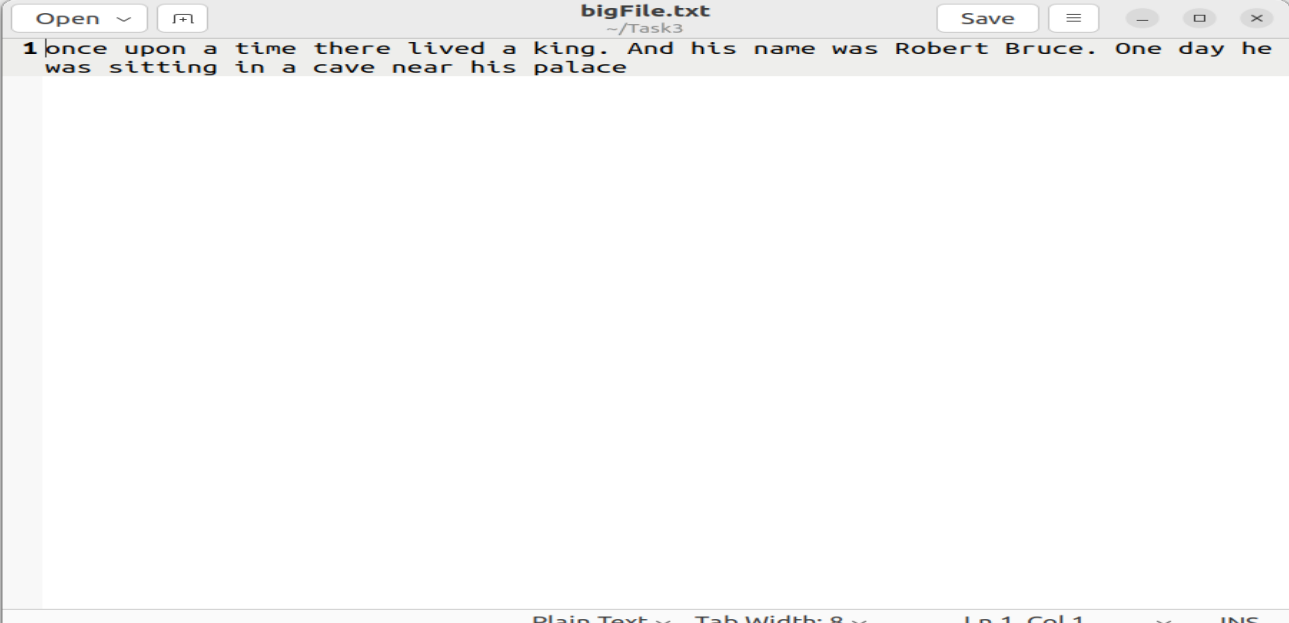
```
openssl enc -aes-128-cbc -e -in snail.bmp -out encrypted-cbc.bmp \
-K 00112233445566778889aabbccddeeff \
-iv 01020304050607083241231213124f23
```

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.

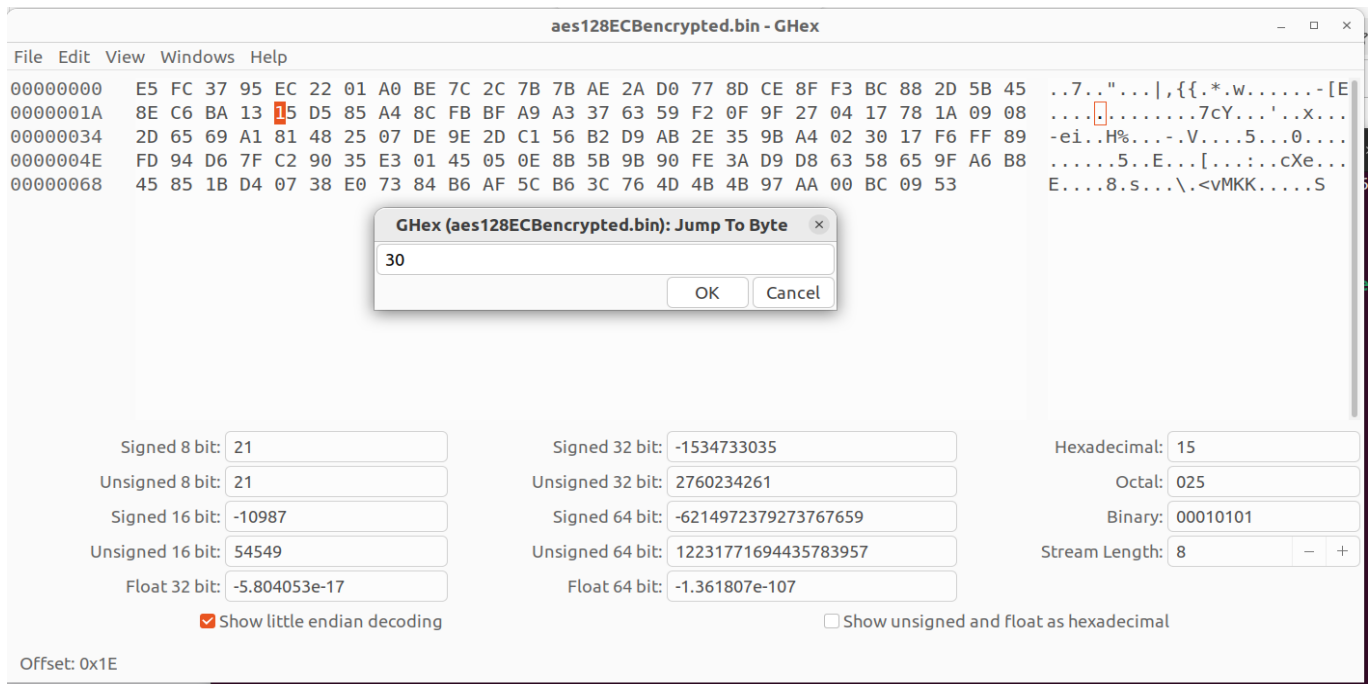
Task – 3: : Encryption mode – corrupted cipher text (3 Marks)



A screenshot of a text editor window titled "bigFile.txt" with a subtitle "~/Task3". The window has a menu bar with "Open" and "Save" buttons, and a toolbar with icons for file operations. The text area contains the following text: "1|once upon a time there lived a king. And his name was Robert Bruce. One day he was sitting in a cave near his palace". The status bar at the bottom shows "Plain Text", "Tab Width: 8", "Ln 1, Col 1", and "INS".



A screenshot of a text editor window titled "aes128ECBdecrypted.txt" with a subtitle "~/Task3". The window has a menu bar with "Open" and "Save" buttons, and a toolbar with icons for file operations. The text area contains the following text: "1|once upon a timei%02Y~^abB~&Úm};;ing. And his name was Robert Bruce. One day he was sitting in a cave near his palace". The status bar at the bottom shows "Plain Text", "Tab Width: 8", "Ln 1, Col 1", and "INS".



```

rasel@rasel: ~/Task3
rasel@rasel: ~/Task3$ openssl enc -aes-128-ecb -e -in bigFile.txt -out aes128ECBencrypted.bin -K 00112233445566778889aabbccddeeff
rasel@rasel: ~/Task3$ ls
aes128ECBencrypted.bin  bigFile.txt
rasel@rasel: ~/Task3$ cat aes128ECBencrypted.bin
778889aabbccddeeff,{{*w0-[E3%3000007cY00'x0-ei0H%0V0r.5000000008s000\0<vMKK000 Srasel@rraselrrasrarrasel
rasel@rasel: ~/Task3$ ghex aes128ECBencrypted.bin
rasel@rasel: ~/Task3$ rm -rf aes128ECBencrypted.bin
rasel@rasel: ~/Task3$ openssl enc -aes-128-ecb -e -in bigFile.txt -out aes128ECBencrypted.bin -K 00112233445566778889aabbccddeeff
rasel@rasel: ~/Task3$ ghex aes128ECBencrypted.bin
^C
rasel@rasel: ~/Task3$ cat aes128ECBencrypted.bin
778889aabbccddeeff,{{*w0-[E3%3000007cY00'x0-ei0H%0V0r.5000000008s000\0<vMKK000 Srasel@rasel: ~/Task3$
rasel@rasel: ~/Task3$ openssl enc -aes-128-ecb -d -in aes128ECBencrypted.bin -out aes128ECBdecrypted.txt -K 00112233445566778889aabbccddeeff
rasel@rasel: ~/Task3$ ls
aes128ECBdecrypted.txt  aes128ECBencrypted.bin  bigFile.txt
rasel@rasel: ~/Task3$ cat aes128ECBencrypted.bin
778889aabbccddeeff,{{*w0-[E3%3000007cY00'x0-ei0H%0V0r.5000000008s000\0<vMKK000 Srasel@rasel: ~/Task3$
cat aes128ECBdecrypted.txt
once upon a time%02Y00bB00m}0;ing. And his name was Robert Bruce. One day he was sitting in a cave near his
palace
rasel@rasel: ~/Task3$

```

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 00112233445566778889aabbccddeeff
```

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

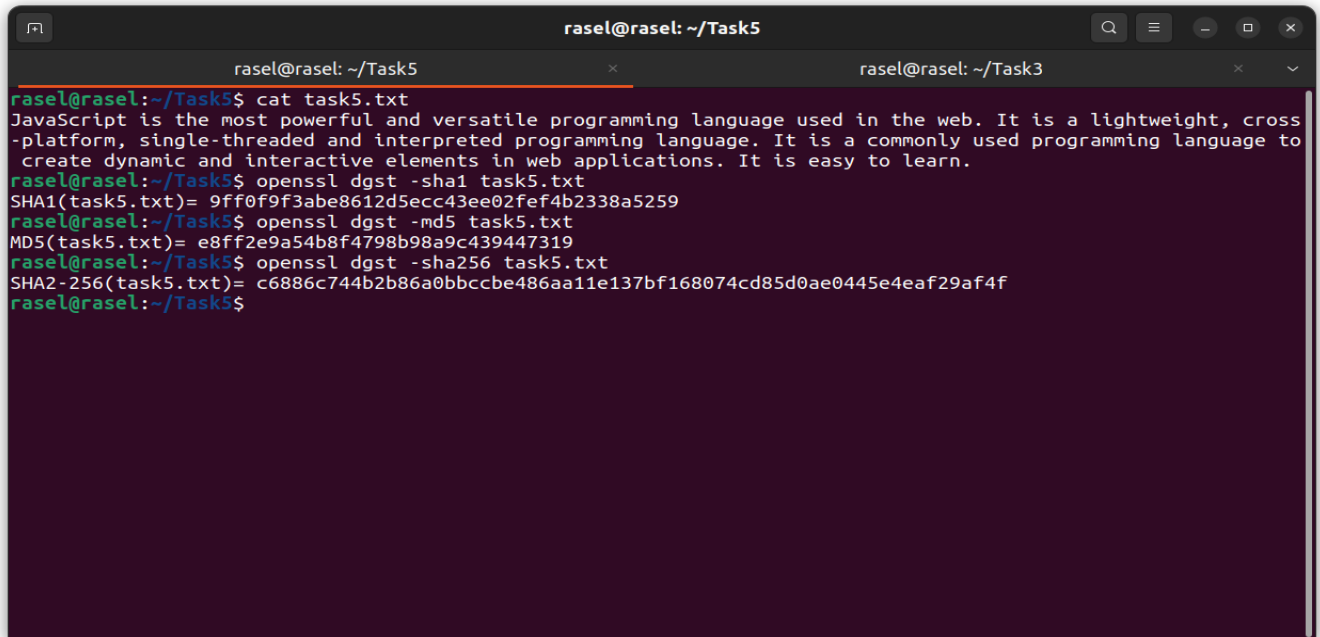
SHA-1 SHA-1 produces a 160-bit hash value.

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.

A terminal window titled 'rasel@rasel: ~/Task5' showing the execution of three commands to generate hashes for a file named 'task5.txt'. The first command is 'cat task5.txt', which displays the content of the file: '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.' The second command is 'openssl dgst -sha1 task5.txt', which outputs the SHA1 hash: 'SHA1(task5.txt)= 9ff0f9f3abe8612d5ecc43ee02fef4b2338a5259'. The third command is 'openssl dgst -md5 task5.txt', which outputs the MD5 hash: 'MD5(task5.txt)= e8ff2e9a54b8f4798b98a9c439447319'. The fourth command is 'openssl dgst -sha256 task5.txt', which outputs the SHA256 hash: 'SHA2-256(task5.txt)= c6886c744b2b86a0bbccbe486aa11e137bf168074cd85d0ae0445e4eaf29af4f'.

```
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
```

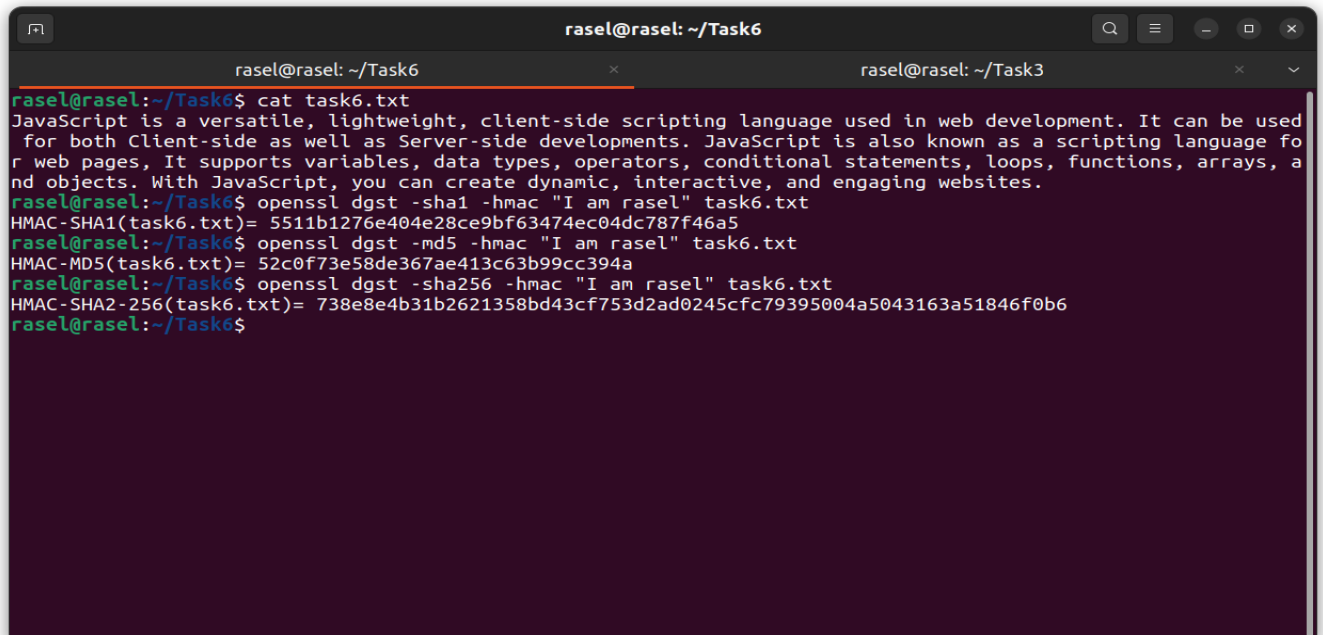
Observation

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.

A terminal window titled 'rasel@rasel: ~/Task6' showing a series of commands and their outputs. The user first cat's 'task6.txt' which contains text about JavaScript. Then, they use 'openssl dgst -sha1 -hmac "I am rasel" task6.txt' to get a SHA1 HMAC, 'openssl dgst -md5 -hmac "I am rasel" task6.txt' for MD5, and 'openssl dgst -sha256 -hmac "I am rasel" task6.txt' for SHA256.

```
rasel@rasel:~/Task6$ cat 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 fo
r web pages, It supports variables, data types, operators, conditional statements, loops, functions, arrays, a
nd objects. With JavaScript, you can create dynamic, interactive, and engaging websites.
rasel@rasel:~/Task6$ openssl dgst -sha1 -hmac "I am rasel" task6.txt
HMAC-SHA1(task6.txt)= 5511b1276e404e28ce9bf63474ec04dc787f46a5
rasel@rasel:~/Task6$ openssl dgst -md5 -hmac "I am rasel" task6.txt
HMAC-MD5(task6.txt)= 52c0f73e58de367ae413c63b99cc394a
rasel@rasel:~/Task6$ openssl dgst -sha256 -hmac "I am rasel" task6.txt
HMAC-SHA2-256(task6.txt)= 738e8e4b31b2621358bd43cf753d2ad0245cfc79395004a5043163a51846f0b6
rasel@rasel:~/Task6$
```

```
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

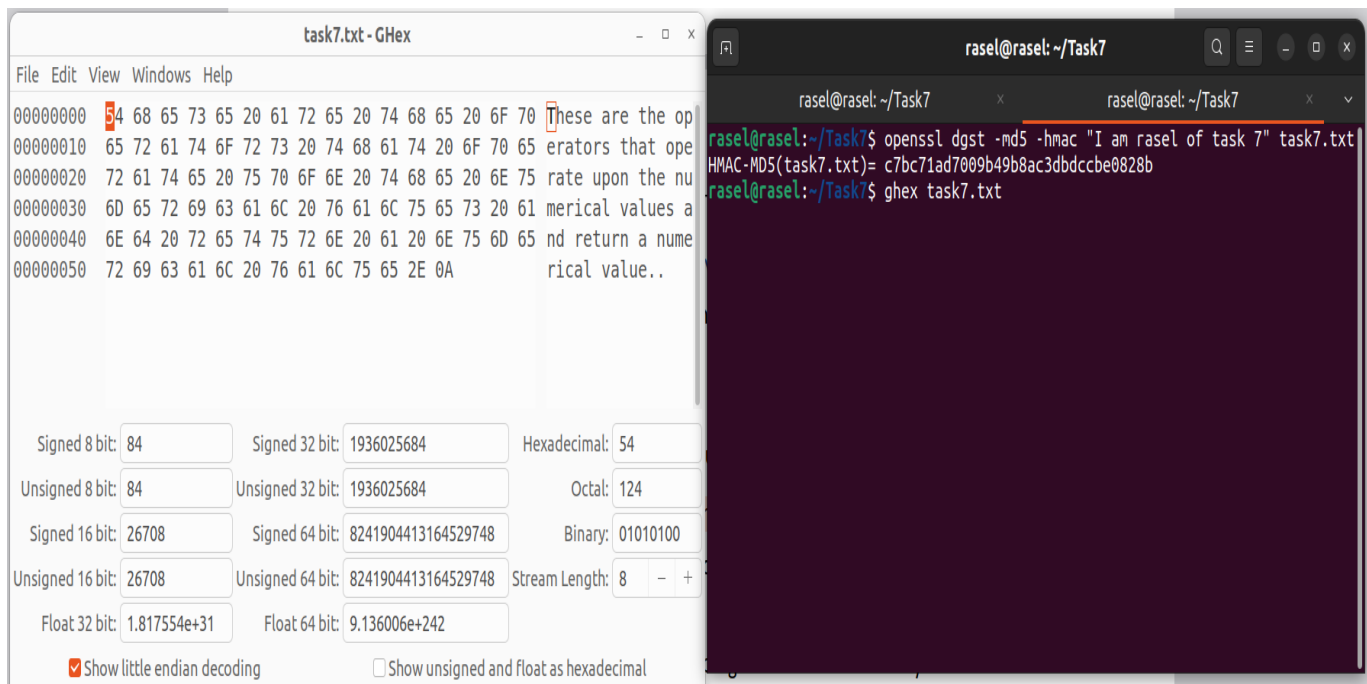
Command:

```
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

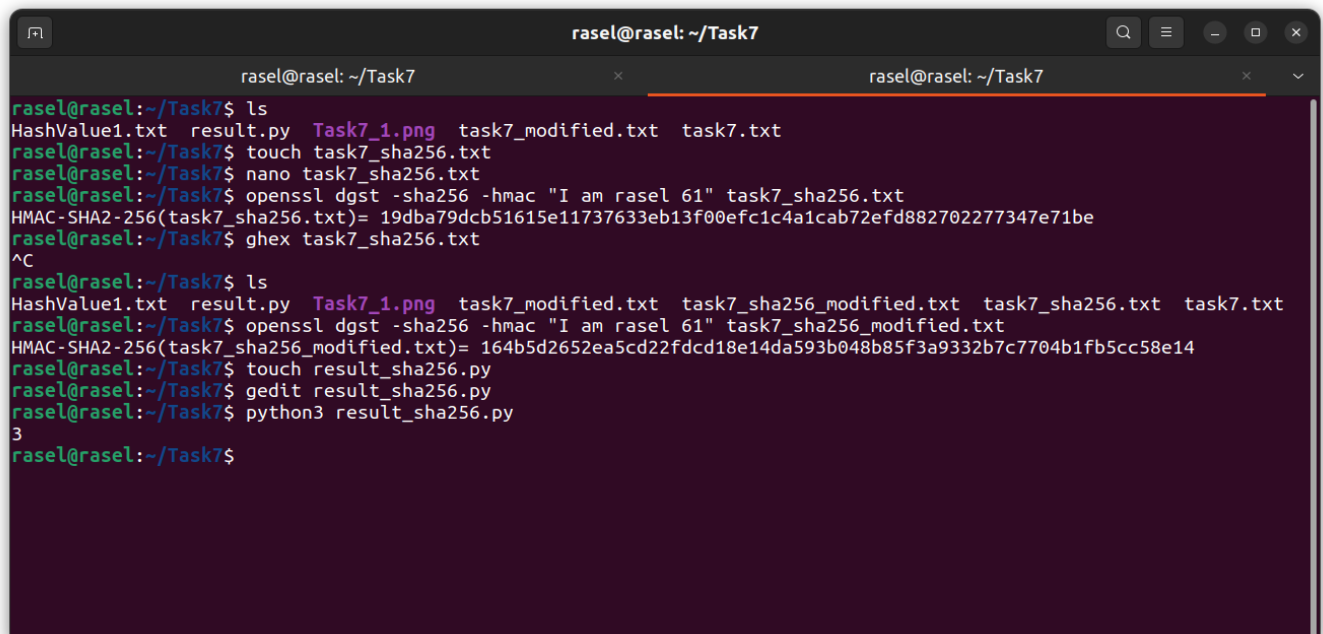
Command:

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

Output

```
output:hl = "19dba79dcb51615e11737633eb13f00efc1c4a1cab72efd882702277347e71be"
```

Modify:

A terminal window titled 'rasel@rasel: ~/Task7' showing a series of commands and their outputs. The user lists files, creates a new file 'task7_sha256.txt', uses 'nano' to edit it with the text 'I am rasel 61', and then runs 'openssl dgst -sha256 -hmac "I am rasel 61" task7_sha256.txt'. The output shows the HMAC-SHA2-256 hash. The user then runs 'ghex task7_sha256.txt' and presses '^C'. They list files again, showing a new file 'task7_sha256_modified.txt' has been added. They then run 'openssl dgst -sha256 -hmac "I am rasel 61" task7_sha256_modified.txt', which outputs a different hash. Finally, they create 'result_sha256.py', edit it with 'gedit', and run 'python3 result_sha256.py', which outputs the number '3'.

```
rasel@rasel:~/Task7$ ls
HashValue1.txt  result.py  Task7_1.png  task7_modified.txt  task7.txt
rasel@rasel:~/Task7$ touch task7_sha256.txt
rasel@rasel:~/Task7$ nano task7_sha256.txt
rasel@rasel:~/Task7$ openssl dgst -sha256 -hmac "I am rasel 61" task7_sha256.txt
HMAC-SHA2-256(task7_sha256.txt)= 19dba79dcb51615e11737633eb13f00efc1c4a1cab72efd882702277347e71be
rasel@rasel:~/Task7$ ghex task7_sha256.txt
^C
rasel@rasel:~/Task7$ ls
HashValue1.txt  result.py  Task7_1.png  task7_modified.txt  task7_sha256_modified.txt  task7_sha256.txt  task7.txt
rasel@rasel:~/Task7$ openssl dgst -sha256 -hmac "I am rasel 61" task7_sha256_modified.txt
HMAC-SHA2-256(task7_sha256_modified.txt)= 164b5d2652ea5cd22fdcd18e14da593b048b85f3a9332b7c7704b1fb5cc58e14
rasel@rasel:~/Task7$ touch result_sha256.py
rasel@rasel:~/Task7$ gedit result_sha256.py
rasel@rasel:~/Task7$ python3 result_sha256.py
3
rasel@rasel:~/Task7$
```

Similar Bit

:
same bit =3