Task 1: AES Encryption using different modes

- **AES encryption using AES-256-cbc mode**

  openssl enc -aes-256-cbc -base64 -in first.txt
  openssl enc -aes-256-cbc -base64 -in first.txt -out encfirst
  openssl enc -aes-256-cbc -d -base64 -in encfirst -out decfirst

  Encryption or decryption password: pass

- **AES encryption using AES-256-ecb mode**

  openssl enc -aes-256-ecb -base64 -in second.txt
  openssl enc -aes-256-ecb -base64 -in second.txt -out encsecond
  openssl enc -aes-256-ecb -d -base64 -in encsecond -out decsecond

  Encryption or decryption password: pass

- **AES encryption using AES-128-cbc mode**

  openssl enc -aes-128-cbc -base64 -in third.txt
  openssl enc -aes-128-cbc -base64 -in third.txt -out encthird
  openssl enc -aes-128-cbc -d -base64 -in encthird -out decthird

  Encryption or decryption password: pass

Task 2: Encryption Mode- ECB vs CBC

- **Observation:**
  ECB is useful for random strings. But it is quite vulnerable for patterns such as image files. Repeated ciphertext provides suitable predictions of the original text. But in the case of CBC, it performs quite well for these situations. Since every output is different from the previous outputs, there are no patterns of ciphertext.
  We found a vague idea of the original image when using ECB. But CBC showed no such vulnerability.
- **Screenshots:**

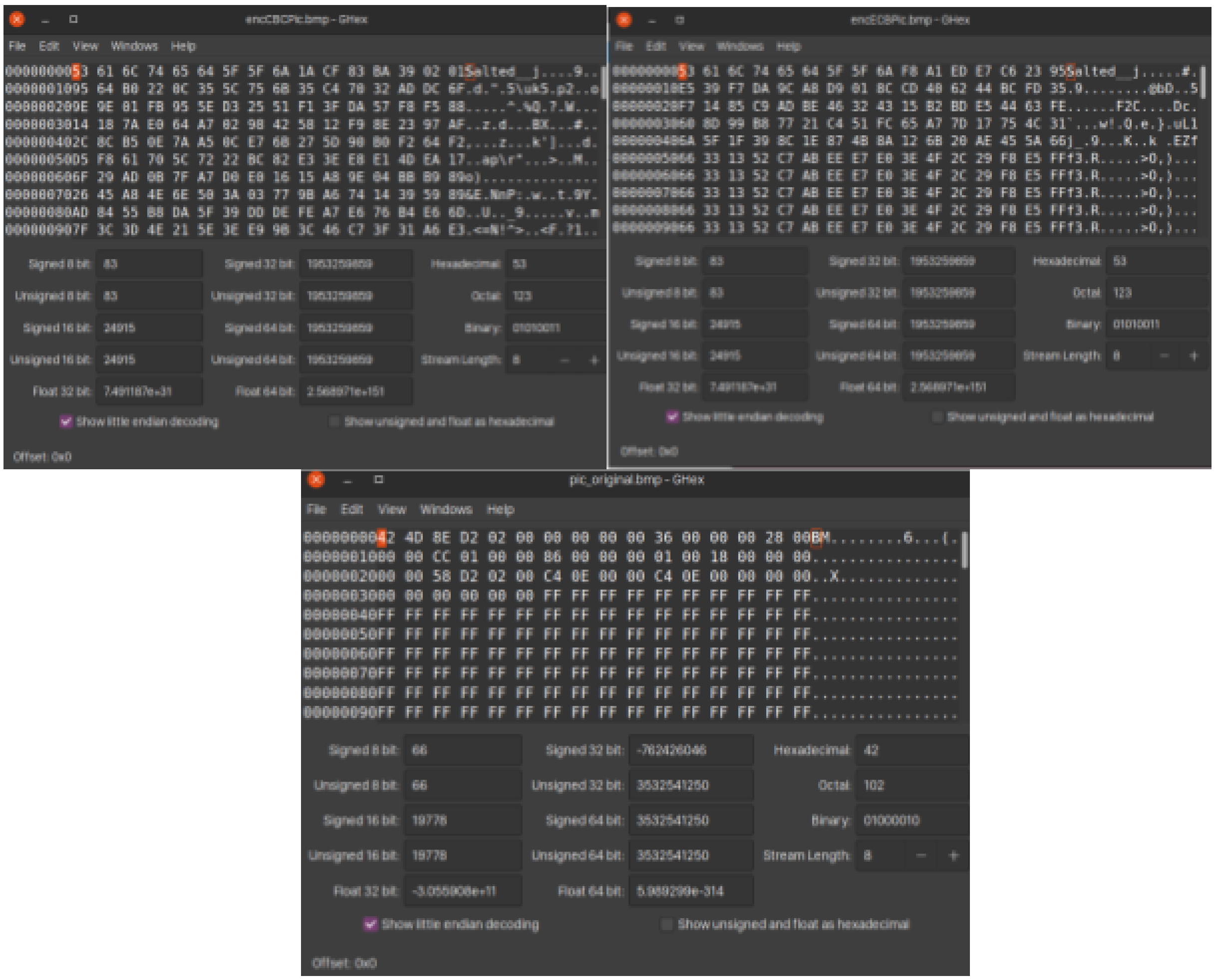
  ECB mode-

```
corei99@corei99:~/Downloads/Security_Assignment/Task3$ openssl enc -aes-256-ecb
-e -in pic_original.bmp -out encECBPic.bmp -k pass
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
```

CBC mode-

```
corei99@corei99:~/Downloads/Security_Assignment/Task3$ openssl enc -aes-256-cbc
-e -in pic_original.bmp -out encCBCPic.bmp -k pass
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
```

Ghex Editor



Task 3: Encryption Mode- Corrupted CipherText

ECB mode: In ECB mode, each block is encrypted independent of the previous blocks. So only the 30th block, which is corrupted, should change without affecting any other block.
Our presumption was correct as the output only changed for the 30th block. The rest of the information was completely recovered.

Corrupted Output:

```
corei99@corei99:~/Downloads/Security_Assignment/Task3/Problem3$ cat decECBProble
m3
k�k0�[��&��dipen-source, public, blockchain-based distributed
computing platform and operating system featuring smart contract functionality
```

CBC mode: In CBC mode, each block is XORed with the previous ciphertext. Thus, any changes in x-th block should affect the (x+1)-th block only. The rest of the block should remain uncorrupted.
As it turns out, only the 30th and 31st block were corrupted. The rest were completely recovered.

Corrupted Output:

```
corei99@corei99:~/Downloads/Security_Assignment/Task3/Problem3$ cat decCBCProble
m3

2�cO�r��(kd�pen-source, p�blic, blockchain-based distributed
computing platform and operating system featuring smart contract functionality
```

OFB mode: We should be able to recover all the blocks except the corrupted block. Since any of the decrypted blocks is not dependent on any previous ciphertexts, the corruption is isolated to individual blocks.
Our presumption proved to be correct. Only the 30th block was affected. The rest of the blocks were recovered completely.

Corrupted Output:

```
corei99@corei99:~/Downloads/Security_Assignment/Task3/Problem3$ cat decOFBProble
m3
Ethereum is aQ open-source, public, blockchain-based distributed
computing platform and operating system featuring smart contract functionality
```

CFB mode: Since in CFB, the XOR process is the same, the corrupted blocks are the same as in CBC. Only 30th and 31st were affected.

Corrupted Output:

```
corei99@corei99:~/Downloads/Security_Assignment/Task3/Problem3$ cat decCFBProble
m3
Ethereum is aw o��>����-���卄��ic, blockchain-based distributed
computing platform and operating system featuring smart contract functionality
```

Task 4: Generating Message Digest

Md5 Algorithm:
Hash Command: *openssl dgst -md5 init.txt*

```
corei99@corei99:~/Downloads/Security_Assignment/Task3/Problem4$ openssl dgst -md
5 init.txt
MD5(init.txt)= d5a896f1d9667ca740b9e17a24ad40d4
```

Sha1 Algorithm:
Hash Command: *openssl dgst -sha1 init.txt*

```
corei99@corei99:~/Downloads/Security_Assignment/Task3/Problem4$ openssl dgst -sh
a1 init.txt
SHA1(init.txt)= 79e8670a0e24d082c1568723bcb79a49a4c783b5
```

SHA256 Algorithm:
Hash Command: *openssl dgst -sha256 init.txt*

```
corei99@corei99:~/Downloads/Security_Assignment/Task3/Problem4$ openssl dgst -sh
a256 init.txt
SHA256(init.txt)= a7ffdf20facaa53366e8985d996093e1153ec736242a39b5b041d15d1c238f
96
```

Task 5: Keyed Hash and HMAC

HMAC-md5 command: *openssl dgst -md5 -hmac “youwannaknow28121996” init.txt*
Screenshot

```
corei99@corei99:~/Downloads/Security_Assignment/Task3/Problem5$ openssl dgst -md
5 -hmac "youwannaknow28121996" init.txt
HMAC-MD5(init.txt)= ff52d0607e53de9993f472a22fd35fb3
```

HMAC-SHA1 command: *openssl dgst -sha1 -hmac “youwannaknow28121996” init.txt*
Screenshot

```
corei99@corei99:~/Downloads/Security_Assignment/Task3/Problem5$ openssl dgst -sh
a1 -hmac "youwannaknow28121996" init.txt
HMAC-SHA1(init.txt)= bec880644cf020676ac2a51628393e2292e6a077
```

HMAC-SHA256 command: *openssl dgst -sha256 -hmac “youwannaknow28121996” init.txt*
Screenshot

```
corei99@corei99:~/Downloads/Security_Assignment/Task3/Problem5$ openssl dgst -sh
a256 -hmac "youwannaknow28121996" init.txt
HMAC-SHA256(init.txt)= 25f4c3ed4afb161b60cc8e2aa1e4ac32b9db6691a47d6dca98ffa871b
040fe7e
```