OPENSSL

Cristina Alcaraz
Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga
alcaraz@lcc.uma.es

PRÁCTICA - OBJETIVOS

Actividades a realizar en la práctica

- Cifrar
- Descifrar
- Estudiar el efecto avalancha
  - Modificando el texto
  - Modificando la clave
- Cifrar/descifrar derivando el modo de operación
- Producir efecto avalancha mediante un mecanismo HASH propio (doHASH)

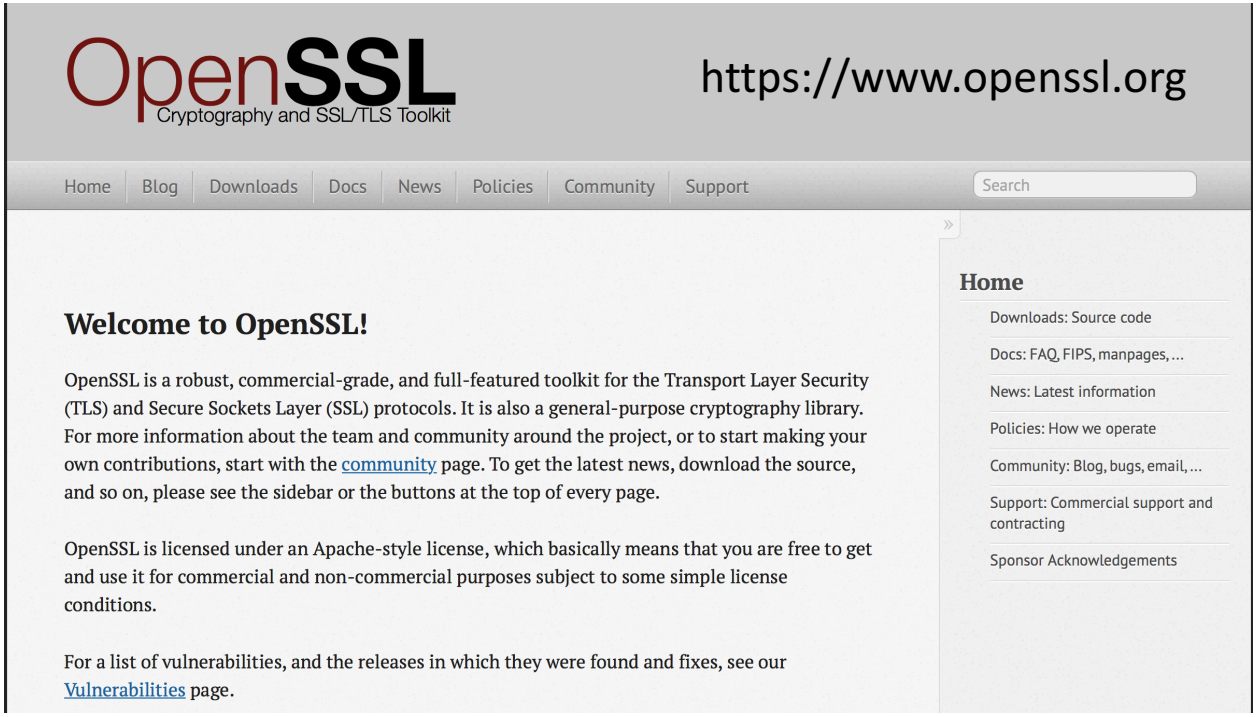
OPENSSL

Objetivos

- **OpenSSL**
  - Características
  - Comandos en openSSL
- **Criptografía simétrica**
  - Características y conceptos (salt, padding)
  - Comandos en openSSL
  - Ejemplos concretos en openSSL
- **Efecto avalancha**
  - Pasos a realizar y comandos en openSSL
  - Ejemplos concretos en openSSL
- **doHASH**
  - Objetivos para los ejercicios

OpenSSL

- OpenSSL es una herramienta open source que permite la implementación fácil de protocolos y algoritmos SSL (Secure Socket Layer) y TLS Transport Layer Security)
- OpenSSL incluye una librería criptográfica de propósito general y mecanismos para ser aplicados en línea de comandos
  - Los comandos openSSL son genéricos y válidos para cualquier plataforma y sistema operativo



OpenSSL

- Verificar la versión:
  - **$ openssl version**
  - **$ openssl version** –a (incluye más información)

- Verificar el benchmarking del sistema cuando éste opera con algoritmos criptográficos
  - **$openssl speed**
  - **$openssl speed** aes
  - **$openssl speed** rsa
  - …

```
To get the most accurate results, try to run this
program when this computer is idle.
Doing md2 for 3s on 16 size blocks: 508376 md2's in 3.00s
Doing md2 for 3s on 64 size blocks: 279644 md2's in 3.00s
Doing md2 for 3s on 256 size blocks: 86804 md2's in 2.99s
Doing md2 for 3s on 1024 size blocks: 22843 md2's in 2.99s
Doing md2 for 3s on 8192 size blocks: 3075 md2's in 3.00s
Doing mdc2 for 3s on 16 size blocks: 3344096 mdc2's in 3.00s
Doing mdc2 for 3s on 64 size blocks: 877061 mdc2's in 2.99s
Doing mdc2 for 3s on 256 size blocks: 225828 mdc2's in 3.00s
Doing mdc2 for 3s on 1024 size blocks: 58922 mdc2's in 3.00s
Doing mdc2 for 3s on 8192 size blocks: 6859 mdc2's in 2.96s
Doing md4 for 3s on 16 size blocks: 11885699 md4's in 2.99s
Doing md4 for 3s on 64 size blocks: 9801213 md4's in 3.00s
Doing md4 for 3s on 256 size blocks: 5849693 md4's in 3.00s
Doing md4 for 3s on 1024 size blocks: 2211701 md4's in 3.00s
Doing md4 for 3s on 8192 size blocks: 328291 md4's in 3.00s
Doing md5 for 3s on 16 size blocks: 9606758 md5's in 3.00s
Doing md5 for 3s on 64 size blocks: 7007445 md5's in 3.00s
Doing md5 for 3s on 256 size blocks: 3510396 md5's in 2.98s
Doing md5 for 3s on 1024 size blocks: 1247455 md5's in 2.99s
Doing md5 for 3s on 8192 size blocks: 178001 md5's in 2.99s
Doing hmac(md5) for 3s on 16 size blocks: 9147567 hmac(md5)'s in 2.99s
Doing hmac(md5) for 3s on 64 size blocks: 6302226 hmac(md5)'s in 2.99s
Doing hmac(md5) for 3s on 256 size blocks: 3717630 hmac(md5)'s in 2.99s
Doing hmac(md5) for 3s on 1024 size blocks: 1301928 hmac(md5)'s in 3.00s
Doing hmac(md5) for 3s on 8192 size blocks: 168181 hmac(md5)'s in 2.99s
Doing sha1 for 3s on 16 size blocks: 10060098 sha1's in 3.00s
Doing sha1 for 3s on 64 size blocks: 6519922 sha1's in 2.99s
Doing sha1 for 3s on 256 size blocks: 3800726 sha1's in 2.99s
Doing sha1 for 3s on 1024 size blocks: 1299924 sha1's in 3.00s
Doing sha1 for 3s on 8192 size blocks: 205039 sha1's in 3.00s
```

OpenSSL - ayuda

- Existen ayudas para identificar los comandos que son estándar y aplicables al entorno que se usa
  - **$ openssl help**
  - **$ openssl list-standard-commands**
  - **$ openssl list-message-digest-commands**
  - **$ openssl list-cipher-commands**

OpenSSL - ayuda

$ openssl help

```
Standard commands
asn1parse      ca             ciphers        crl            crl2pkcs7
dgst           dh             dhparam        dsa            dsaparam
ec             ecparam        enc            engine         errstr
gendh          gendsa         genrsa         nseq           ocsp
passwd         pkcs12         pkcs7          pkcs8          prime
rand           req            rsa            rsautl         s_client
s_server       s_time         sess_id        smime          speed
spkac          verify         version        x509

Message Digest commands (see the `dgst' command for more details)
md2            md4            md5            mdc2           rmd160
sha            sha1

Cipher commands (see the `enc' command for more details)
aes-128-cbc    aes-128-ecb    aes-192-cbc    aes-192-ecb    aes-256-cbc
aes-256-ecb    base64         bf             bf-cbc         bf-cfb
bf-ecb         bf-ofb         cast           cast-cbc       cast5-cbc
cast5-cfb      cast5-ecb      cast5-ofb      des            des-cbc
des-cfb        des-ecb        des-ede        des-ede-cbc    des-ede-cfb
des-ede-ofb    des-ede3       des-ede3-cbc   des-ede3-cfb   des-ede3-ofb
des-ofb        des3           desx           rc2            rc2-40-cbc
rc2-64-cbc     rc2-cbc        rc2-cfb        rc2-ecb        rc2-ofb
rc4            rc4-40         seed           seed-cbc       seed-cfb
seed-ecb       seed-ofb
```

OpenSSL - ayuda

$ openssl list-standard-commands

```
asn1parse
ca
ciphers
crl
crl2pkcs7
dgst
dh
dhparam
dsa
dsaparam
ec
ecparam
enc
engine
errstr
gendh
gendsa
genrsa
nseq
ocsp
passwd
pkcs12
pkcs7
pkcs8
prime
rand
req
rsa
rsautl
s_client
s_server
s_time
sess_id
smime
speed
spkac
verify
version
x509
```

$ openssl list-message-digest-commands

```
md2
md4
md5
mdc2
rmd160
sha
sha1
```

IMPORTANTE:

- 3DES:
  - TAM_KEY_BITS = 168;
  - TAM_BLOQUE_BYTES = 24;
  - Equivalente a **des-ede3**
- 2DES:
  - TAM_KEY_BITS = 112;
  - TAM_BLOQUE_BYTES = 14;
  - equivalente a **des-ede**

$ openssl list-cipher-commands

```
aes-128-cbc
aes-128-ecb
aes-192-cbc
aes-192-ecb
aes-256-cbc
aes-256-ecb
base64
bf
bf-cbc
bf-cfb
bf-ecb
bf-ofb
cast
cast-cbc
cast5-cbc
cast5-cfb
cast5-ecb
cast5-ofb
des
des-cbc
des-cfb
des-ecb
des-ede
des-ede-cbc
des-ede-cfb
des-ede-ofb
des-ede3
des-ede3-cbc
des-ede3-cfb
des-ede3-ofb
des-ofb
des3
desx
rc2
rc2-40-cbc
rc2-64-cbc
rc2-cbc
rc2-cfb
rc2-ecb
rc2-ofb
rc4
rc4-40
seed
seed-cbc
seed-cfb
seed-ecb
seed-ofb
```

OpenSSL - ayuda

- Es también posible obtener información concreta de un determinado comando usando la opción -h:
  - Ejemplo:
    - **$openssl** enc -h

```
options are
-in <file>     input file
-out <file>    output file
-pass <arg>    pass phrase source
-e             encrypt
-d             decrypt
-a/-base64     base64 encode/decode, depending on encryption flag
-k             passphrase is the next argument
-kfile         passphrase is the first line of the file argument
-md            the next argument is the md to use to create a key
                 from a passphrase.  One of md2, md5, sha or sha1
-K/-iv         key/iv in hex is the next argument
-[pP]          print the iv/key (then exit if -P)
-bufsize <n>   buffer size
-engine e      use engine e, possibly a hardware device.
Cipher Types
-aes-128-cbc               -aes-128-cfb               -aes-128-cfb1
-aes-128-cfb8              -aes-128-ecb               -aes-128-ofb
-aes-192-cbc               -aes-192-cfb               -aes-192-cfb1
-aes-192-cfb8              -aes-192-ecb               -aes-192-ofb
-aes-256-cbc               -aes-256-cfb               -aes-256-cfb1
-aes-256-cfb8              -aes-256-ecb               -aes-256-ofb
-aes128                    -aes192                    -aes256
-bf                        -bf-cbc                    -bf-cfb
-bf-ecb                    -bf-ofb                    -blowfish
-cast                      -cast-cbc                  -cast5-cbc
-cast5-cfb                 -cast5-ecb                 -cast5-ofb
-des                       -des-cbc                   -des-cfb
-des-cfb1                  -des-cfb8                  -des-ecb
-des-ede                   -des-ede-cbc               -des-ede-cfb
-des-ede-ofb               -des-ede3                  -des-ede3-cbc
-des-ede3-cfb              -des-ede3-cfb1             -des-ede3-cfb8
-des-ede3-ofb              -des-ofb                   -des3
-desx                      -desx-cbc                  -rc2
-rc2-40-cbc                -rc2-64-cbc                -rc2-cbc
-rc2-cfb                   -rc2-ecb                   -rc2-ofb
-rc4                       -rc4-40                    -seed
-seed-cbc                  -seed-cfb                  -seed-ecb
-seed-ofb
```

OpenSSL – algoritmos de cifrado

- El comando **ciphers** permite saber qué lista de algoritmos y modos de cifrado están disponibles

  - **$openssl ciphers** -v
    - ➔ lista todos los algoritmos disponibles en el sistema
  - **$openssl ciphers** –v -tls1
    - ➔ lista los algoritmos específicos de TLS1
  - **$openssl ciphers** –v ‘HIGH’
    - ➔ lista los algoritmos que trabajan con claves de más de 128 bits
  - **$openssl ciphers** – v ‘AES+HIGH’
    - ➔ lista aquellos modos de cifrado trabajando con AES y con claves superior a 128 bits

OpenSSL – algoritmos de cifrado

$ openssl ciphers -v

```
DHE-RSA-AES256-SHA      SSLv3 Kx=DH       Au=RSA  Enc=AES(256)  Mac=SHA1
DHE-DSS-AES256-SHA      SSLv3 Kx=DH       Au=DSS  Enc=AES(256)  Mac=SHA1
AES256-SHA              SSLv3 Kx=RSA      Au=RSA  Enc=AES(256)  Mac=SHA1
EDH-RSA-DES-CBC3-SHA    SSLv3 Kx=DH       Au=RSA  Enc=3DES(168) Mac=SHA1
EDH-DSS-DES-CBC3-SHA    SSLv3 Kx=DH       Au=DSS  Enc=3DES(168) Mac=SHA1
DES-CBC3-SHA            SSLv3 Kx=RSA      Au=RSA  Enc=3DES(168) Mac=SHA1
DES-CBC3-MD5            SSLv2 Kx=RSA      Au=RSA  Enc=3DES(168) Mac=MD5
DHE-RSA-AES128-SHA      SSLv3 Kx=DH       Au=RSA  Enc=AES(128)  Mac=SHA1
DHE-DSS-AES128-SHA      SSLv3 Kx=DH       Au=DSS  Enc=AES(128)  Mac=SHA1
AES128-SHA              SSLv3 Kx=RSA      Au=RSA  Enc=AES(128)  Mac=SHA1
DHE-RSA-SEED-SHA        SSLv3 Kx=DH       Au=RSA  Enc=SEED(128) Mac=SHA1
DHE-DSS-SEED-SHA        SSLv3 Kx=DH       Au=DSS  Enc=SEED(128) Mac=SHA1
SEED-SHA                SSLv3 Kx=RSA      Au=RSA  Enc=SEED(128) Mac=SHA1
RC2-CBC-MD5             SSLv2 Kx=RSA      Au=RSA  Enc=RC2(128)  Mac=MD5
RC4-SHA                 SSLv3 Kx=RSA      Au=RSA  Enc=RC4(128)  Mac=SHA1
RC4-MD5                 SSLv3 Kx=RSA      Au=RSA  Enc=RC4(128)  Mac=MD5
RC4-MD5                 SSLv2 Kx=RSA      Au=RSA  Enc=RC4(128)  Mac=MD5
EDH-RSA-DES-CBC-SHA     SSLv3 Kx=DH       Au=RSA  Enc=DES(56)   Mac=SHA1
EDH-DSS-DES-CBC-SHA     SSLv3 Kx=DH       Au=DSS  Enc=DES(56)   Mac=SHA1
DES-CBC-SHA             SSLv3 Kx=RSA      Au=RSA  Enc=DES(56)   Mac=SHA1
DES-CBC-MD5             SSLv2 Kx=RSA      Au=RSA  Enc=DES(56)   Mac=MD5
EXP-EDH-RSA-DES-CBC-SHA SSLv3 Kx=DH(512)  Au=RSA  Enc=DES(40)   Mac=SHA1 export
EXP-EDH-DSS-DES-CBC-SHA SSLv3 Kx=DH(512)  Au=DSS  Enc=DES(40)   Mac=SHA1 export
EXP-DES-CBC-SHA         SSLv3 Kx=RSA(512) Au=RSA  Enc=DES(40)   Mac=SHA1 export
EXP-RC2-CBC-MD5         SSLv3 Kx=RSA(512) Au=RSA  Enc=RC2(40)   Mac=MD5  export
EXP-RC2-CBC-MD5         SSLv2 Kx=RSA(512) Au=RSA  Enc=RC2(40)   Mac=MD5  export
EXP-RC4-MD5             SSLv3 Kx=RSA(512) Au=RSA  Enc=RC4(40)   Mac=MD5  export
EXP-RC4-MD5             SSLv2 Kx=RSA(512) Au=RSA  Enc=RC4(40)   Mac=MD5  export
```

OpenSSL – algoritmos de cifrado

$ openssl ciphers –v ‘HIGH’

```
ADH-AES256-SHA:DHE-RSA-AES256-SHA:DHE-DSS-AES256-SHA:AES256-SHA:ADH-AES128-SHA:DHE-RSA-AES128-SHA:DHE-DS
S-AES128-SHA:AES128-SHA:ADH-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:EDH-DSS-DES-CBC3-SHA:DES-CBC3-SHA:DES-CBC3
-MD5
```

$ openssl ciphers -v ‘AES+HIGH’

```
ADH-AES256-SHA:DHE-RSA-AES256-SHA:DHE-DSS-AES256-SHA:AES256-SHA:ADH-AES128-SHA:DHE-RSA-AES128-SHA:DHE-DS
S-AES128-SHA:AES128-SHA
```

CRIPTOGRAFÍA SIMÉTRICA

Cifrar mensajes en openSSL

- Formato general
  - **$ openssl enc** -modo de cifrado **-in** file.txt
  - **$ openssl enc** -modo de cifrado **-in** file.txt **-out** file.txt.enc
  - **$ echo** “texto” **| openssl enc** -modo de cifrado
  - Ejemplos:
    - openssl enc -aes-256-cbc -in input.txt -out output.txt
    - openssl enc -base64 -in file.txt
    - $ echo “hola” | openssl enc –base64
- Cifrando texto en línea de comandos:
  - Es importante destacar que el comando echo añade una nueva línea a la cadena, sin que se vea esa línea en pantalla. Para quitarla se usa la opción –n
    - $ echo **-n** ”hola" | openssl enc -base64

Cifrar mensajes en openSSL

- Otras opciones de cifrado a considerar:
  - **-e** especifica la acción de cifrado (es la de por defecto)
  - **-d** representa la acción de descifrado
  - **-nosalt** elimina la aleatoriedad que el comando enc incluye en el texto
  - **-iv** añade un específico vector de inicialización
  - **-K** incluye una clave específica Y ES DIFERENTE A **-k**
  - **-nopad** para desactivar el padding estándar por defecto (PKCS#5)
  - **-a** / **-base64** conversión a formato base64
    - Sin embargo, esta opción añade una línea cada 64 caracteres. Para quitar esta acción, se debe usar además las opciones **–A**
    - Ej: openssl enc -aes-256-cbc **-a -A** -in input.txt -out output.txt

Ej: openssl enc -aes-256-cbc -nosalt **-a -A** -in input.txt -out output.txt -iv 89FB9D7A7B191B9FC8A529467E794E04 -K 987654321

Descifrar mensajes en openSSL

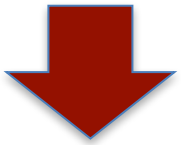
- Simplemente añadir la opción **–d**:

```
Ej: openssl enc -d -aes-256-cbc -nosalt -a -in output.txt -out salida.txt -iv
89FB9D7A7B191B9FC8A529467E794E04 -K 987654321
```

Generar claves y IV en openSSL

- Se usa la herramienta **enc** haciendo uso de las siguientes opciones:
  - **-p** para imprimir por pantalla la clave, el vector de inicialización y el valor salt, y posteriormente cifrar el mensaje
  - **-P** solo imprime por pantalla la clave, el vector de inicialización y el valor salt, sin cifrar el mensaje
  - **-k** corresponde con la "semilla" de entrada para generar la clave

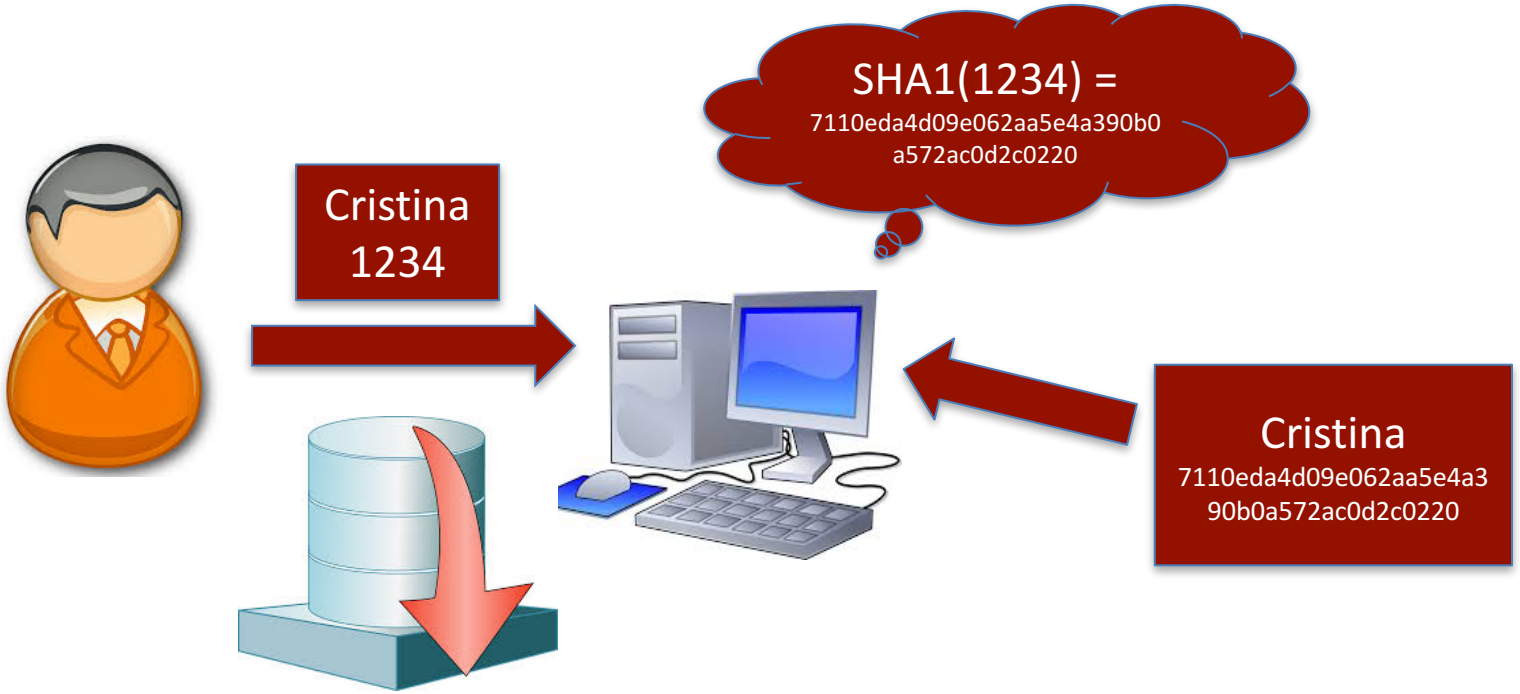
Ej: openssl enc -aes-256-cbc **–P –k** 1234



```
salt=D8F320EA11F0296E
key=F4C17D45FA5AC9BA9B3F433B4DC628E495B5A729EEC20892AECF1C1EDDAD8043
iv =5484F12C0DFDA95C647C2B0DCD305412
```

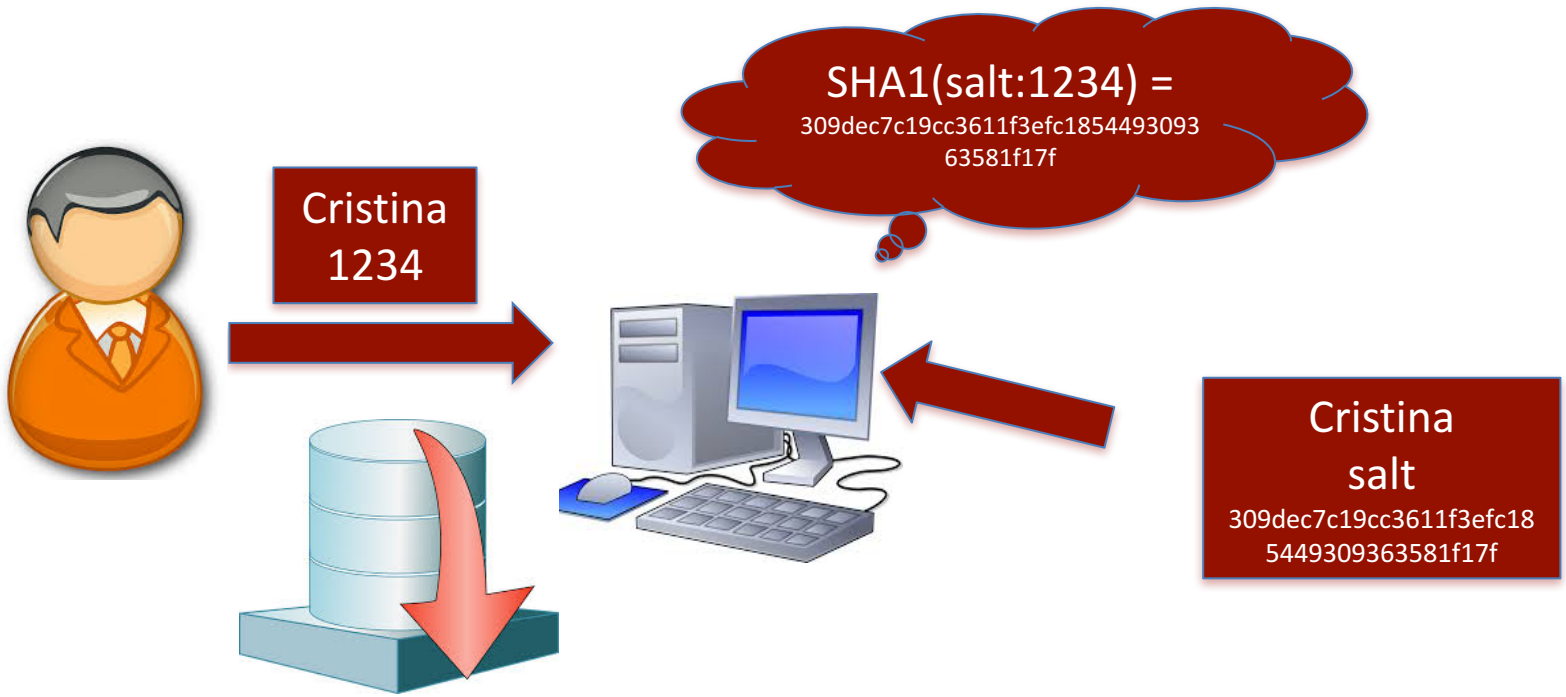
SALT: caso práctico - contraseñas con salt

- SALT no es más que una secuencia de valores aleatorios (de tamaño fijo) que se combina con el HASH para producir más aleatoriedad
- Las cuentas almacenadas en un disco duro de un sistema y protegidas con contraseñas, suelen tener asociado un HASH a dichas contraseñas. **¡Nunca se debe almacenar las contraseñas en claro!**
- Cuando el usuario quiere entrar al equipo se le pide la contraseña, se hace el hash y se compara con el hash almacenado



SALT: contraseñas con salt

- Si alguien roba el fichero con los HASH puede hacer fácilmente un ataque de diccionario
- Para dificultar los ataques de diccionario se usa "**Salt**" → valores aleatorios que se asocia al HASH



Padding

- Los cifrados en bloque están diseñados para trabajar con mensajes compuestos de bloques de un tamaño específico
  - Ejemplo: AES-128 trabaja con bloques de 128 bits (16 bytes)
  - Problema: Supongamos que usamos AES-128 (16 bytes),
  - ¿qué ocurre cuando queremos cifrar un mensaje que ocupa, por ejemplo, 20 bytes?
    - Tendremos un primer bloque de 16 bytes, y un segundo bloque de 4 bytes
    - El primer bloque lo podemos cifrar sin problemas
    - Al segundo bloque tenemos que añadirle algo al final ("**padding**")

Padding

- Esquemas de Padding:
  - ISO 10126: añadir bytes aleatorios, excepto el último, que indicará la longitud del padding
    - |12 63 12 65 E7 82 A7 C1|B7 02 9E 29 4E 8C 7B 05|
  - ISO/IEC 7816-4: añadir ceros, excepto el primero, que siempre tendrá el valor 80:
    - |12 63 12 65 e7 82 a7 c1|b7 02 9e 80 00 00 00 00|
  - Zero Padding: simplemente añadir ceros
    - |12 63 12 65 e7 82 a7 c1|b7 02 9e 00 00 00 00 00|
    - Problema: si el mensaje original acaba en alguna secuencia de ceros, no es posible determinar dónde empieza el padding
  - PKCS#5, PKCS#7: Si necesitamos N bytes de padding, usamos N veces el valor N
    - |12 63 12 65 e7 82 a7 c1|b7 02 9e 05 05 05 05 05|

Padding

- Un aspecto a tener en cuenta es que si se usa padding sobre un mensaje que no lo necesita, necesariamente se añadirá un bloque nuevo al final

  - Por ejemplo con PKCS#5, PKCS#7:

    |12 63 12 65 e7 82 a7 c1|08 08 08 08 08 08 08 08|

Conversiones hex-to-string/ string-to-hex

- Util_alumno.java

- Linux:
  - String to hex: **xxd -ps**
  - Hex to string: **xxd -r –p** (preferible)
  - Hex to string: **xxd –b** (opción 2)

```
Usage:
       xxd [options] [infile [outfile]]
    or
       xxd -r [-s [-]offset] [-c cols] [-ps] [infile [outfile]]
Options:
    -a          toggle autoskip: A single '*' replaces nul-lines. Default off.
    -b          binary digit dump (incompatible with -ps,-i,-r). Default hex.
    -c cols     format <cols> octets per line. Default 16 (-i: 12, -ps: 30).
    -E          show characters in EBCDIC. Default ASCII.
    -g          number of octets per group in normal output. Default 2.
    -h          print this summary.
    -i          output in C include file style.
    -l len      stop after <len> octets.
    -ps         output in postscript plain hexdump style.
    -r          reverse operation: convert (or patch) hexdump into binary.
    -r -s off   revert with <off> added to file positions found in hexdump.
    -s [+][-]seek  start at <seek> bytes abs. (or +: rel.) infile offset.
    -u          use upper case hex letters.
    -v          show version: "xxd V1.10 27oct98 by Juergen Weigert".
```

Ej:

- **cifrar**: echo -n “Hola” | openssl enc -bf-cbc -iv 89FB9D7A7B191B9FC8A529467E794E04 -K 1234… -nosalt | **xxd -ps (a hexadecimal)**

- **Descifrar**: echo -n bd9bfd3aeaf439f66fd7112c1ac083e260c7dc29c2589d28 | **xxd -r -p** | openssl enc -bf-cbc -iv 89FB9D7A7B191B9FC8A529467E794E04 -K 1234… -nosalt -d

Ejemplos

- **Ejemplo 1**:
  - Cifrado: dado un texto, cifrarlo y pasarlo a formato hexadecimal
    - echo -n Hoy he comido arroz | openssl enc -bf-cbc -iv 89FB9D7A7B191B9FC8A529467E794E04 -K 1234… -nosalt -nopad | **xxd –ps**
  - Descifrado: dado un texto, pasar su contenido a formato binario y descifrarlo
    - echo -n bd9bfd3aeaf439f66fd7112c1ac083e260c7dc29c2589d28 | **xxd -r -p** | openssl enc -bf-cbc -iv 89FB9D7A7B191B9FC8A529467E794E04 -K 1234… -nosalt -d –nopad
- **Ejemplo 2**:
  - Cifrado: dado un texto, cifrarlo, pasarlo a formato hexadecimal y guardarlo en un fichero
    - echo -n Hoy he comido arroz | openssl enc -bf-cbc -iv 89FB9D7A7B191B9FC8A529467E794E04 -K 1234… -nosalt -nopad | **xxd -ps** > file.txt
  - Descifrado: dado un fichero, pasarlo a formato string para descifrar su contenido
    - echo -n $(<file.txt) | **xxd -r -p** | openssl enc -bf-cbc -iv 89FB9D7A7B191B9FC8A529467E794E04 -K 1234… -nosalt -d -nopad > file2.txt

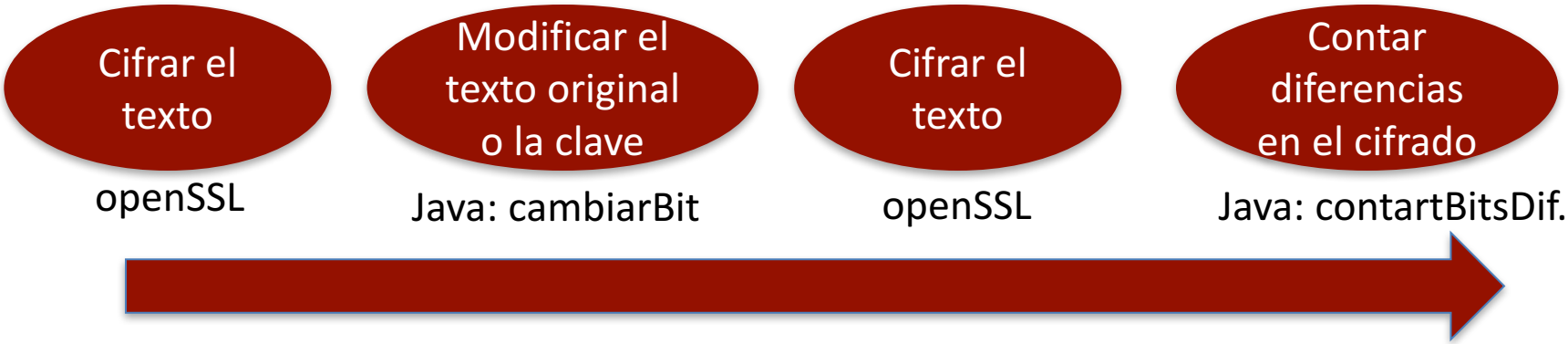
Ejemplos

- **Ejemplo 3**:
  - Cifrado: dado un texto en hexadecimal, pasarlo a string para procesarlo, cifrarlo y pasarlo de nuevo a formato hexadecimal
    - echo -n AECD7E62C357D5F6B4DDE17B66A0F5D0F01AC9329C67475C9C607C40E77A3DF5B1BC6B1C8E5D694C7A18B10D60642227 | **xxd -r -p** | openssl enc -aes-128-ecb -K 4F694C454D350BA29E56E4DDB8CA450C -nosalt -nopad | **xxd -ps -u**
  - Descifrado: dado un texto cifrado en hexadecimal, pasarlo a string para procesarlo, descifrarlo y pasarlo de nuevo a formato hexadecimal
    - echo -n 5FAD7510E13F01AEEBC31130ED445F0D6C7BD452D5B11EEBD103DCB1BB0EACB493D1A7550E473272EB134AC2CD974E23 | **xxd -r -p**| openssl enc -aes-128-ecb -K 4F694C454D350BA29E56E4DDB8CA450C -nosalt -d -nopad | **xxd -ps -u** > file2.txt

EFECTO AVALANCHA

Efecto avalancha

- Pasos a realizar:



- Herramientas:
  - El mecanismo de cifrado de openssl
    - Enc
  - La clase Util de Java conteniendo los métodos:
    - cambiarBit
    - hexStringToBytes
    - hexStringToBytes
    - bytesToHexString
    - contarBitsDiferentes
    - doHash

doHASH

El doHash se puede definir como el "**último bloque del criptograma**", que puede ser los 16 bytes últimos si se aplica AES-128, o los 8 bytes últimos si se aplica DES:

- AES: 128 bits – 16 bytes
- DES: 56 bits – 8 bytes

```
public static byte[] doHash(byte[] encryptedM, int TAM_BLOQUE_BYTES){

    return Arrays.copyOfRange(encryptedM, encryptedM.length-TAM_BLOQUE_BYTES,
    encryptedM.length);

}
```