Marc Monfort Grau

Segona entrega

Clau Secreta

1. El cos finit

Les funcions de l'apartat ***i)*** al ***v)*** estan definides en el fitxer ***cosFinit.py***.
També estan declarades 4 funcions **check1**, **check2**, **check3**, **check4**, que comprova que es compleixin les propietats:

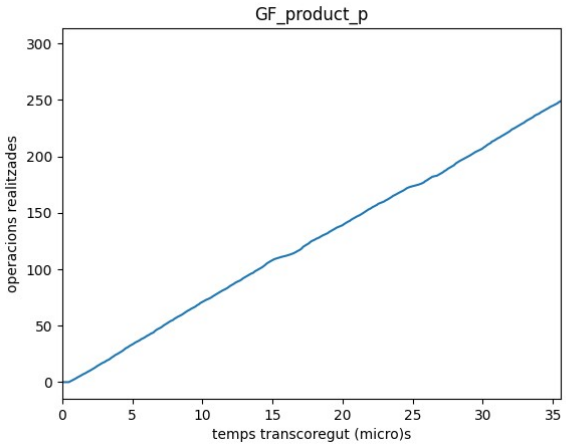
- *GF_product_p(a, b) == GF_product_t (a, b) per tot parell (a, b),*
- *GF_product_p(a, b) == GF_product_p(b, a) per tot parell (a, b),*
- *GF_product_p(a, GF invers(a)) == 1 per tot a != 0.*

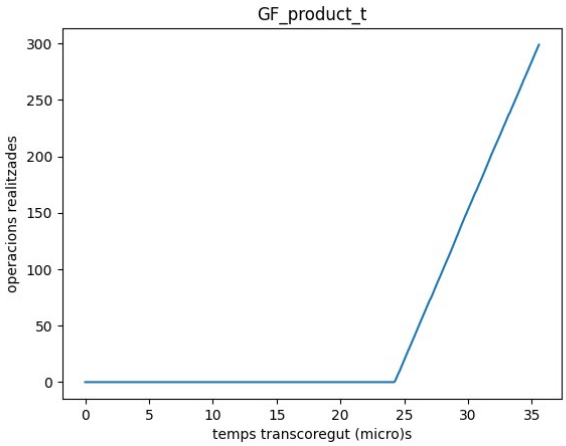
La funció ***compTaula***, genera una taula comparativa del temps d'execució en multiplicar els 256 valors del cos finit, per un altre valor fix (*0x02, 0x03, ...*), utilitzant la funció ***GF_product_p***, i comparant amb el temps d'execució la funció ***GF_product_t***, que fa ús de les taules generades (exponencials i logarítmiques).

	GF_product_p	GF_product_t	(microsegons)
0x02:	2.1842	0.4044	
0x03:	2.6107	0.4789	
0x09:	3.1628	0.4115	
0x0B:	3.659	0.4106	
0x0D:	3.9243	0.4129	
0x0E:	4.2176	0.4129	

(el resultat és la mitjana del temps d'execució en microsegons, després de realitzar 10000 iteracions, utilitzant la llibreria *timeit*)

La funció ***graphTemps*** genera dos gràfics, un primer gràfic del temps d'execució en multiplicar 300 vegades dos polinomis generats de forma aleatòria, utilitzant la funció ***GF_product_p***, i un segon gràfic del temps d'execució en genera les taules exponencials i logaritmiques utilitzant la funció ***GF_tables***, i multiplicar 300 vegades dos polinomis generats de forma aleatòria utilitzant la funció **GF_product_t**.





Com veiem en la taula comparativa i en les gràfiques, després d'haver generat les taules exponencials i logarítmiques, el temps de multiplicar polinomis del cos finit utilitzant les taules, es redueix dràsticament.

2. Advanced Encryption Standard (AES)

2.1 Efecte de les funcions elementals

En el fitxer ***efecteFuncionsElementals.py*** es troba la implementació de l'AES.

```
def checkSubBytes():
    M = random.randrange(2**128) # fixed M
    K = random.randrange(2**128) # fixed K
    C = aes.encryption(M, K)
    for i in range(128):
        Mi = M ^ (1 << i)
        Ci = aes.encryption(Mi, K)
        for j in range(i+1,128):
            Mj = M ^ (1 << j)
            Cj = aes.encryption(Mj, K)
            Mij = M ^ ((1 << i) | (1 << j))
            Cij = aes.encryption(Mij, K)
            if (C != Ci ^ Cj ^ Cij):
                return False # no es lineal (bueno)
    return True # es lineal (malo). Caso cuando ByteSub = ID
```

La funció ***checkSubBytes*** (del mateix fitxer) comprova que quan canviem la funció de l'AES ***ByteSub*** per la identitat, si fixem M i K (missatge i clau), el resultat C (xifrat de M amb K) és igua a *Ci* + *Cj* + *Cij* (on Cx és el xifrat amb la clau K, de M, amb el bit **x** modificat).

Si en canvi utilitzem la funció ***ByteSub*** original de l'AES, aquesta igualtat no es compleix, ja que el xifrat deixa de ser lineal.

```
def checkShiftMix():
    M = random.randrange(2**128) # fixed M
    K = random.randrange(2**128) # fixed K
    C = aes.encryption(M, K)
    for _ in range(10):
        Mi = M ^ (1 << random.randrange(127))
        Ci = aes.encryption(Mi, K)
        print(hex(C^Ci))
```

La funció ***checkShiftMix*** ens permet comprovar els efectes de modificar la funció original de l'AES ***ShiftRows*** i la funció ***MixColumns***, per la identitat.

Quan canviem ***ShiftRows*** per la identitat, els criptogrames C i Ci, com a resultat de xifrar amb la mateixa clau K un missatge M i un missatge Mi (M amb el bit ***i*** modificat) són molt semblants. I només presenten diferències en els bytes de la columna que és trobava el bit modificat.
Exemple de XOR entre C i Ci:

hex = 00000000000000000eb39d33d00000000

Quan canviem ***MixColumns*** per la identitat, els criptogrames C i Ci ara són gairebé idèntics. Només hi ha diferències en el mateix byte al qual li hem modificat el bit.
Exemple de XOR entre C i Ci:

hex = 00000000000000000000000000740000

Quan no fem **cap canvi** a l'algorisme de l'AES, un exemple de XOR entre C i Ci:

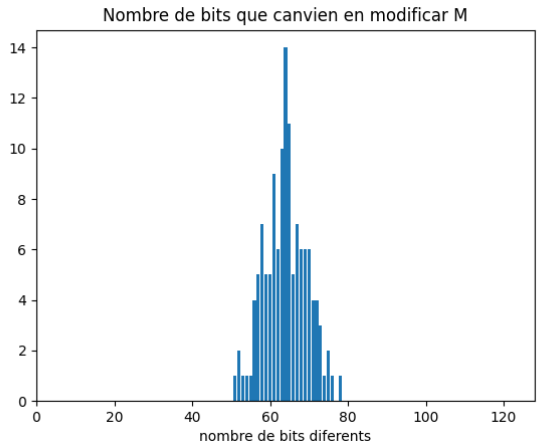
hex = cfd7832ec0d62f15fd42453df47e50eb

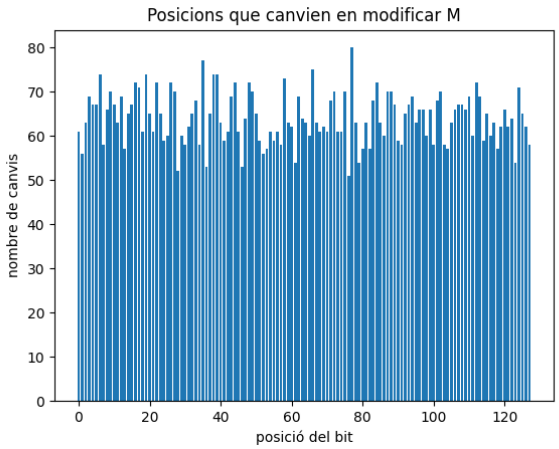
Veiem que tant ***MixColumns*** com ***ShiftRows*** són essencials per a la difusió en el xifrat.

2.2 Propagació de petits canvis

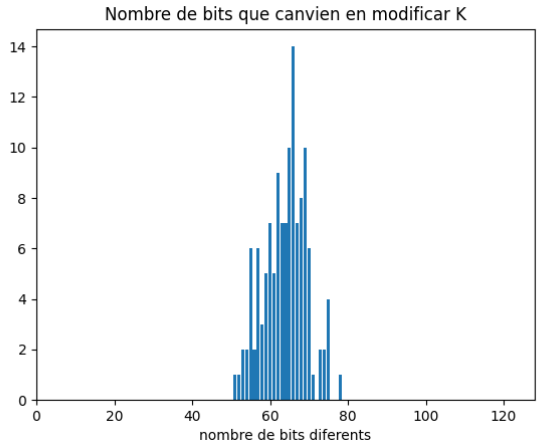
En el fitxer ***propagacioPetitsCanvis.py*** es troba el script que genera els histogrames que surten a continuació.

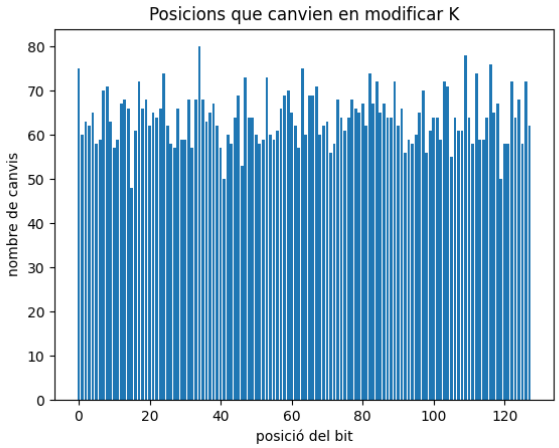
Els dos primers histogrames mostren els canvis en la sortida del xifrat AES quan modifiquem un bit del missatge d'entrada M. Utilitzant sempre la mateixa clau.





Els pròxims dos histogrames mostren els canvis en la sortida del xifrat AES quan modifiquem un bit de la clau K. Utilitzant sempre el mateix missatge.





En les dues situacions veiem que el nombre de bits modificats en el criptograma, en modificar un sol bit del missatge M i utilitzant una mateixa clau (o viceversa) es proper a 65. I les posicions d'aquests bits modificats són bastant homogènies en el rang dels 128 bits del bloc.

Aquest resultat és ideal, ja que el fet de modificar un sol bit fa que aproximadament un 50% dels bits del criptograma de sortida siguin diferents, i que aquests bits diferents estiguin distribuïts homogèniament en tot el criptograma.

2.3 Ús com a funció unidireccional

En el fitxer ***funcioUnidireccional.py*** es troba el script per resoldre els següents apartats.

a) Fixat missatge M = 0x00112233445566778899AABBCCDDEEFF

Aplicant força bruta:

Màxim nombre de zeros trobats = 31

K = 0x00000000000000000000000308ff9ec

Proves realitzades = 928555888

b) Fixada la clau K = 0x0123456789ABCDEFFEDCBA9876543210

Si desxifrem amb la clau K un criptograma C amb tot zeros, podrem trobar un missatge M, tal que en xifrar M amb la clau K, el resultat són tot 0.

Màxim nombre de zeros trobats = 128

M = 0xa9c1017e612ff97efd7587cdd073cb50

Proves realitzades = 1

c) Sense fixar res

Si desxifrem amb qualsevol clau un criptograma C amb tot zeros, podrem trobar un missatge M, tal que en xifrar M amb la clau K, el resultat són tot 0.

Màxim nombre de zeros trobats = 128

M = 0x39e48c87d01430bfc6f7c9818ba138f6

K = 0x45c7253be8577b96fb23e96c65302b2a

Proves realitzades = 1

3 Criptografia de clau secreta

3.1. Per desxifrar el primer fitxer he utilitzat el script ***desxifrar_AES.py.*** El fitxer desxifrat és la imatge ***sabana.jpeg.*** Ha sigut xifrada utilitzant AES en mode CFB amb vector d'inicialització.

3.2. Per resoldre aquest apartat he utilitzat el script ***puerta_trasera.py.*** El fitxer desxifrat és el vídeo ***musica.mp4****.*