

# Cifrari a blocchi

## Esercizi con OpenSSL

**Alfredo De Santis**

Dipartimento di Informatica  
Università di Salerno

[ads@unisa.it](mailto:ads@unisa.it)

<http://www.dia.unisa.it/professori/ads>



Marzo 2017

# Strumenti Necessari

- Per lo svolgimento degli esercizi sono necessari
  - Bless (Editor Esadecimale)
    - `sudo apt-get install bless`
  - Comando **time**
- Potrebbe essere necessaria la consultazione della seguente tabella
  - <https://www.co.tt/files/asciitable.pdf>

# Bless

## Editor Esadecimale

The screenshot displays the Bless hex editor interface. At the top, there is a toolbar with icons for file operations (new, open, save), navigation (back, forward), editing (cut, copy, paste), and search. Below the toolbar, a tab labeled 'Originale.txt' is active. The main editing area shows a hex dump of data with corresponding ASCII text to the right. The hex values are color-coded: red for the first byte of each line and blue for the others. The ASCII text is 'Corso di Sicurezza Su Reti .Corso di Sicurezza Su Reti .Corso di Sicurezza Su Reti .'. Below the hex dump, a conversion panel provides various representations of the selected data (hex '43 6F 72 73').

Signed 8 bit:	67	Signed 32 bit:	1131377267	Hexadecimal:	43 6F 72 73
Unsigned 8 bit:	67	Unsigned 32 bit:	1131377267	Decimal:	067 111 114 115
Signed 16 bit:	17263	Float 32 bit:	239,4471	Octal:	103 157 162 163
Unsigned 16 bit:	17263	Float 64 bit:	7,08125151138374E+16	Binary:	01000011 01101111 01

Show little endian decoding     Show unsigned as hexadecimal    ASCII Text: Cors

Offset: 0 / 108    Selection: None    INS

<http://home.gna.org/bleess/>

# Comando time

TIME(1)

BSD General Commands Manual

TIME(1)

## NAME

**time** -- time command execution

## SYNOPSIS

**time** [-lp] utility

## DESCRIPTION

The **time** utility executes and times utility. After the utility finishes, **time** writes the total time elapsed, the time consumed by system overhead, and the time used to execute utility to the standard error stream. Times are reported in seconds.

## Ad esempio

```
time ls

real 0m0.006s
user 0m0.001s
sys 0m0.002s
```

# Comando time

TIME(1)

BSD General Commands Manual

TIME(1)

## NAME

`time` -- time command execution

## SYNOPSIS

`time` [-lp] utility

## DESCRIPTION

The `time` utility executes and times utility. After the utility finishes, `time` writes the total time elapsed, the time consumed by system overhead, and the time used to execute utility to the standard error stream. Times are reported in seconds.

Ad esempio

Comando

```
time ls  
  
real 0m0.006s  
user 0m0.001s  
sys 0m0.002s
```

# Base 64: Riepilogo

- Schema di codifica che permette di rappresentare dati binari mediante stringhe ASCII
  - Usa 65 caratteri stampabili
    - 26 lettere minuscole, 26 lettere maiuscole, 10 cifre, i caratteri '+' e '/' ed il carattere speciale '='
  - Comunemente utilizzato quando è necessario codificare dati binari che devono essere memorizzati o trasferiti in formato testuale

# Codifica Base64

- **Esercizio 1.0** Studiare il formato di output dalla codifica Base64
  - Spiegare come vengono trasformati i gruppi di bit, come avviene il padding ed il significato di "==" e "="
- **Esercizio 1.1** Codificare un file di testo arbitrario con Base64 e visualizzare il risultato
- **Esercizio 1.2** Codificare un file binario arbitrario con Base64 e visualizzare il risultato
- **Esercizio 1.3** Codificare con Base64 un file di testo contenente una qualsiasi password ed inviarlo tramite email ad un compagno di corso
  - Chiedere al compagno di decodificare la password memorizzata in questo file

# Cifratura con Password

- **Esercizio 2.1** Qual è lo scopo del parametro `-nosalt` in DES?
- **Esercizio 2.2** Cifrare un file arbitrario e decifrarlo usando la password corretta (usando DES e `-nosalt`)
- **Esercizio 2.3** Cifrare un file arbitrario e provare a decifrarlo usando una password non corretta
- **Esercizio 2.4** Confrontare la dimensione di un testo in chiaro e quella del testo cifrato corrispondente
  - Argomentare e motivare le differenze



# Cifratura con Password

- **Esercizio 2.5** Usare "ls -al > dir1.txt" per generare il file dir1.txt
- **Esercizio 2.6** Cifrare dir1.txt mediante DES ECB per ottenere il file cifrato dir1.ecb. Qual è il comando da usare?
- **Esercizio 2.7** Cifrare dir1.txt mediante DES CBC per ottenere il file cifrato dir1.cbc. Qual è il comando da usare?
- **Esercizio 2.8** Cifrare dir1.txt mediante AES 256 ECB per ottenere il file cifrato dir1\_aes.ecb. Qual è il comando da usare?
- **Esercizio 2.9** Cifrare dir1.txt mediante AES 256 CBC per ottenere il file cifrato dir1\_aes.cbc. Qual è il comando da usare?

# Cifratura con Chiave ed IV

- **Esercizio 2.10** Cifrare un file testuale arbitrario, chiamato file.txt, con AES (chiave da 128 bit) in modalità CBC e salvare il risultato nel file cifrato.txt, usando le opzioni -nosalt e -p

```
openssl enc -e -in file.txt -out cifrato.txt -aes-128-cbc  
-nosalt -p
```

- **Esercizio 2.11** Descrivere l'output del suddetto comando
- **Esercizio 2.12** Descrivere come le opzioni -nosalt e -p influenzano il comando stesso

# Cifratura con Chiave ed IV

- **Esercizio 2.13** Eseguire almeno due volte il seguente comando, utilizzando la stessa password

```
openssl enc -e -in file.txt -out cifrato.txt -aes-128-cbc -nosalt -p
```

- **Esercizio 2.14** Eseguire almeno due volte il seguente comando, utilizzando la stessa password

```
openssl enc -e -in file.txt -out cifrato.txt -aes-128-cbc -p
```

- **Esercizio 2.15** Cosa si può notare?
  - Motivare ed argomentare la risposta
- **Esercizio 2.16** Decifrare il file cifrato.txt mediante il seguente comando

```
openssl enc -d -in cifrato.txt -out decifrato.txt -K chiave -iv iv -aes-128-cbc -p
```

# Analisi di Testi Cifrati

- **Esercizio 3.1** Creare i due seguenti file, che differiscano solo per l'ultimo carattere della prima riga

```
Mio esempio1  
SR_2016/2017  
CifraturaSimmetrica
```

Plaintext1.txt

```
Mio esempio2  
SR_2016/2017  
CifraturaSimmetrica
```

Plaintext2.txt

- **Esercizio 3.2** Cifrare Plaintext1.txt e Plaintext2.txt con AES-128-ECB:
  - usando la chiave abcdef0123456789
  - memorizzando le cifrature nei file Cifrato1.txt e Cifrato2.txt, rispettivamente
  - specificando l'opzione -nosalt

# Analisi di Testi Cifrati

- **Esercizio 3.3** Analizzare il contenuto dei file `Cifrato1.txt` e `Cifrato2.txt` mediante uno dei seguenti strumenti
  - Bless
  - Comando `xxd`
  - Comando `hexdump`
- **Esercizio 3.4** Cosa si può notare?
  - Mettere in evidenza le differenze e le similitudini tra i due file
  - Da cosa derivano le similitudini tra i due file?
- **Esercizio 3.5** Ripetere gli esercizi precedenti (da 3.2 a 3.4), utilizzando AES-128-CBC per la cifratura e specificando il vettore di inizializzazione 012345

# Cifratura di Immagini Bitmap

tux.bmp



# Cifratura di Immagini Bitmap

- Il file tux.bmp contiene un'immagine bitmap
- **Esercizio 4.1** Cifrare tale file usando AES con le modalità ECB e CBC
- **Osservazione** Per poter visualizzare il contenuto dell'immagine cifrata è necessario ripristinare il relativo header, contenuto nei primi 54 byte dell'immagine originaria (tux.bmp)
  - Ciò può essere fatto usando l'editor esadecimale Bless
  - Nell'esempio seguente vedremo come fare

# Cifratura di Immagini Bitmap

- Si assuma che
  - tux\_aes\_128\_ecb.bmp sia il file contenente la cifratura di tux.bmp mediante AES a 128 bit in modalità ECB
  - tux\_aes\_128\_cbc.bmp sia il file contenente la cifratura di tux.bmp mediante AES a 128 bit in modalità CBC
- Mediante Bless ripristiniamo l'header bitmap all'interno dell'immagine cifrata
  - Apriamo sia il file tux.bmp che il file tux\_aes\_128\_ecb.bmp
  - Copiamo i primi 54 byte del file tux.bmp in quelli corrispondenti del file tux\_aes\_128\_ecb.bmp
    - La stessa operazione deve essere fatta per il file tux\_aes\_128\_ecb.bmp



# Cifratura di Immagini Bitmap

Hex editor interface showing the encryption of a BMP file. The original file 'tux.bmp' is compared with the encrypted file 'tux\_aes\_128\_ecb.bmp'. The first 12 bytes of the original file are highlighted in orange and correspond to the ASCII string 'BM.....6... (.....X.....'. The rest of the file is filled with 0xFF bytes, indicating encryption.

Signed 8 bit:	-1	Signed 32 bit:	-1	Hexadecimal:	FF FF FF FF
Unsigned 8 bit:	255	Unsigned 32 bit:	4294967295	Decimal:	255 255 255 255
Signed 16 bit:	-1	Float 32 bit:	NaN	Octal:	377 377 377 377
Unsigned 16 bit:	65535	Float 64 bit:	NaN	Binary:	11111111 11111111 11
<input type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal		ASCII Text:	????

Offset: 54 / 900053      Selection: 0 to 53 (54 bytes)      INS

# Cifratura di Immagini Bitmap

**N.B.** Assicurarsi che la selezione avvenga utilizzando l'indicizzazione in formato decimale

➤ Ciò può essere ottenuto cliccando sull'area indicata dal rettangolo in azzurro

Offset: 54 / 900053      Selection: 0 to 53 (54 bytes)      INS

Signed 8 bit:	-1	Unsigned 32 bit:	4294967295	Hexadecimal:	FF FF FF FF
Unsigned 8 bit:	255	Float 32 bit:	NaN	Decimal:	255 255 255 255
Signed 16 bit:	-1	Float 64 bit:	NaN	Octal:	377 377 377 377
Unsigned 16 bit:	65535			Binary:	11111111 11111111 11
<input type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal		ASCII Text:	????

# Cifratura di Immagini Bitmap

tux.bmp x tux\_aes\_128\_ecb.bmp x

```
00000000 42 4D D6 BB 0D 00 00 00 00 00 36 00 00 00 28 00 00 00 BM.....6... (...
00000012 F4 01 00 00 58 02 00 00 01 00 18 00 00 00 00 00 A0 BB .....X.....
00000024 0D 00 13 0B 00 00 13 0B 00 00 00 00 00 00 00 00 00 .....
00000036 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
00000048 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....

```

Signed 8 bit: -1 Signed 32 bit: -1 Header bitmap di 54 byte FF FF FF x

Unsigned 8 bit: 255 Unsigned 32 bit: 4294967295 Decimal: 255 255 255 255

Signed 16 bit: -1 Float 32 bit: NaN Octal: 377 377 377 377

Unsigned 16 bit: 65535 Float 64 bit: NaN Binary: 11111111 11111111 11

Show little endian decoding  Show unsigned as hexadecimal ASCII Text: ????

Offset: 54 / 900053 Selection: 0 to 53 (54 bytes) INS

# Cifratura di Immagini Bitmap

The screenshot shows a hex editor window with a toolbar at the top and a file tab labeled 'tux.bmp'. The main area displays a hex dump of a BMP file header. A region from offset 00000000 to 00000011 is selected and highlighted in orange. A context menu is open over this region, listing the following options: 'Perform Operation', 'Taglia', 'Copia', 'Incolla', and 'Elimina'. The 'Copia' option is currently selected. Below the hex dump, there is a control panel with various data format options and their values:

Signed 8 bit:	-1	Signed 32 bit:	-1	Hexadecimal:	FF FF FF FF
Unsigned 8 bit:	255	Unsigned 32 bit:	4294967295	Decimal:	255 255 255 255
Signed 16 bit:	-1	Float 32 bit:	NaN	Octal:	377 377 377 377
Unsigned 16 bit:	65535	Float 64 bit:	NaN	Binary:	11111111 11111111 11
<input type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal		ASCII Text:	????

At the bottom of the window, the status bar shows 'Offset: 54 / 900053', 'Selection: 0 to 53 (54 bytes)', and 'INS'.

# Cifratura di Immagini Bitmap

tux.bmp x tux\_aes\_128\_ecb.bmp x

00000000	53 61 6C 74 65 64 5F 5F 63 FF 6D 44 62 F7 E1 77 DB 46	Salted__c.mDb..w.F
00000012	BD D1 82 B4 2A 4F C2 3D 6F 03 95 EF B6 9E 7C 91 DC FC	....*O.=o..... ...
00000024	76 05 34 7A FC 02 62 52 73 9A 7E 5B 04 C0 3D 1F FA 25	v.4z..bRs.~[...=..%
00000036	FA B4 BF EB 15 0D 5D BF 6F 5A 55 18 4A F3 23 A4 9E E4	.....].oZU.J.#...
00000048	D5 CA 25 FA A3 B7 5A B1 A8 BE EC E9 9C C2 A1 0D 7B A5	..%...Z.....{.

Signed 8 bit: -6      Signed 32 bit: -88817685      Hexadecimal: FA B4 BF EB x

Unsigned 8 bit: 250      Unsigned 32 bit: 4206149611      Decimal: 250 180 191 235

Signed 16 bit: -1356      Float 32 bit: -4,69253E+35      Octal: 372 264 277 353

Unsigned 16 bit: 64180      Float 64 bit: -1,20527836817238E+283      Binary: 11111010 10110100 10

Show little endian decoding       Show unsigned as hexadecimal      ASCII Text: ????

Offset: 54 / 900079      Selection: 0 to 53 (54 bytes)      INS

# Cifratura di Immagini Bitmap

The screenshot shows a hex editor window with two tabs: 'tux.bmp' and 'tux\_aes\_128\_ecb.bmp'. The active tab displays a hex dump of the encrypted data. A selection of 54 bytes is highlighted in orange, corresponding to the ASCII text 'Salted\_\_c.mDb..w.F...\*O.=o.....|...v.4z..bRs.~[..=..%.....].oZU.J.#... ..%...Z.....{.'. A context menu is open over the selection, with options: 'Perform Operation', 'Taglia', 'Copia', 'Incolla', and 'Elimina'. The 'Incolla' option is highlighted. Below the hex dump is a properties panel with the following data:

Signed 8 bit:	-6	Signed 32 bit:	-68817065	Hexadecimal:	FA B4 BF EB
Unsigned 8 bit:	250	Unsigned 32 bit:	4206149611	Decimal:	250 180 191 235
Signed 16 bit:	-1356	Float 32 bit:	-4,69253E+35	Octal:	372 264 277 353
Unsigned 16 bit:	64180	Float 64 bit:	-1,20527836817238E+283	Binary:	11111010 10110100 10
<input type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal		ASCII Text:	????

At the bottom of the window, the status bar shows: 'Offset: 54 / 900079', 'Selection: 0 to 53 (54 bytes)', and 'INS'.

# Cifratura di Immagini Bitmap

The screenshot shows a hex editor window with two tabs: 'tux.bmp' and 'tux\_aes\_128\_ecb.bmp\*'. The main display shows the hex dump of the encrypted file. The first 54 bytes of the header are highlighted with a green box. A callout box with a green border and a green arrow points to the first byte of this highlighted region, containing the text 'Header bitmap di 54 byte'. Below the hex dump is a control panel with various data type selection buttons and checkboxes.

Signed 8 bit:	-6	Signed 32 bit:	-8	Hex:	A B4 BF EB
Unsigned 8 bit:	250	Unsigned 32 bit:	4206149611	Decimal:	250 180 191 235
Signed 16 bit:	-1356	Float 32 bit:	-4,69253E+35	Octal:	372 264 277 353
Unsigned 16 bit:	64180	Float 64 bit:	-1,20527836817238E+283	Binary:	11111010 10110100 10
<input type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal		ASCII Text:	????

Offset: 54 / 900079      Selection: None      INS

# Cifratura di Immagini Bitmap

tux.bmp x tux\_aes\_128\_ecb.bmp\* x

```
00000000 42 4D D6 BB 0D 00 00 00 00 00 36 00 00 00 28 00 00 00 BM.....6... (...
00000012 F4 01 00 00 58 02 00 00 01 00 18 00 00 00 00 00 A0 BB ....X.....
00000024 0D 00 13 0B 00 00 13 0B 00 00 00 00 00 00 00 00 00 .....
00000036 FA B4 BF EB 15 0D 5D BF 6F 5A 55 18 4A F3 23 A4 9E E4 .....]oZU.J.#...
00000048 D5 CA 25 FA A3 B7 5A B1 A8 BE EC E9 9C C2 A1 0D 7B A5 ..%...Z.....{.
```

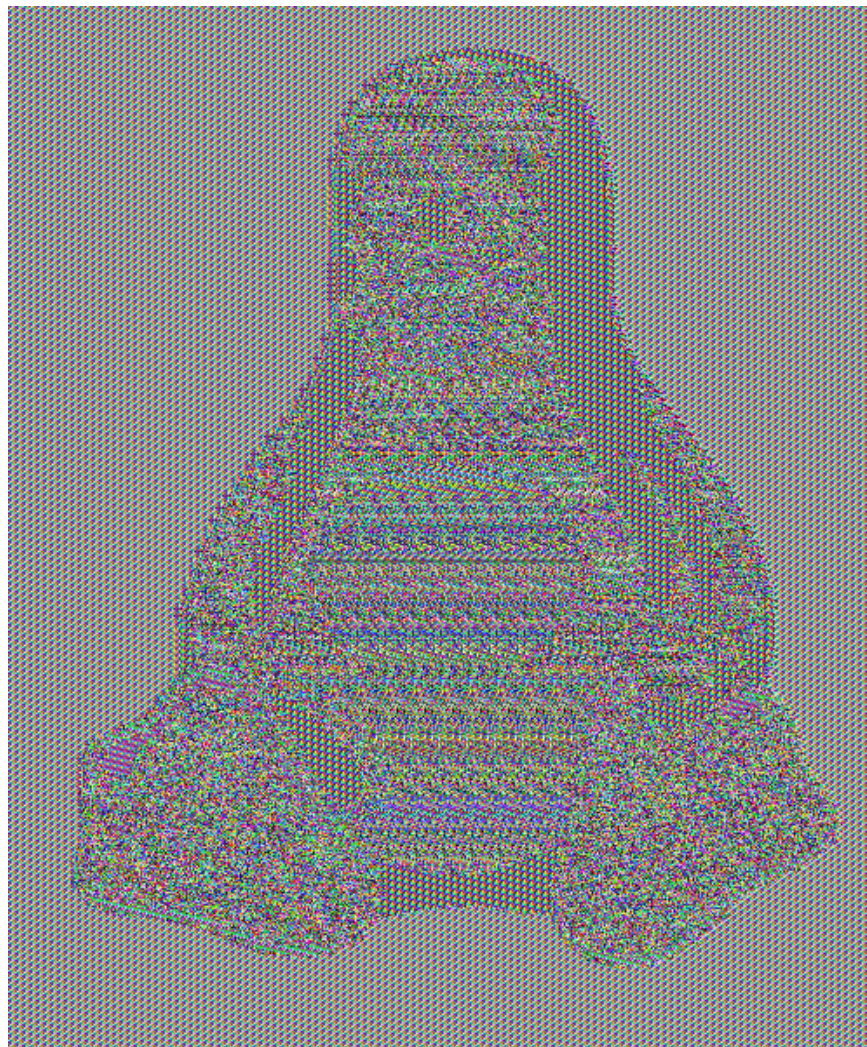
Signed 8 bit:	-6	Signed 32 bit:	-88817685	Hexadecimal:	FA B4 BF EB	x
Unsigned 8 bit:	250	Unsigned 32 bit:	4206149611	Decimal:	250 180 191 235	
Signed 16 bit:	-1356	Float 32 bit:	-4,69253E+35	Octal:	372 264 277 353	
Unsigned 16 bit:	64180	Float 64 bit:	-1,20527836817238E+283	Binary:	1111010 10110100 10	
<input type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal		ASCII Text:	????	

Offset: 54 / 900079 Selection: None INS



# Cifratura di Immagini Bitmap

tux\_aes\_128\_ecb.bmp





# Cifratura di Immagini Bitmap

tux\_aes\_128\_cbc.bmp



# Cifratura di Immagini Bitmap

- **Esercizio 4.2** Visualizzare le immagini cifrate, utilizzando un qualsiasi software per la visualizzazione di immagini
  - A partire dalle immagini cifrate, possono essere ricavate informazioni utili sull'immagine originale? Si argomentino le proprie osservazioni
- **Esercizio 4.3** Si confrontino e discutano le due immagini prodotte cifrando il file originale (tux.bmp) mediante le modalità ECB e CBC, rispettivamente
  - Quale modalità permette di nascondere meglio il contenuto dell'immagine? Quale garantisce maggiore sicurezza e perché?

# Esempio: Testo Cifrato Corrotto

Si consideri il seguente file

```
Corso di Sicurezza Su Reti  
Corso di Sicurezza Su Reti  
Corso di Sicurezza Su Reti  
Corso di Sicurezza Su Reti
```

`Originale.txt`

Cifriamo il file `Originale.txt` mediante AES a 128 bit in modalità ECB

```
openssl enc -aes-128-ecb -e -in Originale.txt -out  
corrotto_ecb.bin
```

# Esempio: Testo Cifrato Corrotto

Apriamo mediante Bless il file `corrotto_ecb.bin`

corrotto\_ecb.bin ✕

00000000	53	61	6C	74	65	64	5F	5F	CB	EE	49	9B	1F	72	6D	13	26	4A	Salted___I..rm.&J
00000012	42	3C	CC	50	81	26	C4	D5	EC	D7	93	41	0E	EE	CB	F2	13	39	B<.P.&.....A.....9
00000024	C9	51	2D	7E	0A	8E	A3	39	4C	7B	B5	3E	AA	8D	7C	B8	1B	38	.Q-~...9L{.>.. ..8
00000036	DE	39	66	B8	5D	09	2D	FE	1C	35	86	CA	3E	3A	C0	07	BC	78	.9f.]..-..5..>:...x
00000048	9C	C2	51	CE	DF	48	18	CC	98	88	F2	DB	E5	59	BB	40	D2	C0	..Q..H.....Y.@..
0000005a	E4	4D	27	45	C1	8E	A2	B1	FB	83	91	DD	35	79	37	95	11	67	.M'E.....5y7..g
0000006c	82	78	99	38	48	AB	54	A3	DF	D5	EB	CC	E8	E1	14	28	6D	66	.x.8H.T.....(mf
0000007e	07	94																	..

Signed 8 bit:       Signed 32 bit:       Hexadecimal:  ✕

Unsigned 8 bit:       Unsigned 32 bit:       Decimal:

Signed 16 bit:       Float 32 bit:       Octal:

Unsigned 16 bit:       Float 64 bit:       Binary:

Show little endian decoding       Show unsigned as hexadecimal      ASCII Text:



# Esempio: Testo Cifrato Corrotto

Consideriamo il valore di un dato byte, ad es. il valore del byte in posizione 48

➤ Valore 3E

corrotto_ecb.bin ✕		
00000000	53 61 6C 74 65 64 5F 5F CB EE 49 9B 1F 72 6D 13 26 4A	Salted___I..rm.&J
00000012	42 3C CC 50 81 26 C4 D5 EC D7 93 41 0E EE CB F2 13 39	B<.P.&.....A.....9
00000024	C9 51 2D 7E 0A 8E A3 39 4C 7B B5 3E AA 8D 7C B8 1B 38	.Q-~...9L{.>.. ..8
00000036	DE 39 66 B8 5D 09 2D FE 1C 35 86 CA 3E 3A C0 07 BC 78	.9f.]]-..5..>:...x
00000048	9C C2 51 CE DF 48 18 CC 98 88 F2 DB E5 59 BB 40 D2 C0	..Q..H.....Y.@..
0000005a	E4 4D 27 45 C1 8E A2 B1 FB 83 91 DD 35 79 37 95 11 67	.M'E.....5y7..g
0000006c	82 78 99 38 48 AB 54 A3 DF D5 EB CC E8 E1 14 28 6D 66	.x.8H.T.....(mf
0000007e	07 94	..

Signed 8 bit:	<input type="text" value="62"/>	Signed 32 bit:	<input type="text" value="1051364732"/>	Hexadecimal:	<input type="text" value="3E AA 8D 7C"/>	✕
Unsigned 8 bit:	<input type="text" value="62"/>	Unsigned 32 bit:	<input type="text" value="1051364732"/>	Decimal:	<input type="text" value="062 170 141 124"/>	
Signed 16 bit:	<input type="text" value="16042"/>	Float 32 bit:	<input type="text" value="0,3331107"/>	Octal:	<input type="text" value="076 252 215 174"/>	
Unsigned 16 bit:	<input type="text" value="16042"/>	Float 64 bit:	<input type="text" value="7,91331658167736E-07"/>	Binary:	<input type="text" value="00111110 10101010 10101010"/>	
<input type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal		ASCII Text:	<input data-bbox="1601 1380 1758 1420" type="text" value="&gt;??"/>	

# Esempio: Testo Cifrato Corrotto

Cambiamo il valore di un bit relativo a tale byte (da 3E a 2E) e salviamo il file **corrotto\_ecb.bin**

The screenshot shows a hex editor window titled "corrotto\_ecb.bin". The hex data is displayed in columns, with the corresponding ASCII text on the right. The byte 2E is highlighted in a red box. Below the hex view is a conversion panel with various data representations for the selected byte.

Signed 8 bit:	46	Signed 32 bit:	782929276	Hexadecimal:	2E AA 8D 7C
Unsigned 8 bit:	46	Unsigned 32 bit:	782929276	Decimal:	046 170 141 124
Signed 16 bit:	11946	Float 32 bit:	7,755838E-11	Octal:	056 252 215 174
Unsigned 16 bit:	11946	Float 64 bit:	6,83407358291895E-84	Binary:	00101110 10101010 10101010 10101010
<input type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal		ASCII Text:	.??

# Esempio: Testo Cifrato Corrotto

Decifriamo il file `corrotto_ecb.bin` ed esaminiamo il risultato della decifratura, sia mediante un editor di testi che mediante Bless

```
openssl enc -aes-128-ecb -d -in corrotto_ecb.bin -out  
corrotto_ecb.txt
```



```
Corso di Sicurez; > ~T`&◇rΔÜ"0Ûç-◇o di Sicurezza Su Reti  
Corso di Sicurezza Su Reti  
Corso di Sicurezza Su Reti
```

`corrotto_ecb.txt`



# Esempio: Testo Cifrato Corrotto

In modo analogo, è possibile valutare le altre modalità operative

Modalità CBC

```
Corso di Sicurezzaæ »»s6 “~î  
1, ®o di Sicurezza Pu Reti  
Corso di Sicurezza Su Reti  
Corso di Sicurezza Su Reti
```

corrotto\_cbc.txt

Modalità CFB

```
Corso di Sicurezza Su Reti  
Corr:ÔíBø' ~®ü7=0ô' "ju Reti  
Corso di Sicurezza Su Reti  
Corso di Sicurezza Su Reti
```

corrotto\_cfb.txt

Modalità OFB

```
Corso di Sicurezza Su Reti  
Corpo di Sicurezza Su Reti  
Corso di Sicurezza Su Reti  
Corso di Sicurezza Su Reti
```

corrotto\_ofb.txt

# Testi Cifrati Corrotti

## ➤ Eseguire i seguenti passi

1. Creare un file testuale che sia grande almeno 64 byte
2. Cifrare il file usando rispettivamente AES-128 in modalità CBC, CFB, ECB ed OFB
3. Simulare la corruzione di un bit nel trentesimo byte del file cifrato, cambiando il valore di tale bit mediante Bless
4. Decifrare il file cifrato corrotto, usando la password corretta

# Testi Cifrati Corrotti

## ➤ **Esercizio 5** Rispondere alle seguenti domande

1. Quante informazioni è possibile recuperare decifrando il file corrotto, se la modalità di cifratura è rispettivamente ECB, CBC, CFB od OFB?
  - Si provi a rispondere a tale domanda prima di svolgere l'esercizio in OpenSSL
  - Si verifichi successivamente, mediante OpenSSL, se la risposta è corretta o sbagliata
2. Quanti byte risultano essere danneggiati nel file decifrato a causa della modifica di un bit nel file cifrato?
3. In quale blocco (o in quali blocchi) si trovano i byte danneggiati?
4. Spiegare perché il numero di byte danneggiati risulta essere diverso quando vengono usate diverse modalità operative per la cifratura ed in che modo, ciascuna modalità operativa, influenza la propagazione dell'errore

**Argomentare e motivare ciascuna risposta**

# Padding

- Per i cifrari a blocchi, quando la dimensione del testo in chiaro non è multipla della dimensione del blocco, può essere necessario il padding
- **Esercizio 6.1** Quali sono le modalità operative che utilizzano il padding? Per le modalità operative che non lo utilizzano, spiegare il perché
- **Esercizio 6.2** Usare le modalità ECB, CBC, CFB, ed OFB per cifrare un dato file (si può scegliere un qualsiasi cifrario a blocchi)
- **Esercizio 6.3** Confrontare la dimensione del file originario rispetto a quella dei file derivanti dalla cifratura con ciascuna modalità operativa

# Padding

- **Esercizio 6.4** Si considerino due differenti file: plaintext1.txt e plaintext2.txt (aventi dimensioni diverse)
  - Cifrare plaintext1.txt e plaintext2.txt specificando chiave ed IV, con un cifrario ed una modalità operativa a scelta. Siano ciphertext1 e ciphertext2 i file derivanti dalle cifrature
  - Decifrare ciphertext1 e ciphertext2 utilizzando l'opzione -nopad. Siano plaintext1nopad.txt e plaintext2nopad.txt i file derivanti dalle decifrature
  - Confrontare mediante un editor di testi il contenuto del file plaintext1.txt rispetto a quello del file plaintext1nopad.txt, così come quello di plaintext2.txt rispetto a plaintext2nopad.txt
  - Si ripetano le suddette operazioni usando Bless
- **Esercizio 6.5** Discutere e motivare i risultati ottenuti al passo precedente
- **Esercizio 6.6** Cosa si può notare osservando le dimensioni dei file?
- **Esercizio 6.7** Ripetere l'esercizio 6.4 utilizzando altre modalità operative, discutendo e motivando i risultati ottenuti

# Prestazioni della cifratura

- Si consideri il seguente comando, che opera su un arbitrario **file**

```
openssl enc ciphertype -e -in file -out cipher.bin -K  
00112233445566778889aabbccddeeff -iv 0102030405060708
```

- **Esercizio 7.1** Sostituire **ciphertype** con uno specifico cifrario fornito da OpenSSL, come `-aes-128-cbc`, `-aes-128-cfb`, etc.
- **Esercizio 7.2** Si provino almeno 3 diversi cifrari e tre diverse modalità operative
- **Esercizio 7.3** Si valutino, mediante il comando **time**, i tempi di esecuzione delle suddette procedure
  - Qual è la più veloce?
  - Qual è la più lenta?

Argomentare e motivare ciascuna risposta

# Prestazioni della cifratura

- **Esercizio 7.4** Cifrare, mediante 2 cifrari a scelta e 4 modalità operative (ECB, CBC, CFB e OFB), un file di dimensioni maggiori di 1GB
- **Esercizio 7.5** Per ciascuna delle suddette configurazioni, mediante il comando **time**, valutare i tempi richiesti e riportarli in Tabella 1
- **Esercizio 7.6** Per ciascuna delle suddette configurazioni, effettuare una stima dei tempi richiesti per cifrare il contenuto dell'intero dell'hard disk e riportarli in Tabella 2

Argomentare e motivare ciascuna risposta

# Prestazioni della cifratura

(File di Grandi Dimensioni)

Cifrario	Cifrario 1	Cifrario 2
Modalità Operativa 1		
Modalità Operativa 2		
Modalità Operativa 3		
Modalità Operativa 4		

Tabella 1

Dimensione File:



# Prestazioni della cifratura

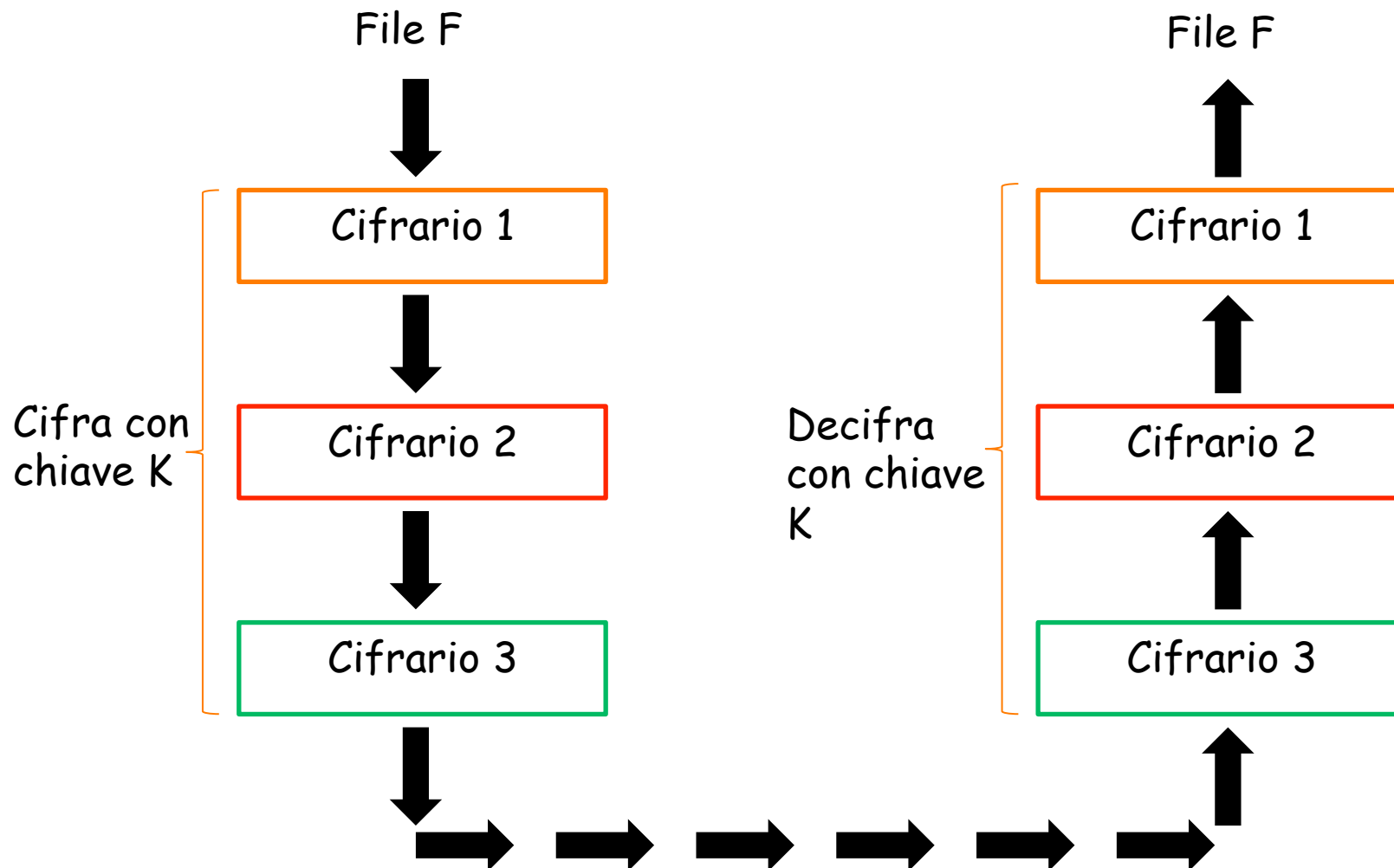
(Contenuto dell'Interno Hard Disk)

Cifrario	Cifrario 1	Cifrario 2
Modalità Operativa 1		
Modalità Operativa 2		
Modalità Operativa 3		
Modalità Operativa 4		

Tabella 2

Dimensione hard disk:

# Cifratura a Cascata



# Cifratura a Cascata

- **Esercizio 8.1** Si realizzi il pattern di cifratura mostrato nell'immagine precedente.  
In dettaglio
  - Il file F deve essere grande almeno 1GB
  - Si utilizzino 3 cifrari differenti
    - Per ciascun cifrario usare una diversa lunghezza della chiave ed una diversa modalità operativa

# Cifratura a Cascata

- **Esercizio 8.2** Si ripeta l'esercizio precedente considerando diverse combinazioni di cifrario/lunghezza chiave/modalità operativa
  - Quale combinazione richiede meno tempo per la cifratura?
  - Quale combinazione richiede più tempo per la decifratura?
  - Quale combinazione richiede complessivamente meno tempo per cifratura e decifratura?
  - Quale combinazione richiede complessivamente più tempo per cifratura e decifratura?
  - Quale combinazione permette di ottenere il ciphertext con la grandezza maggiore?
  - Quale combinazione permette di ottenere il ciphertext con la grandezza minore?

**Argomentare e motivare ciascuna risposta**