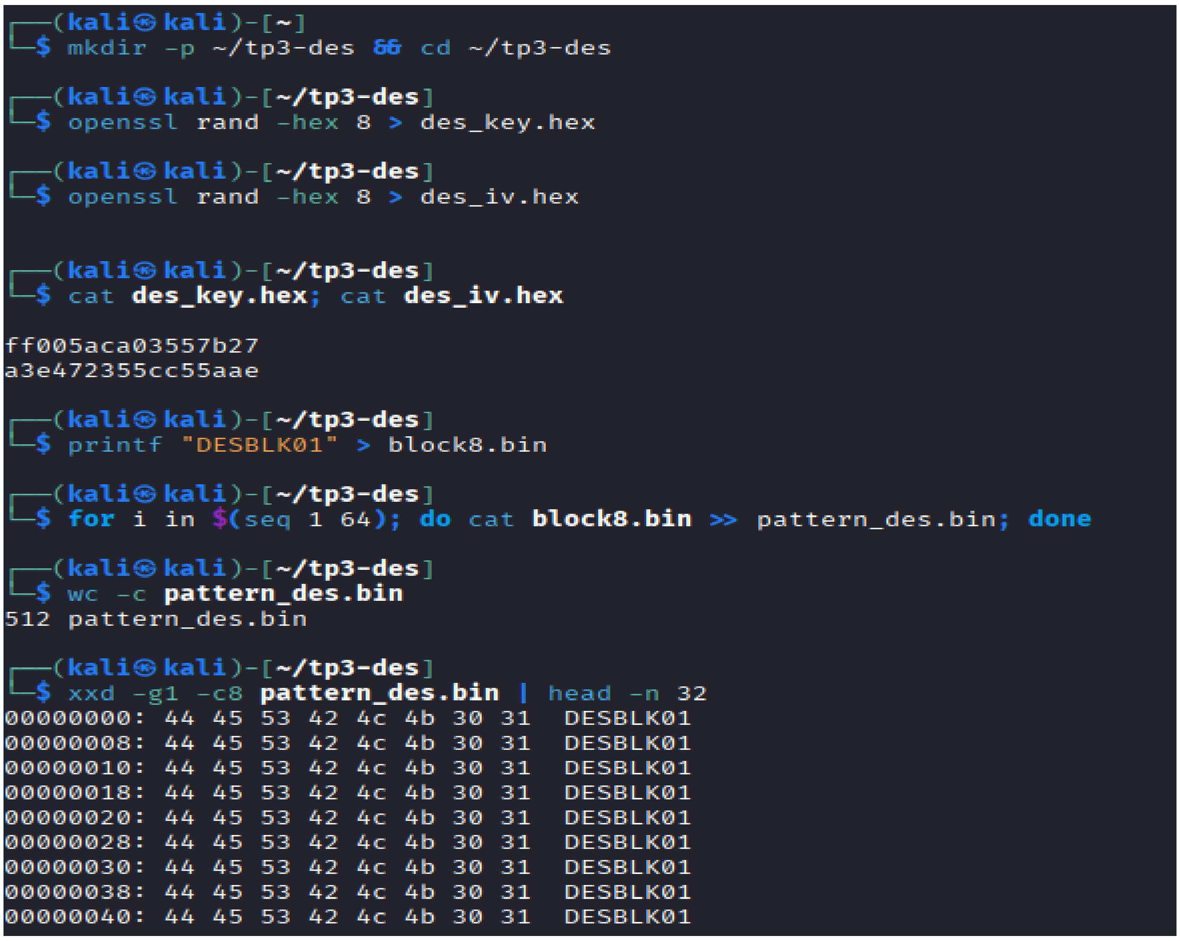
Ecole Supérieure d'Informatique et du Numérique
COLLEGE OF ENGINEERING & ARCHITECTURE

Université Internationale de Rabat
THE INNOVATIVE UNIVERSITY

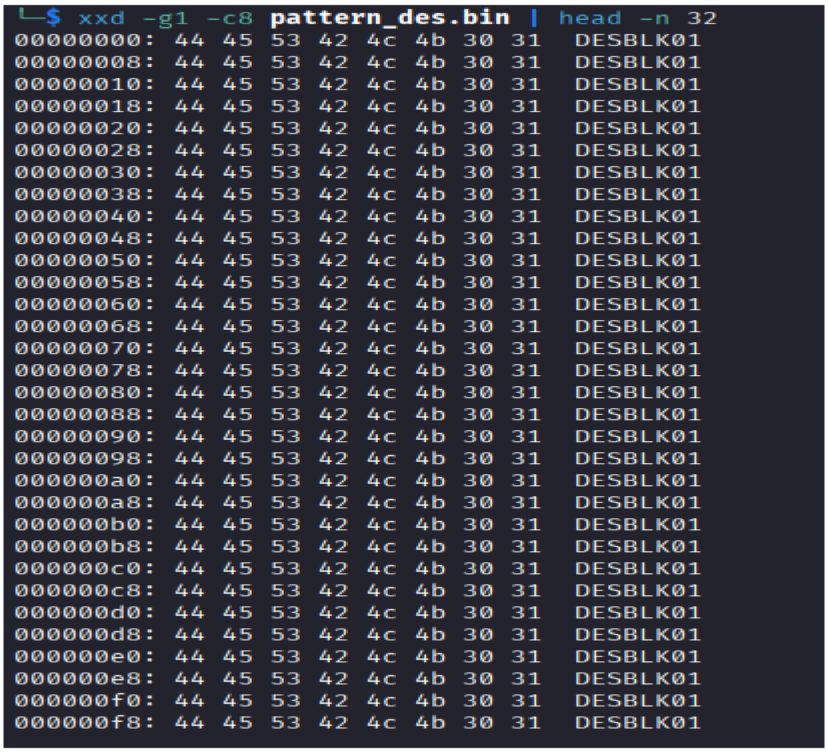
Prepared by: EL M'RABET Zineb & EL BEKKALI Wissal

Lab 4: Block Cipher Modes of Operation: DES and AES (ECB vs CBC)

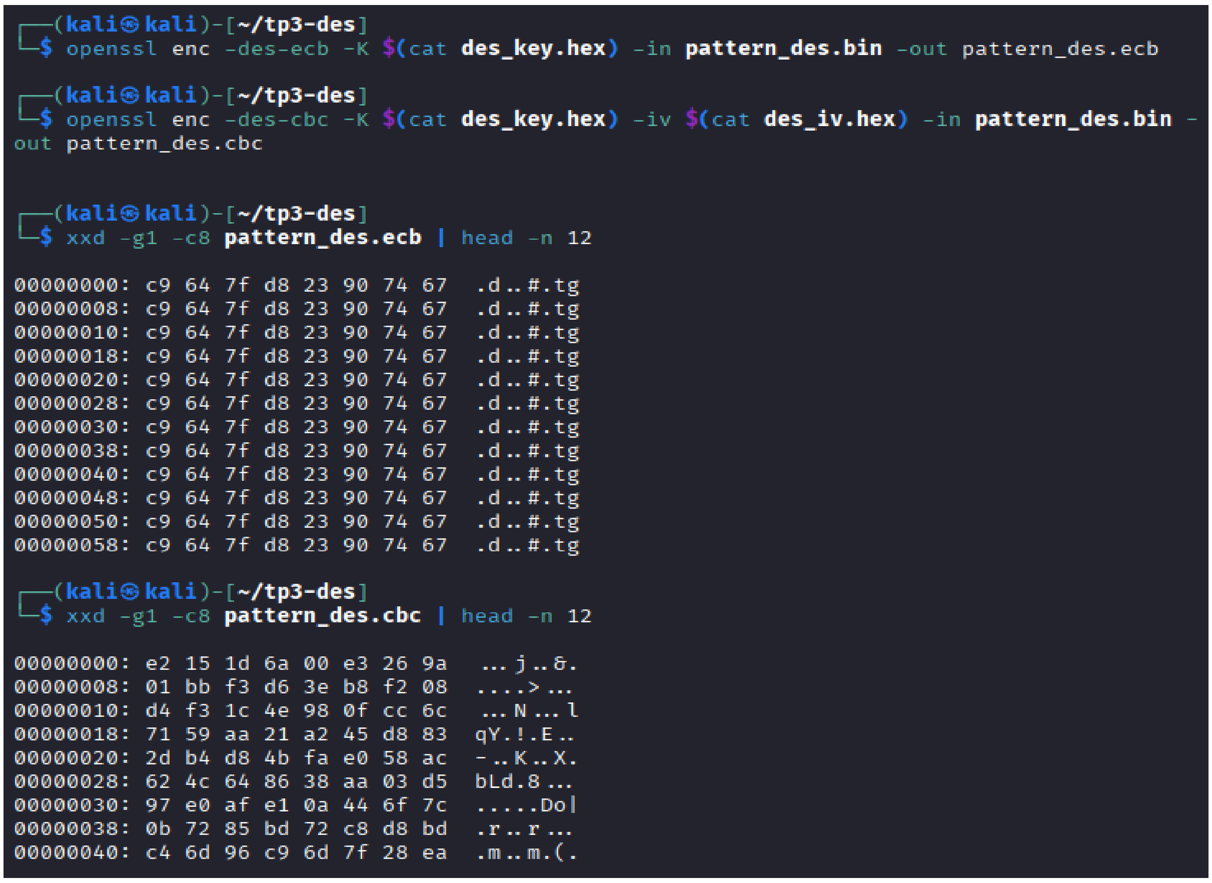
Exercise 1 : Comparing DES in ECB and CBC modes



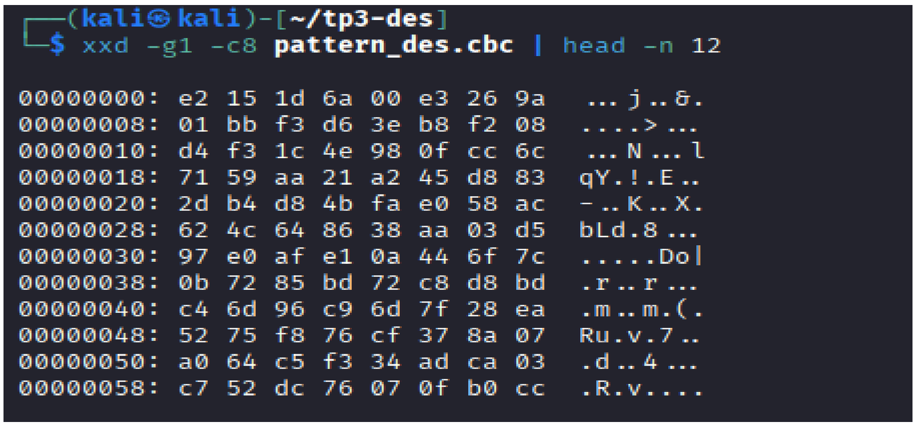
```
┌──(kali㉿kali)-[~]
└─$ mkdir -p ~/tp3-des && cd ~/tp3-des

┌──(kali㉿kali)-[~/tp3-des]
└─$ openssl rand -hex 8 > des_key.hex

┌──(kali㉿kali)-[~/tp3-des]
└─$ openssl rand -hex 8 > des_iv.hex


┌──(kali㉿kali)-[~/tp3-des]
└─$ cat des_key.hex; cat des_iv.hex

ff005aca03557b27
a3e472355cc55aae

┌──(kali㉿kali)-[~/tp3-des]
└─$ printf "DESBLK01" > block8.bin

┌──(kali㉿kali)-[~/tp3-des]
└─$ for i in $(seq 1 64); do cat block8.bin >> pattern_des.bin; done

┌──(kali㉿kali)-[~/tp3-des]
└─$ wc -c pattern_des.bin
512 pattern_des.bin

┌──(kali㉿kali)-[~/tp3-des]
└─$ xxd -g1 -c8 pattern_des.bin | head -n 32
00000000: 44 45 53 42 4c 4b 30 31  DESBLK01
00000008: 44 45 53 42 4c 4b 30 31  DESBLK01
00000010: 44 45 53 42 4c 4b 30 31  DESBLK01
00000018: 44 45 53 42 4c 4b 30 31  DESBLK01
00000020: 44 45 53 42 4c 4b 30 31  DESBLK01
00000028: 44 45 53 42 4c 4b 30 31  DESBLK01
00000030: 44 45 53 42 4c 4b 30 31  DESBLK01
00000038: 44 45 53 42 4c 4b 30 31  DESBLK01
00000040: 44 45 53 42 4c 4b 30 31  DESBLK01
```



```
└─$ xxd -g1 -c8 pattern_des.bin | head -n 32
00000000: 44 45 53 42 4c 4b 30 31  DESBLK01
00000008: 44 45 53 42 4c 4b 30 31  DESBLK01
00000010: 44 45 53 42 4c 4b 30 31  DESBLK01
00000018: 44 45 53 42 4c 4b 30 31  DESBLK01
00000020: 44 45 53 42 4c 4b 30 31  DESBLK01
00000028: 44 45 53 42 4c 4b 30 31  DESBLK01
00000030: 44 45 53 42 4c 4b 30 31  DESBLK01
00000038: 44 45 53 42 4c 4b 30 31  DESBLK01
00000040: 44 45 53 42 4c 4b 30 31  DESBLK01
00000048: 44 45 53 42 4c 4b 30 31  DESBLK01
00000050: 44 45 53 42 4c 4b 30 31  DESBLK01
00000058: 44 45 53 42 4c 4b 30 31  DESBLK01
00000060: 44 45 53 42 4c 4b 30 31  DESBLK01
00000068: 44 45 53 42 4c 4b 30 31  DESBLK01
00000070: 44 45 53 42 4c 4b 30 31  DESBLK01
00000078: 44 45 53 42 4c 4b 30 31  DESBLK01
00000080: 44 45 53 42 4c 4b 30 31  DESBLK01
00000088: 44 45 53 42 4c 4b 30 31  DESBLK01
00000090: 44 45 53 42 4c 4b 30 31  DESBLK01
00000098: 44 45 53 42 4c 4b 30 31  DESBLK01
000000a0: 44 45 53 42 4c 4b 30 31  DESBLK01
000000a8: 44 45 53 42 4c 4b 30 31  DESBLK01
000000b0: 44 45 53 42 4c 4b 30 31  DESBLK01
000000b8: 44 45 53 42 4c 4b 30 31  DESBLK01
000000c0: 44 45 53 42 4c 4b 30 31  DESBLK01
000000c8: 44 45 53 42 4c 4b 30 31  DESBLK01
000000d0: 44 45 53 42 4c 4b 30 31  DESBLK01
000000d8: 44 45 53 42 4c 4b 30 31  DESBLK01
000000e0: 44 45 53 42 4c 4b 30 31  DESBLK01
000000e8: 44 45 53 42 4c 4b 30 31  DESBLK01
000000f0: 44 45 53 42 4c 4b 30 31  DESBLK01
000000f8: 44 45 53 42 4c 4b 30 31  DESBLK01
```



```
┌──(kali㉿kali)-[~/tp3-des]
└─$ openssl enc -des-ecb -K $(cat des_key.hex) -in pattern_des.bin -out pattern_des.ecb

┌──(kali㉿kali)-[~/tp3-des]
└─$ openssl enc -des-cbc -K $(cat des_key.hex) -iv $(cat des_iv.hex) -in pattern_des.bin -
out pattern_des.cbc


┌──(kali㉿kali)-[~/tp3-des]
└─$ xxd -g1 -c8 pattern_des.ecb | head -n 12

00000000: c9 64 7f d8 23 90 74 67  .d..#.tg
00000008: c9 64 7f d8 23 90 74 67  .d..#.tg
00000010: c9 64 7f d8 23 90 74 67  .d..#.tg
00000018: c9 64 7f d8 23 90 74 67  .d..#.tg
00000020: c9 64 7f d8 23 90 74 67  .d..#.tg
00000028: c9 64 7f d8 23 90 74 67  .d..#.tg
00000030: c9 64 7f d8 23 90 74 67  .d..#.tg
00000038: c9 64 7f d8 23 90 74 67  .d..#.tg
00000040: c9 64 7f d8 23 90 74 67  .d..#.tg
00000048: c9 64 7f d8 23 90 74 67  .d..#.tg
00000050: c9 64 7f d8 23 90 74 67  .d..#.tg
00000058: c9 64 7f d8 23 90 74 67  .d..#.tg

┌──(kali㉿kali)-[~/tp3-des]
└─$ xxd -g1 -c8 pattern_des.cbc | head -n 12

00000000: e2 15 1d 6a 00 e3 26 9a  ...j..&.
00000008: 01 bb f3 d6 3e b8 f2 08  ....>...
00000010: d4 f3 1c 4e 98 0f cc 6c  ...N...l
00000018: 71 59 aa 21 a2 45 d8 83  qY.!.E..
00000020: 2d b4 d8 4b fa e0 58 ac  -..K..X.
00000028: 62 4c 64 86 38 aa 03 d5  bLd.8...
00000030: 97 e0 af e1 0a 44 6f 7c  .....Do|
00000038: 0b 72 85 bd 72 c8 d8 bd  .r..r...
00000040: c4 6d 96 c9 6d 7f 28 ea  .m..m.(.
```



```
┌──(kali㉿kali)-[~/tp3-des]
└─$ xxd -g1 -c8 pattern_des.cbc | head -n 12

00000000: e2 15 1d 6a 00 e3 26 9a  ...j..&.
00000008: 01 bb f3 d6 3e b8 f2 08  ....>...
00000010: d4 f3 1c 4e 98 0f cc 6c  ...N...l
00000018: 71 59 aa 21 a2 45 d8 83  qY.!.E..
00000020: 2d b4 d8 4b fa e0 58 ac  -..K..X.
00000028: 62 4c 64 86 38 aa 03 d5  bLd.8...
00000030: 97 e0 af e1 0a 44 6f 7c  .....Do|
00000038: 0b 72 85 bd 72 c8 d8 bd  .r..r...
00000040: c4 6d 96 c9 6d 7f 28 ea  .m..m.(.
00000048: 52 75 f8 76 cf 37 8a 07  Ru.v.7..
00000050: a0 64 c5 f3 34 ad ca 03  .d..4...
00000058: c7 52 dc 76 07 0f b0 cc  .R.v....
```

Questions:

Q1. ECB shows repeated ciphertext blocks when identical plaintext blocks exist (same plaintext block → same ciphertext block). CBC does not show that visible repetition.

Q2. ECB encrypts each block independently: same key and same block-cipher operation, so identical plaintext blocks map to identical ciphertext blocks.

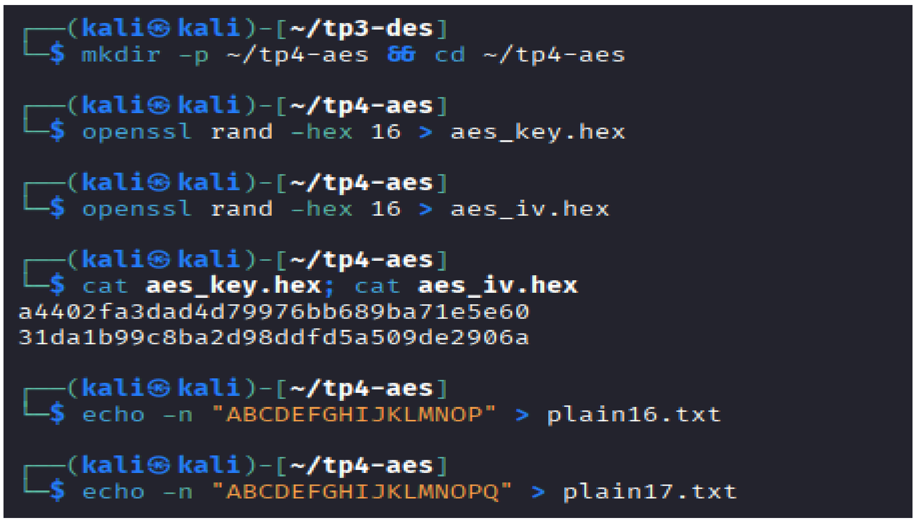
Q3. In CBC each plaintext block is XORed with the previous ciphertext block before encryption. That chaining makes identical plaintext blocks produce different ciphertexts when their preceding ciphertexts differ.

Q4. The Initialization Vector (IV) supplies randomness for the first block. A different IV yields a different ciphertext even for the same plaintext and key.

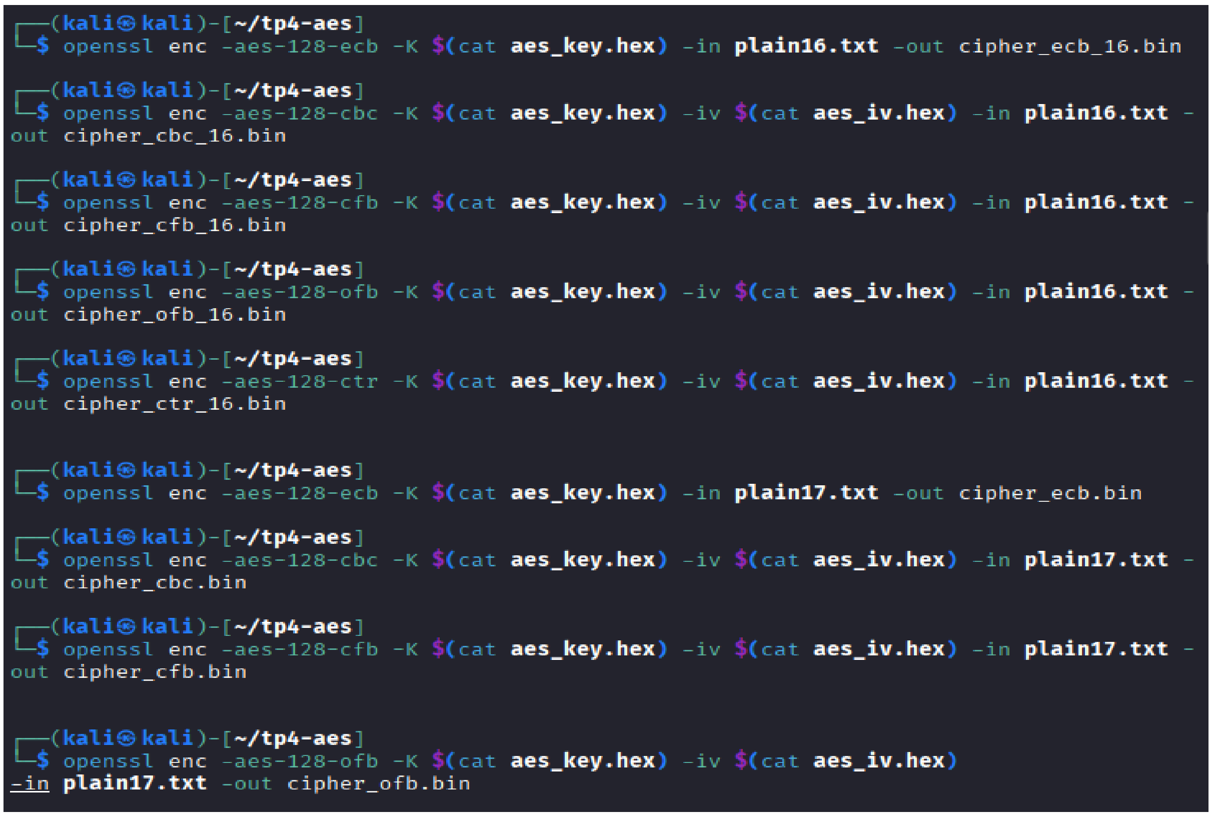
Q5. Reusing an IV with the same key can reveal relationships between messages (e.g., equal first plaintext blocks produce equal first ciphertext blocks). In stream-like modes, reusing a nonce/IV can be catastrophic (keystream reuse).

Q6. Because ECB preserves block-to-block correspondence: repeating structures in plaintext remain visible in ciphertext (e.g., encrypted images still show shapes). ECB lacks sufficient diffusion between blocks, so it is not suitable for structured and visual data.

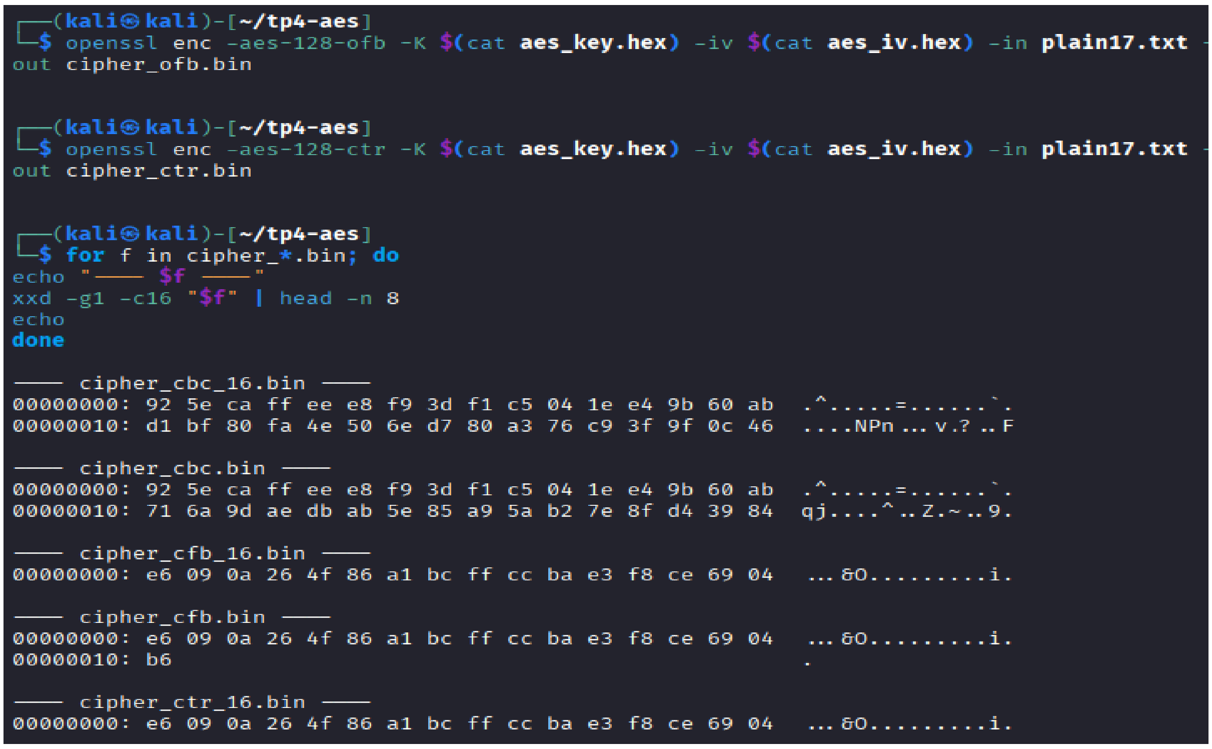
Exercise 2 : AES Padding and Operation Modes (ECB, CBC, CFB, OFB, CTR)



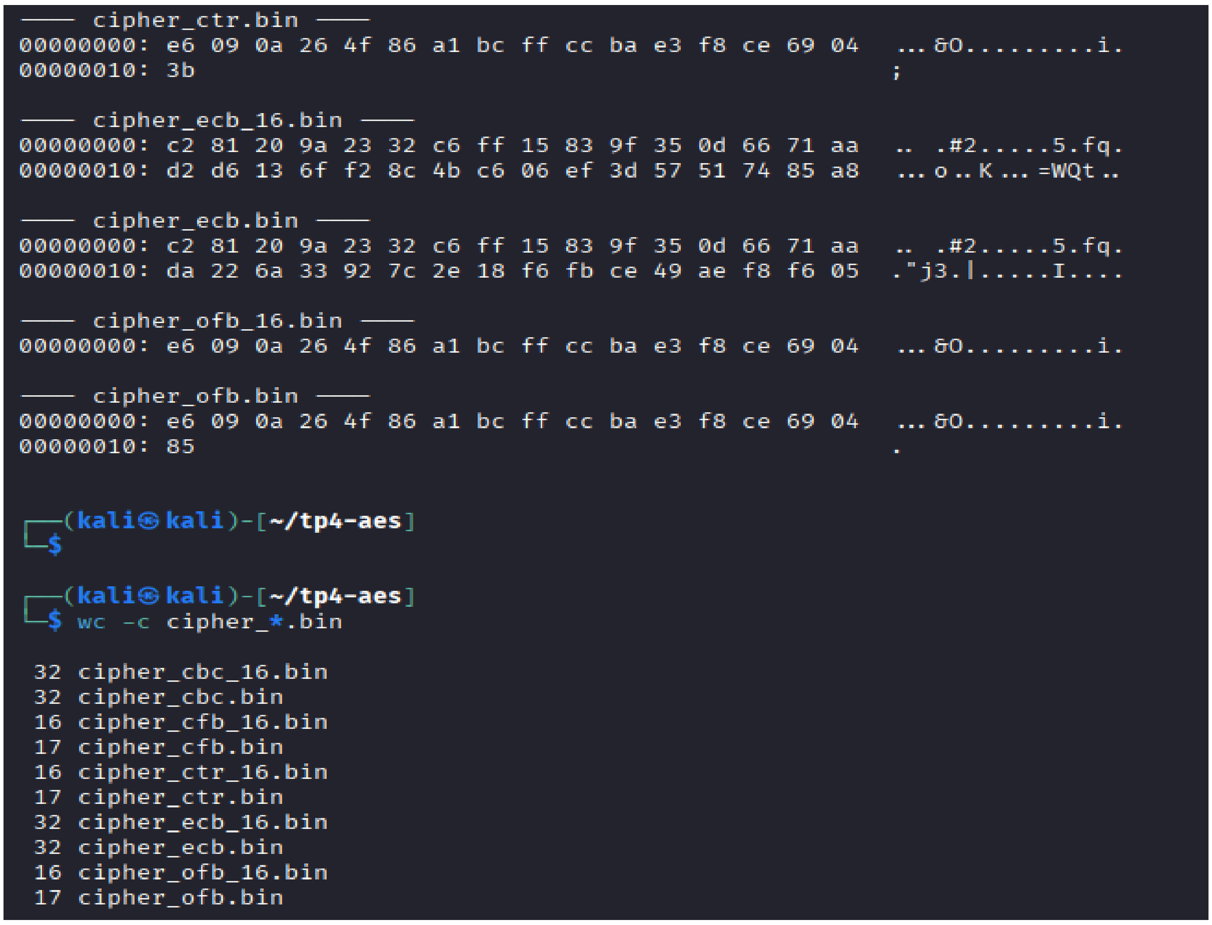
```
┌──(kali㉿kali)-[~/tp3-des]
└─$ mkdir -p ~/tp4-aes && cd ~/tp4-aes

┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl rand -hex 16 > aes_key.hex

┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl rand -hex 16 > aes_iv.hex

┌──(kali㉿kali)-[~/tp4-aes]
└─$ cat aes_key.hex; cat aes_iv.hex
a4402fa3dad4d79976bb689ba71e5e60
31da1b99c8ba2d98ddfd5a509de2906a

┌──(kali㉿kali)-[~/tp4-aes]
└─$ echo -n "ABCDEFGHIJKLMNOP" > plain16.txt

┌──(kali㉿kali)-[~/tp4-aes]
└─$ echo -n "ABCDEFGHIJKLMNOPQ" > plain17.txt
```



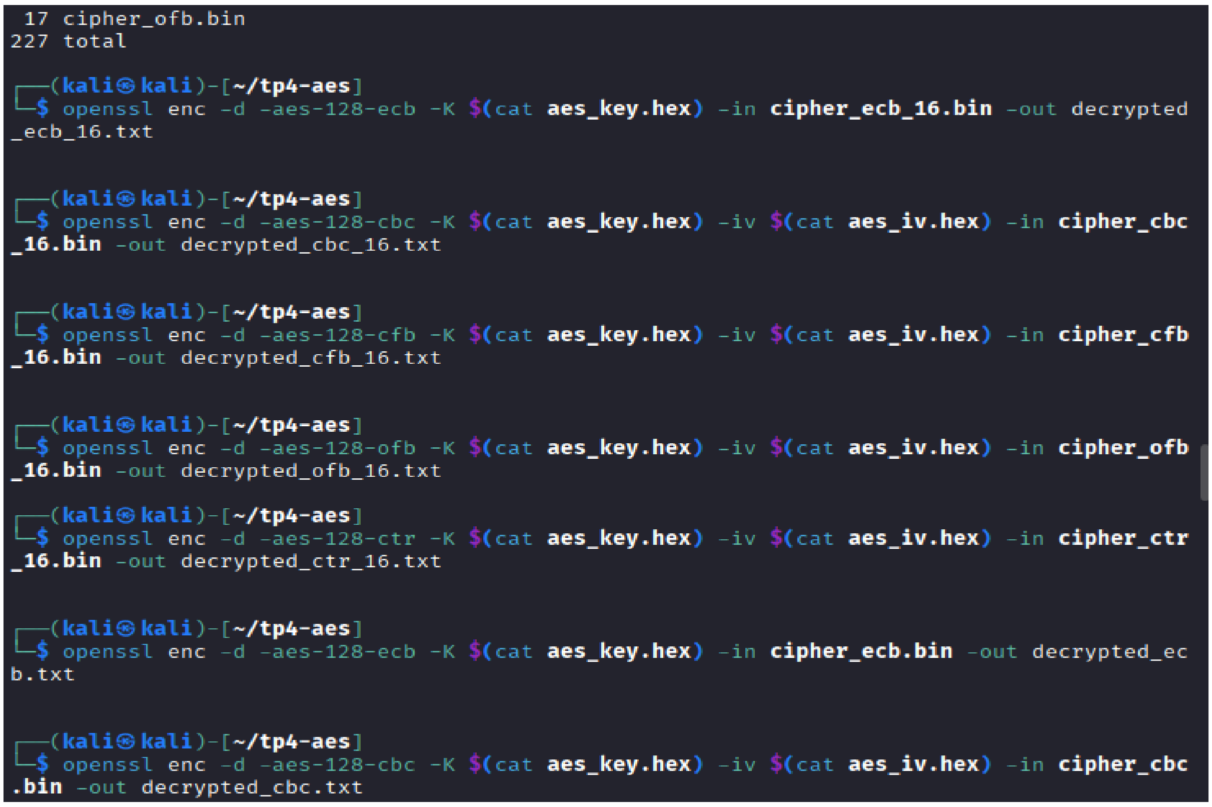
```
┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl enc -aes-128-ecb -K $(cat aes_key.hex) -in plain16.txt -out cipher_ecb_16.bin

┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl enc -aes-128-cbc -K $(cat aes_key.hex) -iv $(cat aes_iv.hex) -in plain16.txt -
out cipher_cbc_16.bin

┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl enc -aes-128-cfb -K $(cat aes_key.hex) -iv $(cat aes_iv.hex) -in plain16.txt -
out cipher_cfb_16.bin

┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl enc -aes-128-ofb -K $(cat aes_key.hex) -iv $(cat aes_iv.hex) -in plain16.txt -
out cipher_ofb_16.bin

┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl enc -aes-128-ctr -K $(cat aes_key.hex) -iv $(cat aes_iv.hex) -in plain16.txt -
out cipher_ctr_16.bin


┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl enc -aes-128-ecb -K $(cat aes_key.hex) -in plain17.txt -out cipher_ecb.bin

┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl enc -aes-128-cbc -K $(cat aes_key.hex) -iv $(cat aes_iv.hex) -in plain17.txt -
out cipher_cbc.bin

┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl enc -aes-128-cfb -K $(cat aes_key.hex) -iv $(cat aes_iv.hex) -in plain17.txt -
out cipher_cfb.bin


┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl enc -aes-128-ofb -K $(cat aes_key.hex) -iv $(cat aes_iv.hex)
-in plain17.txt -out cipher_ofb.bin
```



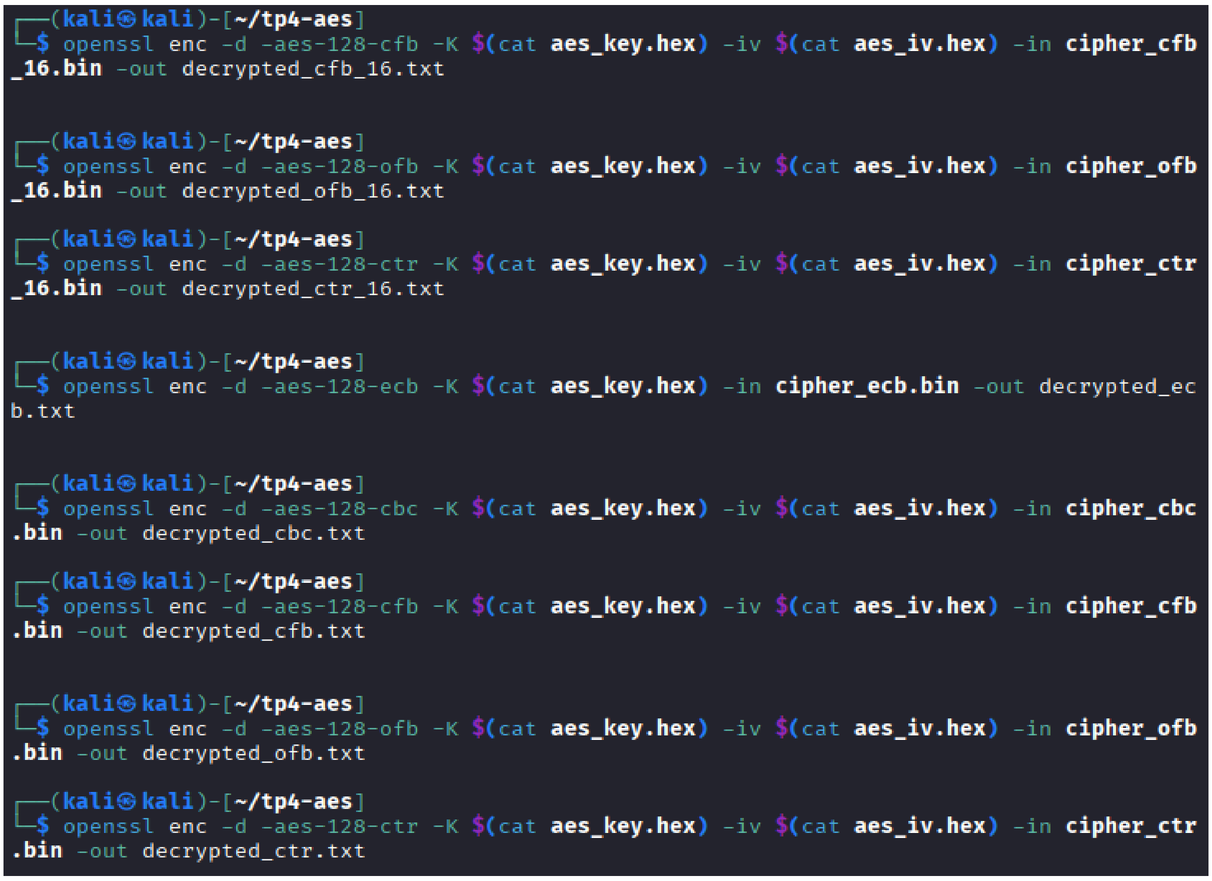
```
┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl enc -aes-128-ofb -K $(cat aes_key.hex) -iv $(cat aes_iv.hex) -in plain17.txt -
out cipher_ofb.bin


┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl enc -aes-128-ctr -K $(cat aes_key.hex) -iv $(cat aes_iv.hex) -in plain17.txt -
out cipher_ctr.bin


┌──(kali㉿kali)-[~/tp4-aes]
└─$ for f in cipher_*.bin; do
echo "——— $f ———"
xxd -g1 -c16 "$f" | head -n 8
echo
done

——— cipher_cbc_16.bin ———
00000000: 92 5e ca ff ee e8 f9 3d f1 c5 04 1e e4 9b 60 ab  .^.....=......`.
00000010: d1 bf 80 fa 4e 50 6e d7 80 a3 76 c9 3f 9f 0c 46  ....NPn...v.?..F

——— cipher_cbc.bin ———
00000000: 92 5e ca ff ee e8 f9 3d f1 c5 04 1e e4 9b 60 ab  .^.....=......`.
00000010: 71 6a 9d ae db ab 5e 85 a9 5a b2 7e 8f d4 39 84  qj....^..Z.~..9.

——— cipher_cfb_16.bin ———
00000000: e6 09 0a 26 4f 86 a1 bc ff cc ba e3 f8 ce 69 04  ...&O.........i.

——— cipher_cfb.bin ———
00000000: e6 09 0a 26 4f 86 a1 bc ff cc ba e3 f8 ce 69 04  ...&O.........i.
00000010: b6                                               .

——— cipher_ctr_16.bin ———
00000000: e6 09 0a 26 4f 86 a1 bc ff cc ba e3 f8 ce 69 04  ...&O.........i.
```



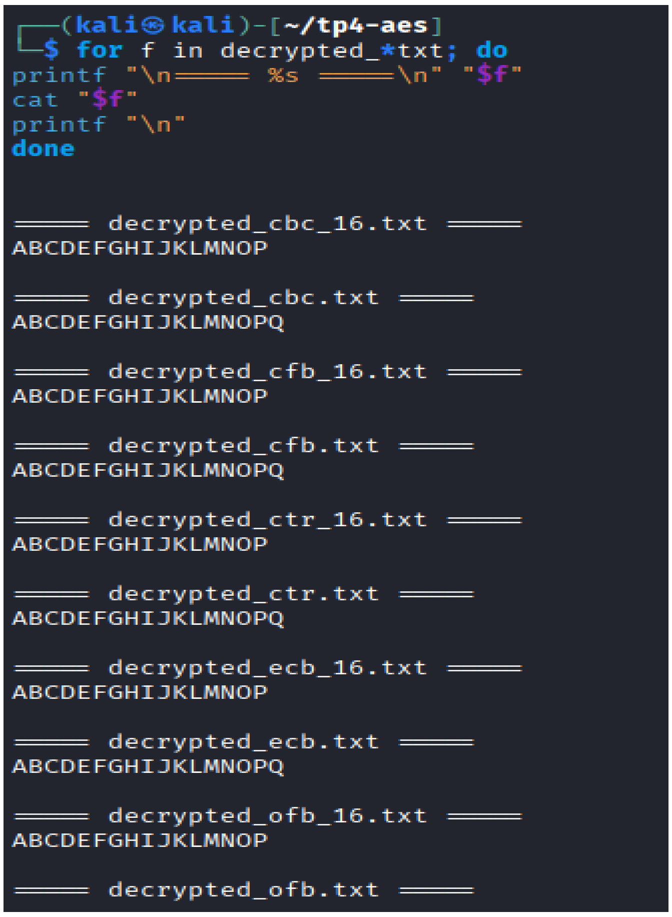
```
——— cipher_ctr.bin ———
00000000: e6 09 0a 26 4f 86 a1 bc ff cc ba e3 f8 ce 69 04  ...&O.........i.
00000010: 3b                                               ;

——— cipher_ecb_16.bin ———
00000000: c2 81 20 9a 23 32 c6 ff 15 83 9f 35 0d 66 71 aa  .. .#2.....5.fq.
00000010: d2 d6 13 6f f2 8c 4b c6 06 ef 3d 57 51 74 85 a8  ...o..K...=WQt..

——— cipher_ecb.bin ———
00000000: c2 81 20 9a 23 32 c6 ff 15 83 9f 35 0d 66 71 aa  .. .#2.....5.fq.
00000010: da 22 6a 33 92 7c 2e 18 f6 fb ce 49 ae f8 f6 05  ."j3.|.....I....

——— cipher_ofb_16.bin ———
00000000: e6 09 0a 26 4f 86 a1 bc ff cc ba e3 f8 ce 69 04  ...&O.........i.

——— cipher_ofb.bin ———
00000000: e6 09 0a 26 4f 86 a1 bc ff cc ba e3 f8 ce 69 04  ...&O.........i.
00000010: 85                                               .


┌──(kali㉿kali)-[~/tp4-aes]
└─$

┌──(kali㉿kali)-[~/tp4-aes]
└─$ wc -c cipher_*.bin

 32 cipher_cbc_16.bin
 32 cipher_cbc.bin
 16 cipher_cfb_16.bin
 17 cipher_cfb.bin
 16 cipher_ctr_16.bin
 17 cipher_ctr.bin
 32 cipher_ecb_16.bin
 32 cipher_ecb.bin
 16 cipher_ofb_16.bin
 17 cipher_ofb.bin
```



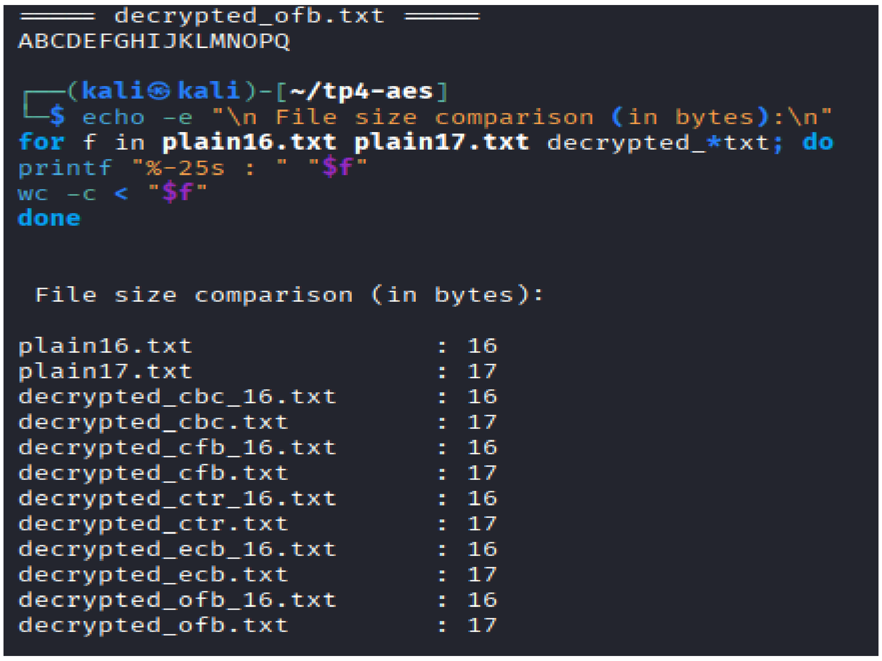
```
 17 cipher_ofb.bin
227 total

┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl enc -d -aes-128-ecb -K $(cat aes_key.hex) -in cipher_ecb_16.bin -out decrypted
_ecb_16.txt


┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl enc -d -aes-128-cbc -K $(cat aes_key.hex) -iv $(cat aes_iv.hex) -in cipher_cbc
_16.bin -out decrypted_cbc_16.txt


┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl enc -d -aes-128-cfb -K $(cat aes_key.hex) -iv $(cat aes_iv.hex) -in cipher_cfb
_16.bin -out decrypted_cfb_16.txt


┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl enc -d -aes-128-ofb -K $(cat aes_key.hex) -iv $(cat aes_iv.hex) -in cipher_ofb
_16.bin -out decrypted_ofb_16.txt

┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl enc -d -aes-128-ctr -K $(cat aes_key.hex) -iv $(cat aes_iv.hex) -in cipher_ctr
_16.bin -out decrypted_ctr_16.txt


┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl enc -d -aes-128-ecb -K $(cat aes_key.hex) -in cipher_ecb.bin -out decrypted_ec
b.txt


┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl enc -d -aes-128-cbc -K $(cat aes_key.hex) -iv $(cat aes_iv.hex) -in cipher_cbc
.bin -out decrypted_cbc.txt
```



```
┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl enc -d -aes-128-cfb -K $(cat aes_key.hex) -iv $(cat aes_iv.hex) -in cipher_cfb
_16.bin -out decrypted_cfb_16.txt


┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl enc -d -aes-128-ofb -K $(cat aes_key.hex) -iv $(cat aes_iv.hex) -in cipher_ofb
_16.bin -out decrypted_ofb_16.txt

┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl enc -d -aes-128-ctr -K $(cat aes_key.hex) -iv $(cat aes_iv.hex) -in cipher_ctr
_16.bin -out decrypted_ctr_16.txt


┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl enc -d -aes-128-ecb -K $(cat aes_key.hex) -in cipher_ecb.bin -out decrypted_ec
b.txt


┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl enc -d -aes-128-cbc -K $(cat aes_key.hex) -iv $(cat aes_iv.hex) -in cipher_cbc
.bin -out decrypted_cbc.txt

┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl enc -d -aes-128-cfb -K $(cat aes_key.hex) -iv $(cat aes_iv.hex) -in cipher_cfb
.bin -out decrypted_cfb.txt


┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl enc -d -aes-128-ofb -K $(cat aes_key.hex) -iv $(cat aes_iv.hex) -in cipher_ofb
.bin -out decrypted_ofb.txt

┌──(kali㉿kali)-[~/tp4-aes]
└─$ openssl enc -d -aes-128-ctr -K $(cat aes_key.hex) -iv $(cat aes_iv.hex) -in cipher_ctr
.bin -out decrypted_ctr.txt
```



```
┌──(kali㉿kali)-[~/tp4-aes]
└─$ for f in decrypted_*txt; do
printf "\n═════ %s ═════\n" "$f"
cat "$f"
printf "\n"
done


═════ decrypted_cbc_16.txt ═════
ABCDEFGHIJKLMNOP

═════ decrypted_cbc.txt ═════
ABCDEFGHIJKLMNOPQ

═════ decrypted_cfb_16.txt ═════
ABCDEFGHIJKLMNOP

═════ decrypted_cfb.txt ═════
ABCDEFGHIJKLMNOPQ

═════ decrypted_ctr_16.txt ═════
ABCDEFGHIJKLMNOP

═════ decrypted_ctr.txt ═════
ABCDEFGHIJKLMNOPQ

═════ decrypted_ecb_16.txt ═════
ABCDEFGHIJKLMNOP

═════ decrypted_ecb.txt ═════
ABCDEFGHIJKLMNOPQ

═════ decrypted_ofb_16.txt ═════
ABCDEFGHIJKLMNOP

═════ decrypted_ofb.txt ═════
```



```
═════ decrypted_ofb.txt ═════
ABCDEFGHIJKLMNOPQ

┌──(kali㉿kali)-[~/tp4-aes]
└─$ echo -e "\n File size comparison (in bytes):\n"
for f in plain16.txt plain17.txt decrypted_*txt; do
printf "%-25s : " "$f"
wc -c < "$f"
done


 File size comparison (in bytes):

plain16.txt               : 16
plain17.txt               : 17
decrypted_cbc_16.txt      : 16
decrypted_cbc.txt         : 17
decrypted_cfb_16.txt      : 16
decrypted_cfb.txt         : 17
decrypted_ctr_16.txt      : 16
decrypted_ctr.txt         : 17
decrypted_ecb_16.txt      : 16
decrypted_ecb.txt         : 17
decrypted_ofb_16.txt      : 16
decrypted_ofb.txt         : 17
```

Mode	Uses IV	Padding	Repetition visible	Suitable for streams	Commentaire
AES-ECB	No	Yes	Yes	No	Encrypts each block independently. Patterns in the data are visible. Bad for repetitive data or images.
AES-CBC	Yes	Yes	No	No	Safer than ECB, requires padding. Sequential encryption, parallelizable decryption.
AES-CTR	Yes	No	No	Yes	Turns AES into a stream cipher. Parallelizable. Very efficient for streams and files.
AES-CFB	Yes	No	No	Yes	Turns AES into a stream cipher. Sequential encryption.
AES-OFB	Yes	No	No	Yes	Turns AES into a stream cipher. IV must be unique for security.

Questions:

Q1. ECB and CBC (block-oriented modes) require plaintext sizes that are multiples of the block size (AES block = 16 bytes). Padding schemes are used to fill the last block.

Q2.

- CBC: yes each ciphertext block depends on the previous ciphertext block (chaining).
- CFB: yes it has feedback behavior so blocks/segments depend on previous outputs.
- OFB and CTR: plaintext blocks are XORed with a keystream independent of the plaintext; the keystream generation is independent of the plaintext blocks (OFB uses feedback for keystream, CTR uses a counter), so plaintext blocks do not directly depend on earlier plaintext blocks.

Q3. Because ECB encrypts each block separately with no mixing between blocks. Identical input blocks produce identical cipher blocks, when data contains repeated patterns (like image tiles), those patterns remain discernible after encryption.

Q4.

- CTR: supports parallel encryption/decryption and allows random access to encrypted blocks, it's useful for high throughput and multi-core parallelism.
- OFB: produces a keystream that can be consumed byte-by-byte, making it suitable for low-latency streaming. Both avoid padding and are well-suited for streaming/real-time use-cases.

Q5. Which modes allow parallel encryption/decryption?

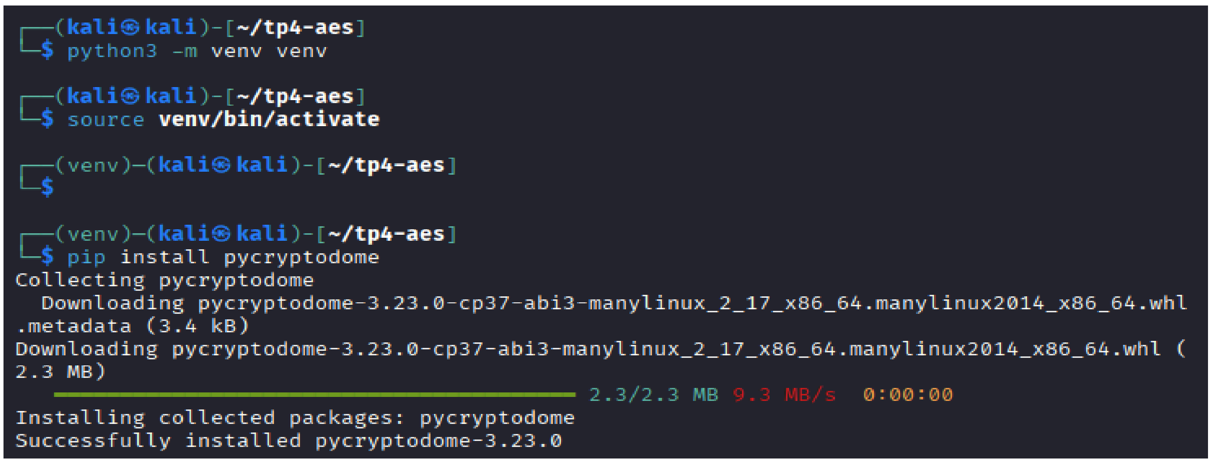
- CTR: both encryption and decryption are easily parallelizable (blocks independent).

- CBC: encryption is inherently sequential (each ciphertext depends on the previous), but decryption can be parallelized. OFB and CFB are generally sequential for keystream generation and less suited to parallel block encryption.

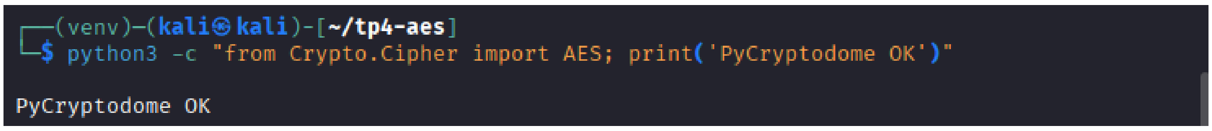
Q6.

- CBC/ECB: reusing the same IV (or using no IV) allows an observer to detect equal first blocks and can aid attacks.
- CTR (and other counter/nonce modes): reusing the same nonce/counter with the same key reuses the keystream; XOR of two ciphertexts equals the XOR of the plaintexts, this leaks everything that differs between messages.

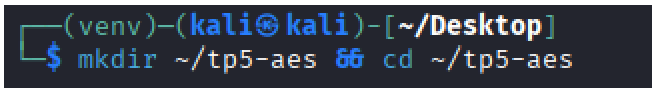
Exercise 3– ECB vs CBC Image Encryption (Python)



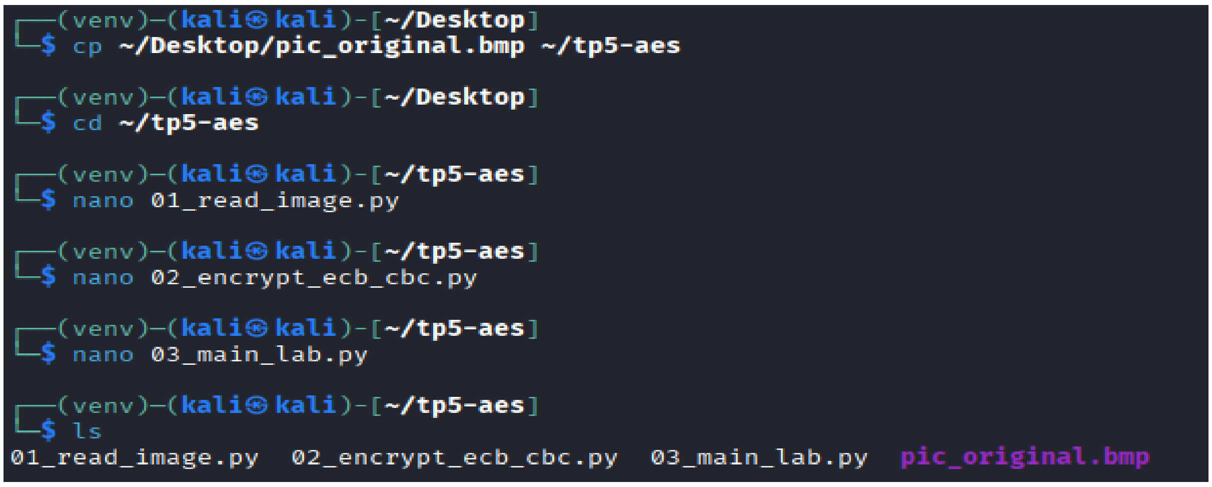
```
┌──(kali㉿kali)-[~/tp4-aes]
└─$ python3 -m venv venv

┌──(kali㉿kali)-[~/tp4-aes]
└─$ source venv/bin/activate

┌──(venv)─(kali㉿kali)-[~/tp4-aes]
└─$

┌──(venv)─(kali㉿kali)-[~/tp4-aes]
└─$ pip install pycryptodome
Collecting pycryptodome
  Downloading pycryptodome-3.23.0-cp37-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
.metadata (3.4 kB)
Downloading pycryptodome-3.23.0-cp37-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (
2.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.3/2.3 MB 9.3 MB/s  0:00:00
Installing collected packages: pycryptodome
Successfully installed pycryptodome-3.23.0
```



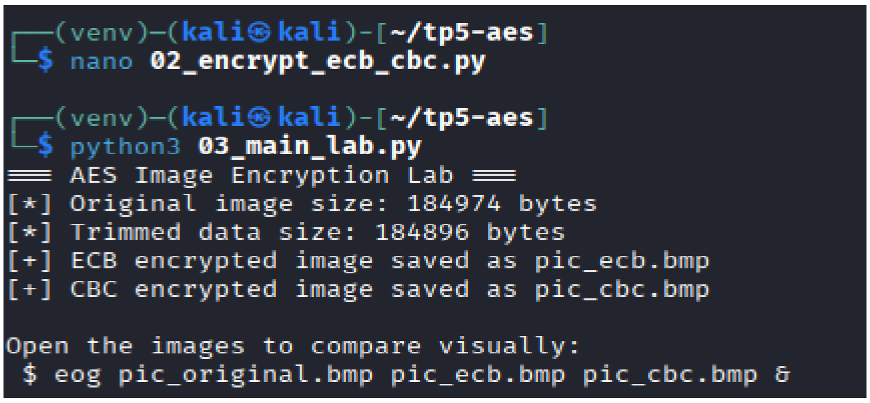
```
┌──(venv)─(kali㉿kali)-[~/tp4-aes]
└─$ python3 -c "from Crypto.Cipher import AES; print('PyCryptodome OK')"

PyCryptodome OK
```



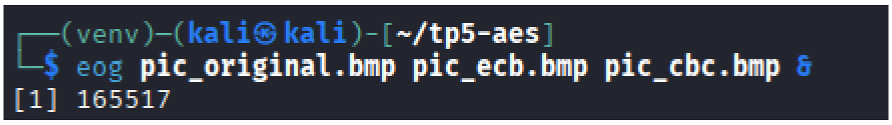
```
┌──(venv)─(kali㉿kali)-[~/Desktop]
└─$ mkdir ~/tp5-aes && cd ~/tp5-aes
```



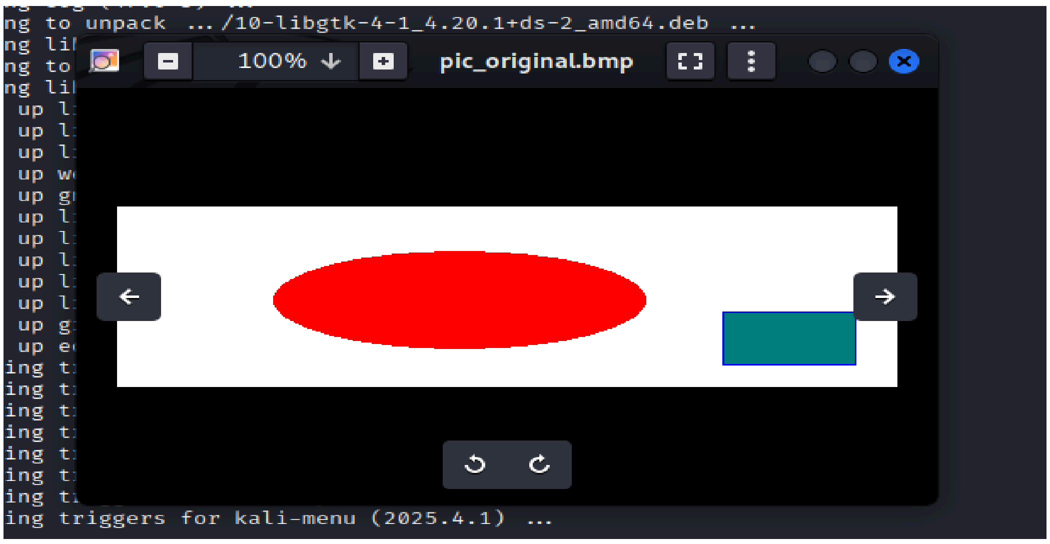
```
┌──(venv)─(kali㉿kali)-[~/Desktop]
└─$ cp ~/Desktop/pic_original.bmp ~/tp5-aes

┌──(venv)─(kali㉿kali)-[~/Desktop]
└─$ cd ~/tp5-aes

┌──(venv)─(kali㉿kali)-[~/tp5-aes]
└─$ nano 01_read_image.py

┌──(venv)─(kali㉿kali)-[~/tp5-aes]
└─$ nano 02_encrypt_ecb_cbc.py

┌──(venv)─(kali㉿kali)-[~/tp5-aes]
└─$ nano 03_main_lab.py

┌──(venv)─(kali㉿kali)-[~/tp5-aes]
└─$ ls
01_read_image.py  02_encrypt_ecb_cbc.py  03_main_lab.py  pic_original.bmp
```



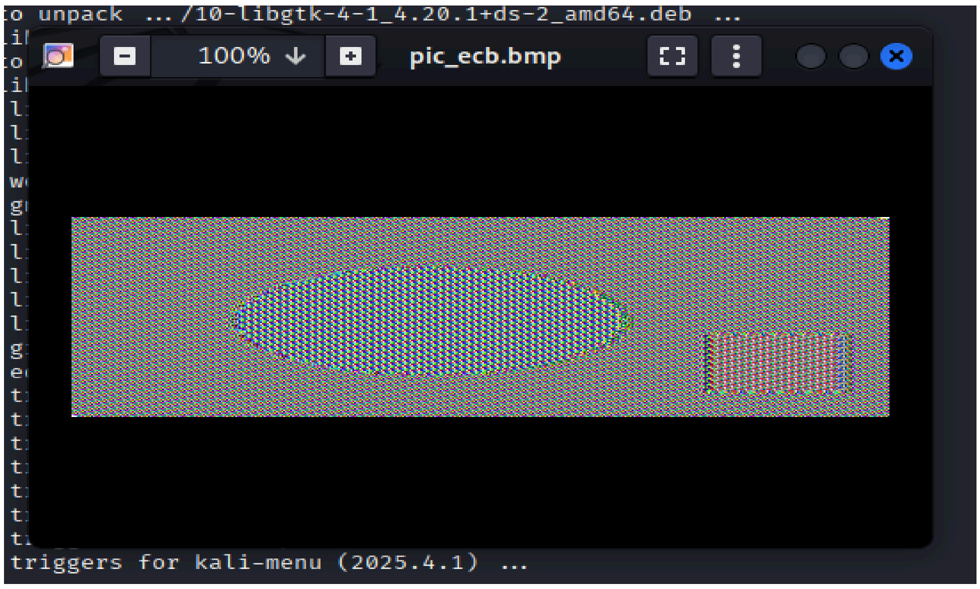
```
┌──(venv)─(kali㉿kali)-[~/tp5-aes]
└─$ nano 02_encrypt_ecb_cbc.py

┌──(venv)─(kali㉿kali)-[~/tp5-aes]
└─$ python3 03_main_lab.py
=== AES Image Encryption Lab ===
[*] Original image size: 184974 bytes
[*] Trimmed data size: 184896 bytes
[+] ECB encrypted image saved as pic_ecb.bmp
[+] CBC encrypted image saved as pic_cbc.bmp

Open the images to compare visually:
 $ eog pic_original.bmp pic_ecb.bmp pic_cbc.bmp &
```



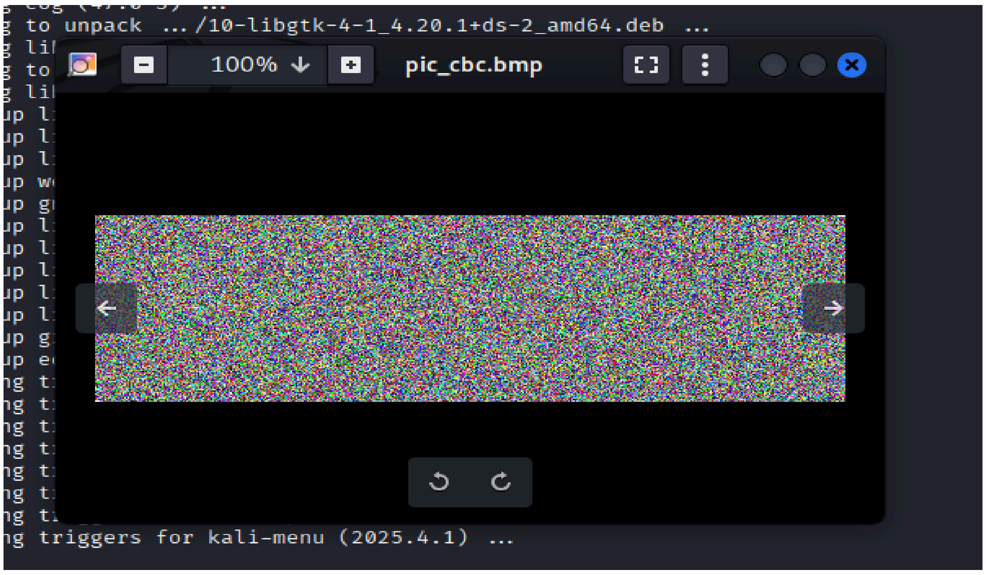
```
┌──(venv)─(kali㉿kali)-[~/tp5-aes]
└─$ eog pic_original.bmp pic_ecb.bmp pic_cbc.bmp &
[1] 165517
```



```
ng to unpack .../10-libgtk-4-1_4.20.1+ds-2_amd64.deb ...
ng li
ng to
ng li
 up l
 up l
 up l
 up w
 up g
 up l
 up l
 up l
 up l
 up l
 up g
 up e
ing t
ing t
ing t
ing t
ing t
ing t
ing t
ing triggers for kali-menu (2025.4.1) ...
```



```
o unpack .../10-libgtk-4-1_4.20.1+ds-2_amd64.deb ...
triggers for kali-menu (2025.4.1) ...
```



```
g to unpack .../10-libgtk-4-1_4.20.1+ds-2_amd64.deb ...
ng triggers for kali-menu (2025.4.1) ...
```

Questions:

Q1. Because identical plaintext blocks (image tiles) become identical ciphertext blocks, the image structure can still be perceived in the encrypted file.

Q2. CBC XORs each plaintext block with the previous ciphertext block (and the first block with the IV), so repeating image blocks are mapped to different ciphertext blocks and the visual structure is masked.

Q3. The IV provides randomness for the first block and ensures that encrypting the same message with the same key but a different IV produces a different ciphertext.

Q4. ECB offers no diffusion between blocks; it does not hide repeated structures or patterns

Comparaison:

AES in ECB mode encrypts each block of data independently, which makes it fast but insecure because identical plaintext blocks produce identical ciphertext, revealing patterns. In contrast, CBC mode links each block to the previous one using an Initialization Vector (IV), ensuring that even identical plaintext blocks produce different ciphertexts. This makes CBC much more secure and suitable for most encryption purposes.