

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:

```
1 import random, string
2
3 def randomword(length):
4     return ''.join(random.choice(string.lowercase) for i in range(length))
5
6 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 de 64 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 código fuente de este script es el siguiente:

```

1  #!/bin/bash
2
3  #
4  # Description: Obtains performance results of ciphers.
5  #
6  # Author: Sergio García Prado
7  #         garciparedes.me
8  #
9
10 password="password"
11
12 CypherAlg=( "aes-256" "des" "camellia-256" "bf")
13 CypherMode=( "ecb" "cbc" "cfb")
14
15 mkdir -p encrypted/$1
16
17
18
19 exec 3>&1 4>&2 >encrypted/$1/readme 2>&1
20
21 for i in "${CypherAlg[@]}"
22 do
23     for j in "${CypherMode[@]}"
24     do
25         echo
26         echo $1 $i-$j
27         echo "Cipher:"
28         /usr/bin/time -v openssl $i-$j -in $1 -out encrypted/$1/$1.$i-$j -k ${password}
29         echo "Decipher:"
30         /usr/bin/time -v openssl $i-$j -d -in encrypted/$1/$1.$i-$j -out $1 -k ${password}
31     done
32 done
33
34 # restore stdout and stderr
35 exec 1>&3 2>&4

```

Los resultados obtenidos han sido los siguientes:

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

BLOWFISH	ECB	Cifrado	0.08		0.07	0.06
		Descifrado	0.08		0.08	0.07
	CBC	Cifrado	0.08		0.08	0.06
		Descifrado	0.04		0.03	0.07
	CFB	Cifrado	0.05		0.07	0.09
		Descifrado	0.09		0.05	0.06

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

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 código fuente del script es el siguiente:

```

1  #!/usr/bin/env python
2
3  #
4  # Description: Calculates Aritmetical Median of argv[1] file.
5  #
6  # Author: Sergio Garcia Prado
7  #         garciparedes.me
8  #
9
10 import sys
11
12 with open(sys.argv[1], "rb") as f:
13     byteArray = map(ord, f.read())
14     f.close()
15     size = len(byteArray)
16
17     # Median
18     sum = 0
19     for byte in byteArray:
20         sum += byte
21     median = float(sum)/size
22     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?

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			127.442	127.477	146.810	131.743
	CBC	127.446	126.726			127.327	127.180	127.544	124.071
	CFB	127.601	127.191			127.605	127.403	146.810	123.243
DES	ECB	105.293	130.086			127.496	127.626	126.622	143.194
	CBC	127.561	126.896			127.414	127.630	127.331	127.488
	CFB	127.343	126.673			127.490	127.430	127.544	124.941
CAMELLIA-256	ECB	113.087	128.601			127.406	127.611	144.748	121.664
	CBC	127.635	127.427			127.514	127.400	127.585	121.481
	CFB	127.615	129.325			127.431	127.611	127.462	131.603
BLOWFISH	ECB	147.809	126.845			127.505	127.533	135.873	161.321
	CBC	127.464	125.294			127.496	127.329	127.477	126.747
	CFB	127.522	128.223			127.565	127.475	127.709	128.881

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 código fuente del script es el siguiente:

```

1  #!/usr/bin/env python
2
3  #
4  # Description: Calculates Shannon Entropy of argv[1] file.
5  #
6  # Author: Sergio García Prado
7  #         garciparedes.me
8  #
9
10 import sys
11 import math
12 import pandas
13
14 with open(sys.argv[1], "rb") as f:
15     byteArray = map(ord, f.read())
16     f.close()
17     fileSize = len(byteArray)
18
19     #Obtain frequency table
20     freqList = pandas.Series(byteArray).value_counts(normalize = True)
21
22     # Shannon entropy
23     ent = 0.0
24     for freq in freqList:
25         ent = ent + freq * math.log(freq, 2)
26     ent = -ent
27     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?

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			7.99959	7.99918	4.00186	4.84131
	CBC	7.99951	7.96728			7.99948	7.99916	7.99954	7.50405
	CFB	7.99947	7.96474			7.99952	7.99928	7.99958	7.51336
DES	ECB	4.23143	7.92140			7.99945	7.99912	3.00133	3.97329
	CBC	7.99948	7.97074			7.99950	7.99925	7.99954	7.48632
	CFB	7.99952	7.96930			7.99958	7.99930	7.99952	7.51143
CAMELLIA-	ECB	5.28436	7.94898			7.99959	7.99925	4.00186	4.80631

256	CBC	7.99947	7.96847			7.99956	7.99920	7.99951	7.48066
	CFB	7.99934	7.97072			7.99961	7.99932	7.99957	7.50765
BLOWFISH	ECB	4.04411	7.92089			7.99953	7.99924	3.00139	3.97852
	CBC	7.99949	7.96852			7.99950	7.99931	7.99954	7.52199
	CFB	7.99956	7.96643			7.99951	7.99929	7.99954	7.44861

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.

ENCRYPTED?

