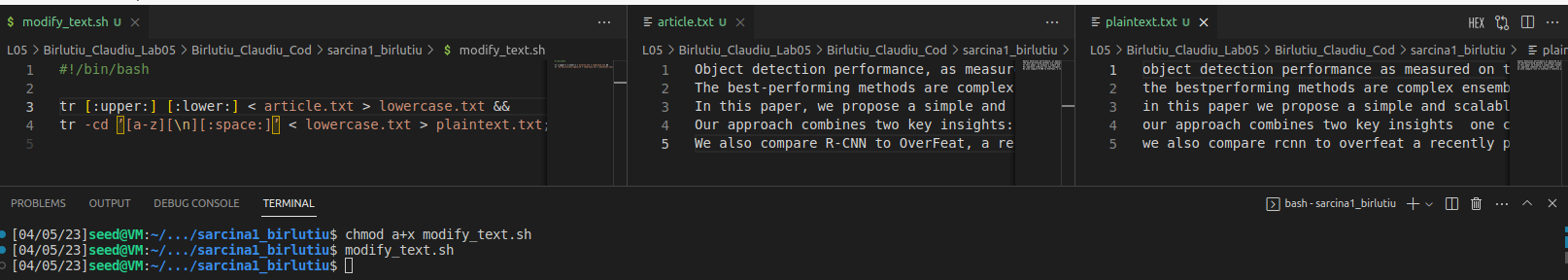
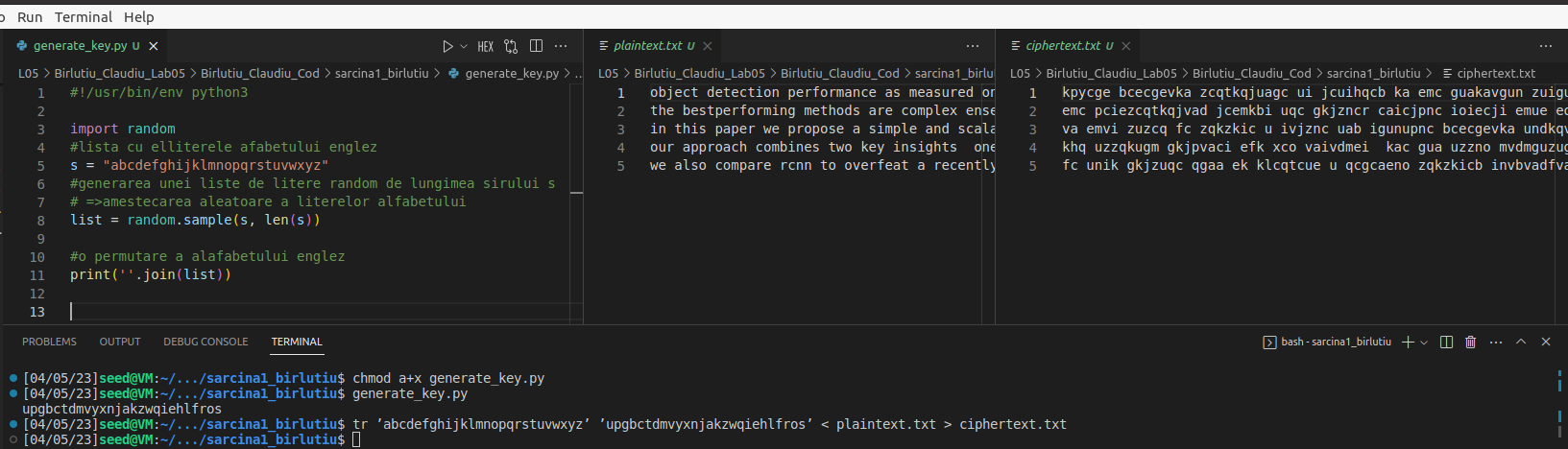
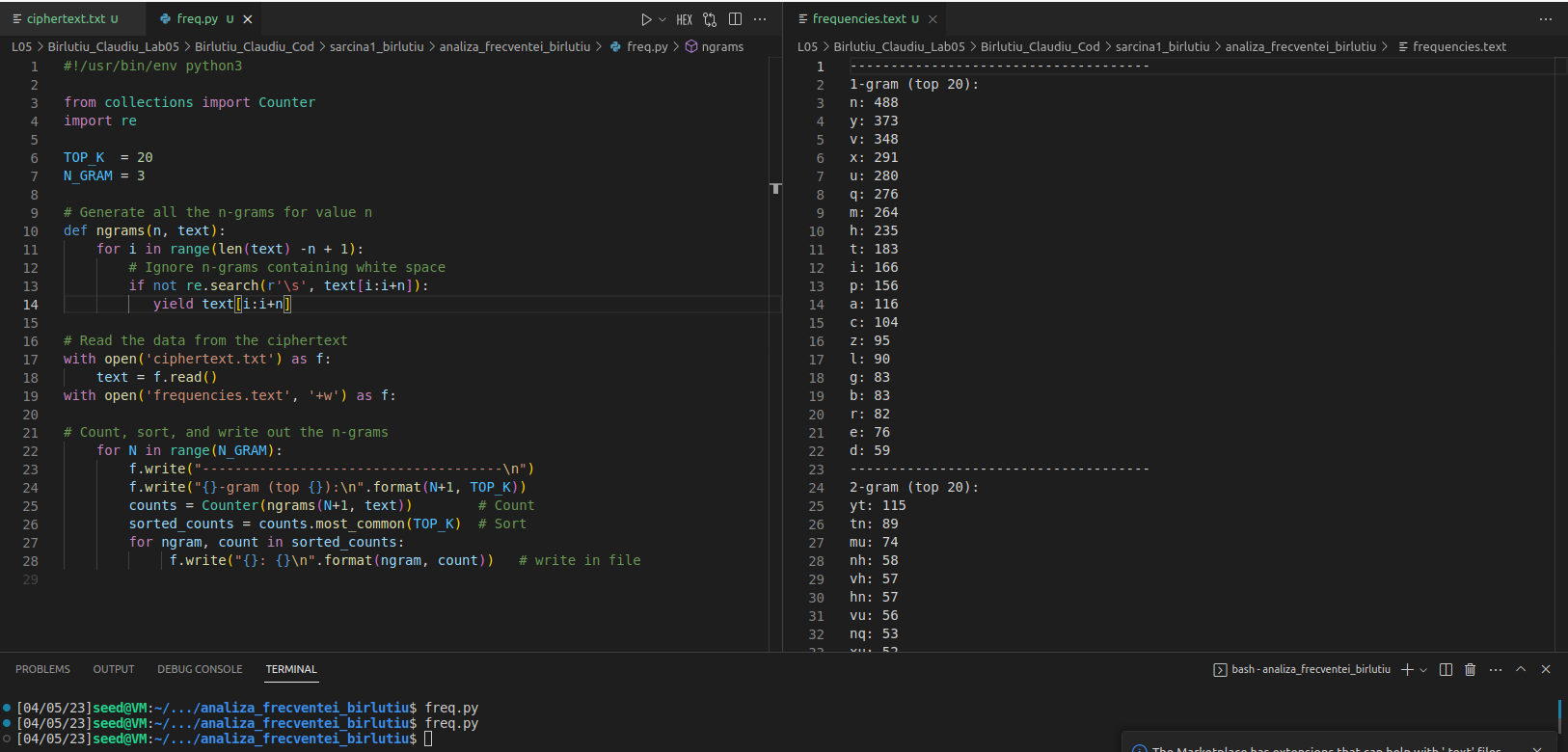
# Raport pentru lucrarea 5: Cifrarea cu cheie secretă

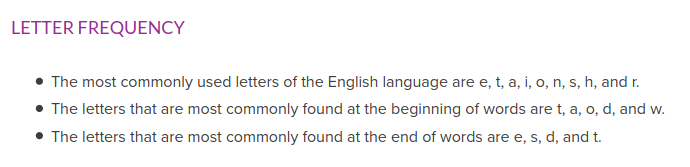
Autor: Birlutiu Claudiu-Andrei, gr 30643

## Sarcina 1: Analiza frecvenței împotriva unui cifru de substituție monoalfabetic

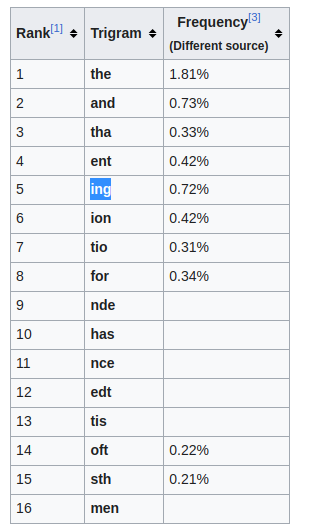
* În continuare am urmat tuorialul de criptare:
  + in prima faza am creat un script bash care ia cifrul și îl transforma în plain text (fără semene de punctuatie și litere mici) – fișierul se numește **modify\_script.sh** și i s-a dat permisiuni de execuție; se observa ca din textul article.txt s-a generat plaintext.txt care reprezintă textul fără cemne de punctuatie și lowercase
  + am creat un script python care genereaza o cheie aleatoare de mapare a literelor alfabetului la o alta litera și am făcut inlocuirile pe plaintext.txt => cipher.txt

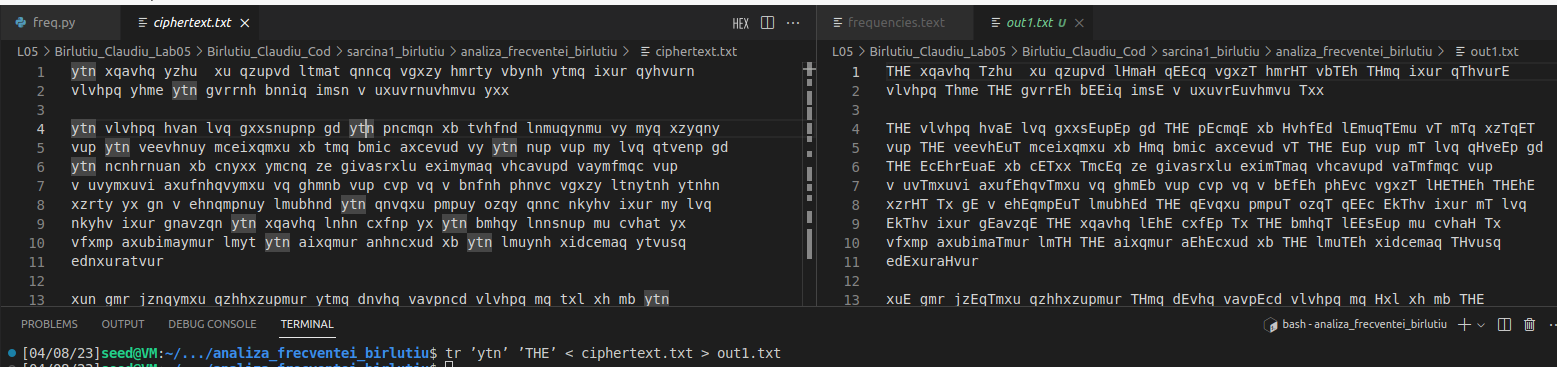


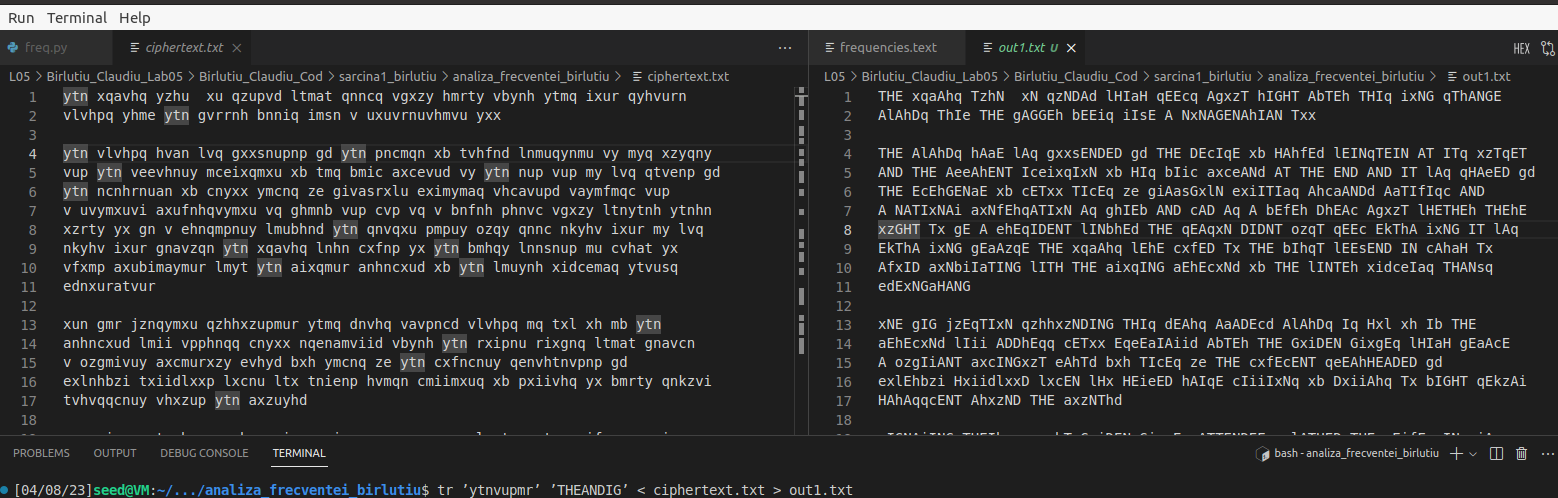
* Am trecut la descifrarea textului criptat din lucrare. Scopul a fost sa analizez frecventa literelor sau grupurilor de litere astfel încât sa găsesc cheia de criptare a mesajului
* am modificat programul python de analiza a textului sa scrie într-un fisier frecventele
* am căutat pe internet care sunt cele mai frecvente litere din limba engleza



* am încercat următoarea analiza în ceea ce privește n-gramele generate: din sursa <https://en.wikipedia.org/wiki/Frequency_analysis>înțelegem ca the – este cea mai comuna tri gram din limba engelza, th- cea mai comuna bi-gram și e este cea mai folosită litera

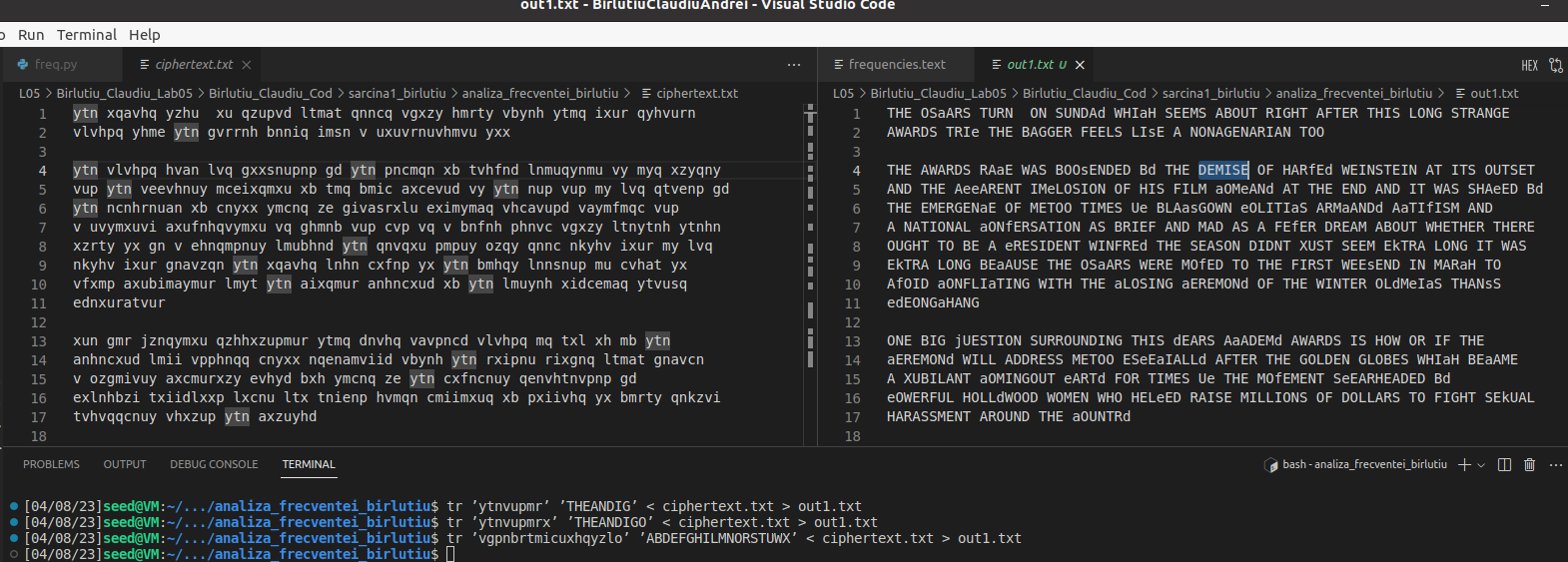


* în urma analizei a reiesit faptul ca n apare de cele mai multe ori (488); yt – apare de asemenea de cel mai multe ori și ytn – apare de asemenea de cele mai multe ori => vom inlocui astfel **y** cu **t**, **n** cu **e** și **t** cu **h**
* **and** este de asemenea o triagrama detsul de întâlnită în textul englezesc, iar la noi pe a doua poziție a triagramelor regasim vup =? încercam astfel sa inlocuim v cu a, u cu n și p cu d
  + de asemenea **ing** este o trigrama frecventa -> iar la noi **mur** (a 3-a cea mai frecventa) s-ar potrivi cu ing având în vedere supozitia anterioara (
  + v = a, u = n , p = d, m = i , r = g



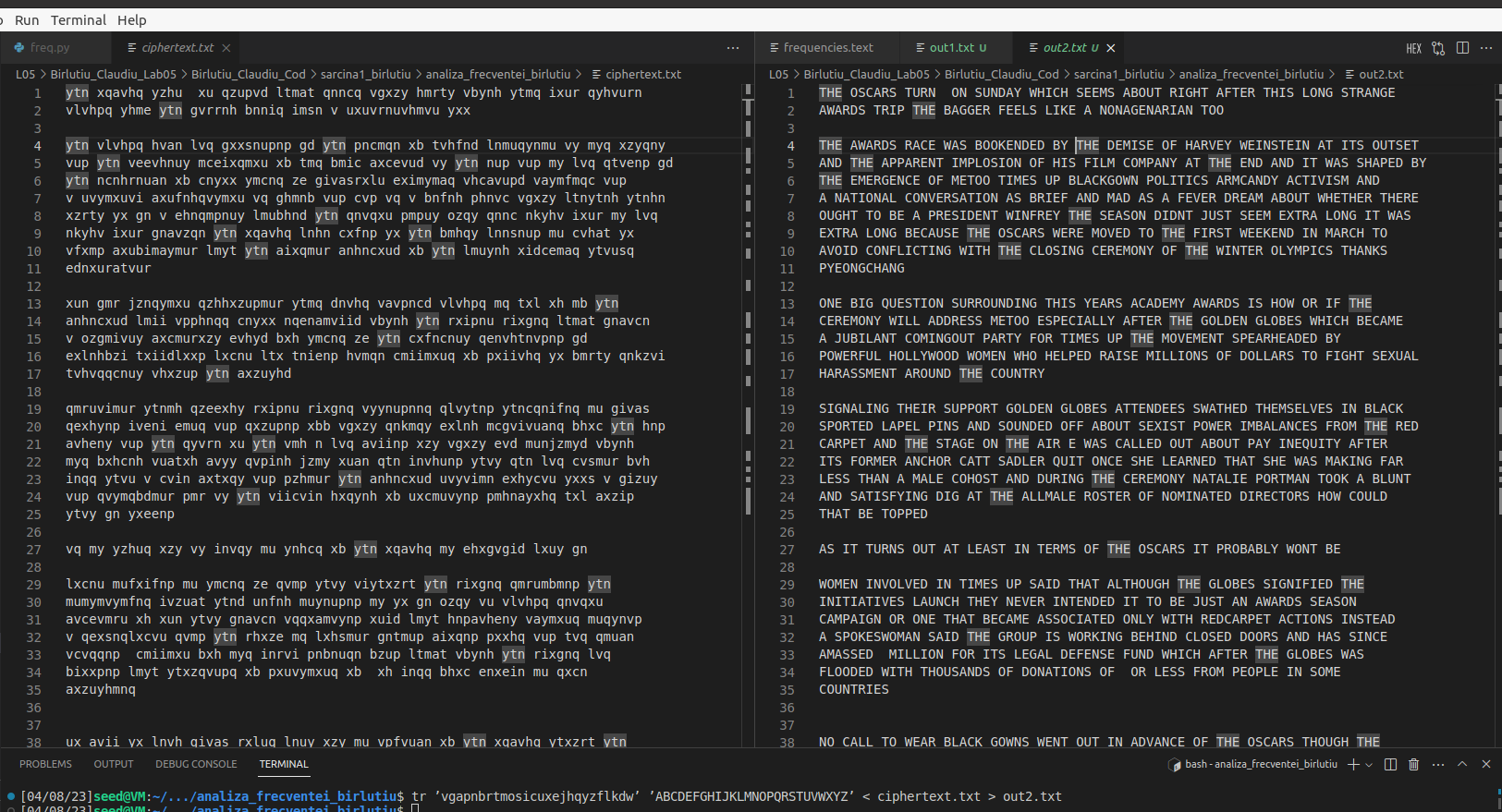
* avem:
  + ytn = the
  + vup = and
  + mur = ing
  + ynh = te\* => ter [0] **h =r**
  + xzy = \*\*t => o\*t => out [2] **z = u**
  + mxu = i\*n => [1] **x = o**
  + gnq = \*e\* => \*es [4] bes => **g =b**
  + ytv = tha
  + nqy = e\*t => [3] **q =s**
  + vii = a\*\* => [7] **i = l (**NATIONAi)
  + bxh = \*or = [5] for **b=f**
  + lvq = \*as = [6] was  **l = w**
  + nuy = ent
  + vyn = ate
  + uvy = nat
  + lmu = win
  + nvh = ear
  + cmu = \*in [8] => min **c=m** (DEcIqE = DmcISE = DEMISE)
  + tmq = his
  + vhp => ard

=>

* a b c d e f g h i j k l m n o p q r s t u v w x y z
* v g \* p n b r t m i c u x h q y z l o
* încercam cu valorile descoprite iar apoi vom identifica și pe celelalte lipsa
* abdefghilmnorstuwx (ABDEFGHILMNORSTUWX) = vgpnbrtmicuxhqyzlo
* identificam restul corespondetelor din cuvinte: OSaARS => **a = c;** TRIe => **e =p ;** LIsE => **s = k;** Bd **=> d =y;** HARfEY = > **f = v;** SEkIST=> k=x; jUESTION = > **j = q;** EkTRA = > **k =x; (modificare); (**XUST=JUST) => **o= j;** (eRIwE = PRIwE) => **w =z**

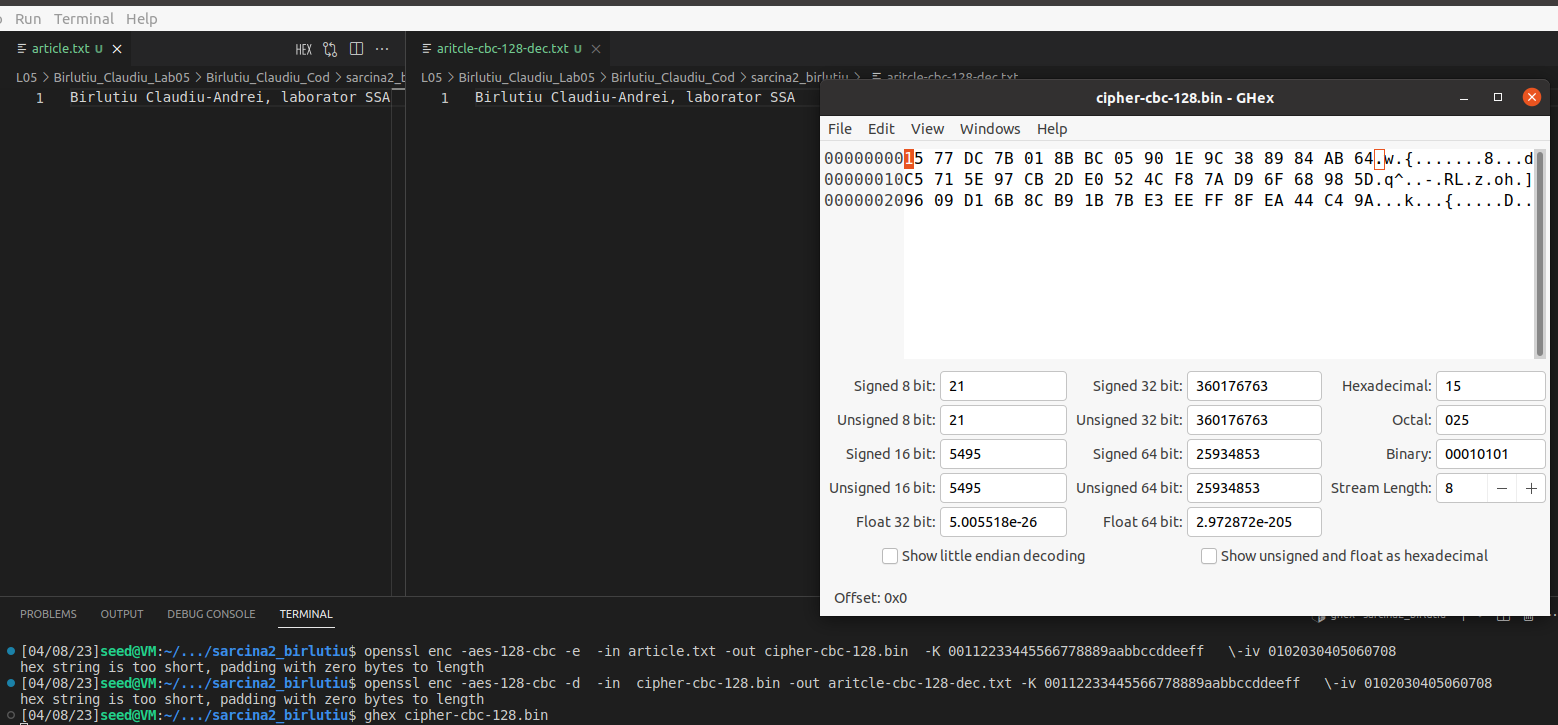
**REZULTAT:**

ab**c**defghi**jk**lmno**pq**rstu**v**wx**y**z(AB**C**DEFGHIJ**K**LMNO**PQ**RSTU**V**WX**YZ**)=vg**a**pnbrtm**os**icux**ej**hqyz**f**lk**dw**

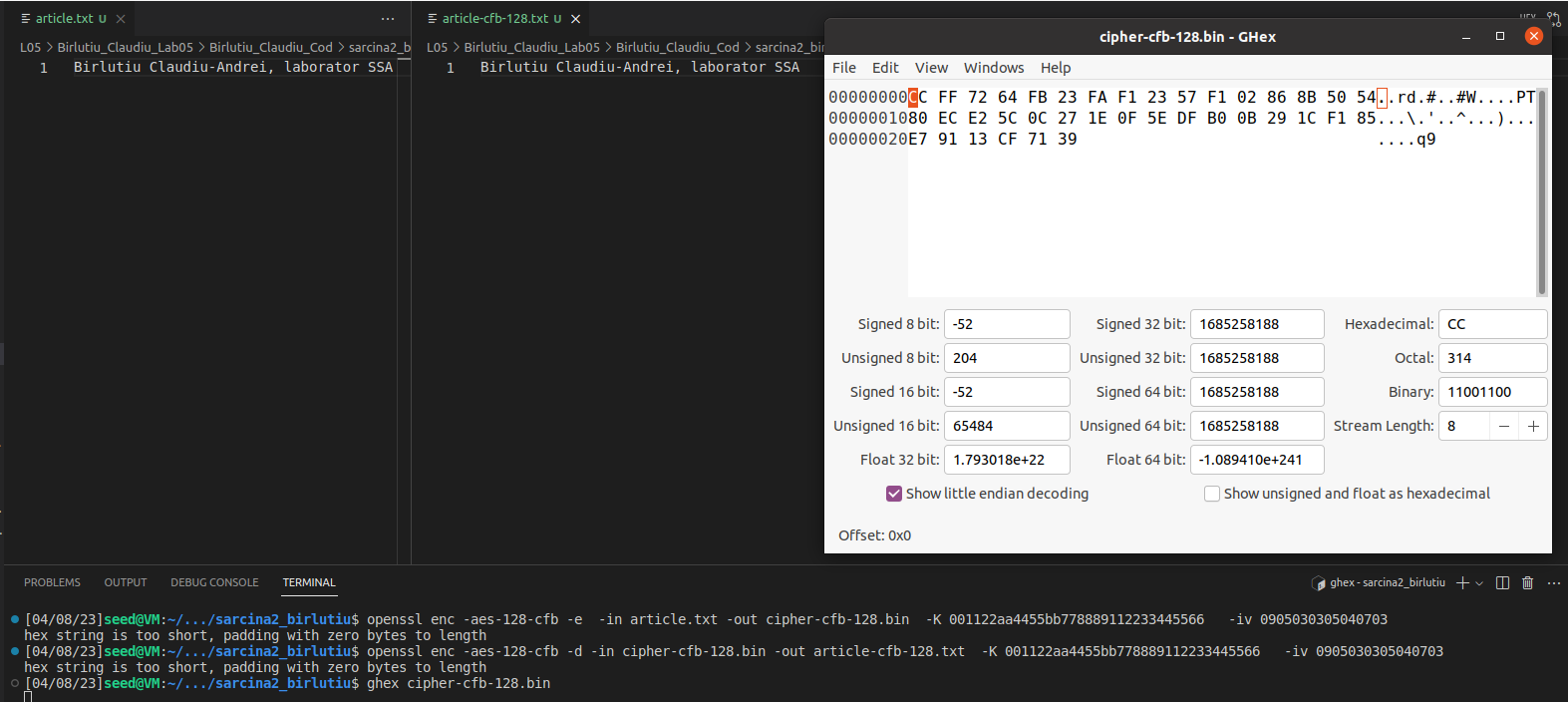


***Sarcina 2: Criptarea cu diferite cifruri și în diverse moduri***

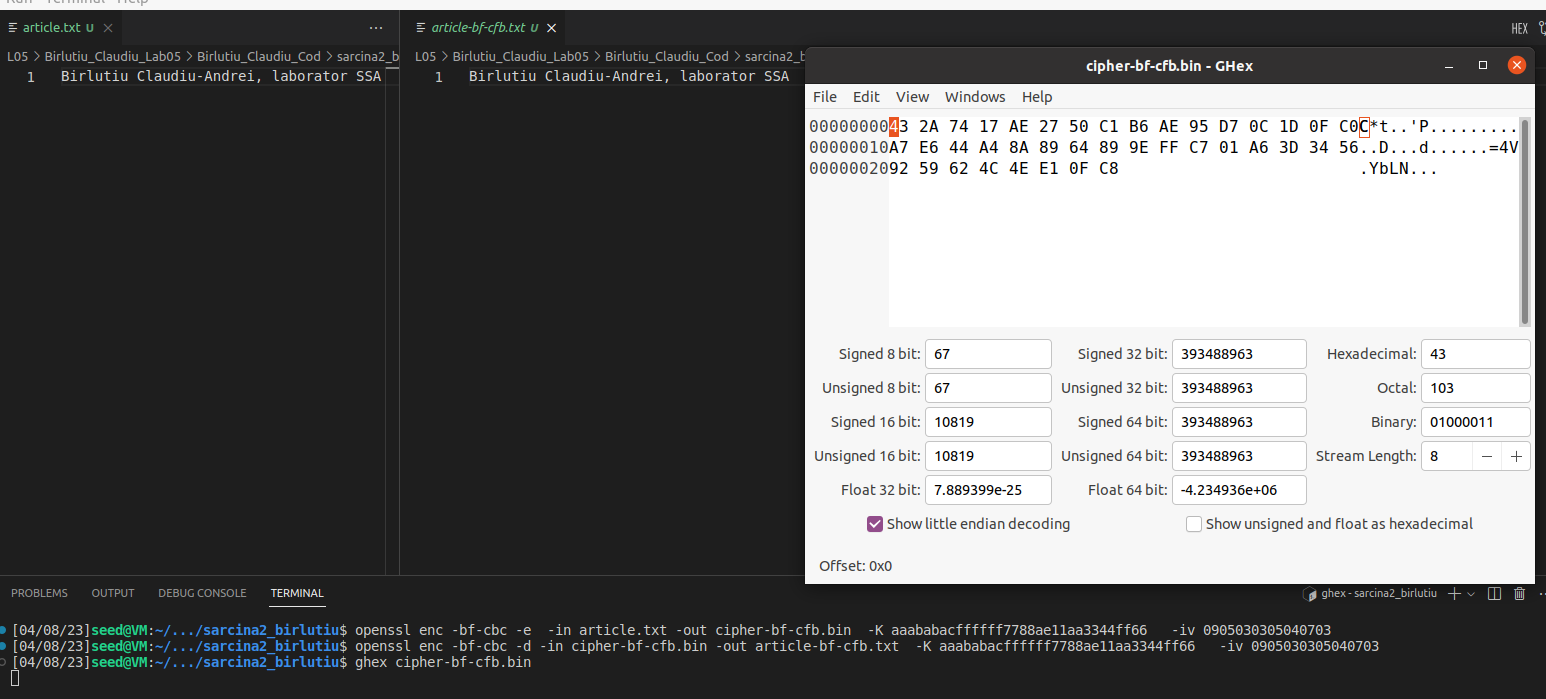
* am încercat urmatorele comenzi de cifrare a textului article.txt
  + **openssl enc -aes-128-cbc -e -in article.txt -out cipher-cbc-128.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708**
  + comanda cripteaza fisierul "article.txt" folosind cifrul AES cu o cheie de 128 de biti si modul de operare CBC, generand fisierul criptat "cipher.bin".
  + parametrul "-K" specifica cheia de criptare, in format hexazecimal. In acest caz, cheia specificata este "00112233445566778889aabbccddeeff".
  + parametrul "-iv" specifica vectorul de initializare (IV), in format hexazecimal. IV-ul este un parametru necesar pentru modurile de operare CBC si OFB. In acest caz, IV-ul specificat este "0102030405060708"



* + am decriptat de asemnea fișierul: **openssl enc -aes-128-cbc -e -in cipher-cbc-128.bin -out aritcle-cbc-128.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708**
* openssl enc **-aes-128-cfb -**e -in article.txt -out cipher-cfb-128.bin -K 001122aa4455bb778889112233445566 -iv 0905030305040703
* openssl enc **-aes-128-cfb** -d -in cipher-cfb-128.bin -out article-cfb-128.txt -K 001122aa4455bb778889112233445566 -iv 0905030305040703



* openssl enc **-bf-cbc** -e -in article.txt -out cipher-bf-cfb.bin -K aaababacffffff7788ae11aa3344ff66 -iv 0905030305040703
* openssl enc **-bf-cbc** -d -in cipher-bf-cfb.bin -out article-bf-cfb.txt -K aaababacffffff7788ae11aa3344ff66 -iv 0905030305040703



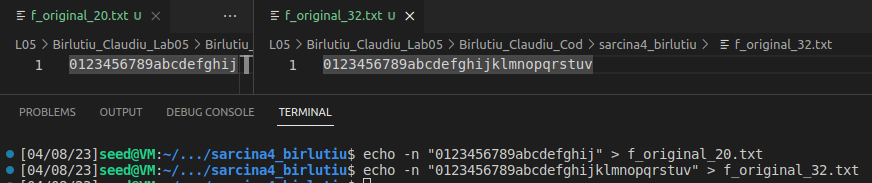
## Sarcina 3: Modul de criptare -- ECB vs. CBC

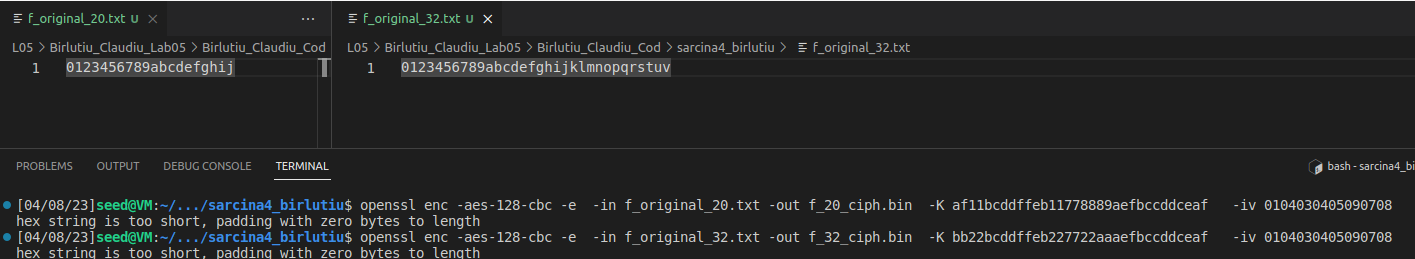
* In continuare vom encodifica o imagine cu extensia .bmp cu cele 2 modalitati de criptare ECB so CBC si vom modifica headerul imaginii criptate pentru a putea fi vizulizata cu ajutorul comenzilor afisate in alborator;
* din docmunenatre am gasit ca:
  + **modul ECB**: funcționează prin criptarea fiecărui bloc de date BMP separat, utilizând aceeași cheie de criptare pentru fiecare bloc astfle ca blocurile de date identice vor fi criptate în același mod, ceea ce poate face ca datele criptate să fie vulnerabile
  + **modul CBC:** funcționează prin criptarea fiecărui bloc BMP utilizând cheia de criptare, dar combină rezultatele criptării cu blocul BMP anterior prin aplicarea unei operații de XOR →datele criptate să fie mai sigure, deoarece blocurile identice de date vor fi criptate în moduri diferite
* openssl enc **-aes-128-ecb** -e -in pic\_original.bmp -out ebc\_picture.bmp -K aaababacffffff7788ae11aa3344ff66
  + Se vor rula urmatoarele comenzi pentru ca fisierul ebc\_picture.bmp sa fie tratat ca si un fisier bmp.
    - * head -c 54 pic\_original.bmp > header
      * tail -c +55 ebc\_picture.bmp > body
      * cat header body > pic\_ebc\_modified.bmp
    - se observa cum blocurile de date identice se cripteaza la fel si se pt distige formele obiectelor
* openssl enc -**aes-128-cbc** -e -in pic\_original.bmp -out cbc\_picture.bmp -K aaababacffffff7788ae11aa3344ff66 -iv 0905030305040703
  + Se vor rula urmatoarele comenzi pentru ca fisierul ebc\_picture.bmp sa fie tratat ca si un fisier bmp.
    - * head -c 54 pic\_original.bmp > header
      * tail -c +55 cbc\_picture.bmp > body
      * cat header body > pic\_cbc\_modified.bmp
      * in cazul encodarii cu CBC se observa ca nu se disting formele din imaginea initiala, deci gradul de securitate e mai ridicat, cee ce demonstreaza ca CBC reduce vulnerabilitatea descifrarii mesajeor

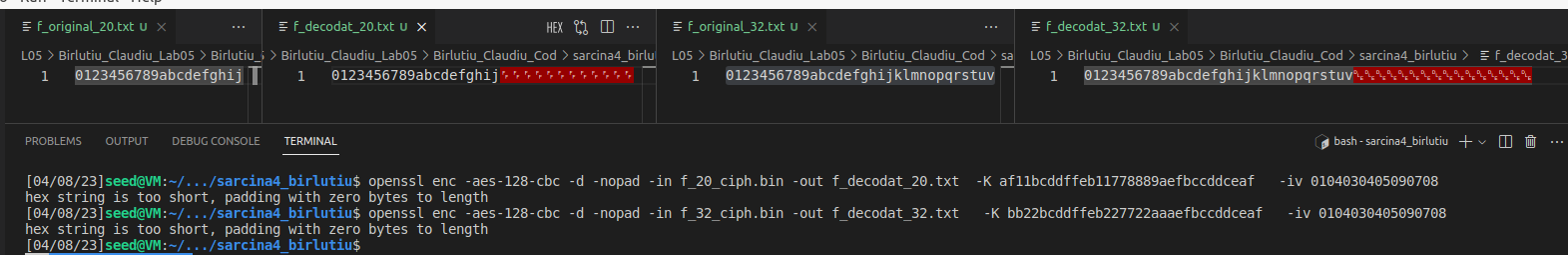
## Sarcina 4 : Caractere de completare pentru textul în clar

### Care sunt caracterele de completare în cifrarea AES atunci când lungimea textului în clar este 20 octeți și 32 octeți

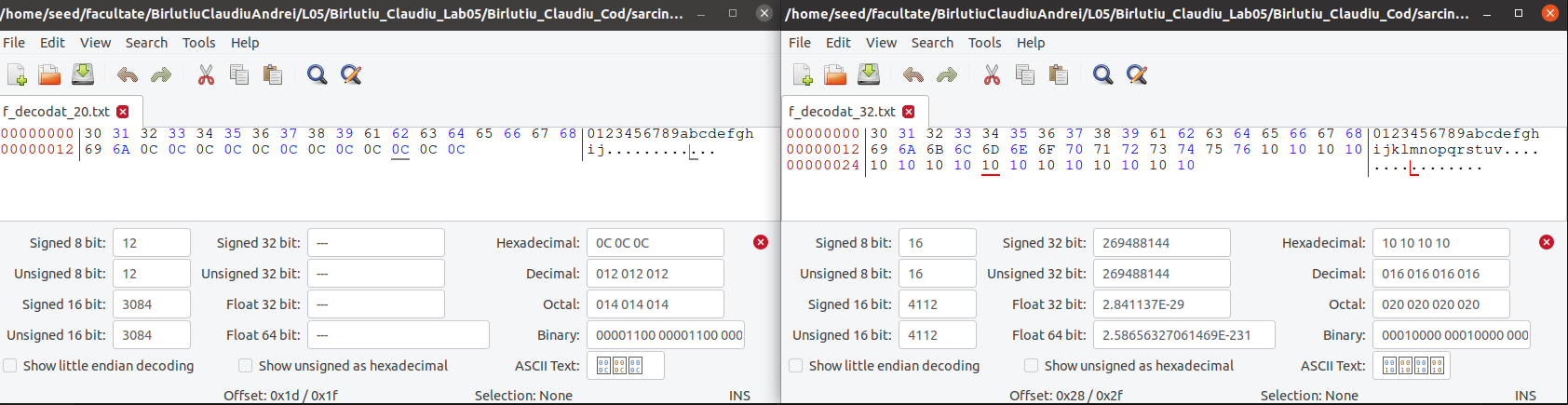
* Am ales ca și studiu de caz modul **CBC** pentru criptare, care este un mod care introduce caractere complementare și vom observa acets lucru prin executarea pasilor ce s-au menționat în lucrarea de laborator și vom folosi bless editor pentru a observa caracterele introduse în plus
* cream un fisier cu 20 de caractere și unul cu 32 de caractere (octeti)



* vom encodifica cele 2 fisiere cu cu modul cbc aes pe 128 de biti:
  + openssl enc -aes-128-cbc -e -in f\_original\_20.txt -out f\_20\_ciph.bin -K af11bcddffeb11778889aefbccddceaf -iv 0104030405090708
  + openssl enc -aes-128-cbc -e -in f\_original\_32.txt -out f\_32\_ciph.bin -K bb22bcddffeb227722aaaefbccddceaf -iv 0104030405090708
* vom decoda cele 2 fisier binare cu optiunea **nopad** astfel încât caracterele adaugate în plus se vor vedea în fieserele noi decodate
  + openssl enc -aes-128-cbc -d -nopad -in f\_20\_ciph.bin -out f\_decodat\_20.txt -K af11bcddffeb11778889aefbccddceaf -iv 0104030405090708
  + openssl enc -aes-128-cbc -d -nopad -in f\_32\_ciph.bin -out f\_decodat\_32.txt -K bb22bcddffeb227722aaaefbccddceaf -iv 0104030405090708
* în imaginea de mai jos se observa cum au fost adaugate caractere complementare la final; acestea nu sunt afisabile, dar putem sa le vizualizam cu un editor de text cum ar fi bless

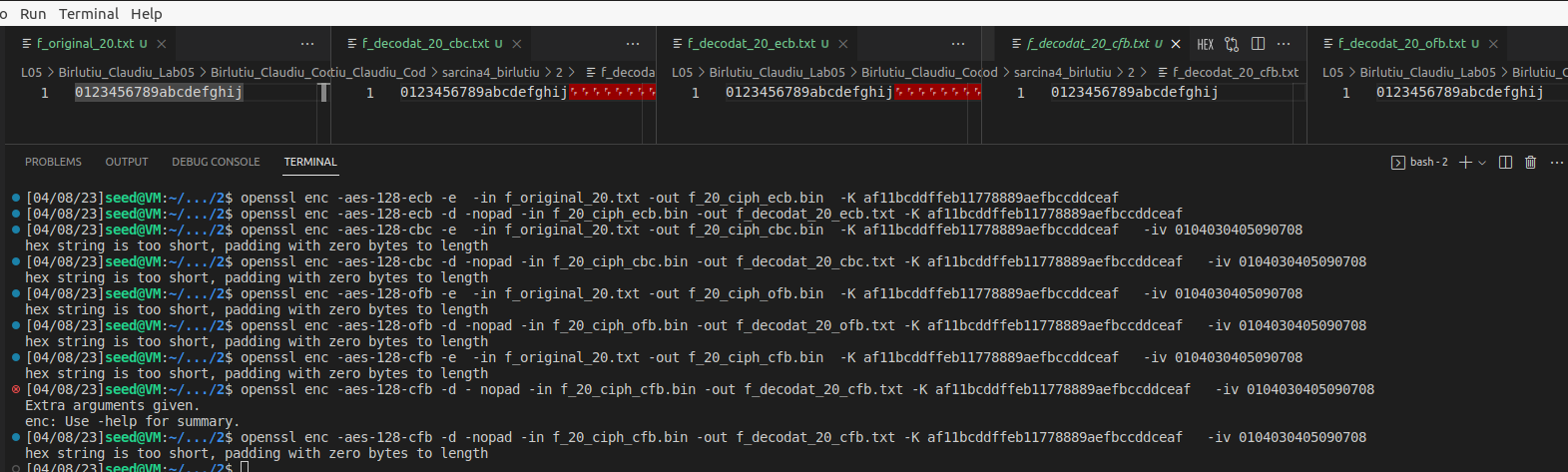


* am examinat prin intermediul tool-ului bless ce caratere s-au adaugat în plus și am observat ce în cadrul fisierului cu 20 de caractere s-a aduagat octetul 0x0C(form feed) de 12 ori => 32 de octeti, iar pentru cel de 32 de caractere s-a pus 0x10 (linefeed) de 16 ori => 48 de octeti; astfel numărul de octeti a devenit **multiplu** de 8



1. **Considerați modurile ECB, CBC, CFB, și OFB de cifrare a unui fișier. Care au caractere de completare și care nu au?**

* Din documentare am aflat ca modurile **ECB** și **CBC** folosesc o dimensiune fixă a blocului de date pentru criptare și de aceea vor utiliza caractere de completare. Datele trebuie să fie în mod explicit completate cu caractere suplimentare, astfel încât să poată fi împărțite în blocuri egale.
* În cauzl modurilor **CFB** și **OFB** permit o dimensiune variabilă a blocului de date și astfle nu este necesara completarea datelor -> aceste moduri de criptare criptează o secvență de biți și generează un flux de ieșire pentru a fi combinat cu datele de intrare în loc sa cripteze blocuri de date întregi
  + - **EBC**
* openssl enc -aes-128-ecb -e -in f\_original\_20.txt -out f\_20\_ciph\_ecb.bin -K af11bcddffeb11778889aefbccddceaf
* openssl enc -aes-128-ecb -d -nopad -in f\_20\_ciph\_ecb.bin -out f\_decodat\_20\_ecb.txt -K af11bcddffeb11778889aefbccddceaf
* **CBC**:
  + openssl enc -aes-128-cbc -e -in f\_original\_20.txt -out f\_20\_ciph\_cbc.bin -K af11bcddffeb11778889aefbccddceaf -iv 0104030405090708
  + openssl enc -aes-128-cbc -d -nopad -in f\_20\_ciph\_cbc.bin -out f\_decodat\_20\_cbc.txt -K af11bcddffeb11778889aefbccddceaf -iv 0104030405090708
* **OFB**
  + openssl enc -aes-128-ofb -e -in f\_original\_20.txt -out f\_20\_ciph\_ofb.bin -K af11bcddffeb11778889aefbccddceaf -iv 0104030405090708
  + openssl enc -aes-128-ofb -d -nopad -in f\_20\_ciph\_ofb.bin -out f\_decodat\_20\_ofb.txt -K af11bcddffeb11778889aefbccddceaf -iv 0104030405090708
* **CFB**:
  + openssl enc -aes-128-cfb -e -in f\_original\_20.txt -out f\_20\_ciph\_cfb.bin -K af11bcddffeb11778889aefbccddceaf -iv 0104030405090708
  + openssl enc -aes-128-cfb -d - nopad -in f\_20\_ciph\_cfb.bin -out f\_decodat\_20\_cfb.txt -K af11bcddffeb11778889aefbccddceaf -iv 0104030405090708
* se poate observa n următoarea imagine care dintre moduri aduaga caractere complementare



## Sarcina 5: Propagarea erorilor -- Text cifrat alterat

### Câtă informație puteți recupera din descifrarea fișierului alterat, dacă modul de cifrare a fost ECB, CBC, CFB, respectiv OFB?

### Răspunsul dat înainte de efectuarea sarcinii

### Răspunsul după execuția acestei sarcini.

### Explicați de ce.

## Sarcina 6. Vectorul inițial (IV) și erori comune

### 6.1 Sarcina 6.1. Unicitatea IV

### 6.2 Sarcina 6.2. Eroare comună: folosirea aceluiași IV

### 6.3 Sarcina 6.3. Eroare comună: folosirea unui IV predictibil

## Sarcina 7: Programarea folosind biblioteca criptografică