Facultat d'Informàtica de Barcelona

Quadrimestre de tardor, 2022/2023

Segona pràctica de Criptografia

Clau Secreta

Guillem González Valdivia
(guillem.gonzalez.valdivia@Estudiantat.upc.edu)
Daniel Morón Roces
(daniel.moron.roces@Estudiantat.upc.edu)

Índex

1 Chacha20

1.1 Propagació de petits canvis

Amb una clau K de 256 bits qualsevol hem fet una estadística dels bits que canvien en la sortida per diferents valors del Counter = 2,3, ..., 4096 comparant amb Counter = 1.

Primerament, hem calculat per cada valor del Counter el nombre de bits que han variat respecte a la sortida amb Counter = 1, aquesta es la primera gràfica de cada apartat. Respecte a la segona comparació el que hem fet ha sigut comparar les posicions dels bits que han canviat respecte a la sortida amb Counter = 1. A les diferents gràfiques que hem anat generant es pot veure la freqüència tant del nombre total de bits que canvien com les seves posicions corresponents.

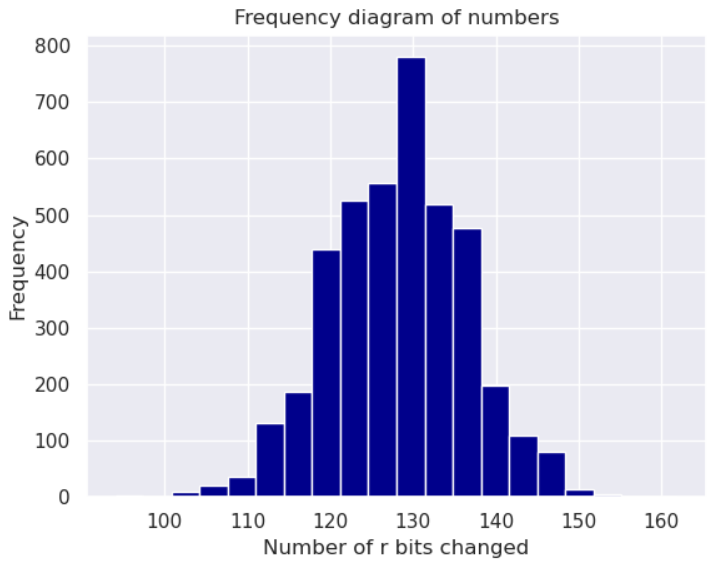

Figura 1: Freqüència del nombre de bits que han variat respecte a la sortida amb Counter=1

Es pot veure que aquest histograma segueix una distribució normal, i té sentit perquè al emmagatzemar el nombre de bits que canvien es van guardant valors de manera aleatòria i això provoca que es mostri la gràfica amb aquesta distribució.

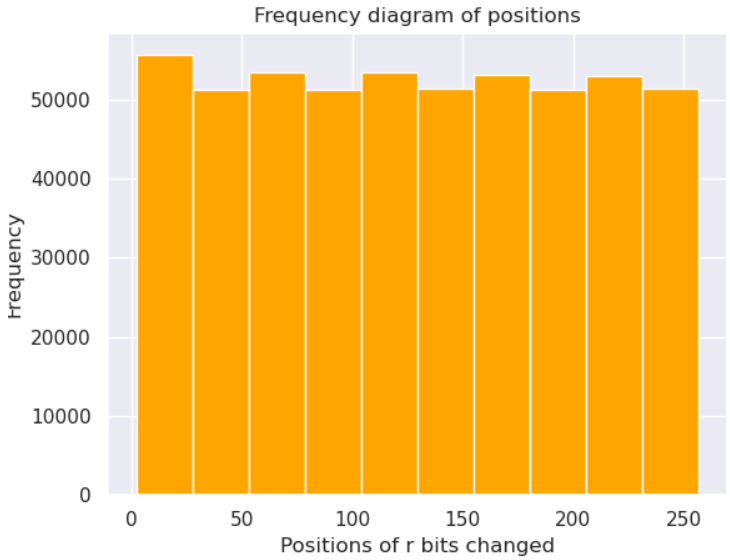

Figura 2: Freqüència de les posicions dels bits que han variat respecte a la sortida amb Counter=1

En canvi, en aquesta segona es pot observar que es una gràfica més constant, ja que realment les posicions dels bits que canvien si es fan suficients proves es pot veure que tenen una freqüència molt semblant.

1.2 Efectes de les funcions elementals

1.2.1 Eliminació dels QUARTERROUND de les Column Rounds

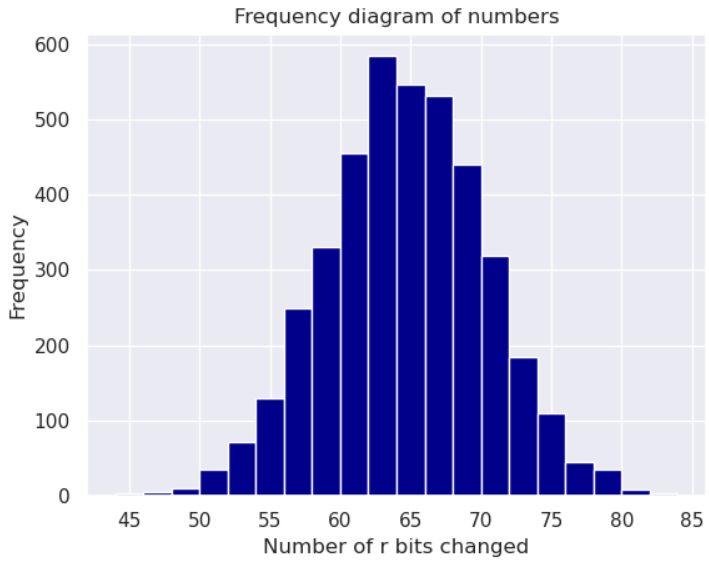

Figura 3: Freqüència de les posicions dels bits que han variat respecte a la sortida amb Counter=1

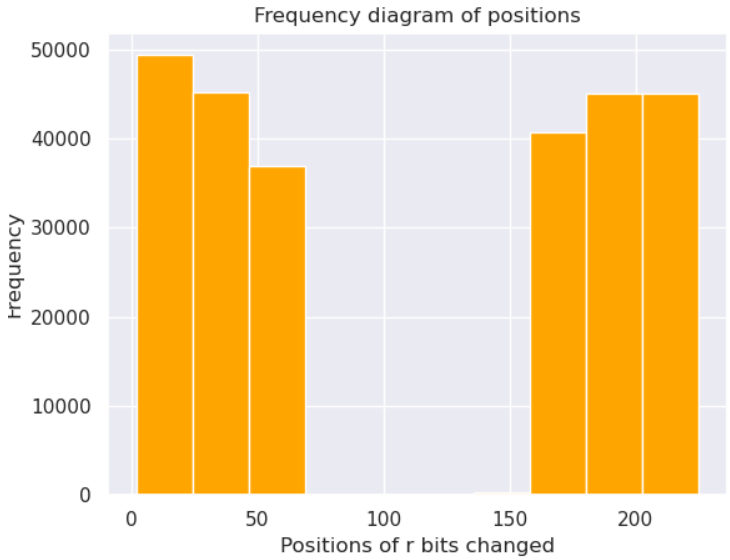

Figura 4: Freqüència de les posicions dels bits que han variat respecte a la sortida amb Counter=1

En aquest cas, al eliminar els QUARTERROUND de les Column Rounds ha provocat que aproximadament la meitat de les posicions no es modifiquin, concretament de les posicions que hem eliminat a l'hora de fer les rondes. Tot i això, es pot veure que es continua generant la distribució normal.

1.2.2 Eliminació dels QUARTERROUND de les Diagonal Rounds

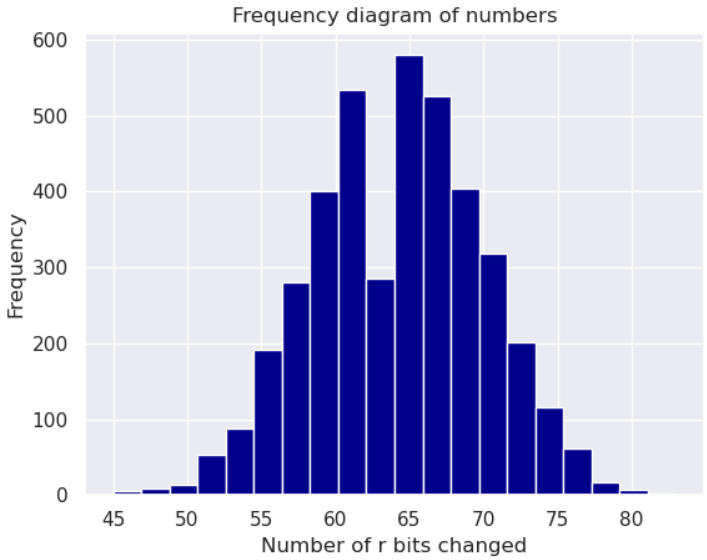

Figura 5: Freqüència de les posicions dels bits que han variat respecte a la sortida amb Counter=1

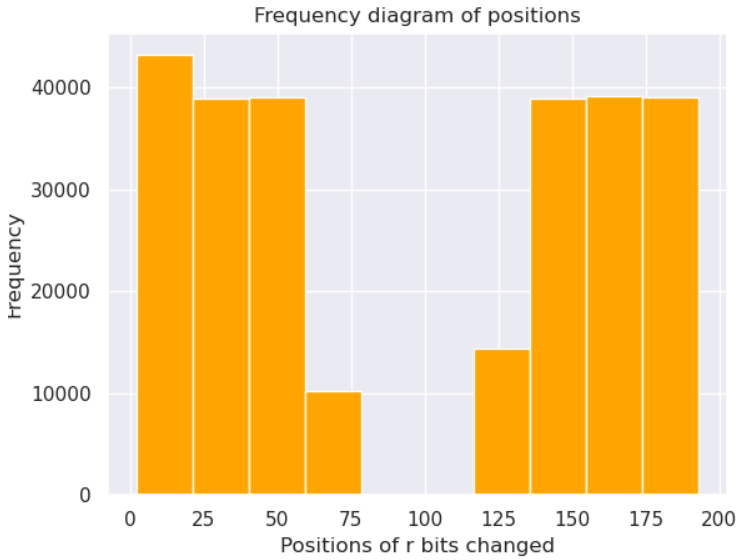

Figura 6: Freqüència de les posicions dels bits que han variat respecte a la sortida amb Counter=1

En aquest apartat, podem veure que la freqüència de les posicions han disminuit però no tant en comparació amb l'eliminació de Column Rounds, això és perquè a les rondes hi ha menys opcions a l'hora de modificar-se bits en posicions que pertanyin a les Diagonals.

1.2.3 Eliminació de dos QUARTERROUND concrets

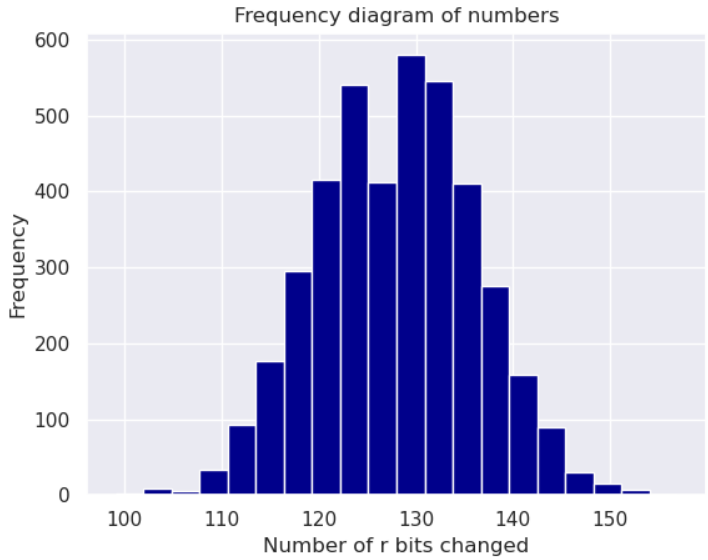

Figura 7: Freqüència de les posicions dels bits que han variat respecte a la sortida amb Counter=1

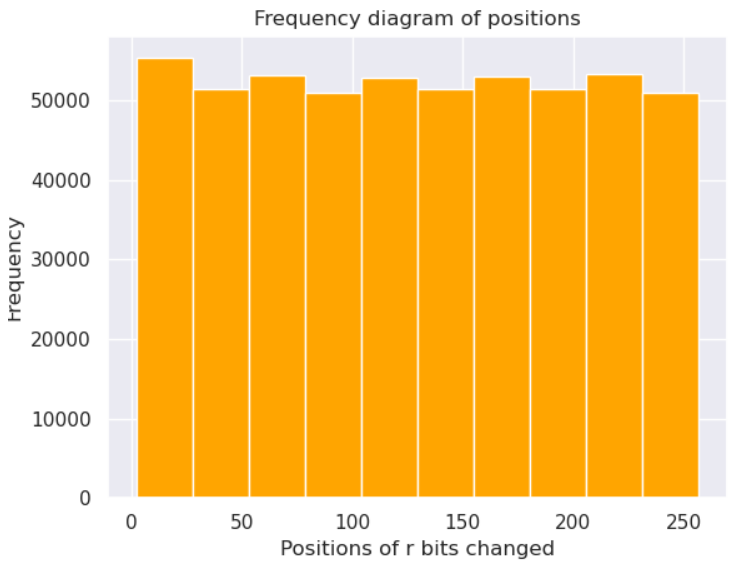

Figura 8: Freqüència de les posicions dels bits que han variat respecte a la sortida amb Counter=1

Finalment, veiem en aquesta secció que no hi ha molts canvis respecte a la primera. Això es degut a que al modificar dos QUARTERROUND concrets no es pot apreciar molta diferència perquè un forma part de les Column Rounds i un altre de Diagonal Rounds.

2 El cos finit GF(2^8)

Veiem el codi necessari per fer els càlculs en el cos finit GF(2^8) :

```
from pkgutil import extend_path
import galois
from time import time
from komm import BinaryPolynomial

# Utils

# function for extended Euclidean Algorithm
def gcdExtended(a, b):

    if a == 0 :
        return b,0,1

    gcd,x1,y1 = gcdExtended(b%a, a)

    x = y1 - (b//a) * x1
    y = x1

    return gcd,x,y

# Functions

def irreducible_polynomials():
    """
    Computes the list of irreductible polynomials of GF(2**8)
    """
    res = []

    for i in range(256, 512):
        aux = []
        for div in range(1,i+1):
            i_pol = BinaryPolynomial(i)
            div_pol = BinaryPolynomial(div)

            r = i_pol % div_pol
            if r == 0:
                aux.append(div)
        if aux.sort() == [1, i]:
            res.append(i)

    return res

def GF_product_p(a: int, b: int):
    """
    Computes the product of a and b in finite field GF(2**8)
    """

    res = 0
    while b > 0:
        if b % 2 != 0:
            res = res ^ a
        a <<= 1
```

```
        if a > 0xff:
            a = a ^ m
        b >>= 1
    return res

def GF_es_generador(a: int):
    """
    Returns wether the integer a is generator of the field GF(2**8) or
    ↪ not
    """
    if a == 0:
        return False
    res = 1
    for i in range(1, 256):
        res = GF_product_p(res, a)
        if res == 1:
            if i == 255:
                return True
            else:
                return False
    return False

def GF_tables():
    """
    For a given generator m, computes the exponential table (for every i,
    ↪ exponential[i] = generator**i == a)
    and logarithmic table (for every position a, logarithmic[a] = i)
    """

    gen = 6   # Generador
    global exp
    global log

    exp_res = 1
    exp = []
    for _ in range(1, 256):
        exp.append(exp_res)
        exp_res = GF_product_p(exp_res, gen)
    exp.append(1)

    # Logarithmic
    log_res = 1
    log = [None for _ in range(0, 256)]
    for i in range(0, 255):
        log[log_res] = i
        log_res = GF_product_p(log_res, gen)

    return

def GF_product_t(a, b):
    """
    Computes the product of two integers a and b using exponential and
    ↪ logarithmic tables
    """
    if a == 0x00 or b == 0x00:
        return 0x00
```

```
106         else:
107             return exp[(log[a] + log[b]) % 255]
108 
109     def GF_invers(a: int):
110         """
111         Computes the inverse poly1d of a given a, represented by its integer
    ↪    form
112         """
113         if a == 0:
114             return None
115         else:
116             return exp[255 - log[a]]
117 
118     # Test
119 
120     def property_testing():
121 
122         # PROPERTIES TESTS
123         for a in range(0,256):
124             for b in range(0,256):
125                 # Property n1
126                 assert GF_product_p(a, b) == GF_product_t(a, b)
127                 # Property n2
128                 assert GF_product_p(a, b) == GF_product_p(b, a)
129 
130         for a in range(1,256):
131             # Property n3
132             assert GF_product_p(a, GF_invers(a)) == 1
133 
134 
135     def test():
136 
137         # Precomputations
138 
139         GF_tables()
140 
141         print(exp)
142         print(log)
143 
144         # TESTS
145 
146         a = 125
147 
148         # TEST 1
149         t1 = time()
150         p = GF_product_p(a, 0x02)
151         tf1 = time() - t1
152         t2 = time()
153         t = GF_product_t(a, 0x02)
154         tf2 = time() - t2
155         print("Test número 1: a = " + str(hex(a)) + ", 0x02")
156         print(f"Resultado: {p} == {t}")
157         print(f"Tiempo: {tf1} vs {tf2}\n")
158 
159         # TEST 2
160         t1 = time()
```

```
    p = GF_product_p(a, 0x03)
    tf1 = time() - t1
    t2 = time()
    t = GF_product_t(a, 0x03)
    tf2 = time() - t2
    print("Test número 2: a = " + str(hex(a)) + ", 0x03")
    print(f"Resultado: {p} == {t}")
    print(f"Tiempo: {tf1} vs {tf2}\n")

    # TEST 3
    t1 = time()
    p = GF_product_p(a, 0x09)
    tf1 = time() - t1
    t2 = time()
    t = GF_product_t(a, 0x09)
    tf2 = time() - t2
    print("Test número 3: a = " + str(hex(a)) + ", 0x09")
    print(f"Resultado: {p} == {t}")
    print(f"Tiempo: {tf1} vs {tf2}\n")

    # TEST 4
    t1 = time()
    p = GF_product_p(a, 0x0b)
    tf1 = time() - t1
    t2 = time()
    t = GF_product_t(a, 0x0b)
    tf2 = time() - t2
    print("Test número 4: a = " + str(hex(a)) + ", 0x0b")
    print(f"Resultado: {p} == {t}")
    print(f"Tiempo: {tf1} vs {tf2}\n")

    # TEST 5
    t1 = time()
    p = GF_product_p(a, 0x0d)
    tf1 = time() - t1
    t2 = time()
    t = GF_product_t(a, 0x0d)
    tf2 = time() - t2
    print("Test número 5: a = " + str(hex(a)) + ", 0x0d")
    print(f"Resultado: {p} == {t}")
    print(f"Tiempo: {tf1} vs {tf2}\n")

    # TEST 6
    t1 = time()
    p = GF_product_p(a, 0x0e)
    tf1 = time() - t1
    t2 = time()
    t = GF_product_t(a, 0x0e)
    tf2 = time() - t2
    print("Test número 6: a = " + str(hex(a)) + ", 0x0e")
    print(f"Resultado: {p} == {t}")
    print(f"Tiempo: {tf1} vs {tf2}\n")


    # PROPERTIES TESTING
    property_testing()
```

```
217
218 m = 395
219 GF28 = galois.GF(2**8)
220
221 """Tables"""
222 exp: list = []
223 log: list = []
224
225 if __name__ == '__main__':
226
227     test()
```

Les següents taules comparen el temps d'execució d'algunes operacions. Suposem a un polinomi interpretar per un enter entre 0 i 255 qualsevol:

2.1 Test 1: GF-product-p(a, 0x02)

Polinomi a	Polinomi b	t(Gf-product-p(a,b))	t(Gf-product-t(a,b))
0x7d	0x02	1.6689 μs	1.1920 μs

2.2 Test 2: GF-product-p(a, 0x03)

Polinomi a	Polinomi b	t(Gf-product-p(a,b))	t(Gf-product-t(a,b))
0x7d	0x03	2.8610 μs	1.1920 μs

2.3 Test 3: GF-product-p(a, 0x09)

Polinomi a	Polinomi b	t(Gf-product-p(a,b))	t(Gf-product-t(a,b))
0x7d	0x09	3.0994 μs	0.47683 μs

2.4 Test 4: GF-product-p(a, 0x0b)

Polinomi a	Polinomi b	t(Gf-product-p(a,b))	t(Gf-product-t(a,b))
0x7d	0x0b	4.5299 μs	0.95367 μs

2.5 Test 5: GF-product-p(a, 0x0d)

Polinomi a	Polinomi b	t(Gf-product-p(a,b))	t(Gf-product-t(a,b))
0x7d	0x0d	4.2915 μs	0.95367 μs

2.6 Test 6: GF-product-p(a, 0x0e)

Polinomi a	Polinomi b	t(Gf-product-p(a,b))	t(Gf-product-t(a,b))
0x7d	0x0d	4.0531 μs	0.95367 μs

Podem observar que hi ha una diferencia més notable entre unes operacions que entre unes altres. Això és degut al temps que triga en executar-se el bucle de la multiplicació, de tal manera que, quant més gran sigui el segon polinomi a multiplicar, més triga la multiplicació. També es pot expressar com: si els polinomis són de grau major, la seva multiplicació serà més lenta, és a dir que es triga més temps.

3 Criptografia de clau secreta

En aquest tercer apartat de la pràctica hem agafat una implementació de l'AES amb l'objectiu de desencriptar els fitxers encriptats amb el nostre nom, concretament hem importat l'algorisme a partir del mòdul Crypto.Cipher.

3.1 Amb informació de la clau i el vector iv

Per arribar a aconseguir la informació dels fitxers encriptats, ho vam fer a partir de la llibreria esmentada anteriorment. Primer de tot, vam llegir la clau i el vector iv de la informació que ens proporcionava la carpeta AES d'Atenea. Una vegada obtenida la informació corresponent vam estar provant els diferents modes d'operació de l'AES i finalment va funcionar amb el mode d'operació OFB. Al final del document adjuntem el codi corresponent.

Amb l'objectiu d'obtenir més informació relacionada amb els fitxers, vam executar la comanda file "nom_fitxer", la qual ens va donar les següents respostes per cada cas.

```
$ file solution_DanielMoron
solution_DanielMoron: JPEG image data, JFIF standard 1.01, resolution (DPI), density 72x72, segment length 16, progressive, precision 8, 640x364, components 3

$ file solution_GuillemGonzalez
solution_GuillemGonzalez: JPEG image data, JFIF standard 1.01, resolution (DPI), density 72x72, segment length 16, progressive, precision 8, 640x799, components 3
```

Figura 9: Execucions per veure el tipus de fitxer

Els fitxers resultants després de desencriptar-los correctament són imatges en format '.jpeg', les quals adjuntem a continuació.

Figura 10: Imatge del fitxer encriptat amb el nom Daniel Morón Roces

Figura 11: Imatge del fitxer encriptat amb el nom Guillem González Valdivia

3.2 Sense conèixer la clau i el vector iv

Una vegada desencriptat els primers dos fitxers, en aquest apartat hem hagut de desencriptar els altres dos sense conèixer ni la clau ni el vector iv. Per tal de generar la key hem provat totes les combinacions possibles dins del conjunt especificat a l'enunciat fins arribar a un fitxer amb una extensió que tingui sentit. Al final del document adjuntem el codi corresponent.

4 Codi

En aquesta secció adjuntem els diferents codis que hem utilitzat al llarg d'aquesta pràctica.

4.1 Exercici ChaCha20

```
"""Pure Python 3 implementation of the ChaCha20 stream cipher.
It works with Python 3.5 (and probably also earler Python 3.x).
Based on https://gist.github.com/cathalgarvey/0ce7dbae2aa9e3984adc
Based on Numpy implementation: https://gist.github.com/chiiph/6855750
Based on http://cr.yp.to/chacha.html
More info about ChaCha20: https://en.wikipedia.org/wiki/Salsa20
"""

import struct
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

from numpy import *
import os
import sys
import csv
from random import randint

def yield_chacha20_xor_stream(key, iv, position=0):
  """Generate the xor stream with the ChaCha20 cipher."""
  if not isinstance(position, int):
    raise TypeError
  if position & ~0xffffffff:
    raise ValueError('Position is not uint32.')
  if not isinstance(key, bytes):
    raise TypeError
  if not isinstance(iv, bytes):
    raise TypeError
  if len(key) != 32:
    raise ValueError
  if len(iv) != 8:
    raise ValueError

  def rotate(v, c):
    return ((v << c) & 0xffffffff) | v >> (32 - c)

  def quarter_round(x, a, b, c, d):
    x[a] = (x[a] + x[b]) & 0xffffffff
    x[d] = rotate(x[d] ^ x[a], 16)
    x[c] = (x[c] + x[d]) & 0xffffffff
    x[b] = rotate(x[b] ^ x[c], 12)
    x[a] = (x[a] + x[b]) & 0xffffffff
    x[d] = rotate(x[d] ^ x[a], 8)
    x[c] = (x[c] + x[d]) & 0xffffffff
    x[b] = rotate(x[b] ^ x[c], 7)

  ctx = [0] * 16
  ctx[:4] = (1634760805, 857760878, 2036477234, 1797285236) # text expand
  ↪  32-byte k
```

```
  ctx[4 : 12] = struct.unpack('<8L', key)
  ctx[12] = ctx[13] = position
  ctx[14 : 16] = struct.unpack('<LL', iv)
  while 1:
    x = list(ctx)
    for i in range(10):
      #quarter_round(x, 0, 4,  8, 12)
      quarter_round(x, 1, 5,  9, 13)
      quarter_round(x, 2, 6, 10, 14)
      quarter_round(x, 3, 7, 11, 15)
      quarter_round(x, 0, 5, 10, 15)
      #quarter_round(x, 1, 6, 11, 12)
      quarter_round(x, 2, 7,  8, 13)
      quarter_round(x, 3, 4,  9, 14)
    for c in struct.pack('<16L', *(
        (x[i] + ctx[i]) & 0xffffffff for i in range(16))):
      yield c
    ctx[12] = (ctx[12] + 1) & 0xffffffff
    if ctx[12] == 0:
      ctx[13] = (ctx[13] + 1) & 0xffffffff


def chacha20_encrypt(data, key, iv=None, position=0):
  """Encrypt (or decrypt) with the ChaCha20 cipher."""
  if not isinstance(data, bytes):
    raise TypeError
  if iv is None:
    iv = b'\0' * 8
  if isinstance(key, bytes):
    if not key:
      raise ValueError('Key is empty.')
    if len(key) < 32:
      # TODO(pts): Do key derivation with PBKDF2 or something similar.
      key = (key * (32 // len(key) + 1))[:32]
    if len(key) > 32:
      raise ValueError('Key too long.')

  return bytes(a ^ b for a, b in
      zip(data, yield_chacha20_xor_stream(key, iv, position)))

def byte_xor(ba1, ba2):
    return bytes([_a ^ _b for _a, _b in zip(ba1, ba2)])


def run_tests():
  import binascii
  uh = lambda x: binascii.unhexlify(bytes(x, 'ascii'))
  ciphertext = b'00000000000000000000000000000000'
  key =  uh('00000000000000000000000000000000')
  iv = uh('0000000000000000')
  res_ini = chacha20_encrypt(b'\0' * len(ciphertext), key, iv, 1)
  infoX = []
  infoFreqY = []
  info_pos = []
  for count in range(2, 4097):
    res = chacha20_encrypt(b'\0' * len(ciphertext), key, iv, count)
```

```
    xor_ex = byte_xor(res_ini, res)
    binary_val = bin(int.from_bytes(xor_ex, "little"))
    for pos in range(2, len(binary_val)):
      if binary_val[pos] == '1':
        info_pos.append(pos)

    numberDif = binary_val.count("1")
    infoX.append(count)
    infoFreqY.append(numberDif)

  # 1.1 number of frequencies that r bits have changed ----- GRAPHIC 1
  sns.set()
  plt.hist(infoFreqY, bins=20, facecolor='darkblue')
  plt.title("Frequency diagram of numbers")
  plt.xlabel("Number of r bits changed")
  plt.ylabel("Frequency")
  plt.show()

  # 1.2 number of frequencies that the positions x of bits have changed
  ↪  ---- GRAPHIC 2
  sns.set()
  plt.hist(info_pos, facecolor='orange')
  plt.title("Frequency diagram of positions")
  plt.xlabel("Positions of r bits changed")
  plt.ylabel("Frequency")
  plt.show()
if __name__ == "__main__":
  run_tests()
```

4.2 Exercici 3.1

```python
from curses import KEY_B2
from Crypto.Cipher import AES


def decrypt(key, iv, encryptedText):
    cip = AES.new(key, AES.MODE_OFB, iv)
    return cip.decrypt(encryptedText)

if __name__ == '__main__':
    fileInf = open('AES_daniel.moron.roces_2022_09_29_11_10_26.enc', 'rb')
    fileIV = open('AES_daniel.moron.roces_2022_09_29_11_10_26.iv', 'rb')
    fileKey = open('AES_daniel.moron.roces_2022_09_29_11_10_26.key', 'rb')

    key = fileKey.read()
    iv =  fileIV.read()
    encryptedText = fileInf.read()

    #3.1 Decrypting the plain text encrypted with mode OFB
    out = open("out", 'wb')
    out.write(decrypt(key, iv, encryptedText))


    fileInf.close()
    fileIV.close()
    fileKey.close()
    out.close()

```

4.3 Exercici 3.2

```python
import subprocess
import hashlib

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

cjt =  "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"


def gen_key():
    for fst_chr in cjt:
        for snd_chr in cjt:
            yield f"{fst_chr * 8}{snd_chr * 8}"


def read_file(path):
    with open(path, 'rb') as file:
        return file.read()


def write_file(path, data):
    with open(path, 'wb') as file:
```

```
        file.write(data)


if __name__ == "__main__":
    key_gen = gen_key()
    i = 0
    iter = 0

    while True:
        iter += 1
        try:
            H = hashlib.sha256(next(key_gen).encode()).digest()
            k = H[:16]
            iv = H[16:]

            encText = read_file('AES_daniel.moron.roces_2022_09_20_16_59_19.puerta_trasera.enc')
            aes = AES.new(k, AES.MODE_CBC, iv)
            decText = unpad(aes.decrypt(encText), AES.block_size)
        except ValueError:
            pass
        except StopIteration:
            break

        else:
            write_file('aux_dec', decText)
            file_type = subprocess.run(['file', '-b', 'aux_dec'], stdout=subprocess.PIPE)

            if 'data' not in file_type.stdout.decode('utf-8'):
                write_file(f"decripted{i}", decText)
                i += 1
```

5 Referències

https://www.dlitz.net/software/pycrypto/api/2.6/Crypto.Cipher.AES-module.html