SPSI - PRÁCTICA 3
ENCRIPTACIÓN SIMÉTRICA

Rafael Lachica Garrido

Ejercicio 7) Clasificar los algoritmos de 1 en “algoritmos clásicos”, “algoritmos de cifrado en bloque” y “algoritmos en flujo” para poder proceder con el resto de la práctica.
Fichero entrada: IN.TXT = “HOLA MUNDO”

Algoritmos clásicos: Caesar, Enigma,Vigenere, Xor, Transposición, ROT13.
Algoritmos de cifrado en bloque: Blowfish, Crypt, DES, IDEA, KHUFU, Lucifer, RSA, Tiny.
Algoritmos de flujo: CDROM, getopt, HEX, ISPRIME, Nextfile, Hash, OTP, RANDOM,
Template, LZSS.

Fuente: https://en.wikipedia.org/wiki/Category:Classical_ciphers
https://en.wikipedia.org/wiki/Stream_cipher HYPERLINK "https://en.wikipedia.org/wiki/Block_cipher"cipher

Ejercicio 8) Hacer varios ejemplos de ejecución de los algoritmos, al menos con dos algoritmos de cada uno de los grupos anteriores. Describir en qué consiste la llave.

Algoritmos Clásicos:

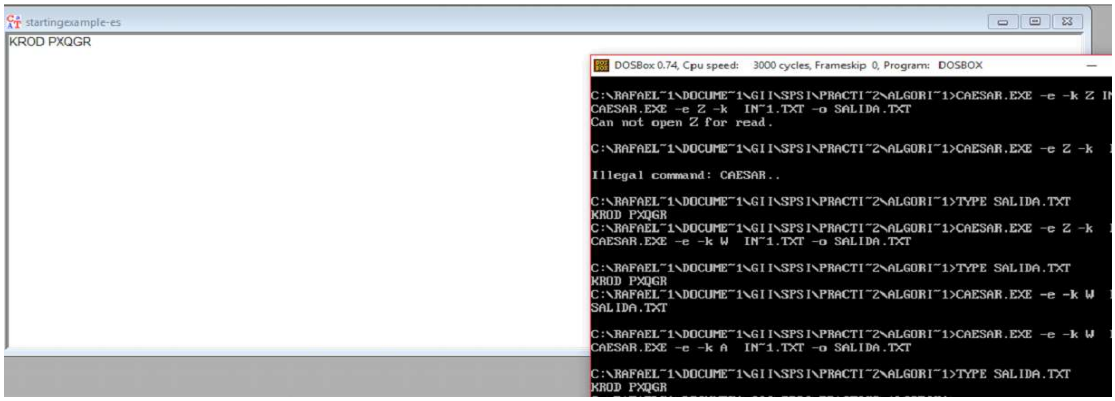
Cesar
Clave K, desplazamiento del alfabeto, empieza desde K.

```
C:\RAFAEL~1\DOCUME~1\GII\SPSI\PRACTI~2\ALGORI~1>CAESAR.EXE -e -k Z IN~1.TXT -o S
SALIDA.TXT

C:\RAFAEL~1\DOCUME~1\GII\SPSI\PRACTI~2\ALGORI~1>TYPE SALIDA.TXT
KROD PXQGR
C:\RAFAEL~1\DOCUME~1\GII\SPSI\PRACTI~2\ALGORI~1>
```

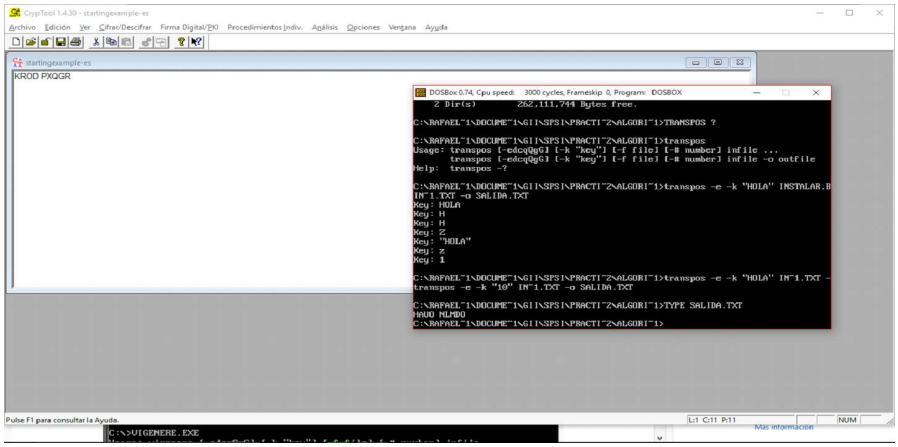
SALIDA.TXT

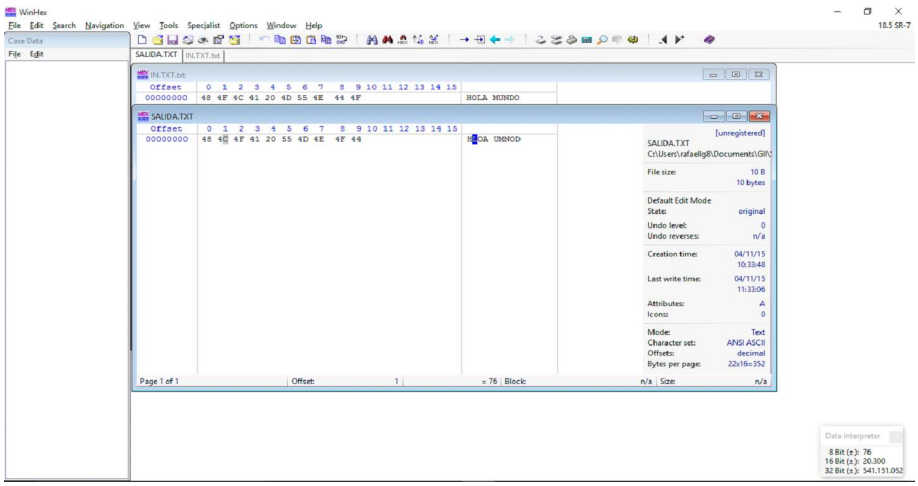
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	4B	52	4F	44	20	50	58	51	47	52							KROD PXQGR



Para César, vemos que pongamos la clave que pongamos siempre hace un desplazamiento de la A → C, B → E y así sucesivamente, independientemente de las claves usadas.

Traspose:
Clave: 1

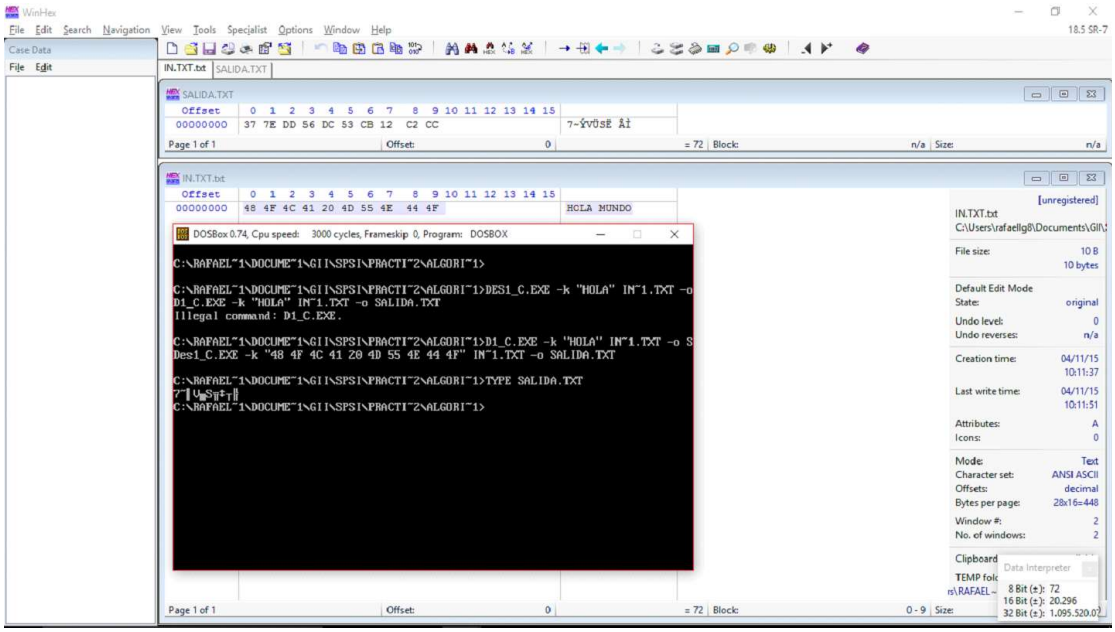




Se hace una transposición de una posición en la matriz que se elabora de letras.

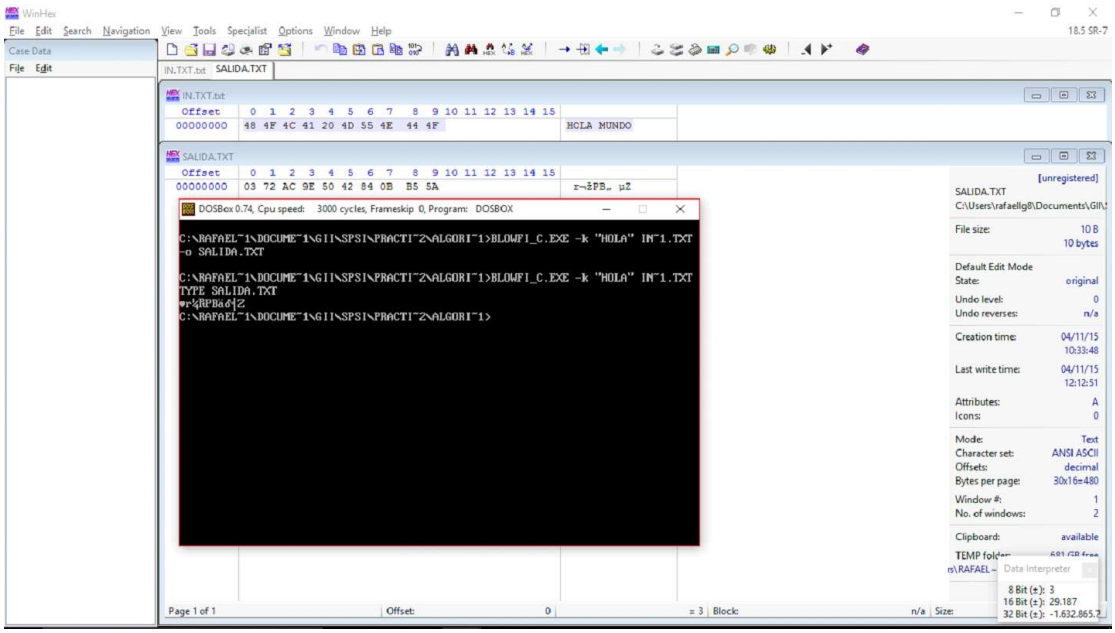
Algoritmos de bloque:

DES_1



Aquí vemos el cifrado en caracteres ASCII de DES.
DES: 64 bit / key, 64 bit bloque, llave en ASCII.
Todo esto lo podemos encontrar en DES_1.c

Blowfish



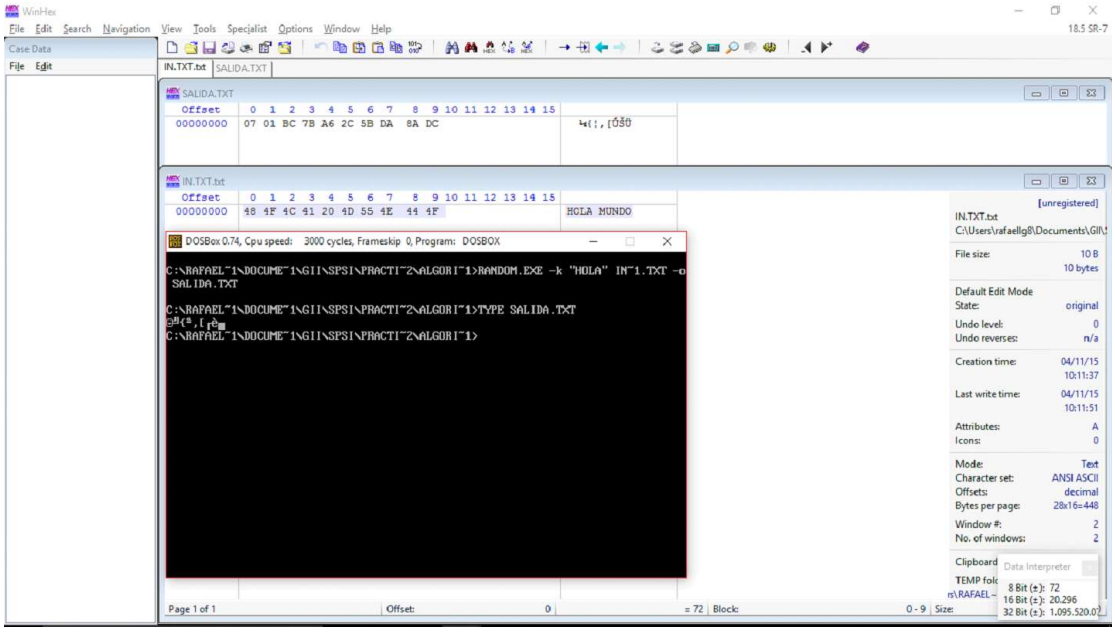
Blowfish usa bloques de 64 bits y claves que van desde los 32 bits hasta 448 bits. Es un codificador de 16 rondas Feistel y usa llaves que dependen de las Cajas-S.

Algoritmos de flujo:

Random:

XOR de la entrada de un archivo con número aleatorios generados por un cadena que es la clave.

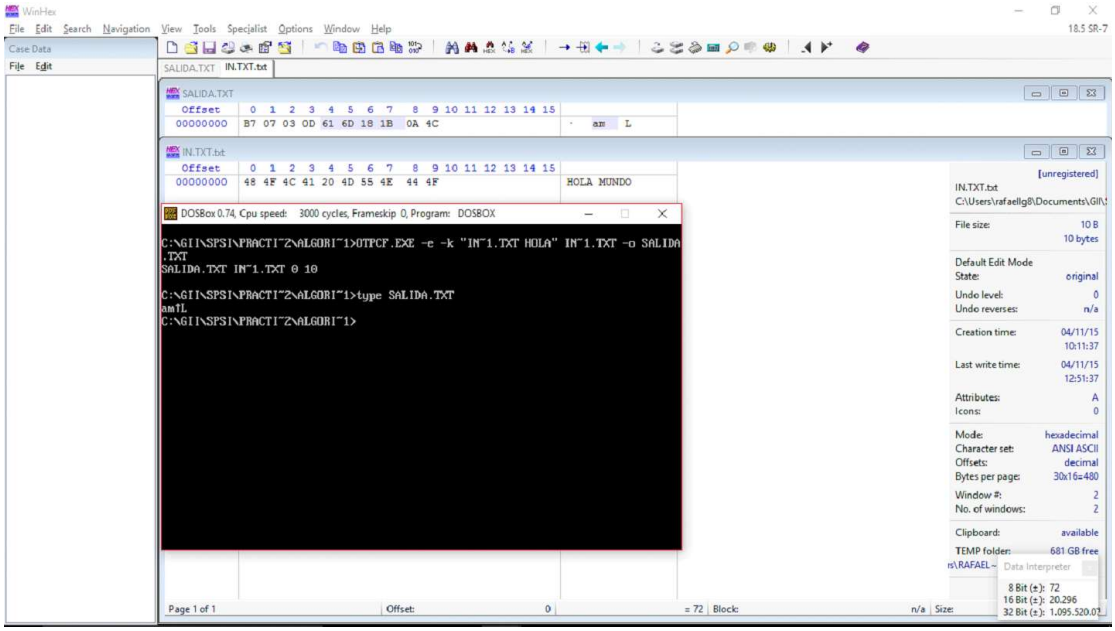
La clave es una cadena ASCII, de como máximo 256 letras.



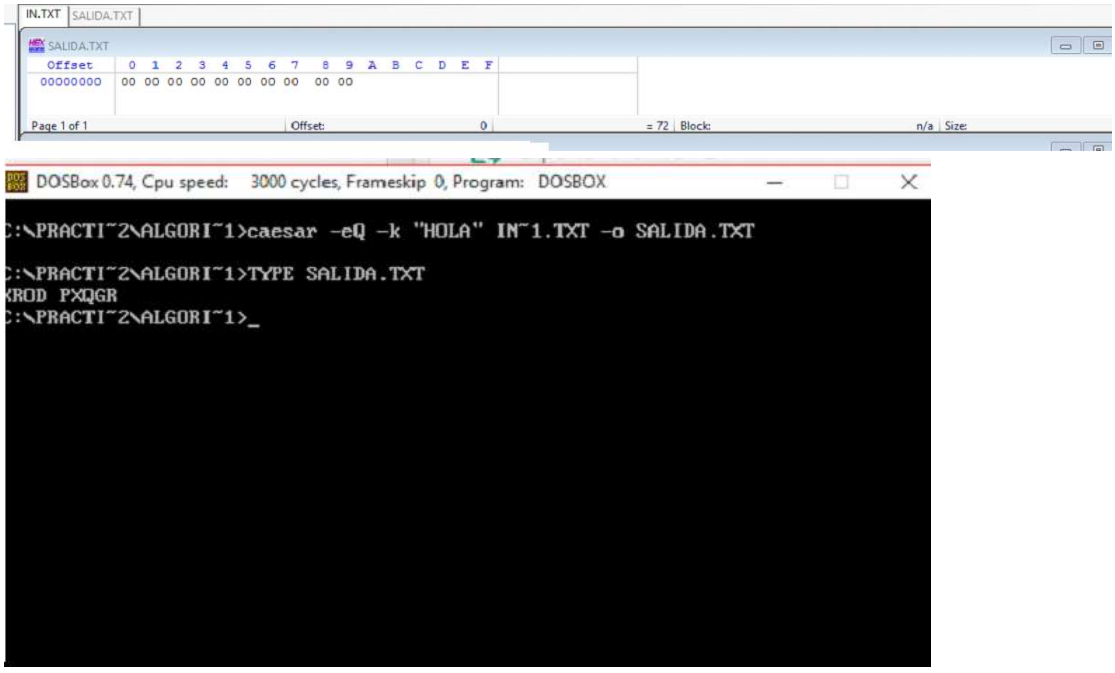
OTP

XOR con un archivo de un solo uso hace un desplazamiento hasta el inicio del archivo.

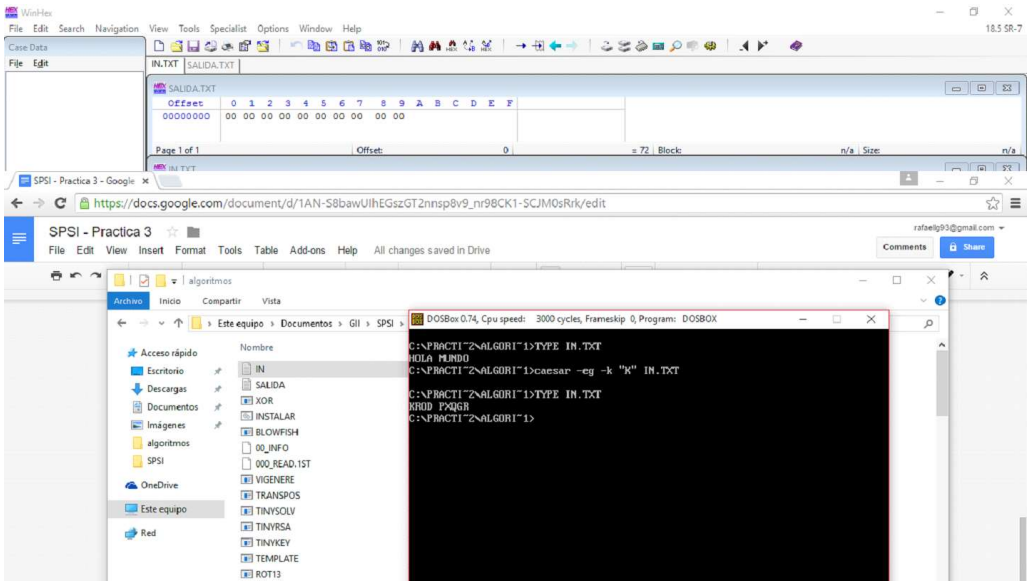
Llave es una cadena ASCII, dado a través de un archivo de texto \" NOMBREARCHIVO.EXT OFFSET \ "".



Ejercicio 9) ¿En qué consiste la opción de borrado seguro de los algoritmos? Hacer algún ejemplo con las diversas opciones de borrado seguro.



Borra el archivo de entrada, y genera el archivo de salida correcto. Opción: -eQ.



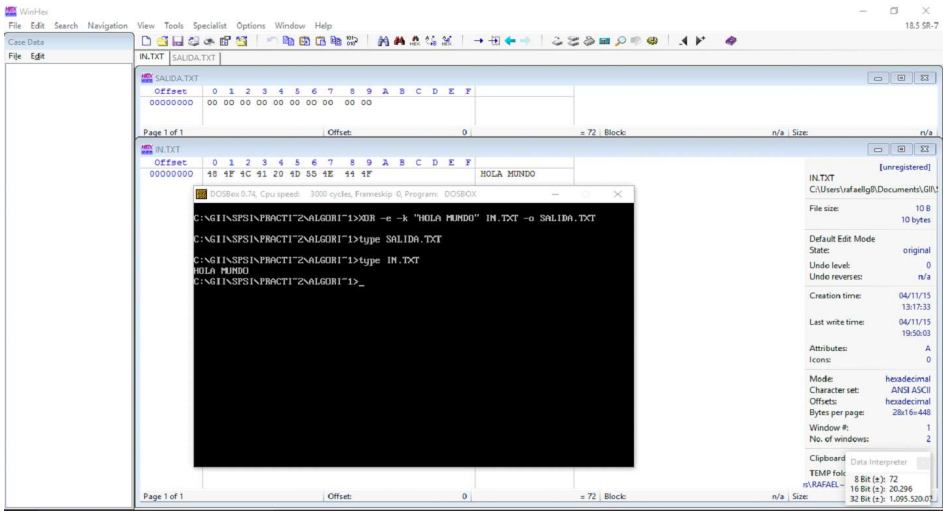
Sobrescribe el archivo de entrada con el resultado. Opción: -eg

Ejercicio 10) ¿Hay algoritmos con un tamaño de bloque distinto de 64 bits?. En caso afirmativo decir cuáles y dar el tamaño del bloque.

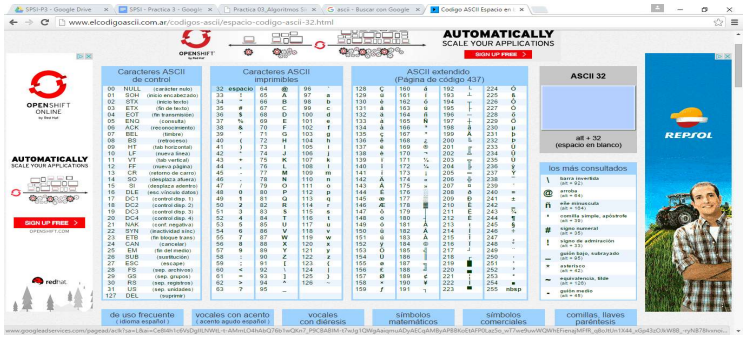
- Lucifer: 48, 32 o 128 bits
- RSA: bloques de datos de 8 o 16 bit.

Ejercicio 11)¿Qué ocurre si con el algoritmo XOR ciframos una cadena de caracteres y utilizamos como llave la misma cadena? ¿Cuál es la salida? ¿Qué ocurre ahora si con el mismo algoritmo XOR ciframos una cadena de caracteres con llave caracteres ESPACIO de la misma longitud que la entrada como texto llano? ¿Cuál es la explicación?

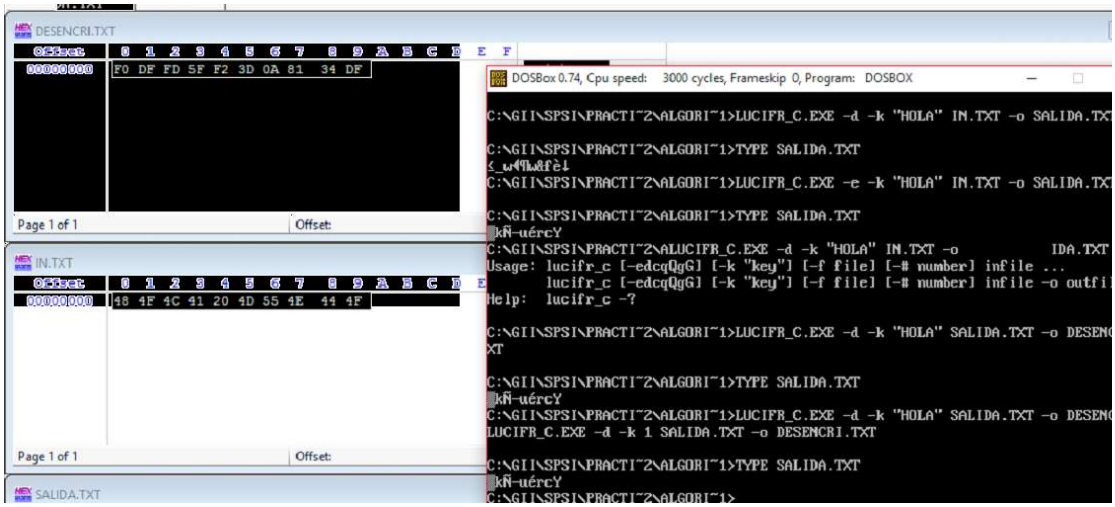
Al hacer XOR con la llave que es la misma que la del texto de entrada dá 0, ya que es como hacer 1 XOR 1. Por eso dá nul.



Al hacer XOR con la clave solo con espacios, pasa el mensaje a minúsculas, esto es porque “espacio “ XOR H → h, hace un desplazamiento hacia la izquierda de 32 posiciones que es el carácter “espacio” en la tabla ASCII. Así si miramos la tabla ASCII, podemos ver la H = 72, y el espacio = 32, 72+32 = 104 → h.



jercicio 12) Dar otros ejemplos de cifrado y descifrado, si encuentras que sean interesantes, de otros algoritmos en particular con otras llaves en particular.



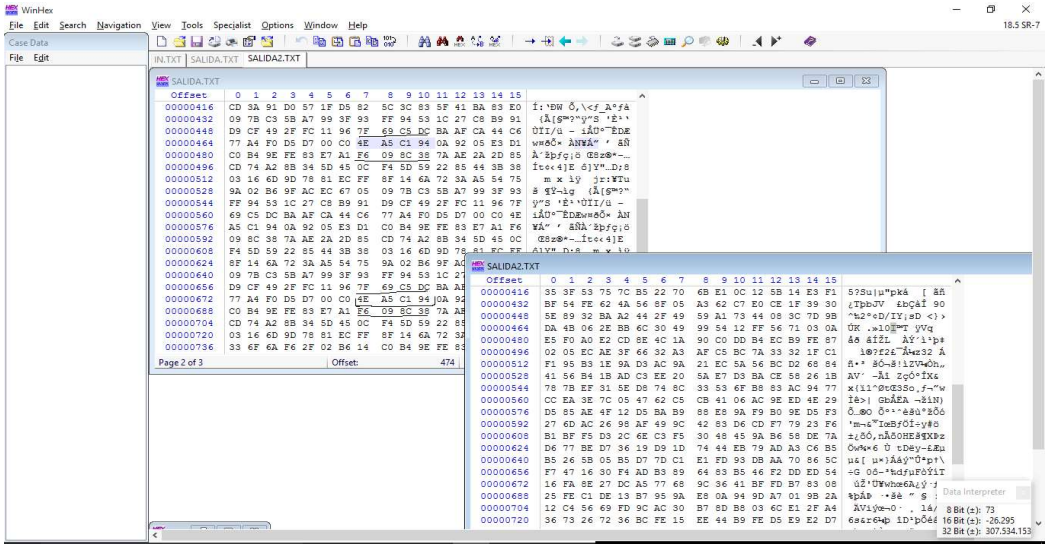
En Lucifer si desencriptamos con la misma clave con la que encriptamos, el

mensaje encriptado no se modifica.

Ejercicio 13) Los algoritmos ejecutables con nombre.exe están implementados en modo ECB. Los algoritmos ejecutables con nombre_C.exe están implementados en modo CBC. ¿Cómo podemos distinguirlos atendiendo a la salida que ofrecen? Poner un ejemplo en uno y otro caso con un algoritmo seleccionado de entre ellos. Indicación: buscar texto llano con repetición por bloques.

En un EBC, si se repite el bloque, se repite el texto cifrado, mientras que en el CBC no se repite el cifrado.

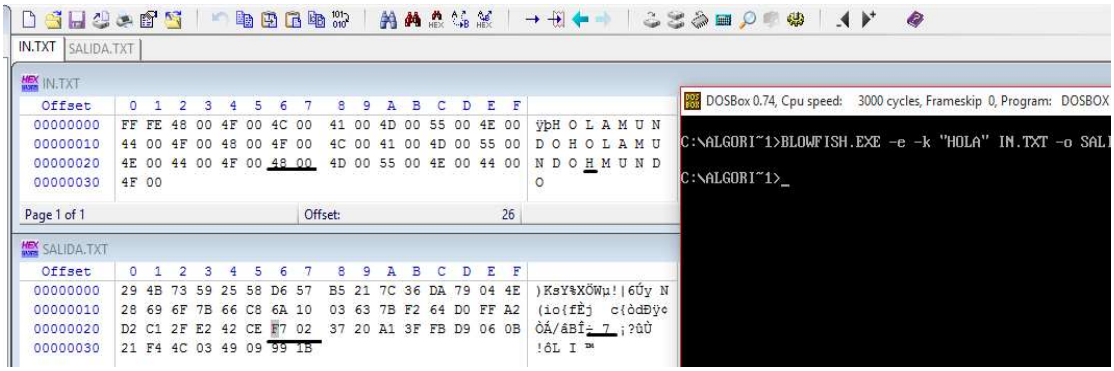
Podemos ver como se repite siempre en EBC, por ejemplo: “ÀN¥Á” en hexadecimal **4E A5 C1 94.** Todos los demás bloques se repiten, mientras CBC no se repiten.



Ejercicio 14.) Los algoritmos están implementados normalmente en modo de "anotaciones al final" o "padding". ¿Cómo podemos comprobarlo?. Poner un ejemplo de ello.

Los algorimtos que tienen padding, intentan "rellenar" todos los bloques enteros de N bits, es decir, si al introducir una llave repetitiva por ejemplo, usamos menos de N bits, intentará rellenar los bits restantes.

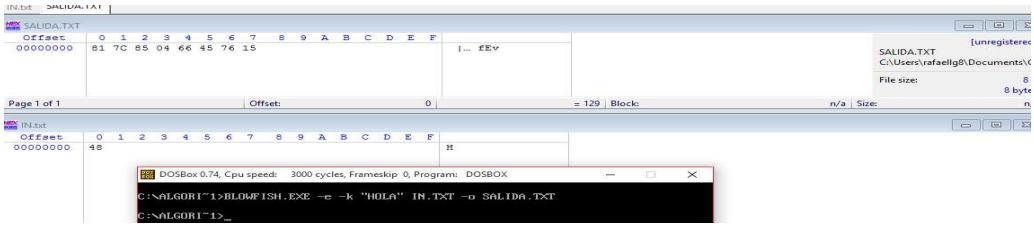
Para ello vamos a usar BLOWFISH_C, con la clave en texto llano "HOLA" que vamos a cifrar el texto en el archivo IN.TXT que contiene la palabra "HOLAMUNDOHOLAMUNDOHOLAMUNDO", vamos a eliminar las últimas "OLA":



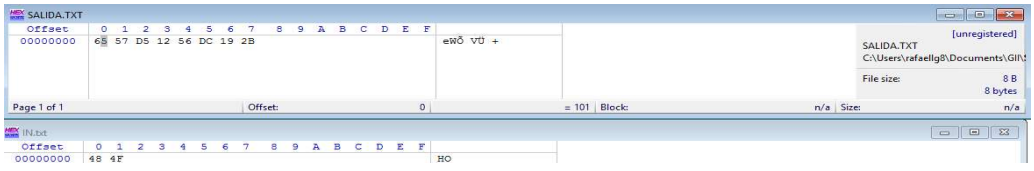
Vemos en la zona subrayada como se "rellena" los bloques vacíos, indicados por 00 en hexadecimal en el texto de entrada, se llenan en el texto e salida. En nuestro caso, como son bloques de 64 bits, ha rellenado hasta un múltiplo del mismo, 256 bits.

Ejercicio 15.) ¿Cómo afecta el padding a un algorimto en modo ECB con repetición de bloques de texto llano de 8,16, o 32 bits?. ¿Y cómo afecta en el caso anterior con bloques de 64 bits?

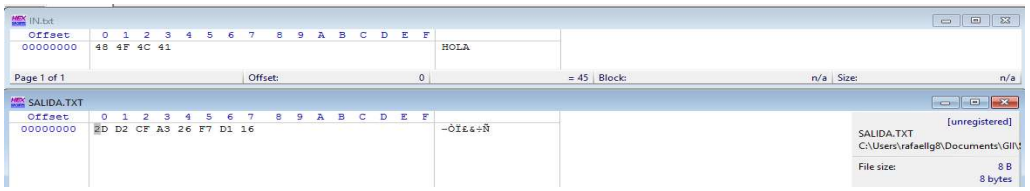
Si metemos menos de 64 bits de bloque, intentará el algoritmo rellenar los bloques hasta llegar a 64 bits.



Podemos apreciar como en el texto de salida, ocupa 8Bytes = 64 bits, por lo que ha rellenado el algoritmo el bloque restante.

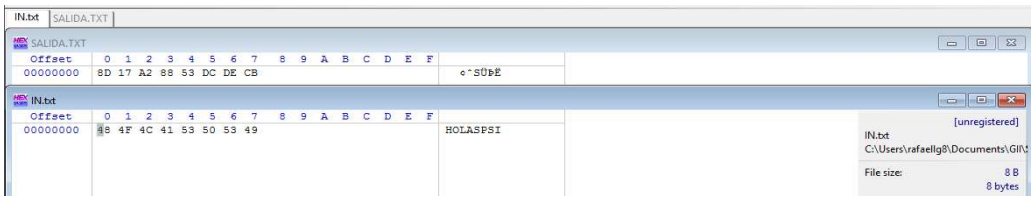


Lo mismo para 16 bits.



Para 32 bits, igual, rellena la mitad restante hasta llegar a 64 bits.

Sin embargo si metemos un bloque de 64 bits no hace padding:



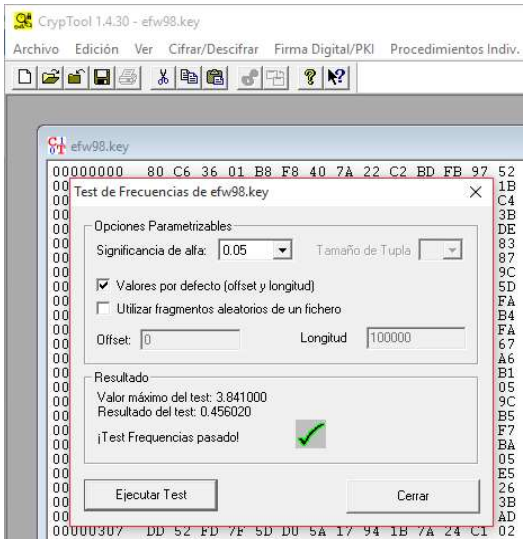
Vemos como coinciden el tamaño de bits tanto para la entrada, abajo, como para la salida, arriba.

Práctica para entrega 3:

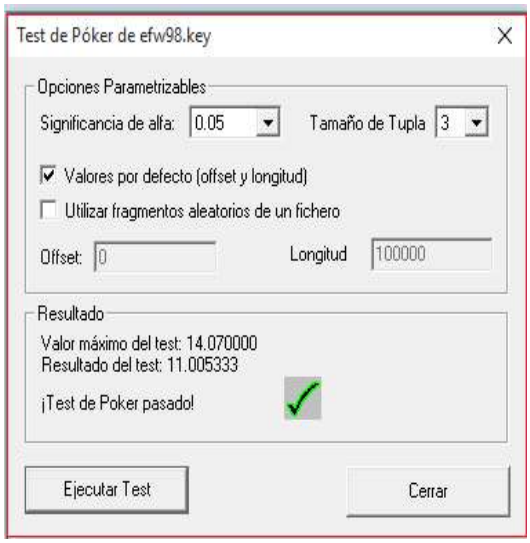
Ejercicio 16) Realizar el análisis de aletoriedad provisto en Cryptool para el Keyfile de enigmawn y fichero de llaves de TrueCrypt, obteniendo ficheros de longitud adecuada por concatenación, si es necesario. Realizar los tests de Frecuencias,Poker,Rachas y Series, así como el integrado FIPS-PUB-140-1. Hace una tabla con los resultados y obtener las conclusiones pertinentes sobre la aletoriedad en uno y otros algoritmos.

1) Clave enigmawm **efw98.key**

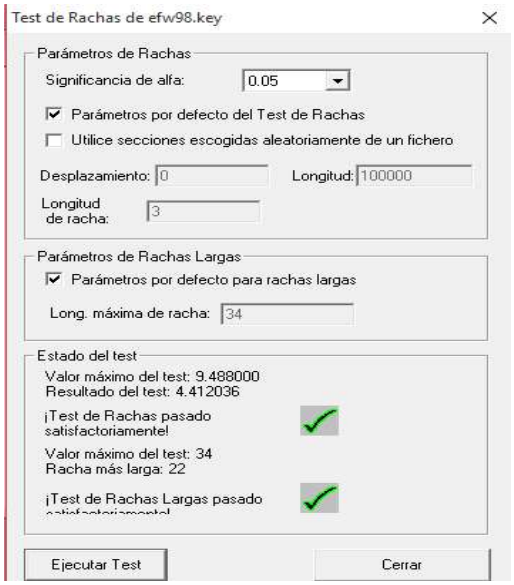
Para ellos abrimos el archivo a través de cryptool versión 1, y ejecutamos los tests:



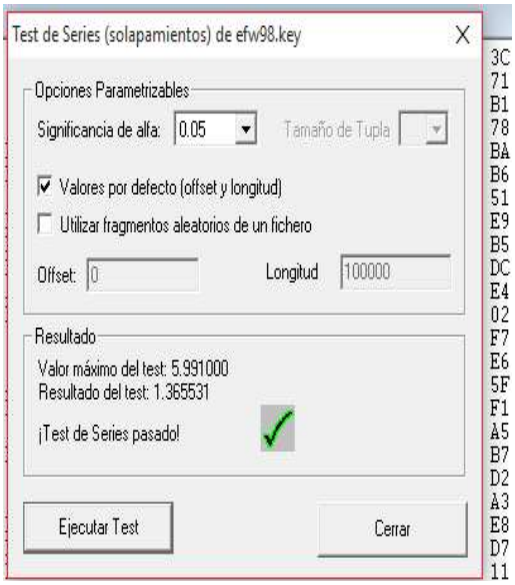
Test de Frecuencias



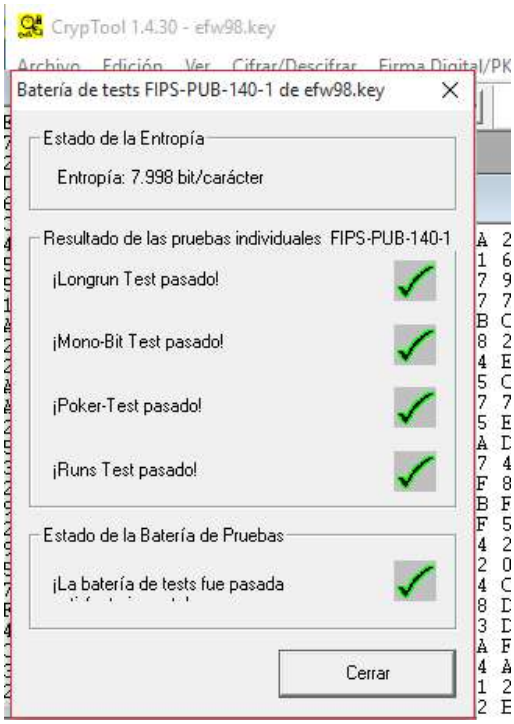
Test de Póker



Test de Rachas



Test de Series

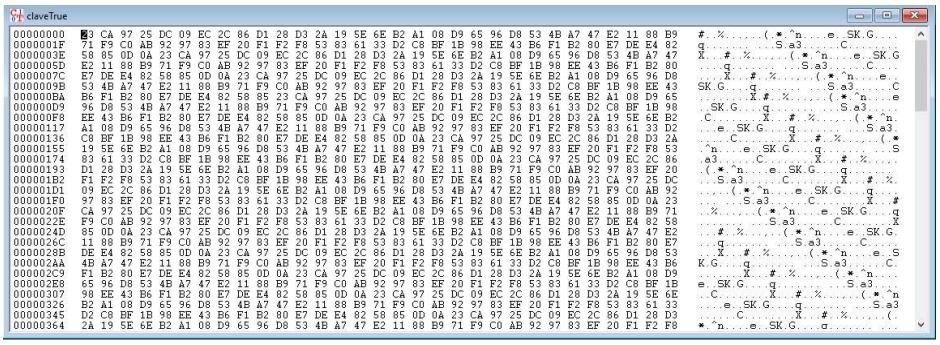


Test FIPS.

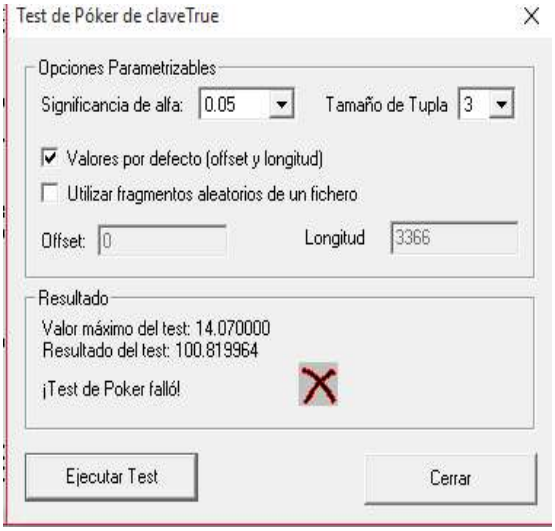
Para este test, nos dice que sólo se usuarán los 2500 primeros Bytes, ya que está preparada para eso y el documento tiene más de 100.000. Aún así ha pasado todos los test de forma existosa.

2) Clave Wirphool TrueCrypt

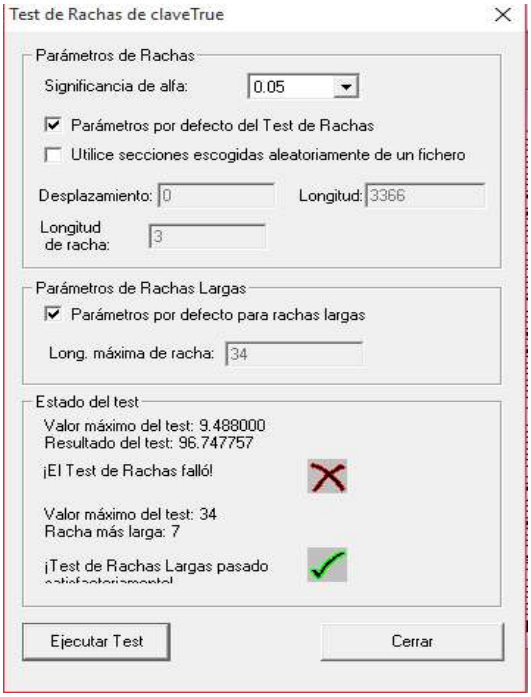
Para crear una clave, vamos a Tools --> Generate Key:

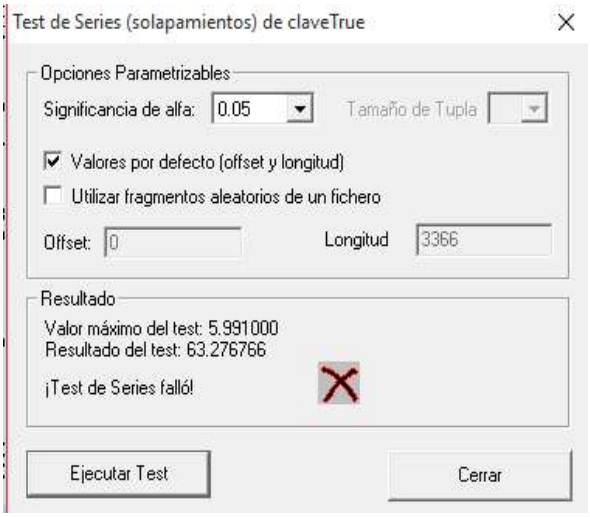


Como sólo nos ocupa 64 Bytes, lo abrimos con el bloc de notas y copiamos pegamos la clave hasta conseguir 3600 Bytes, algo más de lo que nos pide.

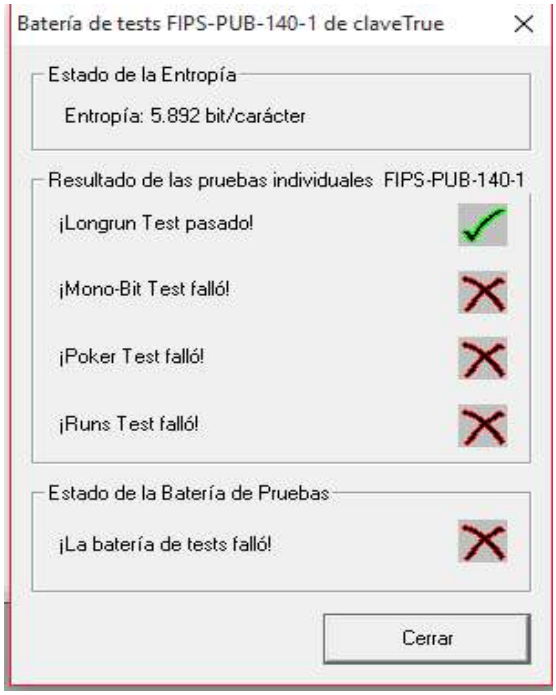


Test de Frecuencias Test de Póker





Test de Rachas Test de Series



Batería FIPS-PUB

Resultados:

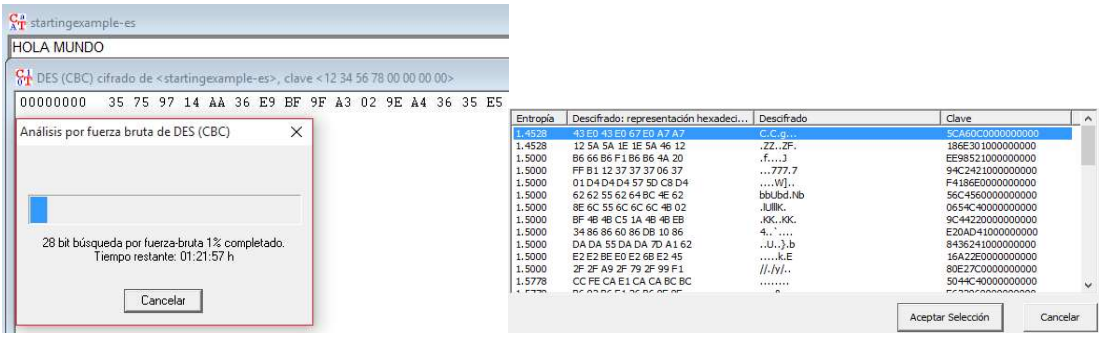
LLAVE	**Test Frecuencias**	**Test Póker**	**Test Rachas**	**Test de Series**	**FIPS**
Enigmaw	0.55	11.00	4.41 racha 22	1.36	7.99
TrueCrypt	36.68	100.81	96.74 racha 7	63.27	5.89

Podemos ver como en el archivo de TrueCrypt, se disparan los valores, debido a que es una clave concatenada, los únicos valores en los que se mantiene y saca un valor algo mejor es en la entropía de FIPS, aunque fallan los demás test para TrueCrypt.

Ejercicio 17). Examinar las opciones de cifrado simétricas en CrypTool con varios algoritmos. Probar un ataque sobre alguno de ellos en función de la longitud de la llave. Analizar los tiempos de ataque necesarios para determinar 32 bits,40 bits,48 bits,56 bits y 64 bits de llave.

Ciframos un texto llano, con **DES CBC**, con :

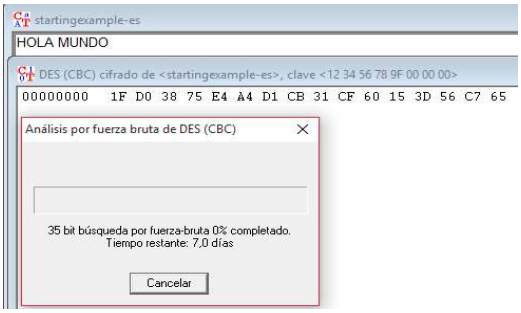
- Llave de 32 bits



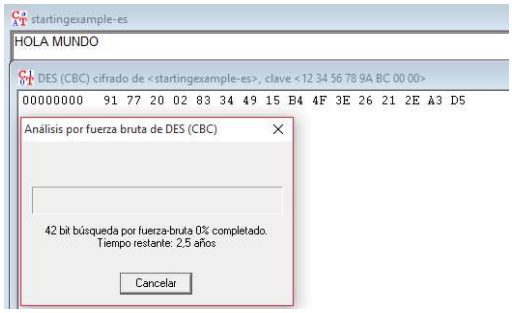
Tendríamos un tiempo de 1:21:00h, para una clave de 32 bits.

He usado un texto llano corto para que tarde menos.

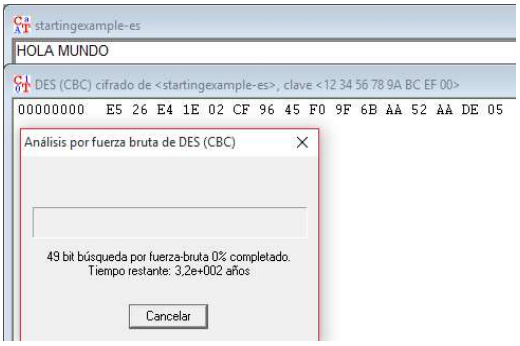
- Llave 40 bits: 7 días



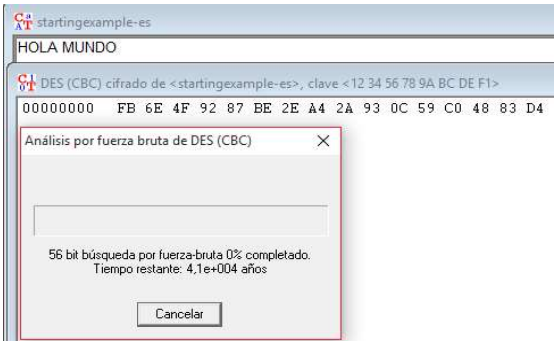
- Llave 48 bits: 2,5 años



- Llave 56 bits: 3.2e+2 años



- Llave 64 bits: 4.1e+0.04 años



Vemos que a partir de una clave mayor de 32 bits, es prácticamente inviable e imposible romper por fuerza bruta una clave de DES CBC.

Ejercicio 18). Con los códigos disponibles de algoritmos simétricos, que puedes encontrar en los enlaces de 6 construye tu propio ejemplo de ejecución de algoritmo simétrico con elección de Modo de Cifrado, Padding, Borrado Seguro, Generación de LLaves etc.

```
"""
Rafael Lachica Garrido
SPSI - Cryptografia con DES3
"""
from Crypto.Cipher import DES3
import binascii
import time
def cifrado():
nombre = raw_input("Nombre archivo a cifrar\n")
textoplano = open(nombre,'r').read()
start_time = time.time()
clave = 'mysecretpassword'
cifrado = DES3.new(clave, DES3.MODE_ECB)
textocifrado = cifrado.encrypt(textoplano)
end_time = time.time()
print ("Tiempo encriptado ms: "+ str((end_time-start_time)/1000))
textocifrado=textocifrado.encode('hex')
print "Texto cifrado: "
print (textocifrado)
archivoCifrado = open('cifrado','w')
archivoCifrado.write((textocifrado)) #Escribimos el resultado al archivo
def descifrado():
start_time = time.time()
key = 'mysecretpassword'
ciphertext = (open('cifrado','r').read())
ciphertext = binascii.unhexlify(ciphertext)
decobj = DES3.new(key, DES3.MODE_ECB)
plaintext = decobj.decrypt(ciphertext)
end_time = time.time()
print ("Tiempo desencriptado ms: "+ str((end_time-start_time)/1000))
# Imprimimos el resultado en el archivo salida
print "Texto plano: \n"
print plaintext
salida = open('salida','w')
salida.write(plaintext)
'''
Menu Principal
'''
menu = input('\nSi desea cifrar pulse 1, si desea descifrar pulse 2')
if (menu==1):
cifrado()
elif (menu==2):
descifrado()
else:
cifrado()
descifrado()
```

Para el cifrado del texto he usado DES3, de tipo ECB, sin padding. Para que funcione de forma correcto hay que insertar una clave de 16 Bytes, y un mensaje múltiplo de 8 Bytes.
He usado para python el módulo **pycrypto.**

Fuente: https://pypi.python.org/pypi/pycrypto

Texto a cifrar:
SPSISPSIRAFAELLACHICAGARRIDOSPSISPSIRAFAELLACHICAGARRIDOSPSI
SPSIRA
FAELLACHICAGARRIDOSPSISPSIRAFAELLACHICAGARRIDOSPSISPSIRAFAEL
LACHI
CAGARRIDOSPSISPSIRAFAELLACHICAGARRIDOSPSISPSIRAFAELLACHICAGA
RRIDO
SPSISPSIRAFAELLACHICAGARRIDO

Clave: “mysecretpassword”

Tiempo usado cifrar/descifrar:
Tiempo
encriptado ms: 2.36034393311e07
Tiempo
desencriptado ms: 2.42948532104e07