

UNIVERSIDAD DE GRANADA
E.T.S.I INFORMÁTICA Y TELECOMUNICACIONES
GRADO EN INGENIERÍA INFORMÁTICA (GII)

Memoria Práctica 1: Criptosistemas Simétricos

Seguridad y Protección de Sistemas Informáticos
CURSO 2017-2018

Arthur Rodríguez Nesterenko - DNI Y1680851W

16 de octubre de 2017

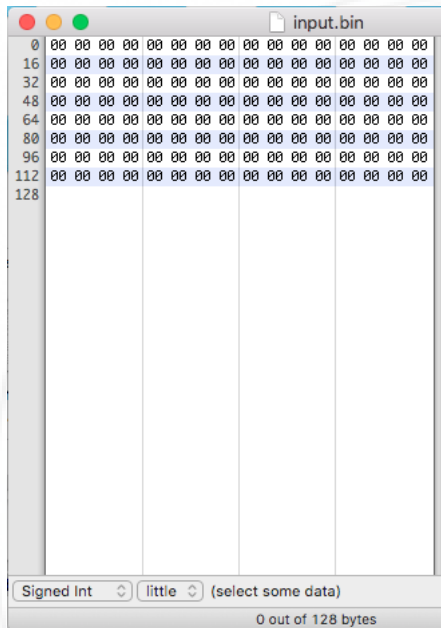
Índice

1. Creación de los ficheros input.bin e input1.bin	3
2. Tareas a realizar: uso de OpenSSL para criptosistemas simétricos	3
2.1. Algoritmo DES: modos ECB, CBC y OFB	4
2.1.1. Punto 3: Cifrar input.bin con DES en modos ECB, CBC y OFB usando como claves una débil y otra semidébil, con vector de inicialización a vuestra elección, y explicad los diferentes resultados.	4
2.1.2. Punto 4: Cifrad input.bin e input1.bin con DES en modo ECB y clave a elegir, pero no débil ni semidébil. Explicad la forma de los resultados obtenidos.	8
2.1.3. Punto 5: Cifrad input.bin e input1.bin con DES en modo CBC y clave y vector de inicialización a elegir. Comparad con los resultados obtenidos en el apartado anterior.	9
2.2. Algoritmo AES 128, 192 y 256. Modos ECB, CBC y OFB	10
2.2.1. Punto 6: Repetid los puntos 4 a 5 con AES-128 y AES-256.	10
2.2.2. Punto 7: Cifrad input.bin con AES-192 en modo OFB, clave y vector de inicialización a elegir. La salida es output.bin.	15
2.2.3. Punto 8: Descifra output.bin utilizando la misma clave y vector de inicialización que en 7.	15
2.2.4. Punto 9: Vuelve a cifrar output.bin con AES-192 en modo OFB, clave y vector de inicialización del punto 7. Compara el resultado obtenido con el punto 8, explicando el resultado.	15
2.3. Descripción de un cifrado simétrico alternativo: CAMELLIA	16

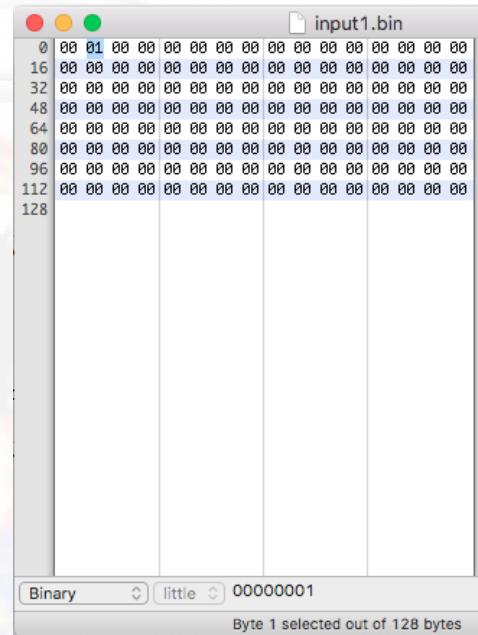
1. Creación de los ficheros input.bin e input1.bin

Partiremos de un archivo binario de 1024 bits, todos ellos con valor 0, al que llamaremos **input.bin**. Posteriormente realizaremos la misma operación, creando un fichero del mismo tamaño pero esta vez con un único bit con valor 1 dentro de los primeros 40 bits (todo lo demás se mantiene a 0). A este archivo se le conocerá como **input1.bin**.

Usando el editor hexadecimal iHex, procedemos a crear los ficheros binarios. En las capturas de pantalla que se ven a continuación se aprecia el contenido de estos dos ficheros, que serán la base para el desarrollo de esta práctica.



(a) Fichero **input.bin**: 1024 bits a 0



(b) Fichero **input1.bin**: fichero del mismo tamaño que antes pero con un bit a 1 dentro de los primeros 40.

2. Tareas a realizar: uso de OpenSSL para criptosistemas simétricos

Tras haber creado los archivos binarios que nos servirán de base para llevar a cabo una serie de ejercicios o tareas, nos disponemos a realizar las mismas siguiendo el guión de la práctica. Para cada tarea que se propone, expondremos los comandos utilizados y mostraremos capturas de pantallas de la salida de los mismos, seguidos de la explicación pertinente a los distintos resultados y escenarios.

2.1. Algoritmo DES: modos ECB, CBC y OFB

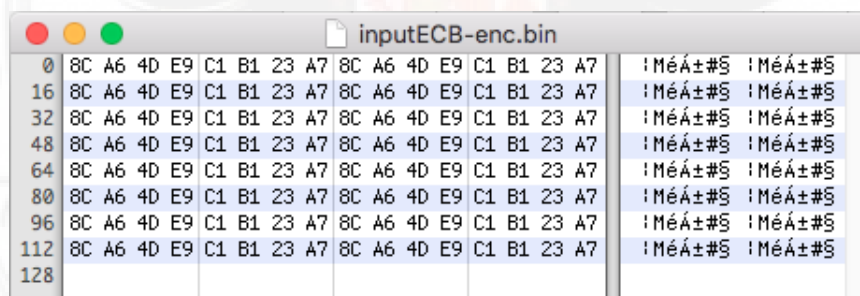
2.1.1. Punto 3: Cifrar input.bin con DES en modos ECB, CBC y OFB usando como claves una débil y otra semidébil, con vector de inicialización a vuestra elección, y explicad los diferentes resultados.

DES es un algoritmo **simétrico de cifrado por bloques**, donde se utiliza una **red de Feistel de 16 rondas** para cifrar **bloques de 64 bits** haciendo uso de claves del mismo tamaño que del tamaño del bloque, donde 56 bits son de clave y 8 bits de paridad. En cada una de las rondas el algoritmo genera dos subclaves de ronda que dependerán del tipo de clave a utilizar, ya sea débil, semidébil o de otro tipo.

La implementación de los algoritmos como tal **actúa únicamente sobre un bloque** por lo que será necesario especificar un modo de cifrado de los consecutivos bloques de un texto en claro, en nuestro caso, ECB, CBC y OFB. A continuación se muestran las salidas correspondientes a aplicar este algoritmo con distintos modos, utilizando claves débiles y semidébiles. Cabe destacar que se utilizará la opción **-nopad** en todos los comandos con el fin de eliminar el bloque de padding que es introducido en el criptograma para ajustar el texto en claro y que sea un múltiplo del tamaño del bloque utilizado.

1. DES con claves débiles

Comando utilizado: `openssl enc -nopad -des-ecb -K 0101010101010101 -in input.bin -out ./Ej3/debil/inputECB-enc.bin`



Offset	Hex Data	ASCII Data
0	8C A6 4D E9 C1 B1 23 A7 8C A6 4D E9 C1 B1 23 A7	¡MéÁ±#§ ¡MéÁ±#§
16	8C A6 4D E9 C1 B1 23 A7 8C A6 4D E9 C1 B1 23 A7	¡MéÁ±#§ ¡MéÁ±#§
32	8C A6 4D E9 C1 B1 23 A7 8C A6 4D E9 C1 B1 23 A7	¡MéÁ±#§ ¡MéÁ±#§
48	8C A6 4D E9 C1 B1 23 A7 8C A6 4D E9 C1 B1 23 A7	¡MéÁ±#§ ¡MéÁ±#§
64	8C A6 4D E9 C1 B1 23 A7 8C A6 4D E9 C1 B1 23 A7	¡MéÁ±#§ ¡MéÁ±#§
80	8C A6 4D E9 C1 B1 23 A7 8C A6 4D E9 C1 B1 23 A7	¡MéÁ±#§ ¡MéÁ±#§
96	8C A6 4D E9 C1 B1 23 A7 8C A6 4D E9 C1 B1 23 A7	¡MéÁ±#§ ¡MéÁ±#§
112	8C A6 4D E9 C1 B1 23 A7 8C A6 4D E9 C1 B1 23 A7	¡MéÁ±#§ ¡MéÁ±#§
128		

Figura 2.1: Input cifrado con DES en modo ECB

Resultado: como se puede observar, la salida que ofrece DES en modo ECB es justo lo que esperamos del **Electronic Code Book**. DES utiliza un tamaño de bloque de 64 bits y en nuestra captura este bloque está representado por dos de las columnas de cada fila. Así, por tanto, se puede ver como el cifrado en modo ECB genera el **mismo criptograma en cada bloque de 64 bits**, puesto que el cifrado es independiente en cada bloque, manteniéndose la **confusión** y **difusión** localmente en cada uno de estos bloques.

Comando utilizado: `openssl enc -nopad -des-cbc -K 0101010101010101 -iv A716F8964346068F -in input.bin -out ./Ej3/debil/inputCBC-enc.bin`

Address	Hex Data
0	03 D8 A8 35 C6 37 58 E3 A7 16 F8 96 43 46 06 8F
16	03 D8 A8 35 C6 37 58 E3 A7 16 F8 96 43 46 06 8F
32	03 D8 A8 35 C6 37 58 E3 A7 16 F8 96 43 46 06 8F
48	03 D8 A8 35 C6 37 58 E3 A7 16 F8 96 43 46 06 8F
64	03 D8 A8 35 C6 37 58 E3 A7 16 F8 96 43 46 06 8F
80	03 D8 A8 35 C6 37 58 E3 A7 16 F8 96 43 46 06 8F
96	03 D8 A8 35 C6 37 58 E3 A7 16 F8 96 43 46 06 8F
112	03 D8 A8 35 C6 37 58 E3 A7 16 F8 96 43 46 06 8F
128	

Figura 2.2: Input cifrado con DES en modo CBC

Resultado: DES en modo **Cipher Block Chaining** funciona de forma ligeramente distinta. En este modo, la salida del cifrado de un bloque (de 64 bits) se le **suma aplicando un XOR** a la entrada del siguiente bloque de forma que es necesario emplear un vector de inicialización **IV** para realizar esta operación sobre el primer bloque.

La salida obtenida sigue el siguiente patrón: el primer bloque de 64 bits cifrado se corresponde concretamente con el $e_k(IV) \oplus c_1$, esto es, el vector de inicialización cifrado al que posteriormente se le realiza un XOR con un bloque de 64 bits que contiene únicamente ceros. El segundo bloque de 64 bits (y todos los restantes) siguen el mismo mecanismo. Esta vez se suma la salida del cifrado del bloque anterior a la entrada del bloque c_{i+1} .

Pero ocurre que dado que DES es un cifrado simétrico, las **operaciones de cifrado y descifrado son equivalentes**, por lo que tras volver a cifrar nuestro bloque (el IV cifrado) obtenemos como salida en el segundo bloque el vector de inicialización utilizado: **A716F8964346068F**. Esta operación se repite de tal forma que cada dos bloques cifrados volvemos a obtener el VI, puesto que la operación de cifrado actúa únicamente sobre el IV (en este caso).

Comando utilizado: `openssl enc -nopad -des-ofb -K 0101010101010101 -iv A716F8964346068F -in input.bin -out ./Ej3/debil/inputOFB-enc.bin`

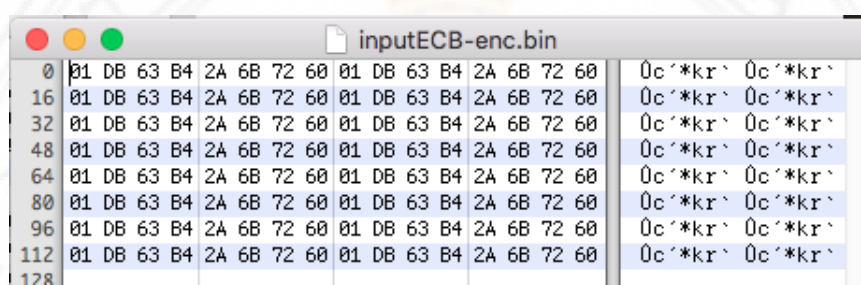
Address	Hex Data
0	03 D8 A8 35 C6 37 58 E3 A7 16 F8 96 43 46 06 8F
16	03 D8 A8 35 C6 37 58 E3 A7 16 F8 96 43 46 06 8F
32	03 D8 A8 35 C6 37 58 E3 A7 16 F8 96 43 46 06 8F
48	03 D8 A8 35 C6 37 58 E3 A7 16 F8 96 43 46 06 8F
64	03 D8 A8 35 C6 37 58 E3 A7 16 F8 96 43 46 06 8F
80	03 D8 A8 35 C6 37 58 E3 A7 16 F8 96 43 46 06 8F
96	03 D8 A8 35 C6 37 58 E3 A7 16 F8 96 43 46 06 8F
112	03 D8 A8 35 C6 37 58 E3 A7 16 F8 96 43 46 06 8F
128	

Figura 2.3: Input cifrado con DES en modo OFB

Resultado: Tras ejecutar DES en modo OFB, nos damos cuenta de que se obtiene exactamente la misma salida que en el caso anterior. ¿Por qué razón se da esto? **Output FeedBack** tiene un funcionamiento muy parecido al de CBC: utiliza el IV cifrado para realizar un XOR con el primer bloque. Para el segundo bloque se utiliza nuevamente como entrada el criptograma obtenido en la salida de la operación anterior; si tenemos en cuenta que cada bloque de 64 bits son ceros, realizar las operaciones de cifrado y XOR en distinto orden en nuestro caso es equivalente, dado que cada una de las **subclaves de ronda son iguales (por tanto no importa el orden)** y porque **cada bloque es idéntico al anterior**.

2. DES con claves semidébiles

Comando utilizado: `openssl enc -nopad -des-ecb -K 01FE01FE01FE01FE -in input.bin -out ./Ej3/semidebil/inputECB-enc.bin`



Offset	Hex	ASCII
0	01 DB 63 B4 2A 6B 72 60 01 DB 63 B4 2A 6B 72 60	Ûc*kr`Ûc*kr`
16	01 DB 63 B4 2A 6B 72 60 01 DB 63 B4 2A 6B 72 60	Ûc*kr`Ûc*kr`
32	01 DB 63 B4 2A 6B 72 60 01 DB 63 B4 2A 6B 72 60	Ûc*kr`Ûc*kr`
48	01 DB 63 B4 2A 6B 72 60 01 DB 63 B4 2A 6B 72 60	Ûc*kr`Ûc*kr`
64	01 DB 63 B4 2A 6B 72 60 01 DB 63 B4 2A 6B 72 60	Ûc*kr`Ûc*kr`
80	01 DB 63 B4 2A 6B 72 60 01 DB 63 B4 2A 6B 72 60	Ûc*kr`Ûc*kr`
96	01 DB 63 B4 2A 6B 72 60 01 DB 63 B4 2A 6B 72 60	Ûc*kr`Ûc*kr`
112	01 DB 63 B4 2A 6B 72 60 01 DB 63 B4 2A 6B 72 60	Ûc*kr`Ûc*kr`
128		

Figura 2.4: Input cifrado con DES en modo ECB

Resultado: La salida de esta ejecución es predecible teniendo en cuenta el ejemplo de modo ECB visto en la ejecución anterior. La seguridad que aporta el modo ECB es cuestionable dado que la confusión y difusión se mantienen **únicamente de forma local en cada bloque**. Por esta razón, independientemente de la clave utilizada, en nuestro caso (archivo de 1024 ceros) obtendremos siempre el mismo patrón: cada bloque de 64 bits se cifra de forma independiente pero siempre dando la misma salida, debido al hecho de que las subclaves de ronda son las mismas y el algoritmo se aplica de forma independiente a cada bloque.

Teniendo en cuenta estas propiedades, en nuestros ficheros en concreto (input.bin e input1.bin) encontraremos **siempre** un patrón que se repite por cada bloque de tamaño X (en caso de DES, 64 bits). La diferencia principal radicará en la clave utilizada, que afectará al criptograma de salida una vez se haya aplicado el algoritmo con este modo.

Comando utilizado: `openssl enc -nopad -des-cbc -K 01FE01FE01FE01FE -iv A716F8964346068F -in input.bin -out ./Ej3/semidebil/inputCBC-enc.bin`

inputCBC-enc.bin																	
0	2D	48	FA	D8	94	57	E2	17	32	65	B6	EF	A7	E9	C4	99	-Húø Wâ 2eŋiSéÄ
16	DD	93	C2	F2	4D	35	B6	67	7E	10	D3	70	AD	51	40	06	Ý ÂðM5ŋg~ Óp-Q@
32	C0	80	7C	E8	ED	8A	32	42	1B	27	8F	E2	5F	E8	9E	C6	À€ èí 2B ' â_è È
48	A6	93	99	8E	D3	C2	07	B3	78	24	BB	51	42	7D	CC	20	! ÓÄ ðx\$»QB}Î
64	E9	4B	52	ED	46	F7	01	84	E0	3A	A3	40	95	CD	CD	CC	éKRíF+ à:É@ ííî
80	2F	A6	B5	E2	9C	8E	31	B9	86	B7	5A	13	9D	EF	2B	8B	/!µâ 1¹ ·Z î+
96	A1	ED	75	63	3C	7E	B0	69	77	82	98	CE	8C	3B	84	26	¡íuc<~*iw Î ; &
112	DC	E1	0E	A7	FB	AE	F6	60	A3	FC	9F	B9	8A	56	51	57	Üá Sû0ö`Éü ¹ VQW
128																	

Figura 2.5: Input cifrado con DES en modo CBC

Resultado: El escenario de este resultado cambia completamente debido a la naturaleza de la clave: al tratarse de una clave semidébil que solo genera dos o cuatro claves de rondas diferentes, al estar alternadas se introduce la suficiente aleatoriedad para que el criptograma de salida sea ininteligible a la vista humana. En esta caso podemos comprobar como no existe ningún patrón a simple vista que nos pueda sugerir alguna forma de tratar de descifrar este criptograma.

Esto ocurre porque una clave semidébil genera un numero determinado de subclaves de ronda (2 o 4) que ya **si son distintas de las demás** y por lo tanto el orden en el que se realicen las operaciones **afecta a la salida del algoritmo**. Teniendo en cuenta además que CBC es un ecadenamiento de bloques, el IV se modificará en algún punto y a partir de ese momento, esa modificación se arrastrará a los demás bloques, dando como resultado una salida como la que se puede ver en la figura 2.5

Comando utilizado: `openssl enc -nopad -des-ofb -K 01FE01FE01FE01FE -iv A716DB4E4346068F -in input.bin -out ./Ej3/semidebil/inputOFB-enc.bin`

inputOFB-enc.bin																	
0	CF	78	F4	FF	3C	E1	F0	02	A9	62	15	18	68	18	B2	E1	Ïxôÿ<áð @b h ºá
16	28	5C	35	CE	90	1B	31	D2	D0	3E	B1	48	9F	FE	7C	07	(\5Î 10ð>±H þ
32	5F	F5	07	1D	A4	40	37	45	88	F5	F5	9B	A0	78	52	21	_õ ¢@7E õõ xR!
48	2B	45	B3	97	27	FE	8A	AF	25	05	F9	6E	81	99	98	D1	+E³ 'þ -% ùn Ñ
64	8E	CD	E5	09	CC	8F	2F	E4	6D	97	63	75	D0	E4	54	23	íâ î /äm cuðäT#
80	B1	6A	BA	F7	0C	F5	EA	97	49	99	FE	BF	2B	39	CC	50	±j²÷ õë I þ¿+9ÎP
96	1E	BB	99	75	DD	88	5A	78	F1	40	C7	6D	2D	43	1A	00	» uÝ Zxñ@Çm-C
112	52	1F	BC	FE	3F	93	42	F0	B4	76	A0	11	77	94	9F	B0	R ðp? Bð'v w °
128																	

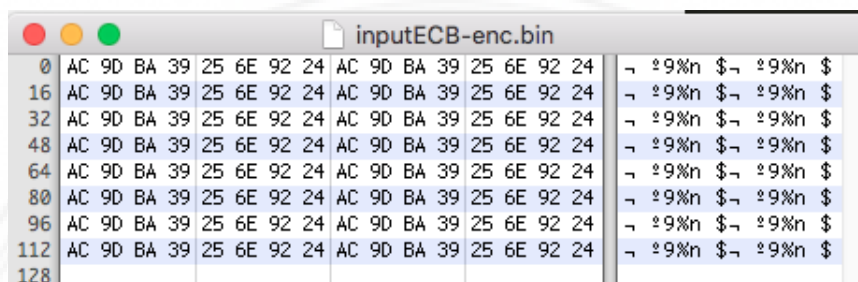
Figura 2.6: Input cifrado con DES en modo OFB

Resultado: Por la misma razón que antes, se pierde esa similitud que compartían los modos CBC y OFB para las claves débiles: la aleatoriedad que producen las claves semidébiles es suficiente para alterar el criptograma ,y el IV altera de forma distinta al primer bloque. Por esta razón no se pueden apreciar similitudes entre las salidas de DES en modos CBC y OFB para claves semidébiles, puesto que ahora el

orden si ha cobrado protagonismo y las operaciones realizadas, a pesar de hacerse sobre el mismo IV, no darán los mismos resultados.

2.1.2. Punto 4: Cifrad input.bin e input1.bin con DES en modo ECB y clave a elegir, pero no débil ni semidébil. Explicad la forma de los resultados obtenidos.

Comando utilizado: `openssl enc -nopad -des-ecb -K 0B2C749F8A15D6A3 -in input.bin -out ./Ej4/inputECB-enc.bin`

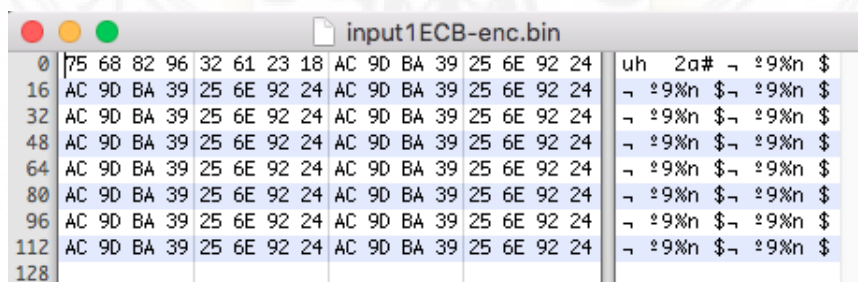


Offset	Hex 1	Hex 2	Hex 3	Hex 4	ASCII
0	AC	9D	BA	39	25
16	AC	9D	BA	39	25
32	AC	9D	BA	39	25
48	AC	9D	BA	39	25
64	AC	9D	BA	39	25
80	AC	9D	BA	39	25
96	AC	9D	BA	39	25
112	AC	9D	BA	39	25
128					

Figura 2.7: Input cifrado con DES en modo ECB con clave a elegir

Resultado: No se aprecia ninguna diferencia notable en cuanto a las salidas de los ejemplos anteriores. La razón principal es que el funcionamiento del modo ECB es, por decirlo de alguna forma, **independiente de la clave**. Ya sea una clave débil, semidébil o de otro tipo, **las subclaves de ronda que se generan son siempre las mismas** cada vez que se aplica el algoritmo, aunque ninguna de las subclaves de ronda sean iguales entre si, y dado que se cifra de forma independiente cada bloque de 64 bits, la salida será siempre la misma por cada bloque y lo único que cambiará será el criptograma en sí (que sí es dependiente de la clave elegida).

Comando utilizado: `openssl enc -nopad -des-ecb -K 0B2C749F8A15D6A3 -in input1.bin -out ./Ej4/input1ECB-enc.bin`



Offset	Hex 1	Hex 2	Hex 3	Hex 4	Hex 5	ASCII
0	75	68	82	96	32	uh 2a#
16	AC	9D	BA	39	25	
32	AC	9D	BA	39	25	
48	AC	9D	BA	39	25	
64	AC	9D	BA	39	25	
80	AC	9D	BA	39	25	
96	AC	9D	BA	39	25	
112	AC	9D	BA	39	25	
128						

Figura 2.8: Input1 cifrado con DES en modo ECB con clave a elegir

Resultado: El resultado obtenido en este caso es más que evidente: nuestro primer bloque de 64 bits contiene un único bit a 1, por tanto el cifrado de ese bloque será completamente distinto del de los demás a pesar de estar utilizando la misma clave que en el caso anterior. En este caso, sin embargo, **la difusión se sigue manteniendo local a cada bloque**, por lo que no se dará el caso ideal ni mucho menos, que consistiría en que se cambiasen aproximadamente la mitad de los caracteres del criptograma; en nuestro caso, esto solo ocurre a nivel del bloque que se cifra, sin afectar a ninguno de los demás bloques, cuyos criptogramas se corresponden con los de la figura 2.7.

2.1.3. Punto 5: Cifrad input.bin e input1.bin con DES en modo CBC y clave y vector de inicialización a elegir. Comparad con los resultados obtenidos en el apartado anterior.

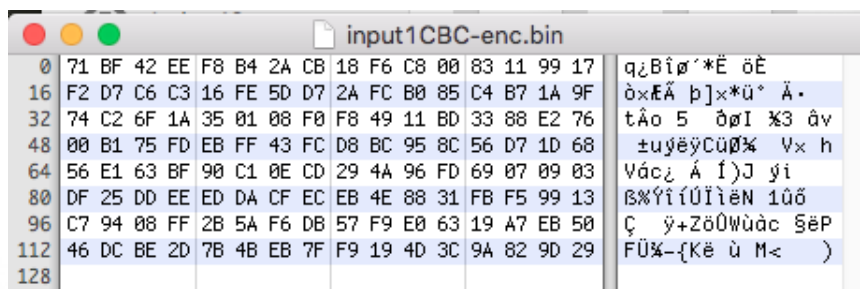
Comando utilizado: `openssl enc -nopad -des-cbc -K 0B2C749F8A15D6A3 -iv 6DAC134E0AFF46DD -in input.bin -out ./Ej5/inputCBC.bin`

0	B1 48 4A 23 36 D8 45 65 FA 45 15 8E 28 51 09 6B	±HJ#6øEeúE (Q k
16	A7 BE 88 D8 52 D4 4E 7C AA 5C 38 57 3B 73 1C 92	S% ØRÖN ²\8W;s
32	A9 4C 5E F9 4D A9 D3 68 40 3A 80 7C E6 D0 27 30	0L^ùM00h0:el0'0
48	A0 E5 86 15 F1 72 A1 CA B8 AF 61 E9 5D D5 69 BC	ð ñrjÊ,-aé]öi%
64	39 5A 3A 95 14 98 CD FE C1 57 0B 7D 2A B8 07 F3	9Z: ípÁW }*» ó
80	CF FF FF 26 41 B3 AD 2B 80 84 60 7A 0B 47 1E 88	İÿÿ&A³-+€ `z G
96	A9 90 DF A3 78 7D D4 17 3B 9D DB 1C C2 70 86 C0	0 B£x}Ü ; Ü Âp À
112	DB AB 8C A8 7F 09 62 74 15 70 35 93 98 D0 3F 67	Ü« ~ bt p5 0?g
128		

Figura 2.9: Input cifrado con DES en modo CBC con clave a elegir

Resultado: La salida que obtenemos al ejecutar DES en modo CBC con una clave cualquiera (ni débil ni semidébil) es un claro ejemplo de que las propiedades de **difusión y confusión ya no se mantienen locales a cada bloque** como en el modo ECB, sino que la salida del cifrado de un bloque que se suma a la entrada del otro y así para todo el conjunto de bloques, permitiría que **un cambio en un único bloque** (digamos el primero) cause una salida totalmente distinta a la que tenemos actualmente, que es justo el caso que veremos a continuación. Además, contar con una clave que no sea débil ni semidébil puede no generar ninguna subclave de ronda que sea igual que alguna generada en algún paso previo, añadiendo mas aleatoriedad al criptograma de salida.

Comando utilizado: `openssl enc -nopad -des-cbc -K 0B2C749F8A15D6A3 -iv 6DAC134E0AFF46DD -in input1.bin -out ./Ej5/input1CBC-enc.bin`



0	71	BF	42	EE	F8	B4	2A	CB	18	F6	C8	00	83	11	99	17	q¿Bîø´*Ë öÈ
16	F2	D7	C6	C3	16	FE	5D	D7	2A	FC	B0	85	C4	B7	1A	9F	òxÆÃ þ]×*Ü* Ä.
32	74	C2	6F	1A	35	01	08	F0	F8	49	11	BD	33	88	E2	76	tÂo 5 òøI %3 âv
48	00	B1	75	FD	EB	FF	43	FC	D8	BC	95	8C	56	D7	1D	68	±uyëÿCÜø% V× h
64	56	E1	63	BF	90	C1	0E	CD	29	4A	96	FD	69	07	09	03	Vác¿ Á í)J yi
80	DF	25	DD	EE	ED	DA	CF	EC	EB	4E	88	31	F8	F5	99	13	ß%ÝííÜÏîN 10ö
96	C7	94	08	FF	2B	5A	F6	DB	57	F9	E0	63	19	A7	EB	50	Ç ŷ+ZöÜWùâc ŠëP
112	46	DC	BE	2D	7B	4B	EB	7F	F9	19	4D	3C	9A	82	9D	29	FÜ%-(Kë ù M<)
128																	

Figura 2.10: Input1 cifrado con DES en modo CBC con clave a elegir

Resultado: En el punto 4 se observaba como el cambio de un 0 por un 1 en el primer bloque no afectaba **de ninguna forma** al cifrado del resto de los bloques. El encadenamiento que propone CBC en este caso permite que se **arrastre un ligero cambio en el texto en claro** para que provoque numerosos cambios (en la mitad de los caracteres del criptograma total) de forma que se cumpla la propiedad de difusión. La otra propiedad, la confusión, se cumple de la misma forma dado que ahora el texto cifrado no solo va a depender de unos pocos caracteres de la clave de forma directa, sino que ligado con la difusión, dependerá de forma indirecta de los demás caracteres de la clave que hayan modificado los caracteres del criptograma.

2.2. Algoritmo AES 128, 192 y 256. Modos ECB, CBC y OFB

2.2.1. Punto 6: Repetid los puntos 4 a 5 con AES-128 y AES-256.

1. **AES 128: modos ECB y CBC con input e input1** Algoritmo de encriptación simétrico por bloques donde cada bloque que se cifra es de 128 bits y se utilizan claves de 128, 192 y 256 bits. Para representar los bloques se utilizan polinomios de grado tres en el grupo multiplicativo F_{256} . Veamos a continuación como varían las salidas de este algoritmo aplicadas sobre los mismos ficheros utilizados anteriormente.

Comando utilizado: `openssl enc -nopad -aes-128-ecb -K C4D18A0B2C749F8A15D6A3AF3E5729B6 -in input.bin -out ./Ej6/inputAES-128ECB.bin`

Offset	Hex	ASCII
0	ED 20 72 74 2D 68 33 38 04 6D 7E 1C FA 98 10 3F	í rt-h38 m~ ú ?
16	ED 20 72 74 2D 68 33 38 04 6D 7E 1C FA 98 10 3F	í rt-h38 m~ ú ?
32	ED 20 72 74 2D 68 33 38 04 6D 7E 1C FA 98 10 3F	í rt-h38 m~ ú ?
48	ED 20 72 74 2D 68 33 38 04 6D 7E 1C FA 98 10 3F	í rt-h38 m~ ú ?
64	ED 20 72 74 2D 68 33 38 04 6D 7E 1C FA 98 10 3F	í rt-h38 m~ ú ?
80	ED 20 72 74 2D 68 33 38 04 6D 7E 1C FA 98 10 3F	í rt-h38 m~ ú ?
96	ED 20 72 74 2D 68 33 38 04 6D 7E 1C FA 98 10 3F	í rt-h38 m~ ú ?
112	ED 20 72 74 2D 68 33 38 04 6D 7E 1C FA 98 10 3F	í rt-h38 m~ ú ?
128		

Figura 2.11: Input cifrado con AES-128 modo ECB con clave a elegir

Resultado: La única diferencia que se introduce al utilizar AES-128 en modo ECB frente a utilizar DES radica en el tamaño del bloque: **AES tiene un tamaño de bloque de 128 bits** (las 4 columnas de una fila), por tanto ahora el patrón de repetición se verá cada 128 bits. La clave de 128 bits sirve únicamente para ajustarse al tamaño del bloque, además de que el vector de inicialización también ha de ajustarse al tamaño del bloque y por tanto será de 128 bits.

Comando utilizado: `openssl enc -nopad -aes-128-ecb -K C4D18A0B2C749F8A15D6A3AF3E5729B6 -in input1.bin -out ./Ej6/input1AES-128ECB.bin`

Offset	Hex	ASCII
0	8A D7 B8 14 98 3A 8E 29 3B 69 50 50 5A 7C 4C DE	×, :);iPPZ Lb
16	ED 20 72 74 2D 68 33 38 04 6D 7E 1C FA 98 10 3F	í rt-h38 m~ ú ?
32	ED 20 72 74 2D 68 33 38 04 6D 7E 1C FA 98 10 3F	í rt-h38 m~ ú ?
48	ED 20 72 74 2D 68 33 38 04 6D 7E 1C FA 98 10 3F	í rt-h38 m~ ú ?
64	ED 20 72 74 2D 68 33 38 04 6D 7E 1C FA 98 10 3F	í rt-h38 m~ ú ?
80	ED 20 72 74 2D 68 33 38 04 6D 7E 1C FA 98 10 3F	í rt-h38 m~ ú ?
96	ED 20 72 74 2D 68 33 38 04 6D 7E 1C FA 98 10 3F	í rt-h38 m~ ú ?
112	ED 20 72 74 2D 68 33 38 04 6D 7E 1C FA 98 10 3F	í rt-h38 m~ ú ?
128		

Figura 2.12: Input1 cifrado con AES-128 modo ECB con clave a elegir

Resultado: Como bien se ha dicho anteriormente, la modificación de un carácter del texto en claro afectará únicamente al criptograma correspondiente al bloque de 128 bits donde se encuentre la modificación, dado que ECB cifra cada bloque de forma independiente a los demás. Por ello, la primera fila no coincide con el de la figura 2.11, mientras que todas las demás sí.

Comando utilizado: `openssl enc -nopad -aes-128-cbc -K C4D18A0B2C749F8A15D6A3AF3E5729B6 -iv 7D0BD48A8E6B96FF4C110FBFF61B806E -in input.bin -out ./Ej6/inputAES-128CBC.bin`

0	76	DE	A7	7F	C5	36	88	C0	D7	49	BC	2C	98	82	64	5C	vD\$ Á6 ÀxI%, d\
16	9C	EA	80	54	65	C3	01	62	E7	4D	1F	D5	9B	03	7E	EC	êeTeA bçM Ò ~ì
32	65	10	93	C1	48	53	28	2F	95	77	D4	27	0B	F2	4F	13	e ÁHS(/ wÔ' ò0
48	85	F7	DC	AA	8B	C6	25	45	CE	DB	BC	F0	53	16	0B	B1	±Ü² E%EI0%0S ±
64	A6	BE	9F	4A	69	B5	2D	22	79	4E	9A	5A	7F	B8	88	6D	!% Jip-"yN Z , m
80	F2	52	6D	A8	1A	F3	9A	F0	30	E2	23	A7	58	93	91	8F	òRm" ó òòâ#5X
96	64	93	52	38	1E	B5	68	71	01	E3	47	F5	46	C0	58	03	d R8 µhq áGôFAX
112	A5	C6	01	E3	5D	7E	57	43	10	28	CD	0B	35	B0	FC	A8	%E äj~wC (f 5*ü"
128																	

Figura 2.13: Input cifrado con AES-128 modo CBC con clave a elegir

Resultado: Si bien llegado al modo CBC no se puede hacer otra cosa que remarcar el hecho de que el encadenamiento inherente al modo es el que permite que las propiedades de **difusión y confusión sean globales al criptograma de salida total**. Por esa misma razón, la modificación de un único bit del primer bloque hará que la salida del siguiente bloque, a pesar de contener la misma clave y vector de inicialización que para el caso anterior, sea completamente distinta de principio a fin.

Comando utilizado: `openssl enc -nopad -aes-128-cbc -K C4D18A0B2C749F8A15D6A3AF3E5729B6 -iv 7D0BD48A8E6B96FF4C110FBFF61B806E -in input1.bin -out ./Ej6/input1AES-128CBC.bin`

0	11	B5	49	D1	43	47	D3	A4	27	14	FC	42	C4	30	82	63	µIÑCGó" úBÄ0 c
16	7E	83	AC	06	01	0F	A6	DC	2B	92	CC	73	DE	1C	C0	A4	~ - !Ü+ lsb À
32	0E	59	C5	AE	51	64	EC	FF	6A	62	2A	33	6B	EE	18	AF	YÅ@Qdìyjb*3kî -
48	3C	03	F2	0B	00	0C	95	05	7A	D3	78	44	B4	1E	F6	8D	< ò zÔxD' ò
64	B3	50	9A	C5	6C	D5	05	B8	7D	0D	AD	E8	C2	C8	92	E6	'P ÅlÖ ,} -èÂÊ e
80	89	1E	26	13	F0	64	18	8F	E3	A0	C6	66	BD	08	78	CC	& òd á f% xÌ
96	83	FD	8C	8A	65	09	8C	42	3F	9B	DF	40	E7	BF	FA	CD	y e B? B@ç¿ÚÍ
112	05	AB	4C	D9	F7	96	9A	40	E3	C3	85	19	D0	6E	F9	3B	«LÜ÷ @äÄ ðnù;
128																	

Figura 2.14: Input1 cifrado con AES-128 modo CBC con clave a elegir

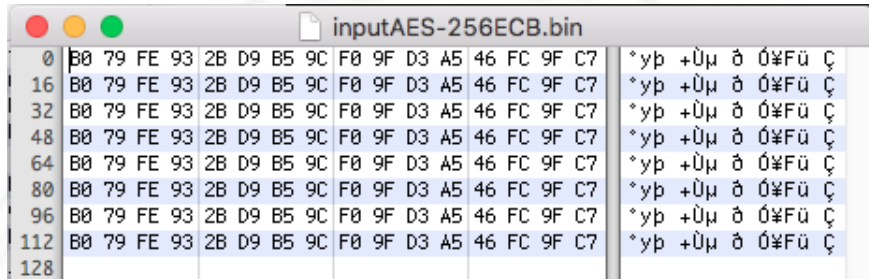
Resultado: En este caso se puede comprobar como el modo CBC introduce la confusión y difusión a nivel global del criptograma. Cambiar un bit 0 por un 1 ha conseguido cambiar la mayor parte de los dígitos hexadecimales correspondientes a la salida del criptograma con respecto a la de la figura 2.13.

2. AES 256: modos ECB y CBC con input e input1.

Como comentario global a este apartado, cabe destacar que la única diferencia con el apartado anterior (AES-128) radica en el tamaño de la clave empleada para cifrar el texto. El **tamaño de bloque** y del **vector de inicialización** se mantienen en 128 bits, por lo que se pueden ver de la misma forma los aspectos que se han comentado en cada una de las figuras y subapartados anteriores.

Comando utilizado: `openssl enc -nopad -aes-256-ecb -K`

`C94A324B7221AA8A8760DA0717C80256EF4308EC6068B7144AA3BBA4A5F98007`
`-in input.bin -out ./Ej6/inputAES-256ECB.bin`



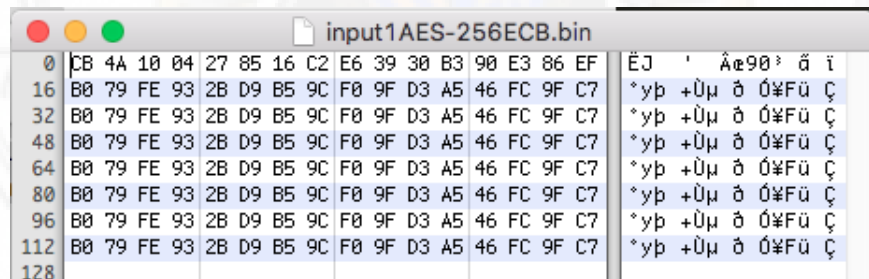
Offset	Hex	ASCII
0	B0 79 FE 93 2B D9 B5 9C F0 9F D3 A5 46 FC 9F C7	*yb +0μ ð Ó%Fü Ç
16	B0 79 FE 93 2B D9 B5 9C F0 9F D3 A5 46 FC 9F C7	*yb +0μ ð Ó%Fü Ç
32	B0 79 FE 93 2B D9 B5 9C F0 9F D3 A5 46 FC 9F C7	*yb +0μ ð Ó%Fü Ç
48	B0 79 FE 93 2B D9 B5 9C F0 9F D3 A5 46 FC 9F C7	*yb +0μ ð Ó%Fü Ç
64	B0 79 FE 93 2B D9 B5 9C F0 9F D3 A5 46 FC 9F C7	*yb +0μ ð Ó%Fü Ç
80	B0 79 FE 93 2B D9 B5 9C F0 9F D3 A5 46 FC 9F C7	*yb +0μ ð Ó%Fü Ç
96	B0 79 FE 93 2B D9 B5 9C F0 9F D3 A5 46 FC 9F C7	*yb +0μ ð Ó%Fü Ç
112	B0 79 FE 93 2B D9 B5 9C F0 9F D3 A5 46 FC 9F C7	*yb +0μ ð Ó%Fü Ç
128		

Figura 2.15: Input cifrado con AES-256 modo ECB con clave a elegir

Resultado: Repetición del criptograma cada 128 bits.

Comando utilizado: `openssl enc -nopad -aes-256-ecb -K`

`C94A324B7221AA8A8760DA0717C80256EF4308EC6068B7144AA3BBA4A5F98007`
`-in input1.bin -out ./Ej6/input1AES-256ECB.bin`



Offset	Hex	ASCII
0	B0 79 FE 93 2B D9 B5 9C F0 9F D3 A5 46 FC 9F C7	*yb +0μ ð Ó%Fü Ç
16	B0 79 FE 93 2B D9 B5 9C F0 9F D3 A5 46 FC 9F C7	*yb +0μ ð Ó%Fü Ç
32	B0 79 FE 93 2B D9 B5 9C F0 9F D3 A5 46 FC 9F C7	*yb +0μ ð Ó%Fü Ç
48	B0 79 FE 93 2B D9 B5 9C F0 9F D3 A5 46 FC 9F C7	*yb +0μ ð Ó%Fü Ç
64	B0 79 FE 93 2B D9 B5 9C F0 9F D3 A5 46 FC 9F C7	*yb +0μ ð Ó%Fü Ç
80	B0 79 FE 93 2B D9 B5 9C F0 9F D3 A5 46 FC 9F C7	*yb +0μ ð Ó%Fü Ç
96	B0 79 FE 93 2B D9 B5 9C F0 9F D3 A5 46 FC 9F C7	*yb +0μ ð Ó%Fü Ç
112	B0 79 FE 93 2B D9 B5 9C F0 9F D3 A5 46 FC 9F C7	*yb +0μ ð Ó%Fü Ç
128		

Figura 2.16: Input1 cifrado con AES-256 modo ECB con clave a elegir

Resultado: Primer bloque de 128 bits distinto de los demás, mientras que los restantes son iguales a los que aparecen en la figura 2.15.

Comando utilizado: `openssl enc -nopad -aes-256-cbc -K C94A324B7221AA8A8760DA0717C80256EF4308EC6068B7144AA3BBA4A5F98007 -iv AEF07C99850B4F6DE18D974F92372C17 -in input.bin -out ./Ej6/inputAES-256CBC.bin`

Offset	Hex	ASCII
0	49 2E F2 70 5A 75 D0 BE 4A 01 DB 86 6E 69 E7 FD	I.òpZu0%J Ò niçý
16	57 94 A0 46 F1 3C 1C F7 3C DA 3E 7E 54 00 DB B3	W Fñ< ÷<Ù>~T Ò¸
32	B0 AE C9 22 84 52 EB 8F 17 BD B5 DB 67 1F B3 9C	*@É" Rê %µ0g ¸
48	48 A5 38 60 8C 1E 82 31 44 FF 45 A2 E3 23 90 D3	H#8` 1DÿEçâ# Ó
64	77 F2 DC 77 1D 23 50 5E E2 0D 0A EE CA A7 A3 6A	wòÜw #P^â iÊSËj
80	DF 76 78 6D 25 B0 58 59 75 49 A2 6A D3 7D B3 14	ßvxm%*XYuIçjÓ}¸
96	BD 32 8A 58 DE 3A A0 DC 50 2F EE B8 D0 CC 32 51	%2 Xb: ÜP/i,0I2Q
112	3E 93 11 EF 50 98 6A 64 6A 50 A6 71 19 83 F2 D6	> iP jdjP!q òÜ
128		

Figura 2.17: Input cifrado con AES-256 modo CBC con clave a elegir

Resultado: Las propiedades de confusión y difusión, globales al criptograma, se reflejan en la salida obtenida, y por ello a simple vista es imposible identificar algún patrón como los que introduce ECB en los bloques de cierto tamaño.

Comando utilizado: `openssl enc -aes-256-cbc -K C94A324B7221AA8A8760DA0717C80256EF4308EC6068B7144AA3BBA4A5F98007 -iv AEF07C99850B4F6DE18D974F92372C17 -in input1.bin -out ./Ej6/input1AES-256CBC.bin`

Offset	Hex	ASCII
0	8E 48 8A C9 88 14 16 99 55 75 3F 9D 7B EA A6 92	H É Uu? {ê!
16	57 7D 9F 61 14 88 9C CE E5 3B 4E DF C8 DF 30 8E	W} a Îâ;N&Ëß0
32	3E D5 F5 ED FA 48 5B 7E DF AA 48 47 25 65 7B 57	>ÖöíúH[~B²HG%e{W
48	C8 D9 D9 56 E8 E9 93 85 E1 FB D8 A9 2F 26 EB 17	ÈÜÜVèè áÜØ0/ðè
64	9F 9F 4C 30 30 73 2E 9F B9 33 7A DD A4 50 18 75	L00s. ¹3zÝ0P u
80	07 BD 80 8F 27 AD 00 DA 77 E1 DC D0 06 0E 46 26	%e '- ÜwáÜ0 F&
96	5A 29 81 D8 50 ED AF 59 25 5F EF BE 0E EA C1 90	Z) ØPÍ-Y%_i% éÁ
112	8C C4 CF AB A2 DE 9E A3 5C 0D AC 5E 3A 1D D8 68	ÄÎ«çp È\ -^: Øh
128	50 12 25 37 1C 25 CB F4 A9 44 5F 51 D9 2E 9C 39	P %7 %Êô0D_QÜ. 9
144		

Figura 2.18: Input1 cifrado con AES-256 modo CBC con clave a elegir

Resultado: La salida es distinta por las mismas propiedades que antes, un único cambio afecta a gran parte del criptograma obtenido en la salida total.

2.2.2. Punto 7: Cifrad input.bin con AES-192 en modo OFB, clave y vector de inicialización a elegir. La salida es output.bin.

Comando utilizado: `openssl enc -nopad -aes-192-ofb
-K 2A457B21AA6C0B249EF43FDA07178A15D6A8B714490E1FD3
-iv 91DB4ADE39F62314B2C7F43017C8CD64 -in input.bin -out ./Ej7/output.bin`

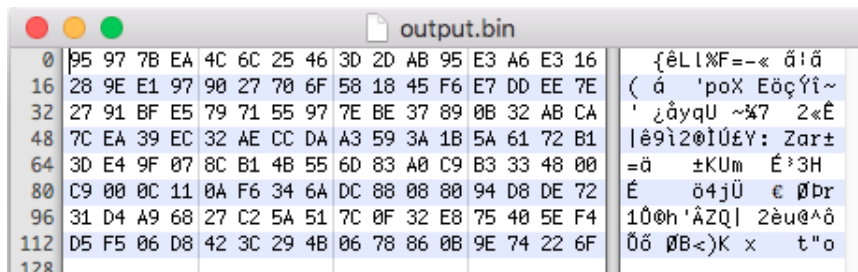


Figura 2.19: Input cifrado con AES-192 con clave e IV a elegir.

2.2.3. Punto 8: Descifra output.bin utilizando la misma clave y vector de inicialización que en 7.

Comando utilizado: `openssl aes-192-ofb -d
-K 2A457B21AA6C0B249EF43FDA07178A15D6A8B714490E1FD3
-iv 91DB4ADE39F62314B2C7F43017C8CD64 -in ./Ej7/output.bin -out ./Ej8/output-dec.bin`

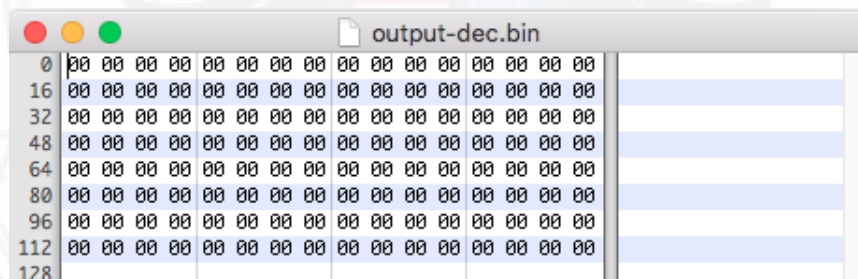


Figura 2.20: Output descifrado con AES-192 con la misma clave e IV que en punto 7.

2.2.4. Punto 9: Vuelve a cifrar output.bin con AES-192 en modo OFB, clave y vector de inicialización del punto 7. Compara el resultado obtenido con el punto 8, explicando el resultado.

Comando utilizado: `openssl enc -aes-192-ofb
-K 2A457B21AA6C0B249EF43FDA07178A15D6A8B714490E1FD3
-iv 91DB4ADE39F62314B2C7F43017C8CD64 -in ./Ej8/output-dec.bin -out ./Ej9/output-enc.bin`

Offset	Hex	ASCII
0	95 97 7B EA 4C 6C 25 46 3D 2D AB 95 E3 A6 E3 16	{èLl%F=-« ä:ä
16	28 9E E1 97 90 27 70 6F 58 18 45 F6 E7 DD EE 7E	(ä 'poX EöçYi~
32	27 91 BF E5 79 71 55 97 7E BE 37 89 0B 32 AB CA	' ¿äyqu ~%7 2«Ê
48	7C EA 39 EC 32 AE CC DA A3 59 3A 1B 5A 61 72 B1	è9ì2@IÜ&Y: Zar±
64	3D E4 9F 07 8C B1 4B 55 6D 83 A0 C9 B3 33 48 00	=ä ±KUm É'3H
80	C9 00 0C 11 0A F6 34 6A DC 88 08 80 94 D8 DE 72	É ö4jÜ € øbr
96	31 D4 A9 68 27 C2 5A 51 7C 0F 32 E8 75 40 5E F4	10@h'ÂZQ 2èu@^ô
112	D5 F5 06 D8 42 3C 29 4B 06 78 86 0B 9E 74 22 6F	Öö øB<)K x t"o
128		

Figura 2.21: Output descifrado con AES-192 con la misma clave e IV que en punto 7.

Resultado: Tras la operación cifrado-descifrado-cifrado se puede comprobar como el algoritmo AES es un algoritmo **simétrico**, independientemente del tamaño de la clave. En este caso, tanto cifrar como descifrar son equivalentes por lo que es fácil comprobar como la operación no altera de ninguna forma el texto en claro. El cifrado que se realiza con una clave y el posterior descifrado realizado con la misma clave nos dan la salida del fichero **input.bin**, que contiene nuestros 1024 bits a cero.

Una nueva operación de cifrado nos sirve únicamente para comprobar como, siempre y cuando se utilice la misma clave y el mismo IV, se obtendrá la misma salida cifrada una y otra vez, haciendo énfasis en la simetría del algoritmo.

2.3. Descripción de un cifrado simétrico alternativo: CAMELLIA

A continuación se presenta la descripción de un tipo de cifrado simétrico que se encuentra disponible en mi implementación de OpenSSL.

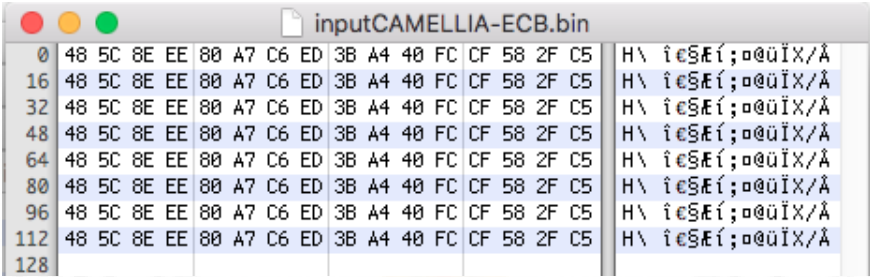
Para encontrar los cifrados simétricos disponibles en mi implementación de OpenSSL he utilizado la salida del comando **openssl list-cipher-algorithms**, encontrando un algoritmo de cifrado llamado **Camellia** que describiremos a continuación.

Camellia [1] es un algoritmo de cifrado simétrico de bloque. El tamaño de bloque es de 128 bits y los tamaños de clave son, como en AES, 128, 192 y 256 bits. Desarrollado por Mitsubishi Electric en Japón, es un algoritmo que utiliza una **red de Feistel de 18 rondas** cuando utiliza claves de **128 bits** o de **24 rondas**, con claves de **192 y 256 bits**, en donde cada 6 rondas se aplica una transformación lógica denominada *Función FL* o su inversa.

La seguridad de Camellia como algoritmo de cifrado está muy probada, donde incluso se considera que utilizar la clave de 128 bits para cifrar es prácticamente irrompible por fuerza bruta. Además, como bien especifica en el RFC 4312 [2], en el momento de su implementación (y actualmente) **no se conocen claves débiles ni semidébiles** para el algoritmo, por lo que se considera un algoritmo robusto en la actualidad.

Como última tarea en el guión, se propone repetir los puntos 3 a 5 con el cifrado presentado anteriormente (el 3 si el cifrado elegido tuviese claves débiles o semidébiles). En nuestro caso solo repetiremos los pasos 4 y 5, esto es, el algoritmo Camellia128 con modos ECB y CBC. Veamos los resultados a continuación.

Comando utilizado: `openssl enc -nopad -camellia-128-ecb -K D020AC919FB8F68273CDB0A4781A5511 -in input.bin -out ./Ej11/inputCAMELLIA-ECB.bin`

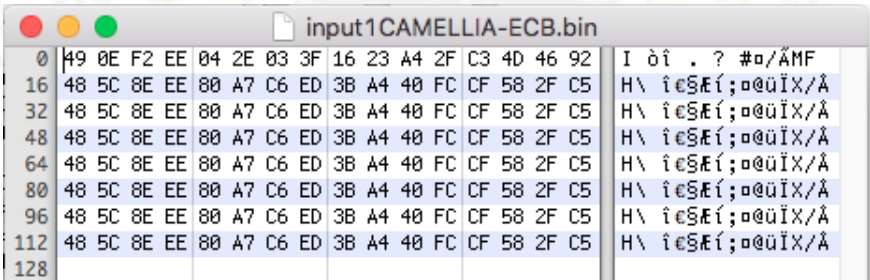


Offset	Hex	ASCII
0	48 5C 8E EE 80 A7 C6 ED 3B A4 40 FC CF 58 2F C5	H\ i e S f ; o u i x / Å
16	48 5C 8E EE 80 A7 C6 ED 3B A4 40 FC CF 58 2F C5	H\ i e S f ; o u i x / Å
32	48 5C 8E EE 80 A7 C6 ED 3B A4 40 FC CF 58 2F C5	H\ i e S f ; o u i x / Å
48	48 5C 8E EE 80 A7 C6 ED 3B A4 40 FC CF 58 2F C5	H\ i e S f ; o u i x / Å
64	48 5C 8E EE 80 A7 C6 ED 3B A4 40 FC CF 58 2F C5	H\ i e S f ; o u i x / Å
80	48 5C 8E EE 80 A7 C6 ED 3B A4 40 FC CF 58 2F C5	H\ i e S f ; o u i x / Å
96	48 5C 8E EE 80 A7 C6 ED 3B A4 40 FC CF 58 2F C5	H\ i e S f ; o u i x / Å
112	48 5C 8E EE 80 A7 C6 ED 3B A4 40 FC CF 58 2F C5	H\ i e S f ; o u i x / Å
128		

Figura 2.22: Input cifrado con CAMELLIA-128 modo ECB con clave a elegir

Resultado: Independientemente del funcionamiento interno del algoritmo, el cual no ha sido detallado en su totalidad, aplicar un algoritmo de cifado simétrico por bloques en modo ECB da como resultado un patrón similar a los que se han obtenido con algoritmos cuyo tamaño de bloque es de 128 bits. En la captura anterior se puede ver como cada bloque de 128 bits se ha cifrado de forma independiente, mostrando la misma salida para cada uno de ellos.

Comando utilizado: `openssl enc -nopad -camellia-128-ecb -K D020AC919FB8F68273CDB0A4781A5511 -in input1.bin -out ./Ej11/input1CAMELLIA-ECB.bin`



Offset	Hex	ASCII
0	49 0E F2 EE 04 2E 03 3F 16 23 A4 2F C3 4D 46 92	I ò i . ? # o / Å M F
16	48 5C 8E EE 80 A7 C6 ED 3B A4 40 FC CF 58 2F C5	H\ i e S f ; o u i x / Å
32	48 5C 8E EE 80 A7 C6 ED 3B A4 40 FC CF 58 2F C5	H\ i e S f ; o u i x / Å
48	48 5C 8E EE 80 A7 C6 ED 3B A4 40 FC CF 58 2F C5	H\ i e S f ; o u i x / Å
64	48 5C 8E EE 80 A7 C6 ED 3B A4 40 FC CF 58 2F C5	H\ i e S f ; o u i x / Å
80	48 5C 8E EE 80 A7 C6 ED 3B A4 40 FC CF 58 2F C5	H\ i e S f ; o u i x / Å
96	48 5C 8E EE 80 A7 C6 ED 3B A4 40 FC CF 58 2F C5	H\ i e S f ; o u i x / Å
112	48 5C 8E EE 80 A7 C6 ED 3B A4 40 FC CF 58 2F C5	H\ i e S f ; o u i x / Å
128		

Figura 2.23: Input1 cifrado con CAMELLIA-128 modo ECB con clave a elegir

Resultado: Atendiendo a los resultados anteriores, una modificación en un único bit en el primer bloque hace que el criptograma correspondiente a éste sea completamente

distinto del de los demás, siguiendo el mismo comportamiento que en DES y AES, pero en este caso con un bloque de 128 bits. Veamos que ocurre con el modo CBC.

Offset	Hex	ASCII
0	1E BB C3 1E F5 DF D4 E1 55 F2 9D 98 45 69 80 FC	»Ã õßÓáUò Eieü
16	08 05 A6 0C BE 55 D1 1D BB 49 B7 00 08 29 61 7C	! %UÑ »I.)a
32	A3 91 0D 00 72 D9 11 9A 40 E8 01 9A C4 A4 6D 3E	é rÛ @è Åom>
48	8B C9 BC 5A 65 6C F2 BA 7D BE FC 6F B2 E8 C1 45	É%Zelò%}¼üo²èÁE
64	2B A0 10 5B B7 52 E2 D0 B6 71 AD C6 EB 1D 8D 15	+ [·Râ0¶q-Æè
80	AB 8E 5D 8F 63 F6 65 16 74 2C 77 9C DC 99 72 B8	«] cõe t,w Ü r,
96	35 85 F6 FF 44 D6 94 98 52 F5 B1 F4 24 27 2D 66	5 öÿDÖ Rõ±ô\$'-f
112	CB 19 55 39 AF F7 D6 00 5C 66 A1 12 62 CE 29 0C	É U9^-±0 \fj bÍ)
128		

Figura 2.24: Input cifrado con CAMELLIA-128 modo CBC con clave a elegir

Resultado: La salida obtenida es, en términos comparativos, equivalente a lo que podríamos obtener aplicando AES-128 a nuestro fichero input: combinaciones de caracteres hexadecimales prácticamente aleatorias. Siguiendo el esquema propuesto anteriormente para los algoritmos de bloque que utilizan este modo, en caso de realizar una pequeña modificación en uno de los caracteres del texto en claro, es lógico pensar que la salida será completamente distinta a ésta, resultado que veremos a continuación.

Offset	Hex	ASCII
0	8A 6C FC 45 DC 0D B7 E3 41 22 0F 39 A4 A2 84 23	lÜEÜ .äA" 9□¢ #
16	C8 32 2D 14 08 04 7E D1 C9 7C 2F 33 3D D2 67 6D	È2- ~ÑÉ /3=0gm
32	A7 B6 B5 F4 7E EA 87 85 40 39 7C 66 08 10 62 EF	Ş¶mô~ê @9 f bİ
48	EC 15 D1 15 17 F0 D3 FC 26 21 CB 16 78 E4 21 34	ì Ñ õóü&!É xä!4
64	16 21 E9 E8 C7 0E 42 2D A2 9D F9 FF ED D8 4A 9C	!éeÇ B-¢ ùÿíØJ
80	34 67 B5 3E FA 6E 73 8F 66 25 50 FF CF 53 CA 57	4gm>úns f%PÿİSÉW
96	E3 1A 13 B9 64 F4 29 41 D5 36 B0 2D 20 D8 E8 F7	á ¹dó)A06*- Øè+
112	6B 48 8B AA E8 2B A8 DD EE 4C 3B 82 9D 40 46 56	kH ²è+~ÝİL; @FV
128		

Figura 2.25: Input1 cifrado con CAMELLIA-128 modo CBC con clave a elegir

Resultado: Efectivamente se ha hecho latente el cambio realizado en un único bit dentro del primer bloque, encadenando los cambios que éste produce a los 7 bloques restantes y obteniendo por tanto una salida que nada tiene que ver con la que se puede ver en la figura 2.24, dando por hecho que el modo de funcionamiento con el que se aplica un algoritmo nos da mucha información de cual es la salida que obtendremos, solo en términos comparativos, porque en este caso la **difusión y confusión** son propiedades que se cumplen en la totalidad del criptograma y no localmente en cada bloque.

Referencias

- [1] [https://en.wikipedia.org/wiki/Camellia_\(cipher\)](https://en.wikipedia.org/wiki/Camellia_(cipher)), Camellia Cipher System.
- [2] <https://tools.ietf.org/html/rfc4312>, RFC 4312: The Camellia Cipher Algorithm and Its Use With IPsec.

