Criptografía
Práctica SL4
Informe de prácticas

1. **Obtenga y/o genere fichero de texto, una imagen BMP, un fichero formado por datos aleatorios y un fichero formado por un único byte repetido y encripte y desencripte estos ficheros usando diferentes algoritmos (AES, DES, CAMELLIA, BLOWFISH) y diferentes modos de cifrado (ECB, CBC, CFB). Recopile cuantos resultados pueda de este proceso (memoria usada, tiempo necesario para encriptar y desencriptar,…). Muestre en una o varias tablas los resultados obtenidos (tiempos, tamaños,…) y coméntelos.**

   Los ficheros utilizados durante los test son los siguientes (Se ha tratado de conseguir que todos tuvieran un tamaño similar: 350-400KB):

   - **Imagen BMP**: Se ha utilizado la imagen que se ofrece junto con el guion de la práctica (test.bmp).
   - **Fichero de Texto**: Obtenido de la red, consiste en parte de un libro en inglés.
   - **Fichero de Texto Aleatorio**: Se ha generado mediante un script script en Python y la redirección por línea de comandos. El Script se muestra a continuación.
   - **Fichero de Texto con Byte Repetido**: Se ha generado repitiendo el carácter “a” hasta alcanzar el tamaño requerido.

   El script para generar el fichero aleatorio es el siguiente (Se puede acceder al fichero desde https://github.com/garciparedes/gsi-practices/blob/master/practice-4/random-text.py):

```python
import random, string

def randomword(length):
   return ''.join(random.choice(string.lowercase) for i in range(length))

print randomword(400000)
```

   En cuanto a los algoritmos de cifrado utilizados en la comparativa, se han utilizado los siguientes (Todos ellos son de cifrado simétrico, es decir, utilizan la misma clave tanto para el cifrado como para el descifrado):

   - **AES**: Se basa en cifrado por bloques, tiene un tamaño de bloque fijo de 128 bits y tamaños de llave de 128, 192 o 256 bits. Los cálculos se hacen en un espacio finito o

de Galois. Se basa en 4 operaciones básicas denominadas *subBytes*, *shiftRows*, *mixColumns* y *addRoundKey*, que se repiten un número determinado de veces tanto en el cifrado como en el descifrado. Actualmente es el estándar de seguridad utilizado por el Gobierno de Estados Unidos.

- **DES**: El tamaño del bloque es de 64 bits. La clave tiene una longitud de64 bits, aunque en realidad, sólo 56 de ellos son empleados por el algoritmo. Los ocho bits restantes se utilizan únicamente para comprobar la paridad, y después son descartados. Para el proceso de cifrado utiliza una función matemática denominada F de Feistel. Fue el estándar hasta que en 2001 fue remplazado por AES.

- **CAMELLIA**: Al igual que DES, también utiliza la función de Feistel con 18 o 24 rondas dependiendo del tamaño de la clave. Cada seis rondas, se aplica una capa de transformación lógica: la llamada "función FL" o su inversa. Camellia utiliza cuatro matrices de 8 × 8 bits con transformaciones afines de entrada y salida y operaciones lógicas. El cifrado también utiliza el blanqueamiento de claves de entrada y salida.

- **BLOWFISH**: Tiene un tamaño de bloque de 64 bits y una longitud de clave variable de 32 bits a 448 bits. Utiliza la función de Feistel en 16 rondas y grandes matrices dependientes de la clave. En estructura se asemeja a CAST.

Otro de los puntos a tener en cuenta durante la comparativa es el modo de cifrado de bloque:

- **ECB**: Es el modo de cifrado más simple, en el cual el mensaje es dividido en bloques, cada uno de los cuales se cifra de manera separada. La desventaja de este método es que bloques idénticos de mensaje sin cifrar producirán idénticos textos cifrados. Por esto, no proporciona una auténtica confidencialidad y no es recomendado para protocolos criptográficos, como apunte no usa el vector de inicialización, sino que únicamente se basa en la clave del algoritmo de encriptación.

- **CBC**: En este modo, antes de ser cifrado, a cada bloque de texto se le aplica una operación XOR con el previo bloque ya cifrado. De esta manera, cada bloque cifrado depende de todos los bloques de texto claros usados hasta ese punto. Además, para hacer cada mensaje único se debe usar un vector de inicialización en el primer bloque.

- **CFB**: Este modo es similar al anterior, solo que en lugar de cifrar el texto plano y hacer la operación XOR con el mensaje anterior (o vector de inicialización en el primer caso), lo que hace es cifrar el mensaje anterior (o IV) y después hacer el XOR con el texto plano.

Para tratar de automatizar el proceso de obtención de resultados se ha desarrollado un script para Shell que cifra y descifra los ficheros escribiendo los resultados de tiempo, memoria, etc en otro fichero denominado readme.

El fichero se puede descargar desde https://github.com/garciparedes/gsi-practices/blob/master/practice-4/test-files/cipher.sh y el código fuente del mismo es el siguiente:

```bash
#!/bin/bash

#
# Descrition: Obtains performance results of ciphers.
#
# Author:   Sergio García Prado
#           garciparedes.me
#

password="password"

CypherAlg=( "aes-256" "des" "camellia-256" "bf")
CypherMode=( "ecb" "cbc" "cfb")


mkdir -p encrypted/$1


exec 3>&1 4>&2 >encrypted/$1/readme 2>&1

for i in "${CypherAlg[@]}"
do
    for j in "${CypherMode[@]}"
    do
        echo
        echo $1 $i-$j
        echo "Cipher:"
        /usr/bin/time -v openssl $i-$j -in $1 -out encrypted/$1/$1.$i-$j -k ${password}
        echo "Decipher:"
        /usr/bin/time -v openssl $i-$j -d -in encrypted/$1/$1.$i-$j -out $1 -k  ${password}
    done
done

# restore stdout and stderr
exec 1>&3 2>&4
```

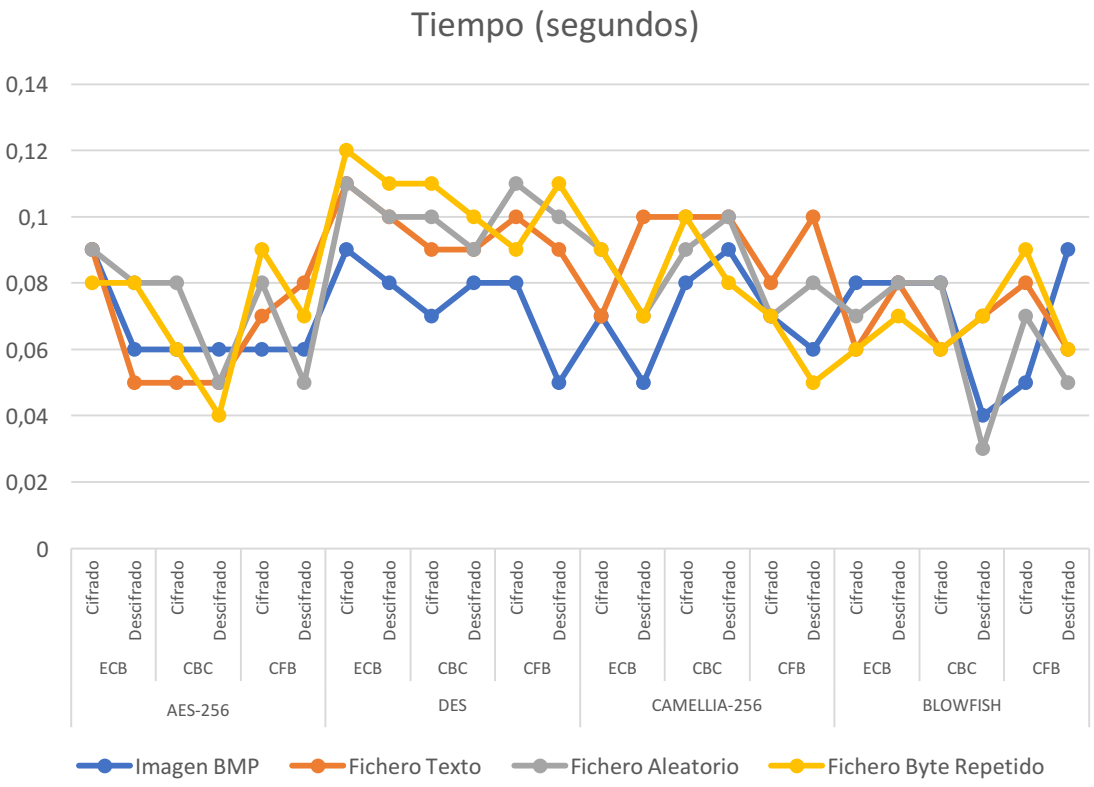
Dicho script se ha ejecutado en una máquina Raspberry Pi B+, la cual tiene una limitada capacidad de cómputo, con la intención de que las variaciones en cuanto a los resultados que se obtengan sean lo más amplias posibles.

Los resultados obtenidos en cuanto a la duración como proceso de usuario en el sistema han sido los siguientes:

Tiempo (segundos)			**Imagen BMP**	**Fichero Texto**	**Fichero Aleatorio**	**Fichero Byte Repetido**
AES-256	ECB	Cifrado	0.09	0.09	0.09	0.08
		Descifrado	0.06	0.05	0.08	0.08
	CBC	Cifrado	0.06	0.05	0.08	0.06
		Descifrado	0.06	0.05	0.05	0.04
	CFB	Cifrado	0.06	0.07	0.08	0.09
		Descifrado	0.06	0.08	0.05	0.07
DES	ECB	Cifrado	0.09	0.11	0.11	0.12
		Descifrado	0.08	0.10	0.10	0.11
	CBC	Cifrado	0.07	0.09	0.10	0.11
		Descifrado	0.08	0.09	0.09	0.10
	CFB	Cifrado	0.08	0.10	0.11	0.09

		Descifrado	0.05	0.09	0.10	0.11
CAMELLIA-256	ECB	Cifrado	0.07	0.07	0.09	0.09
		Descifrado	0.05	0.10	0.07	0.07
	CBC	Cifrado	0.08	0.10	0.09	0.10
		Descifrado	0.09	0.10	0.10	0.08
	CFB	Cifrado	0.07	0.08	0.07	0.07
		Descifrado	0.06	0.10	0.08	0.05
BLOWFISH	ECB	Cifrado	0.08	0.06	0.07	0.06
		Descifrado	0.08	0.08	0.08	0.07
	CBC	Cifrado	0.08	0.06	0.08	0.06
		Descifrado	0.04	0.07	0.03	0.07
	CFB	Cifrado	0.05	0.08	0.07	0.09
		Descifrado	0.09	0.06	0.05	0.06

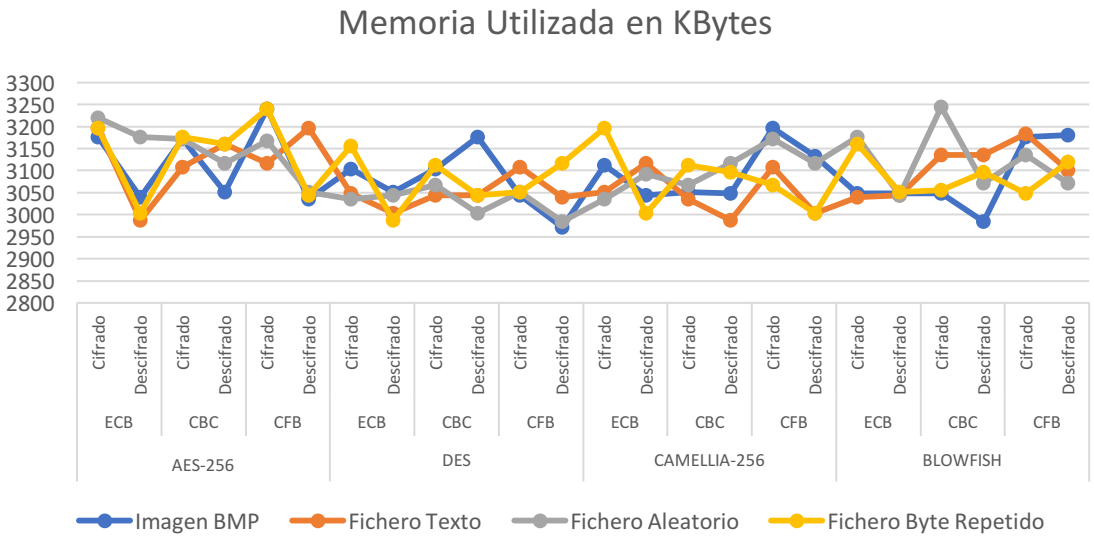
Los resultados en cuanto al tiempo dedicado para el proceso de cifrado y descifrado (con ficheros de tamaños similares) revelan que el algoritmo AES con claves de 256 bytes es el más eficiente en promedio mientras que DES es quién requiere de más tiempo. Debido al tamaño reducido de los ficheros de pruebas, estos resultados podrían no ser del todo fiables, por lo cual, han de ser tomados con cautela.



En el caso de la memoria utilizada por cada uno de los algoritmos para el proceso de cifrado así como el de descifrado se han obtenido los siguientes resultados:

Memoria (kbytes)			Imagen BMP	Fichero Texto	Fichero Aleatorio	Fichero Byte Repetido
AES-256	ECB	Cifrado	3176	3196	3220	3196
		Descifrado	3040	2988	3176	3004
	CBC	Cifrado	3172	3108	3172	3176
		Descifrado	3052	3160	3116	3160
	CFB	Cifrado	3240	3116	3168	3240
		Descifrado	3036	3196	3052	3044
DES	ECB	Cifrado	3104	3048	3036	3156
		Descifrado	3052	3004	3044	2988
	CBC	Cifrado	3104	3044	3068	3112
		Descifrado	3176	3044	3004	3044
	CFB	Cifrado	3044	3108	3052	3052
		Descifrado	2972	3040	2984	3116
CAMELLIA-256	ECB	Cifrado	3112	3052	3036	3196
		Descifrado	3044	3116	3092	3004
	CBC	Cifrado	3052	3036	3068	3112
		Descifrado	3048	2988	3116	3096
	CFB	Cifrado	3196	3108	3172	3068
		Descifrado	3132	3004	3116	3004
BLOWFISH	ECB	Cifrado	3048	3040	3176	3160
		Descifrado	3048	3044	3044	3052
	CBC	Cifrado	3048	3136	3244	3056
		Descifrado	2984	3136	3072	3096
	CFB	Cifrado	3176	3184	3136	3048
		Descifrado	3180	3100	3072	3120

Los resultados en cuanto al uso de memoria por los distintos algoritmos no aportan la suficiente información como para revelar resultados sobre los mismos por su dispersión similar y la dificultad de encontrar patrones sobre los mismos.



2. **Escriba un programa, en el lenguaje de programación que desee, que calcule la media aritmética de los bytes de un fichero.**

   Para el cálculo de la media aritmética de los bytes del fichero se ha creado un script en el lenguaje Python, que recibe el nombre del fichero como argumento al programa e imprime por salida estándar el resultado.

   El fichero se puede descargar desde https://github.com/garciparedes/gsi-practices/blob/master/practice-4/byte-mediam.py El código fuente del mismo es el siguiente:

```python
#!/usr/bin/env python

#
# Description: Calculates Aritmetical Median of argv[1] file.
#
# Author:   Sergio Garcia Prado
#           garciparedes.me
#

import sys

with open(sys.argv[1], "rb") as f:
    byteArr = map(ord, f.read())
    f.close()
    size = len(byteArr)

    # Median
    sum = 0
    for byte in byteArr:
        sum += byte
    median = float(sum)/size
    print median
```

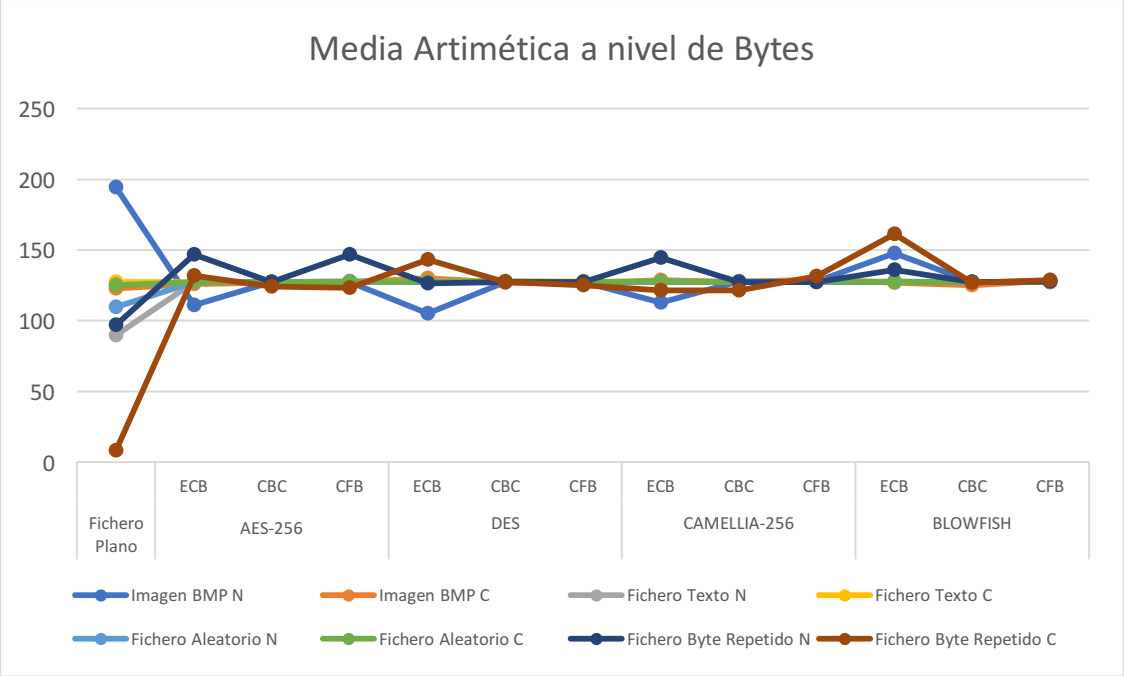
3. **Comprima con *gzip* los cuatro ficheros usados en la cuestión 1. Usando el programa anterior, obtenga la media aritmética de los bytes que componen los cuatro ficheros usados en la cuestión 1, sus correspondientes archivos comprimidos y sus correspondientes archivos encriptados con AES-ECB, AES-CBC, DES-ECB y DES-CBC. Muestre los resultados en una tabla y coméntelos. ¿De qué puede ser un indicador la media aritmética de los bytes de un fichero?**

   La media aritmética a nivel de bytes representa el centro de gravedad de la distribución de bytes, revelando menor cantidad de información cuando esta se encuentre más centrada en el punto medio de la distribución, es decir, $\frac{0+256}{2} = 128$

   Los resultados obtenidos al aplicar el script del ejercicio anterior sobre los ficheros son los siguientes:

Media Aritmética		Imagen BMP		Fichero Texto		Fichero Aleatorio		Fichero Byte Repetido	
		N	C	N	C	N	C	N	C
Fichero Plano		194.263	123.060	89.769	127.444	109.497	124.932	96.999	8.187
AES-256	**ECB**	110.853	126.074	127.587	127.517	127.442	127.477	146.810	131.743
	CBC	127.446	126.726	127.600	127.510	127.327	127.180	127.544	124.071
	CFB	127.601	127.191	127.317	127.887	127.605	127.403	146.810	123.243
DES	**ECB**	105.293	130.086	127.369	127.619	127.496	127.626	126.622	143.194
	CBC	127.561	126.896	127.553	127.624	127.414	127.630	127.331	127.488
	CFB	127.343	126.673	127.510	127.678	127.490	127.430	127.544	124.941
CAMELLIA-256	**ECB**	113.087	128.601	127.675	127.532	127.406	127.611	144.748	121.664
	CBC	127.635	127.427	127.437	127.497	127.514	127.400	127.585	121.481
	CFB	127.615	129.325	127.411	127.518	127.431	127.611	127.462	131.603
BLOWFISH	**ECB**	147.809	126.845	127.618	127.579	127.505	127.533	135.873	161.321
	CBC	127.464	125.294	127.450	127.356	127.496	127.329	127.477	126.747
	CFB	127.522	128.223	127.561	127.386	127.565	127.475	127.709	128.881

La distribución de valores sobre la media aritmética de los ficheros revela variaciones entre los ficheros en texto plano, es decir, sin cifrar, las cuales son reducidas cuando estos se comprimen. En cuanto al modo de cifrado, ECB obtiene resultados peores al resto de alternativas, las cuales se agravan en el caso del cifrador BlowFish.



4. **Escriba un programa, en el lenguaje de programación que desee, que calcule la entropía de la información de un fichero.**

Para el cálculo de la entropía de los bytes del fichero se ha creado un script en el lenguaje Python, que recibe el nombre del fichero como argumento al programa e imprime por salida estándar el resultado. Para el cálculo se ha utilizado la biblioteca "*pandas*", que facilita la obtención de la tabla de frecuencias.

El fichero se puede descargar desde https://github.com/garciparedes/gsi-practices/blob/master/practice-4/byte-entropy.py y el código fuente del mismo es el siguiente:

```python
#!/usr/bin/env python

#
# Description: Calculates Shannon Entropy of argv[1] file.
#
# Author:   Sergio Garcia Prado
#           garciparedes.me
#

import sys
import math
import pandas

with open(sys.argv[1], "rb") as f:
    byteArr = map(ord, f.read())
    f.close()
    fileSize = len(byteArr)

    #Obtain frequency table
    freqList = pandas.Series(byteArr).value_counts(normalize = True)

    # Shannon entropy
    ent = 0.0
    for freq in freqList:
        ent = ent + freq * math.log(freq, 2)
    ent = -ent
    print ent
```

5. **Usando el programa anterior, obtenga la entropía de la información de los cuatro ficheros usados en la cuestión 1, sus correspondientes archivos comprimidos con *gzip* y sus correspondientes archivos encriptados con AES-ECB, AES-CBC, DES-ECB y DES-CBC. Muestre los resultados en una tabla y coméntelos. ¿Por qué $\mathcal{H}(X) \in [0,8]$? ¿Existe alguna relación entre la entropía, $\mathcal{H}(X)$, de los ficheros sin comprimir y el tamaño de los correspondientes ficheros comprimidos?**

   La causa de que el resultado de la entropía de los ficheros se distribuya entre 0 y 8 es que los cálculos se han realizado utilizando un byte como unidad atómica. Dado que cada byte permite 256 posibilidades (2^8), al realizar la operación logaritmo en base 2, los resultados siempre estarán en el rango [0,8].

   La entropía de un fichero es una unidad de medida que determina la cantidad de información

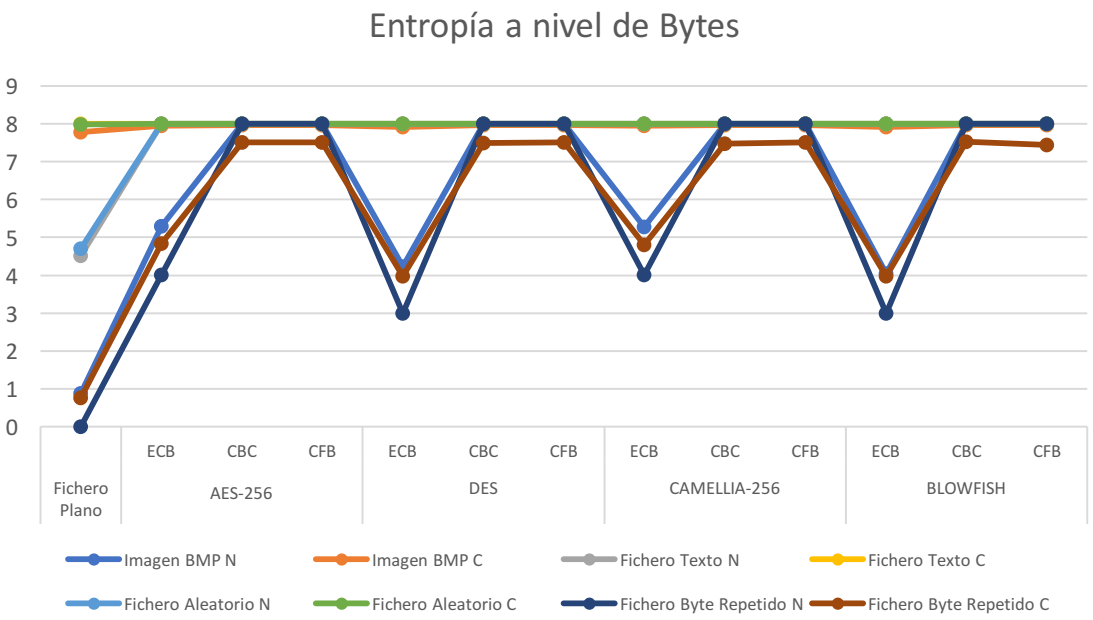
que se puede obtener a partir de la distribución de bytes del mismo.

Los resultados al calcular la entropía a nivel de bytes sobre cada fichero han sido los siguiente:

Entropía		Imagen BMP		Fichero Texto		Fichero Aleatorio		Fichero Byte Repetido	
		N	C	N	C	N	C	N	C
Fichero Plano		0.88504	7.78217	4.52366	7.99818	4.70043	7.99034	0.00051	0.76117
AES-256	**ECB**	5.28978	7.94157	7.99952	7.99853	7.99959	7.99918	4.00186	4.84131
	CBC	7.99951	7.96728	7.99956	7.99884	7.99948	7.99916	7.99954	7.50405
	CFB	7.99947	7.96474	7.99959	7.99890	7.99952	7.99928	7.99958	7.51336
DES	**ECB**	4.23143	7.92140	7.99924	7.99877	7.99945	7.99912	3.00133	3.97329
	CBC	7.99948	7.97074	7.99956	7.99895	7.99950	7.99925	7.99954	7.48632
	CFB	7.99952	7.96930	7.99953	7.99901	7.99958	7.99930	7.99952	7.51143
CAMELLIA-256	**ECB**	5.28436	7.94898	7.99956	7.99880	7.99959	7.99925	4.00186	4.80631
	CBC	7.99947	7.96847	7.99950	7.99888	7.99956	7.99920	7.99951	7.48066
	CFB	7.99934	7.97072	7.99955	7.99889	7.99961	7.99932	7.99957	7.50765
BLOWFISH	**ECB**	4.04411	7.92089	7.99924	7.99876	7.99953	7.99924	3.00139	3.97852
	CBC	7.99949	7.96852	7.99956	7.99882	7.99950	7.99931	7.99954	7.52199
	CFB	7.99956	7.96643	7.99953	7.99885	7.99951	7.99929	7.99954	7.44861

A partir de la siguiente tabla se puede apreciar que todos los algoritmos siguen una distribución similar sobre el conjunto de ficheros, revelando una relativa cantidad de información adicional cuando estos se utilizan con el modo de cifrado de bloques ECB.

En cuanto a los distintos ficheros, los que han sido comprimidos obtienen mejores resultados en los tests. El que más complicaciones presenta para los cifradores, como era de esperar es el que contiene únicamente un byte repetido. Cabe destacar el fichero de imagen comprimido, que tiene un alto grado de entropía.



6. **Encripte la imagen *test.bmp* usando el algoritmo AES con los modos de cifrado ECB y CBC. Salve una copia de la imagen cifrada, copie la cabecera de 54 bytes de la imagen original en el fichero cifrado y muestre (corte y pegue en la respuesta) las tres imágenes. Comente y explique, a la luz de los resultados, por qué sucede esto.**

   La imagen ofrecida para realizar este ejercicio es 'test.bmp' (se renombró a 'image.bmp' para facilitar su reconocimiento en el gestor de archivos). Esta imagen se adjunta a continuación:

   Para realizar el procedimiento descrito en el enuncia se ha creado un script de Shell para simplificar el proceso.
   El fichero se puede descargar desde https://github.com/garciparedes/gsi-practices/blob/master/practice-4/test-files/image-cipher.sh y el código fuente del mismo es el siguiente:

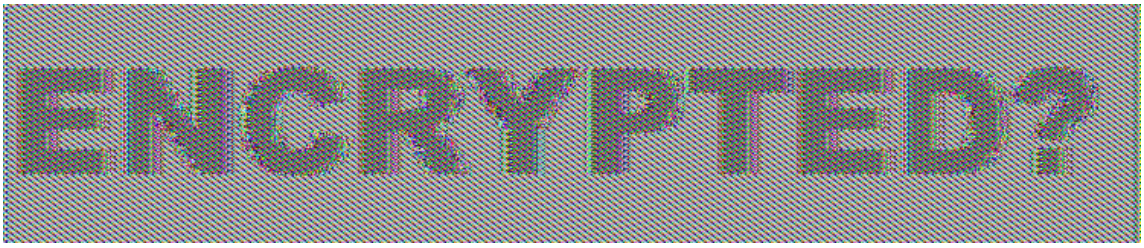
```
head -c 54 image.bmp > header.txt
tail -c +55 image.bmp > body.bin

openssl enc -aes-128-ecb -nosalt -k password -in body.bin -out body.ecb.bin
openssl enc -aes-128-cbc -nosalt -k password -in body.bin -out body.cbc.bin

cat header.txt body.ecb.bin > image.ecb.bmp
cat header.txt body.cbc.bin > image.cbc.bmp

rm body.bin body.ecb.bin body.cbc.bin header.txt
```

Tras examinar los resultados obtenidos se puede apreciar que en el caso del modo de cifrado ECB, que no tiene en cuenta el conjunto de bloques, sino que cifra cada uno independientemente, podemos apreciar lo siguiente. El mensaje se puede visualizar a pesar de que los colores se hayan cambiado. La causa de esto es que los bloques no se han cifrado encadenadamente.



En el caso del cifrado con el modo de CBC, que encadena los bloques esto no sucede, es decir, el mensaje queda completamente ilegible. La conclusión que se saca de este suceso es que el modo de cifrado ECB es vulnerable cuando los bloques se repiten un gran número de veces como sucede en el caso de las imágenes.