

# Cifrarea cu cheie secretă

# Cifru de substituție

- Criptarea se face prin înlocuirea unităților de text clar cu text cifrat, conform unui sistem fix.
- *Unitățile* pot fi litere, perechi de litere, triplete de litere, amestecuri de mai sus și aşa mai departe
- Decriptarea efectuează pur și simplu substituția inversă.
- Două cifruri de substituție tipice :
  - monoalfabetică - substituție fixă pentru întregul mesaj
  - polialfabetică - o serie de substituții în poziții diferite în mesaj

# Cifru de substituție monoalfabetic

- Cifrare și descifrare

```
# Criptare
$ tr 'a-z' 'vgapnbrtmosicuxejhqyzflkdw' < plaintext > ciphertext

# Decriptare
$ tr 'vgapnbrtmosicuxejhqyzflkdw' 'a-z' < ciphertext > plaintext_new
```

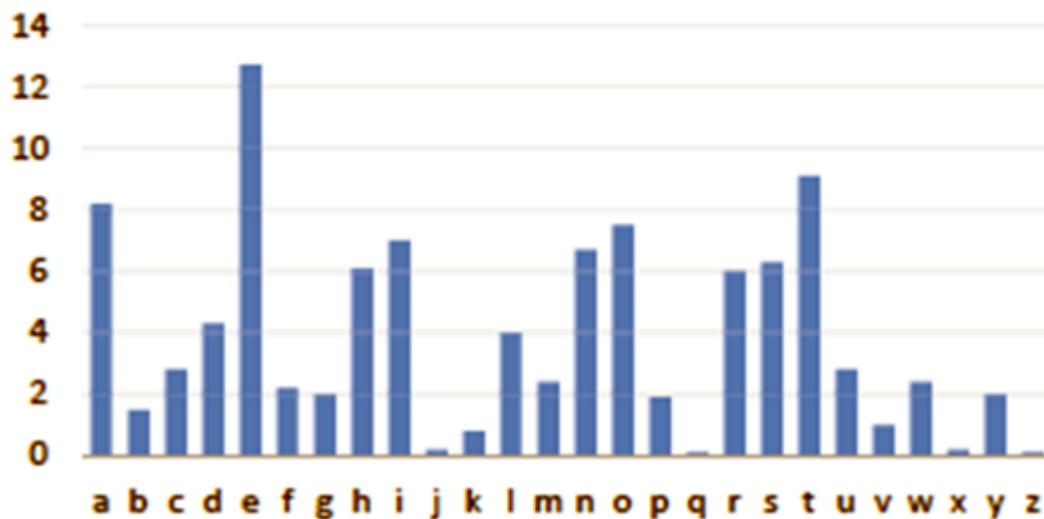
# Spargerea unui cifru de substituție monoalfabetic

- Analiza frecvenței: studiul frecvenței literelor și a grupurilor de litere într-un text cifrat.
- Litere întâlnite frecvent : T, A, E, I, O
- Combinări de 2 litere (bigrame) întâlnite frecvent : TH, HE, IN, ER
- Combinări de 3 litere (trigrame) : THE, AND și ING
  - Pentru limba engleză

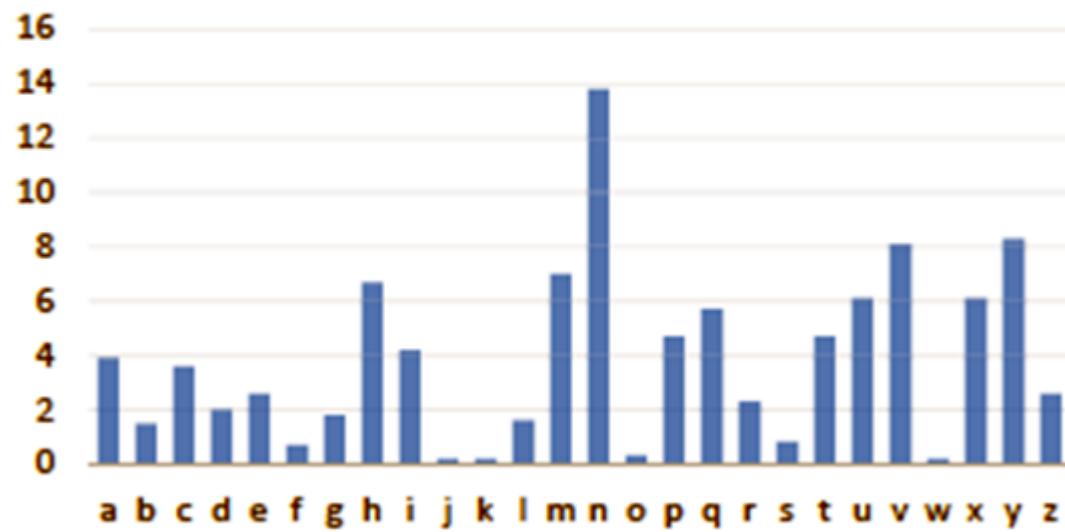
# Spargerea unui cifru de substituție monoalfabetic

- Exemplu de rezultat al analizei frecvenței literelor:

Letter frequencies in English text



Letter frequencies in ciphertext



# Spargerea unui cifru de substituție monoalfabetic

- Exemplu de rezultat al analizei a frecvenței **bigramelor**:

Bigram frequency in English		
TH : 2.71	EN : 1.13	NG : 0.89
HE : 2.33	AT : 1.12	AL : 0.88
IN : 2.03	ED : 1.08	IT : 0.88
ER : 1.78	ND : 1.07	AS : 0.87
AN : 1.61	TO : 1.07	IS : 0.86
RE : 1.41	OR : 1.06	HA : 0.83
ES : 1.32	EA : 1.00	ET : 0.76
ON : 1.32	TI : 0.99	SE : 0.73
ST : 1.25	AR : 0.98	OU : 0.72
NT : 1.17	TE : 0.98	OF : 0.71

Bigram frequency in ciphertext (The top-10 patterns)		
tn : 77	np : 50	
yt : 76	hn : 45	
nh : 61	nu : 44	
nq : 51	mu : 42	
vu : 51	cv : 42	

# Spargerea unui cifru de substituție monoalfabetic

- Exemplu de rezultat al analizei a frecvenței **trigramelor**:

Trigram frequency in English		
THE : 1.81	ERE : 0.31	HES : 0.24
AND : 0.73	TIO : 0.31	VER : 0.24
ING : 0.72	TER : 0.30	HIS : 0.24
ENT : 0.42	EST : 0.28	OFT : 0.22
ION : 0.42	ERS : 0.28	ITH : 0.21
HER : 0.36	ATI : 0.26	FTH : 0.21
FOR : 0.34	HAT : 0.26	STH : 0.21
THA : 0.33	ATE : 0.25	OTH : 0.21
NTH : 0.33	ALL : 0.25	RES : 0.21
INT : 0.32	ETH : 0.24	ONT : 0.20

Trigram frequency in ciphertext (The top-10 patterns)		
ytn : 60	tnh : 13	
vup : 26	pyt : 13	
nhc : 16	hcv : 13	
nhn : 15	tne : 13	
nuy : 14	mrc : 13	

# Spargerea unui cifru de substituție monoalfabetic

- Aplicarea mapărilor parțiale:

```
$ tr ntyhqu EHRSN < ciphertext
```

THE ENmrcv cvaHmNES lERE v SERmES xb EiEaTRxcEaHvNmavi RxTxR ameHER  
cvaHmNES pEfEixeEp vNp zSEp mN THE EvRid Tx cmpTH aENTzRd Tx  
eRxTEaT axccERamvi pmeixcvTma vNp cmimTvRd axcczNmavTmxN ENmrcv lvS  
mNfENTEp gd THE rERcvN ENrmNEER vRTHzR SaHERgmzS vT THE ENp xb  
lxRip lvR m EvRid cxpEis lERE zSEp axccERamviid bRxc THE EvRid S  
vNp vpxeTEp gd cmimTvRd vNp rxfERNcENT SERfmaES xb SEfERvi  
axzNTRmES cxST NxTvgic \$ tr ntyhquvmxbpz EHRSNAIOFDU < ciphertext  
SEfERvi pmhbERENT ENm  
cmimTvRd cxpEis HvfmNi  
vNp mTvimvN cxpEis lER

```
$ tr ntyhquvmxbpz EHRSNAIOFDU < ciphertext
```

THE ENIrcA cAaHINES lERE A SERIES OF EiEaTROcEaHANIaAi ROTOR aIeHER  
cAaHINES DEFElOeED AND USED IN THE EARid TO cIDTH aENTURd TO  
eROTEaT aOccERaIAi DlEiOcATIA AND cIIITARD aOccUNIaATION ENIrcA lAS  
**INFENTED** gd THE rERcAN **ENrINEER** ARTHUR SaHERgiUS AT THE END OF

lORid lAR I EARid cODEis lERE USED aOccERaIAiid **FROc** THE EARid S

AND **ADOeTED** gd cIIITARD AND ro \$ tr ntyhquvmxbpzfrcei EHRSNAIOFDUVGMPL < ciphertext

aOUNTRIES COST NOTAgid NAWI rE THE ENIGMA MAaHINES lERE A SERIES OF ELEATROMEaHANIaAL ROTOR aIPHER  
SEFERAi DIFFERENT ENIrcA cODEi MAaHINES DEVELOPED AND USED IN THE EARld TO MIDTH aENTURd TO

PROTEaT aOMMERaIAL DIPLOMATIA AND MILITARD aOMMUNIaATION ENIGMA lAS  
INVENTED gd THE GERMAN ENGINEER ARTHUR SaHERgiUS AT THE END OF  
lORLD lAR I EARld MODELS lERE USED aOMMERaIALlD FROM THE EARld S  
AND ADOPTED gd MILITARD AND GOVERNMENT SERVIaES OF SEVERAL  
aOUNTRIES MOST NOTAgld NAWI GERMAnD gEFORc AND DURING lORLD lAR II  
SEVERAL DIFFERENT ENIGMA MODELS lERE PRODUaED gUT THE GERMAN  
MILITARD MODELS HAVING A PLUGBOARD lERE THE MOST aOMPLEk oAPANESE  
AND ITALIAN MODELS lERE ALSO IN USE ...

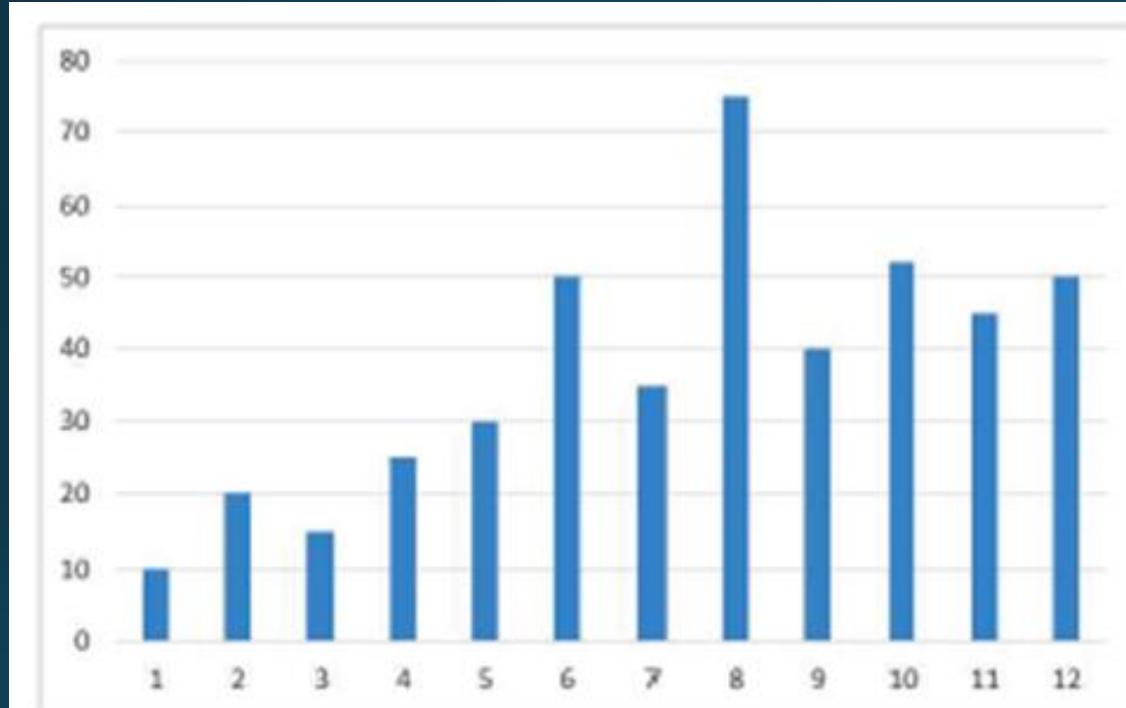
# Data Encryption Standard (DES)

- DES este un cifru bloc – poate cifra doar blocuri de date
- Dimensiunea blocului pentru DES este de 64 biți
- DES folosește chei de 56-biți deși cheia are 64-biți
- Au fost identificate atacuri teoretice. Nici unul nu era destul de practic pentru a cauza preocupări majore.
- DES triplu (Triple DES ) poate rezolva problema lungimii cheii folosite

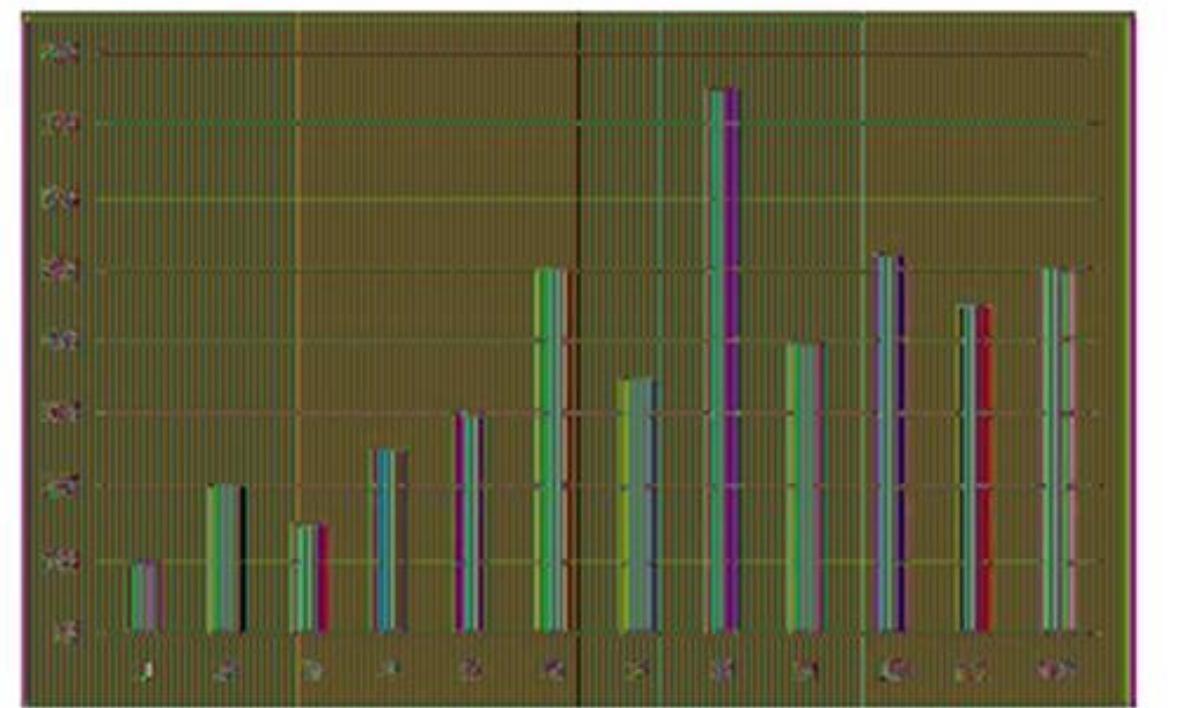
# Advanced Encryption Standard (AES)

- AES este un cifru bloc
- Blocul are 128-biți.
- Trei dimensiuni diferite pentru cheie: 128, 192 și 256 biți

# Moduri de cifrare



(a) The original image (`pic_original.bmp`)

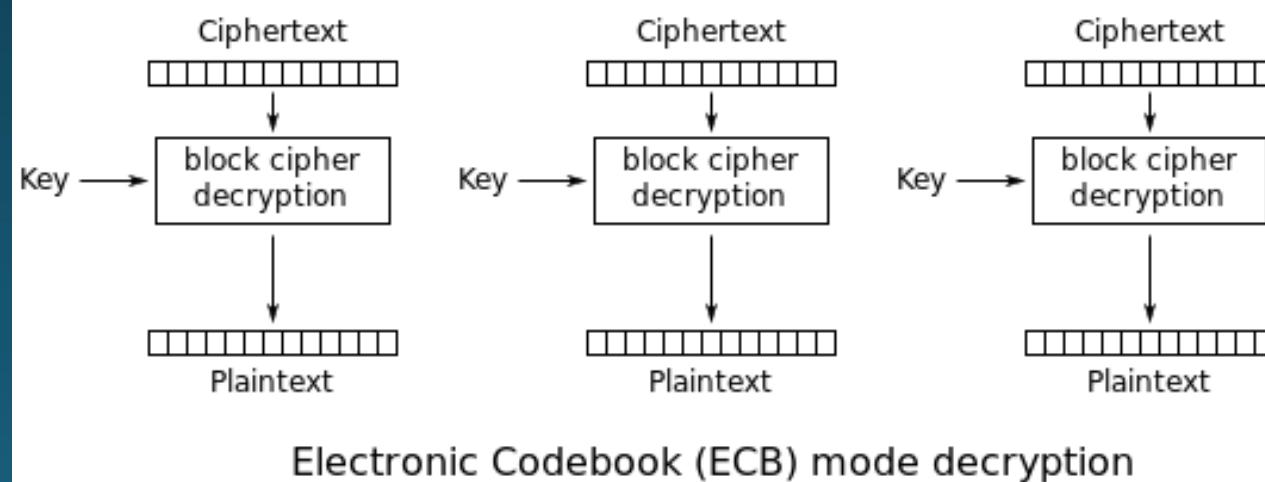
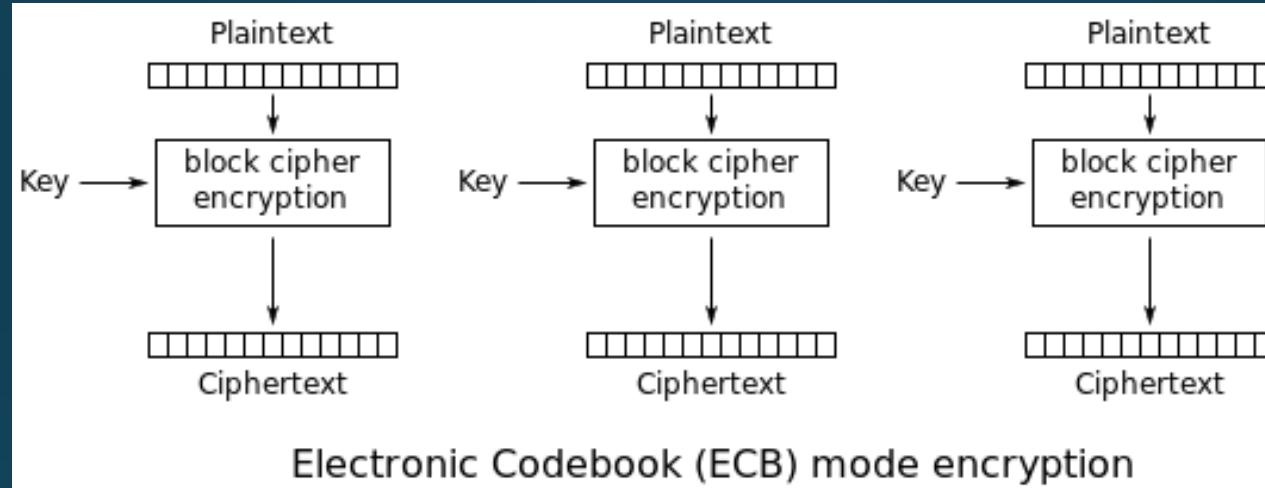


(b) The encrypted image (`pic_encrypted.bmp`)

# Moduri de cifrare

- Modurile de cifrare sau modurile de operare se referă la multiplele feluri în care se poate varia intrarea unui algoritm de criptare.
- Exemple:
  - Electronic Codebook (ECB) - "carte de coduri electronică"
  - Cipher Block Chaining (CBC) – "înlănțuirea blocurilor cifrate"
  - Propagating CBC (PCBC)
  - Cipher Feedback (CFB)
  - Output Feedback (OFB)
  - Counter (CTR)

# Modul "carte de coduri electronică" (Electronic Codebook - ECB)



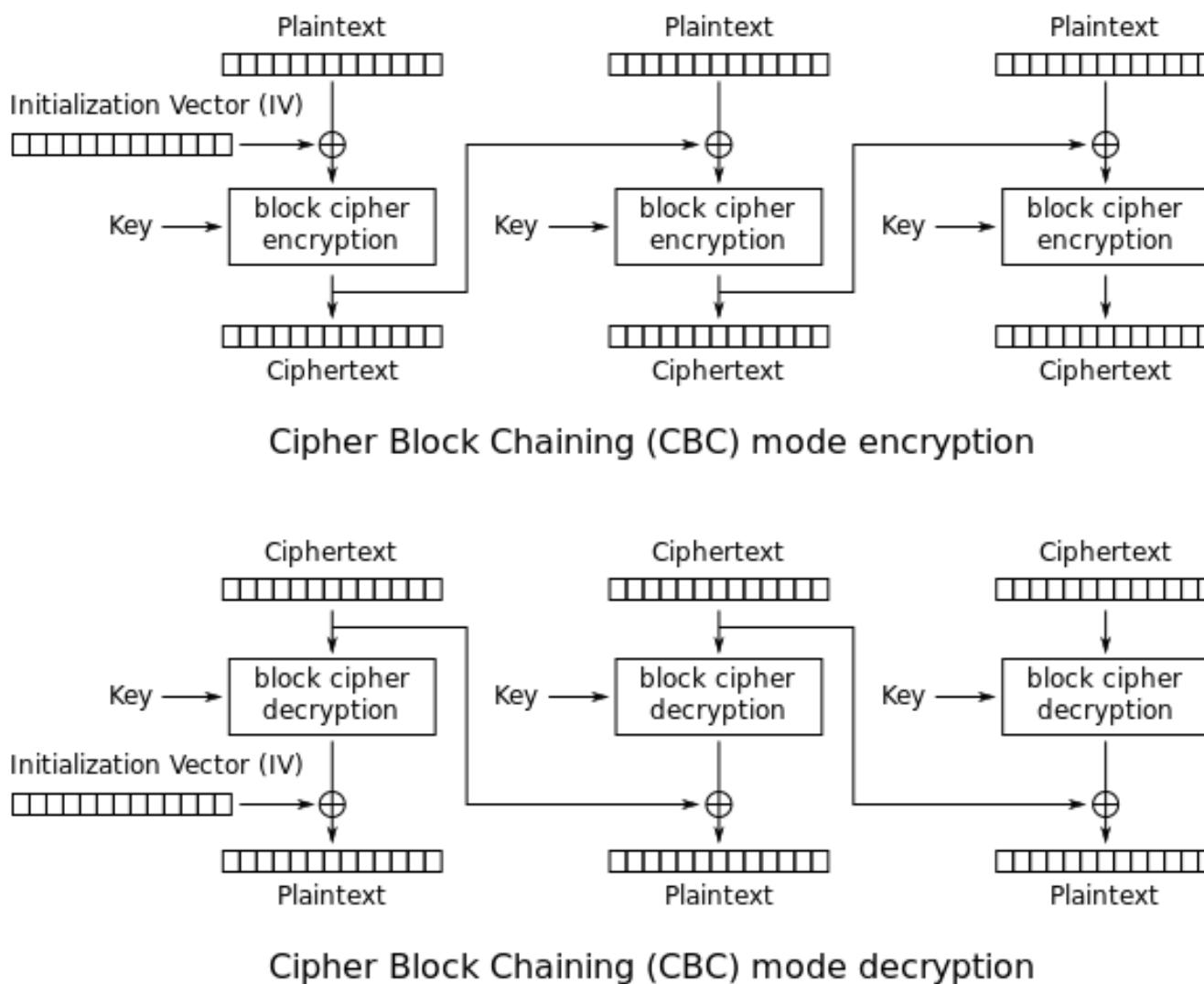
# Modul "carte de coduri electronică" (Electronic Codebook - ECB)

- Folosim comanda openssl **enc**:

```
$ openssl enc -aes-128-ecb -e -in plain.txt -out cipher.txt \
-K 00112233445566778899AABBCCDDEEFF
$ openssl enc -aes-128-ecb -d -in cipher.txt -out plain2.txt \
-K 00112233445566778899AABBCCDDEEFF
```

- Folosim algoritmul AES cu mărimea cheii de 128-biți ( AES-128)
- Opțiunea **-aes-128-ecb** specifică modul ECB
- Opțiunea **-e** indică cifrare
- Opțiunea **-d** indică descifrare
- Opțiunea **-K** se folosește pentru a preciza cheia de cifrare/descifrare

# Modul "înlănțuirea blocurilor cifrate" (Cipher Block Chaining -CBC)



- Scopul principal al vectorului de inițializare (**IV**) este să asigure că chiar dacă două texte în clar sunt identice, textele cifrate corespunzătoare lor sunt diferite, datorită folosirii vectorilor de inițializare diferenți.
- Decriptarea **se poate** paraleliza
- Criptarea **nu se poate** paraleliza

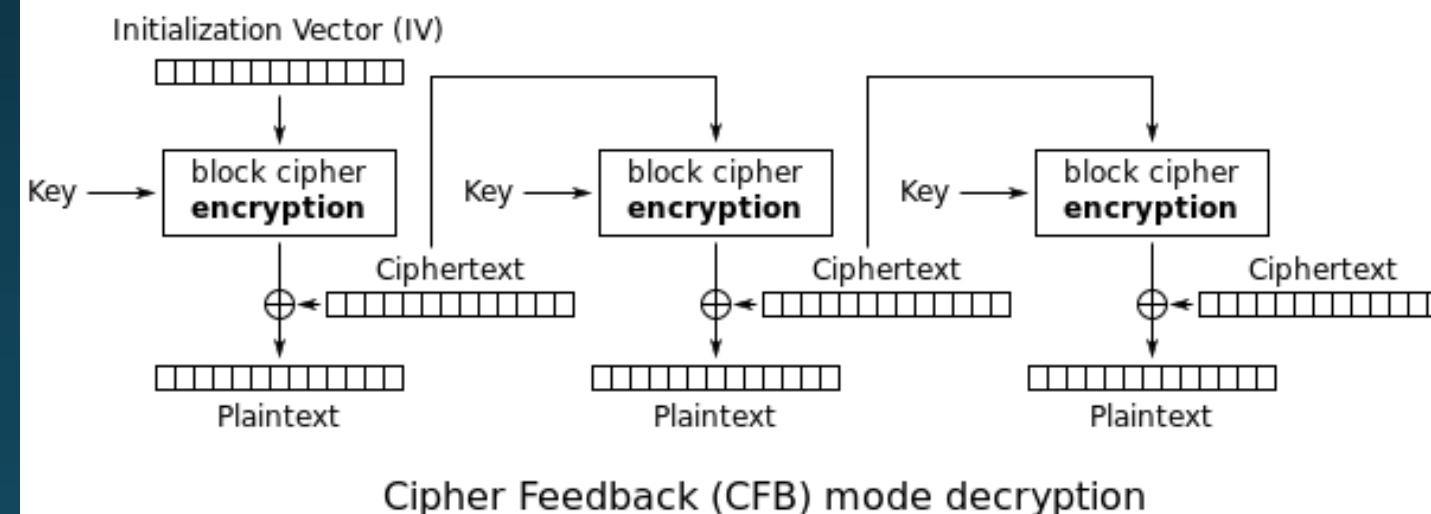
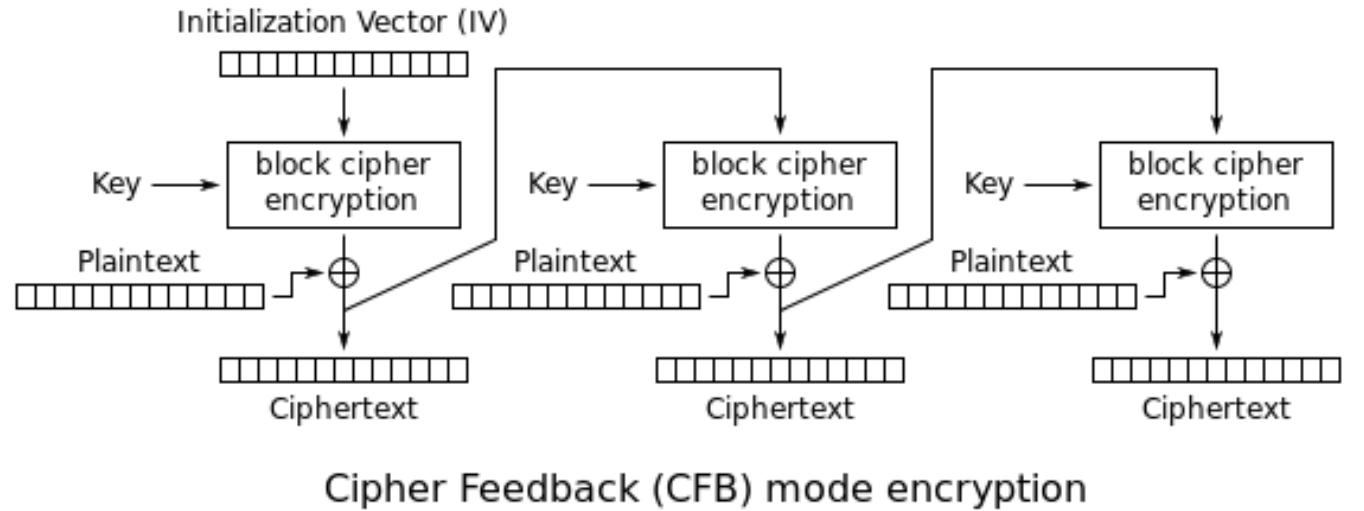
# Modul "înlăntuirea blocurilor cifrate" (Cipher Block Chaining -CBC)

- Folosim comanda `openssl enc` pentru a cifra același text în clar cu aceeași cheie, dar cu **IV** diferit

```
$ openssl enc -aes-128-cbc -e -in plain.txt -out cipher1.txt \
-K 00112233445566778899AABBCCDDEEFF \
-iv 000102030405060708090a0b0c0d0e0f
$ openssl enc -aes-128-cbc -e -in plain.txt -out cipher2.txt \
-K 00112233445566778899AABBCCDDEEFF \
-iv 000102030405060708090a0b0c0d0e0e
$ xxd -p cipher1.txt
52381c7726763ac132752bb29a32a68fc8dbcf20367fdfd03649b3a0d1744567
$ xxd -p cipher2.txt
50a9e3b81cc020d286d86fc7f1d8fb4268f9cd87c08126226c4626dbd4961d58
```

- Folosim AES cu cheie de 128-biți
- Opțiunea **-aes-128-cbc** specifică modul **CBC**
- Opțiunea **-e** indică cifrare
- Opțiunea **-iv** se folosește pentru a preciza vectorul de inițializare (IV)

# Modul Cipher Feedback (CFB)



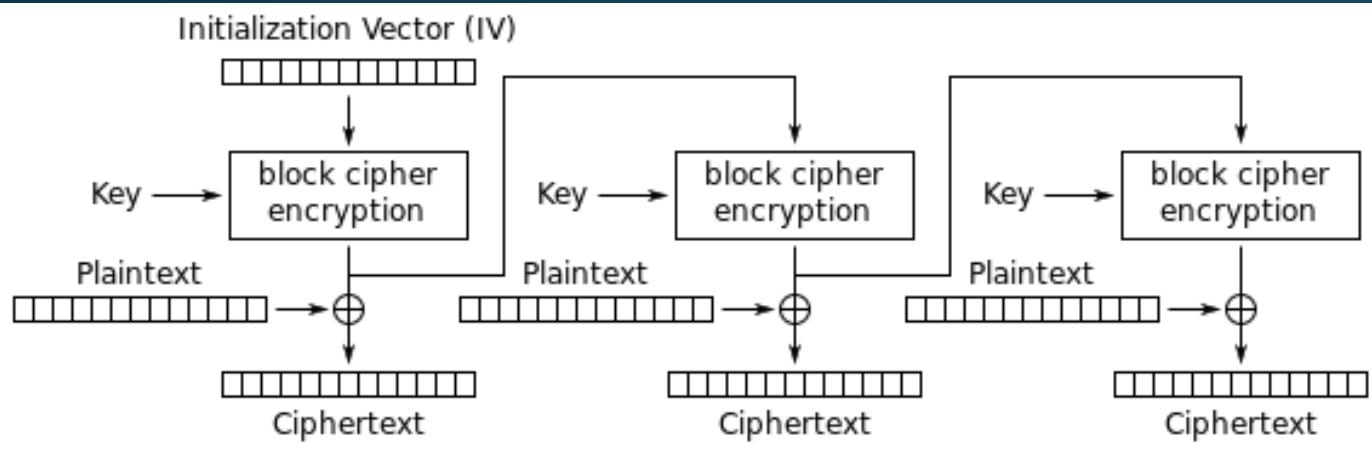
- Un cifru bloc este transformat într-un cifru flux (stream).
- Ideal pentru cifrarea datelor în timp real.
- Nu e nevoie de completare (**padding**) pentru ultimul bloc.
- Decriptarea în modul CFB se poate **paraleliza**
- Criptarea se poate face doar **secvențial**

# Compararea criptării cu CBC și CFB

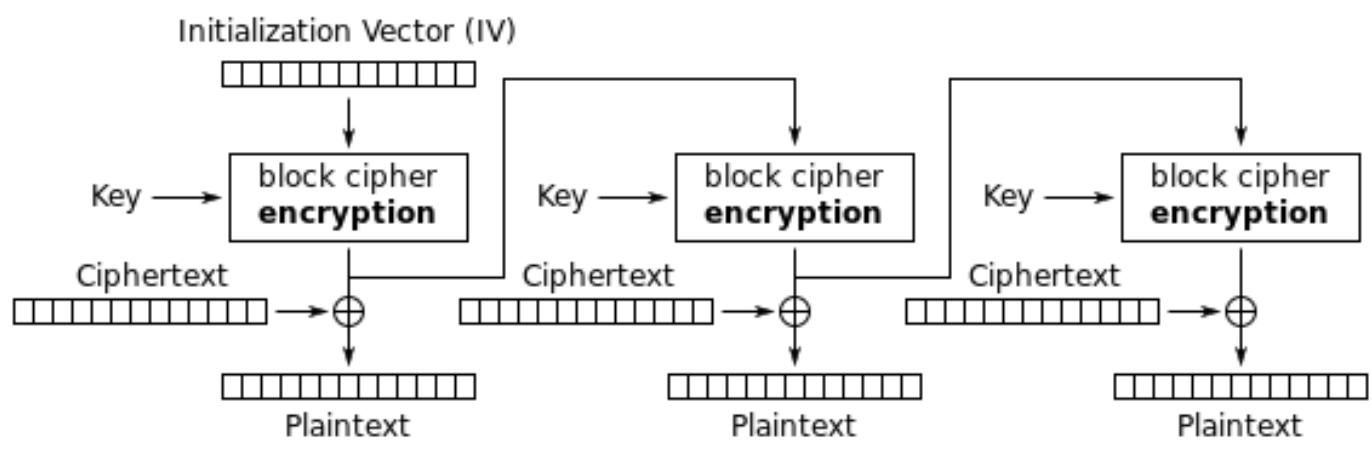
```
$ openssl enc -aes-128-cbc -e -in plain.txt -out cipher1.txt \
-K 00112233445566778899AABBCCDDEEFF \
-iv 000102030405060708090a0b0c0d0e0f
$ openssl enc -aes-128-cfb -e -in plain.txt -out cipher2.txt \
-K 00112233445566778899AABBCCDDEEFF \
-iv 000102030405060708090a0b0c0d0e0f
$ ls -l plain.txt cipher1.txt cipher2.txt
-rw-rw-r-- 1 seed seed 32 Jun 20 13:55 cipher1.txt
-rw-rw-r-- 1 seed seed 21 Jun 20 13:55 cipher2.txt
-rw-rw-r-- 1 seed seed 21 May 11 10:27 plain.txt
```

- Textul în clar este de 21 octeți
- Modul CBC: textul cifrat are 32 octeți din cauza completării (padding)
- Modul CFB: textul cifrat are aceeași lungime ca și textul în clar (21 octeți)

# Modul Output Feedback (OFB)



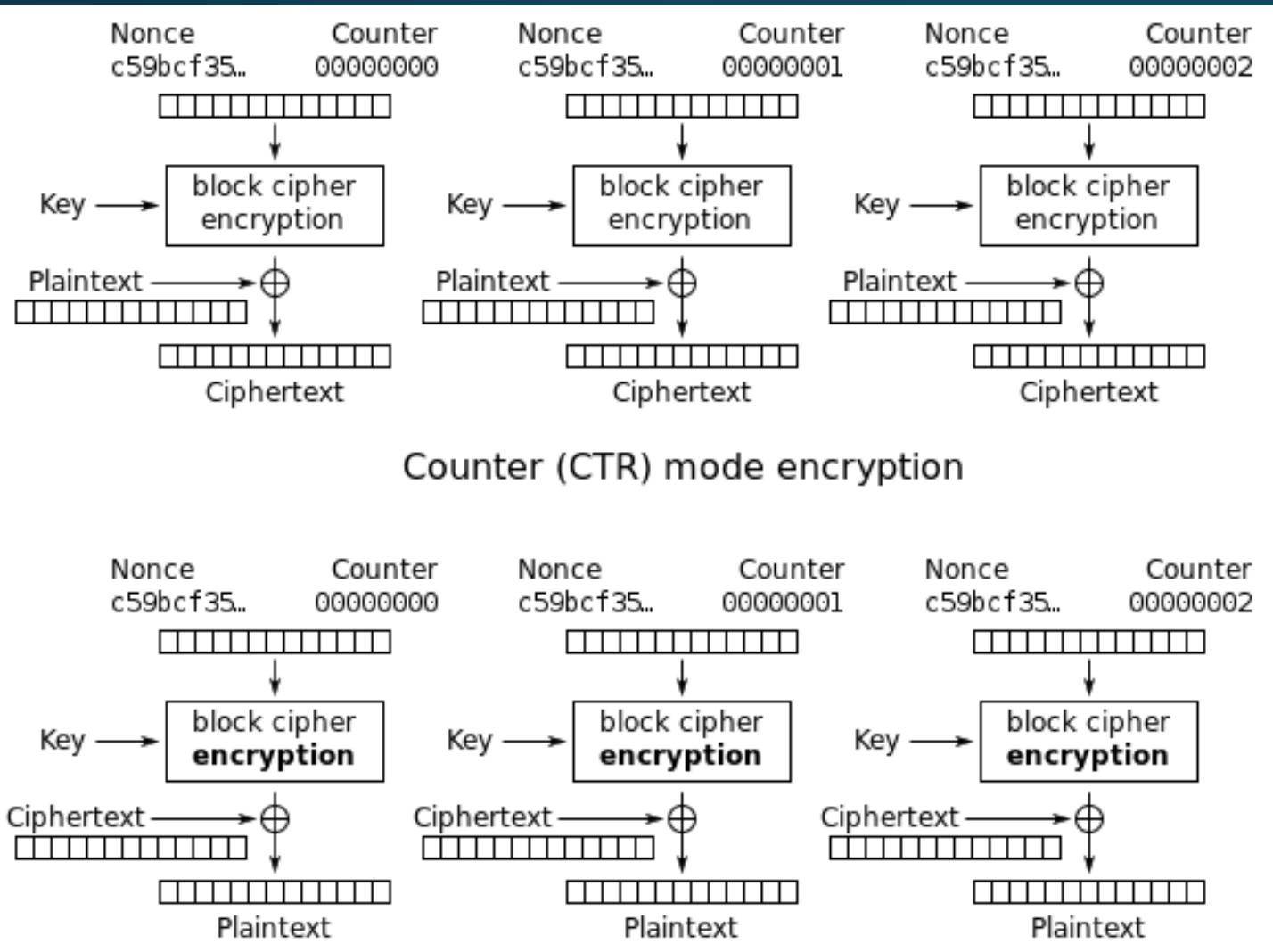
Output Feedback (OFB) mode encryption



Output Feedback (OFB) mode decryption

- Asemănător CFB
  - Folosit ca cifrul flux (stream)
  - Nu are nevoie de completare (padding)
  - Decriptarea se poate paraleliza
- Criptarea în modul OFB se poate paraleliza

# Modul "contor" (Counter - CTR)



- În esență folosește un contor pentru a genera fluxurile de chei
- Nici un flux de chei nu poate fi reutilizat. De aceea valoarea contorului pentru fiecare bloc este prefixată cu o valoare aleatoare numită *nonce*
- Acest *nonce* îndeplinește același rol ca și IV pentru celelalte moduri de criptare
- Atât cifrarea cât și descifrarea pot fi paralelizate
- Fluxul de chei din modul CTR poate fi calculat în paralel în timpul criptării

# Moduri pentru Cifrare autentificată (Authenticated Encryption)

- Nici unul dintre modurile de cifrare discutate până cum nu poate fi folosit pentru a obține autentificarea mesajului
- Au fost gândite o serie de moduri de operare pentru a combina autentificarea mesajului și criptarea.
- Exemplu:
  - GCM (Galois/Counter Mode)
  - CCM (Counter with CBC-MAC)
  - OCB (Offset Codebook Mode)

# Completarea (padding)

- Modurile de cifrare bloc împart textul în clar în blocuri și mărimea fiecărui bloc trebuie să se potrivească cu mărimea blocului cifrului folosit.
- Nu există nici o garanție că mărimea ultimului bloc se va potrivi.
- **Ultimul bloc are nevoie de completare (padding)**, adică, înainte de cifrare trebuie adăugate date suplimentare la ultimul bloc de text în clar, pentru a realiza potrivirea de lungime.
- Schemele de completare trebuie să marcheze clar unde începe completarea, a.î. descifrarea să poată elimina datele de completare.
- O schemă folosită de obicei la completare este [PKCS#5](#)

# Experiment de completare

```
$ echo -n "123456789" > plain.txt
$ openssl enc -aes-128-cbc -e -in plain.txt -out cipher.bin \
    -K 00112233445566778899AABBCCDDEEFF \
    -iv 0102030405060708090a0b0c0d0e0f
$ ls -ld cipher.bin
-rw-rw-r-- 1 seed seed 16 Jun 28 11:15 cipher.bin
$ openssl enc -aes-128-cbc -d -in cipher.bin -out plain2.txt \
    -K 00112233445566778889aabbccddeeff \
    -iv 0102030405060708
$ ls -ld plain2.txt
-rw-rw-r-- 1 seed seed 9 Jun 28 11:16 plain2.txt
```

- Textul în clar (**plain.txt**) are 9 octeți.
- Mărimea textului cifrat (**cipher.bin**) devine 16 octeți

# Experiment de completare

- Cum știe descifrarea unde începe completarea?

```
$ openssl enc -aes-128-cbc -d -in cipher.bin -out plain3.txt \
-K 00112233445566778889aabbcdddeeff \
-iv 0102030405060708 -(@*\textbf{nopad}@*)
```

```
$ ls -ld plain3.txt
-rw-rw-r-- 1 seed seed 16 Jun 28 11:18 plain3.txt
```

```
$ xxd -g 1 plain.txt
00000000: 31 32 33 34 35 36 37 38 39
$ xxd -g 1 plain3.txt
00000000: 31 32 33 34 35 36 37 38 39 07 07 07 07 07 07 07
```

Au fost adăugați 7 octeți de  
**0x07** ca date de completare

# Experiment de completare – un caz special

- Ce se întâmplă dacă lungimea textului în clar este deja multiplu de mărimea blocului (deci nu e nevoie de completare) și ultimii 7 octeți sunt toți 0x07

```
$ openssl enc -aes-128-cbc -e -in plain3.txt -out cipher3.bin \
-K 00112233445566778889aabcccddeeff \
-iv 0102030405060708
$ openssl enc -aes-128-cbc -d -in cipher3.bin -out plain3_new.txt \
-K 00112233445566778889aabcccddeeff \
-iv 0102030405060708 -(@*\textbf{nopad}@*)\n\n
$ ls -ld cipher3.bin
-rw-rw-r-- 1 seed seed 32 Jun 28 11:27 cipher3.bin
$ xxd -g 1 plain3_new.txt
00000000: 31 32 33 34 35 36 37 38 39 07 07 07 07 07 07 07
00000010: 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
```

- Mărimea textului în clar (`plain3.txt`) este 16 octeți
- Mărimea textului descifrat (`plain3_new.txt`) este 32 octeți (s-a adăugat un întreg bloc pe post de completare).
- De aici, în PKCS#5, dacă lungimea intrării este deja un multiplu de mărimea B a blocului, atunci se vor adăuga B octeți de valoare B ca și completare.

# Vectorul de initializare și greșeli comune

- Pentru vectorii de initializare există următoarele cerințe:
  - Se presupune că se stochează sau transmit ca text în clar
  - Nu trebuie să se repete (unicitate).
  - Nu trebuie să fie predictibili

# Experiment - IV nu ar trebui să fie predictibil

- Eve calculează următorul IV

```
IV_bob: 4ae71336e44bf9bf79d2752e234818a5

# Encrypt Bob's vote
$ echo -n "John Smith....." > P1
$ openssl enc -aes-128-cbc -e -in P1 -out C1 \
    -K 00112233445566778899AABBCCDDEEFF \
    -iv 4ae71336e44bf9bf79d2752e234818a5

# Calculate IV_next from IV_bob
$ echo -n 4ae71336e44bf9bf79d2752e234818a5 | xxd -r -p > IV_bob
$ md5sum IV_bob
398d01fdf7934d1292c263d374778e1a

# Therefore, IV_next is 398d01fdf7934d1292c263d374778e1a
```

# Experiment - IV nu ar trebui să fie predictibil

- Eve ghicește că Bob a votat pe John Smith, aşa că ea creează **P1\_guessed** și face XOR cu **IV\_bob** și apoi cu **IV\_next** și, în final construiește numele pentru un candidat.

```
$ echo -n "John Smith....." > P1_guessed

# Convert the ascii string to hex string
$ xxd -p P1_guessed
4a6f686e20536d6974682e2e2e2e2e2e

# XOR P1_guessed with IV_bob
$ xor.py 4a6f686e20536d6974682e2e2e2e2e2e \
          4ae71336e44bf9bf79d2752e234818a5
00887b58c41894d60dba5b000d66368b

# XOR the above result with with IV_next
$ xor.py 00887b58c41894d60dba5b000d66368b \
          398d01fdf7934d1292c263d374778e1a
39057aa5338bd9c49f7838d37911b891

# Convert the above hex string to binary and save to P2
$ echo -n "39057aa5338bd9c49f7838d37911b891" | xxd -r -p > P2
```

# Experiment - IV nu ar trebui să fie predictibil

- Eve dă numele candidatului ei (stocat în P2) mașinii de vot, care cifrează numele folosind IV\_next pe post de IV. Rezultatul este stocat în C2.
- Dacă C1 (votul cifrat al lui Bob) == C2, atunci Eve știe sigur că Bob a votat pentru “John Smith”.

```
$ openssl enc -aes-128-cbc -e -in P2 -out C2 \
-K 00112233445566778899AABBCCDDEEFF \
-iv 398d01fdf7934d1292c263d374778e1a

# Compare C1 and C2
$ xxd -p C1
7380ee1c0f9eb7dae28c1ba6a1a74310114288f771139da8ec99dfb0036e38ce
$ xxd -p C2
7380ee1c0f9eb7dae28c1ba6a1a74310114288f771139da8ec99dfb0036e38ce
```

# Programarea cu API-uri criptografice

```
#!/usr/bin/python3

from Crypto.Cipher import AES
from Crypto.Util import Padding

key_hex_string = '00112233445566778899AABBCCDDEEFF'
iv_hex_string = '000102030405060708090A0B0C0D0E0F'
key = bytes.fromhex(key_hex_string)
iv = bytes.fromhex(iv_hex_string)
data = b'The quick brown fox jumps over the lazy dog'
print("Length of data: {0:d}".format(len(data)))

# Encrypt the data piece by piece
cipher = AES.new(key, AES.MODE_CBC, iv)                      ①
ciphertext = cipher.encrypt(data[0:32])                        ②
ciphertext += cipher.encrypt(Padding.pad(data[32:], 16))       ③
print("Ciphertext: {0}".format(ciphertext.hex()))

# Encrypt the entire data
cipher = AES.new(key, AES.MODE_CBC, iv)                      ④
ciphertext = cipher.encrypt(Padding.pad(data, 16))            ⑤
print("Ciphertext: {0}".format(ciphertext.hex()))

# Decrypt the ciphertext
cipher = AES.new(key, AES.MODE_CBC, iv)                      ⑥
plaintext = cipher.decrypt(ciphertext)                         ⑦
print("Plaintext: {0}".format(Padding.unpad(plaintext, 16)))
```

- Folosim API-urile din pachetul **PyCryptodome**.
- Linia:
  1. Inițializează cîfrul
  2. Cifrează primii 32 octeți de date
  3. Cifrează restul datelor
  4. Inițializează cîfrul (începe un lanț nou)
  5. Cifrează toate datele
  6. Inițializează cîfrul pentru descifrare
  7. Descifrează

# Programarea cu API-uri criptografice

- Modurile care nu necesită completare includ CFB, OFB și CTR.
- Pentru aceste moduri, datele trimise metodei `encrypt()` pot avea lungimi arbitrare și nu e nevoie de completare.
- Exemplul de mai jos ilustrează cifrarea OFB

```
# Encrypt the data piece by piece
cipher = AES.new(key, AES.MODE_OFB, iv)
ciphertext = cipher.encrypt(data[0:20])
ciphertext += cipher.encrypt(data[20:])
```

# Atacuri asupra integrității textului cifrat

- Atacatorul modifică textul cifrat (Linia 2)

```
data = b'The quick brown fox jumps over the lazy dog'

# Encrypt the entire data
cipher = AES.new(key, AES.MODE_OFB, iv)
ciphertext = bytearray(cipher.encrypt(data))          ①

# Change the 10th byte of the ciphertext
ciphertext[10] = 0xE9                                ②

# Decrypt the ciphertext
cipher = AES.new(key, AES.MODE_OFB, iv)
plaintext = cipher.decrypt(ciphertext)                ③
print("Original Plaintext: {}".format(data))
print("Decrypted Plaintext: {}".format(plaintext))
```

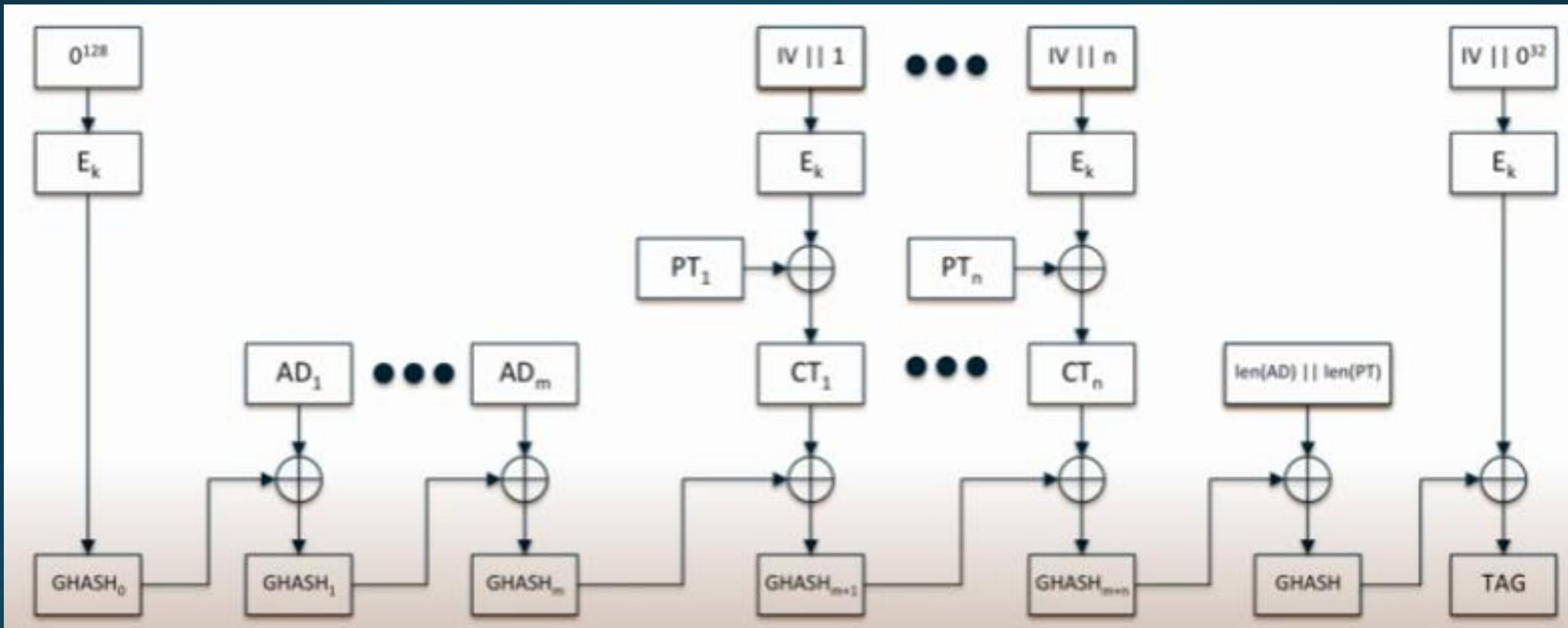
- Rezultatul

```
Original Plaintext: b'The quick brown fox jumps over the lazy dog'
Decrypted Plaintext: b'The quick grown fox jumps over the lazy dog'
```

# Criptarea autentificată

- Pentru a proteja integritatea, expeditorul trebuie să genereze un cod de autentificare a mesajului (Message Authentication Code - MAC) din textul cifrat folosind un secret partajat cu destinatarul.
- Se vor trimite la destinatar MAC și textul cifrat; acesta va calcula un MAC din textul cifrat.
- Dacă MAC este identic cu cel din mesaj, atunci textul cifrat nu a fost modificat.
- Sunt necesare două operații pentru a obține integritatea textului cifrat: una pentru criptarea datelor și o altă pentru generarea MAC.
- **Criptarea autentificată** combină aceste două operații separate într-un singur mod de cifrare. D.e. GCM, CCM (Counter with CBC-MAC), OCB (Offset codebook mode)

# Galois/Counter Mode (GCM)



# Programarea folosind modul GCM

```
#!/usr/bin/python3

from Crypto.Cipher import AES
from Crypto.Util import Padding

key_hex_string = '00112233445566778899AABBCCDDEEFF'
iv_hex_string = '000102030405060708090A0B0C0D0E0F'
key = bytes.fromhex(key_hex_string)
iv = bytes.fromhex(iv_hex_string)
data = b'The quick brown fox jumps over the lazy dog'

# Encrypt the data
cipher = AES.new(key, AES.MODE_GCM, iv) ①
cipher.update(b'header') ②
ciphertext = bytearray(cipher.encrypt(data))
print("Ciphertext: {}".format(ciphertext.hex()))

# Get the MAC tag
tag = cipher.digest() ③
print("Tag: {}".format(tag.hex()))
```

Partea diferită a codului conține generarea și verificarea etichetei (tag)

În linia ③ , folosim **digest()** pentru a obține eticheta de autentificare, care este generată din textul cifrat.

# Programarea folosind modul GCM

```
# Corrupt the ciphertext
ciphertext[10] = 0x00                                ④

# Decrypt the ciphertext
cipher = AES.new(key, AES.MODE_GCM, iv)
cipher.update(b' header')
plaintext = cipher.decrypt(ciphertext)
print("Plaintext: {0}".format(plaintext))

# Verify the MAC tag
try:
    cipher.verify(tag)                            ⑤
except:
    print("*** Authentication failed ***")
else:
    print("*** Authentication is successful ***")
```

În linia ⑥, după ce trimitem textul cifrat cîfrului, invocăm **verify()** pentru a verifica dacă eticheta mai este validă

# Experiment – Modul GCM

- Modificăm textul cifrat prin schimbarea celui de al 10-lea octet la valoarea **0x00**
- Descifrăm textul cifrat modificat și verificăm eticheta

```
$ enc_gcm.py
Ciphertext: ed1759cf244fa97f87de552c1...a11d
Tag: 701f3c84e2da10aae4b76c89e9ea8427
Plaintext: b'The quick brown fox jumps over the lazy dog'
*** Authentication failed ***
```