



IES Zaidín Vergeles

OpenSSL



OpenSSL
Cryptography and SSL/TLS Toolkit

Jose Almirón López, 26 de Abril de 2024

Tabla de contenidos

Openssl con clave simétrica	2
Openssl con clave asimétrica	4
Generar claves	4
Generar claves con contraseña	5
Cifrado con claves asimétricas	5
Funciones Hash	6
MD5	6
SHA1	7
SHA256	7
SHA512	8
HMAC	8
Certificados digitales	9
Encriptación con certificado digital	12
Firma digitalmente con certificado digital	13

.....

Openssl con clave simétrica

OpenSSL es una herramienta de código abierto y una biblioteca de cifrado que proporciona implementaciones de varios protocolos de seguridad y algoritmos de cifrado. Una de las funcionalidades más comunes que ofrece OpenSSL es la capacidad de cifrar y descifrar datos utilizando claves simétricas.

La criptografía simétrica implica el uso de una única clave para cifrar y descifrar datos. En el contexto de OpenSSL, cuando se utiliza cifrado simétrico, una clave secreta se utiliza tanto para cifrar como para descifrar los datos.

Desde Linux, podemos emplear el comando **man openssl** para acceder a la documentación completa de esta herramienta. Ahora, procederemos a cifrar el archivo "**prueba.txt**" utilizando el algoritmo de cifrado simétrico **DES (Data Encryption Standard)** y guardarlo en un nuevo archivo.

openssl enc -des -in prueba.txt -out prueba.enc.des

```
(jose@kali)-[~]
$ echo "Hola Mundo" > prueba.txt

(jose@kali)-[~]
$ openssl enc -des -in prueba.txt -out prueba.enc.des
enter DES-CBC encryption password:
Verifying - enter DES-CBC encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

(jose@kali)-[~]
$ cat prueba.enc.des
Salted__8♦♦@b0♦♦♦♦♦♦♦♦♦♦(♦^♦♦
```

Para descifrar, emplearemos la misma línea de comando, pero añadiendo el argumento **-d** para indicar que queremos descifrar los datos.

```
Archivo Acciones Editar Vista Ayuda
(jose@kali)-[~]
$ openssl enc -des -d -in prueba.enc.des -out prueba2.txt
enter DES-CBC decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

(jose@kali)-[~]
$ cat prueba2.txt
Hola Mundo

(jose@kali)-[~]
$ diff prueba.txt prueba2.txt
```

A continuación, vamos a cifrar el archivo "**prueba.txt**" utilizando el algoritmo de cifrado simétrico **AES (Advanced Encryption Standard)** con una clave de 256 bits y en modo **CBC (Cipher Block Chaining)**. El archivo cifrado resultante se guardará en un nuevo archivo llamado "**prueba.enc.aes-256**".

```
openssl enc -aes-256-cbc -in prueba.txt -out prueba.enc.aes-256
```

```
Archivo Acciones Editar Vista Ayuda
(jose@kali)-[~]
$ openssl enc -aes-256-cbc -in prueba.txt -out prueba.enc.aes-256
enter AES-256-CBC encryption password:
Verifying - enter AES-256-CBC encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

(jose@kali)-[~]
$ cat prueba.enc.aes-256
(o \t1♦2
```

Para descifrar, como en el caso anterior, añadimos el argumento **-d**. Luego, con el comando **diff**, estamos verificando que los archivos sean idénticos.

```
openssl enc -aes-256-cbc -d -in prueba.enc.aes-256 -out prueba3.txt
```

```
(jose@kali)-[~]
$ openssl enc -aes-256-cbc -d -in prueba.enc.aes-256 -out prueba3.txt
enter AES-256-CBC decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

(jose@kali)-[~]
$ cat prueba3.txt
Hola Mundo

(jose@kali)-[~]
$ diff prueba.txt prueba3.txt
```

Estos son los archivos que hemos cifrado y descifrado en su totalidad.

```
(jose@kali)-[~/ssl]
$ ls -l
total 20
-rw-r--r-- 1 jose jose 11 abr 28 17:39 prueba2.txt
-rw-r--r-- 1 jose jose 11 abr 28 17:42 prueba3.txt
-rw-r--r-- 1 jose jose 32 abr 28 17:41 prueba.enc.aes-256
-rw-r--r-- 1 jose jose 32 abr 28 17:37 prueba.enc.des
-rw-r--r-- 1 jose jose 11 abr 28 17:37 prueba.txt
(jose@kali)-[~/ssl]
```

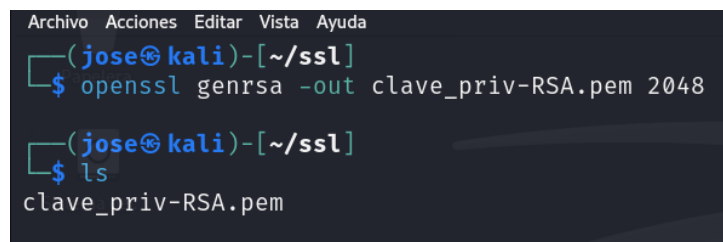
Openssl con clave asimétrica

Cuando hablamos de criptografía asimétrica, también conocida como criptografía de clave pública, utilizamos un par de claves: una pública y otra privada. La clave pública se comparte abiertamente, mientras que la clave privada se mantiene en secreto. Con el comando **openssl list -cipher-algorithms**, podemos visualizar la lista de algoritmos de cifrado que están disponibles en OpenSSL.

Generar claves

El primer paso será generar una clave privada. Hay dos formas de hacerlo: una no requerirá contraseña y simplemente generará la clave privada, mientras que la otra opción solicitará una contraseña adicional para proteger la clave privada.

openssl genrsa -out clave_priv-RSA.pem 2048




```
Archivo Acciones Editar Vista Ayuda
(jose@kali)-[~/ssl]
$ openssl genrsa -out clave_priv-RSA.pem 2048

(jose@kali)-[~/ssl]
$ ls
clave_priv-RSA.pem
```

Una vez que tengamos la clave privada, podremos generar la clave pública a partir de ella.

openssl rsa -in clave_priv-RSA.pem -pubout -out clave_pub-RSA.pem



```
Archivo Acciones Editar Vista Ayuda
(jose@kali)-[~/ssl]
$ openssl rsa -in clave_priv-RSA.pem -pubout -out clave_pub-RSA.pem
writing RSA key

(jose@kali)-[~/ssl]
$ ls
clave_priv-RSA.pem  clave_pub-RSA.pem
```

Generar claves con contraseña

Como mencionamos anteriormente, también podemos optar por crear las claves con una contraseña. En este caso, vamos a proceder a crear una clave privada protegida por una contraseña.

```
openssl genrsa -aes128 -out clave_priv-RSA_con_passw.pem 2048
```

```
(jose@kali)-[~/ssl]
└─$ openssl genrsa -aes128 -out clave_priv-RSA_con_passw.pem 2048
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:

(jose@kali)-[~/ssl]
└─$ ls
clave_priv-RSA_con_passw.pem  clave_priv-RSA.pem  clave_pub-RSA.pem
(jose@kali)-[~/ssl]
```

Ahora, a partir de esta clave privada protegida por contraseña, generamos una clave pública, la cual también estará protegida por contraseña.

```
openssl rsa -in clave_priv-RSA_con_passw.pem -pubout -out  
clave_pub-RSA_con_passw.pem
```

```
(jose@kali)-[~/ssl]
└─$ openssl rsa -in clave_priv-RSA_con_passw.pem -pubout -out clave_pub-RSA_con_passw.pem
Enter pass phrase for clave_priv-RSA_con_passw.pem:
writing RSA key

(jose@kali)-[~/ssl]
└─$ ls
clave_priv-RSA_con_passw.pem  clave_priv-RSA.pem  clave_pub-RSA_con_passw.pem  clave_pub-RSA.pem
(jose@kali)-[~/ssl]
```

Cifrado con claves asimétricas

Para cifrar un archivo, utilizaremos la clave pública de la siguiente manera

```
openssl pkeyutl -encrypt -inkey clave_pub-RSA_con_passw.pem -pubin -in pp.txt -out  
pp.cryptoRSA
```

```
Archivo Acciones Editar Vista Ayuda
(jose@kali)-[~/ssl]
└─$ openssl pkeyutl -encrypt -inkey clave_pub-RSA_con_passw.pem -pubin -in pp.txt -out pp.cryptoRSA

(jose@kali)-[~/ssl]
└─$ cat pp.cryptoRSA
9•De•••i•K#•}•Jn•••F•I•(N?hA|!•••••6R• •••z•8•••I,W[
••
#•••|••0••••m~ @••
•L•••U7•K•0):••|YY•••••@••••b"1N•••Üf•••25•••••••?•C•••c•••<••6•••••f•39b•w•••W•••N•P[DC••a•••••••2•5
••[DE•nH-pi•••••••?••5-••10;••••
m•o
(jose@kali)-[~/ssl]
```

Para descifrarlo, utilizaremos la clave privada de la siguiente manera:

```
openssl pkeyutl -decrypt -inkey clave_priv-RSA_con_passw.pem -in pp.cryptoRSA -out  
datos.recupRSA
```

```
(jose@kali)-[~/ssl]
└─$ openssl pkeyutl -decrypt -inkey clave_priv-RSA_con_passw.pem -in pp.cryptoRSA -out datos.recupRSA
Enter pass phrase for clave_priv-RSA_con_passw.pem:

(jose@kali)-[~/ssl]
└─$ cat datos.recupRSA
Hola Mundo

(jose@kali)-[~/ssl]
└─$ diff pp.txt datos.recupRSA
```

Funciones Hash

Las funciones de hash en OpenSSL se utilizan para generar resúmenes criptográficos de datos, que son valores únicos y fijos de longitud fija que representan los datos originales.

MD5

MD5 (Message Digest Algorithm 5) es un algoritmo de hash criptográfico que produce un resumen de 128 bits (16 bytes). Sin embargo, debido a sus vulnerabilidades conocidas, MD5 ya no se considera seguro para aplicaciones críticas de seguridad, como la autenticación de contraseñas.

```
openssl dgst -md5 dd.txt
```

```
md5sum dd.txt
```

Podemos usar `wc -c` para verificar el tamaño del hash, que debería ser 33 caracteres. Al quitarle uno, obtenemos 32 caracteres. Dado que cada carácter hexadecimal representa 4 bits, multiplicar 32 por 4 nos dará los 128 bits correspondientes al tamaño del hash

```
(jose@kali)-[~/ssl]
└─$ openssl dgst -md5 dd.txt
MD5(dd.txt)= b4b9e397fb7e08bfeaa54090d2989e53

(jose@kali)-[~/ssl]
└─$ echo b4b9e397fb7e08bfeaa54090d2989e53 | wc -c
33

(jose@kali)-[~/ssl]
└─$ md5sum dd.txt
b4b9e397fb7e08bfeaa54090d2989e53  dd.txt
```

SHA1

SHA-1 (Secure Hash Algorithm 1) es un algoritmo de hash criptográfico que genera un resumen de mensaje de 160 bits (20 bytes). Fue desarrollado por la NSA en 1993 y se ha utilizado ampliamente en seguridad de datos. Sin embargo, desde 2005 se han encontrado vulnerabilidades en SHA-1, haciéndolo inseguro para aplicaciones críticas de seguridad.

`openssl dgst -sha1 dd.txt`

`sha1sum dd.txt`

```
(jose@kali)~[/ssl]
$ openssl dgst -sha1 dd.txt
SHA1(dd.txt)= 38beb661c63932eef289638a2443a7fdb8b7ad2d

(jose@kali)~[/ssl]
$ echo 38beb661c63932eef289638a2443a7fdb8b7ad2d | wc -c
41

(jose@kali)~[/ssl]
$ sha1sum dd.txt
38beb661c63932eef289638a2443a7fdb8b7ad2d dd.txt
```

SHA256

SHA-256 es un algoritmo de hash criptográfico que produce un resumen de mensaje de 256 bits (32 bytes). Es altamente seguro y ampliamente utilizado en aplicaciones que requieren integridad y autenticidad de datos.

`openssl dgst -sha256 dd.txt`

`sha256sum dd.txt`

```
(jose@kali)~[/ssl]
$ openssl dgst -sha256 dd.txt
SHA2-256(dd.txt)= 06bd0df149ba6261bbf9439e4f0ecacc1fa51e562cdb0ea95b6ce9293707f4da

(jose@kali)~[/ssl]
$ echo 06bd0df149ba6261bbf9439e4f0ecacc1fa51e562cdb0ea95b6ce9293707f4da | wc -c
65

(jose@kali)~[/ssl]
$ sha256sum dd.txt
06bd0df149ba6261bbf9439e4f0ecacc1fa51e562cdb0ea95b6ce9293707f4da dd.txt

(jose@kali)~[/ssl]
```

.....

SHA512

SHA-512 es un algoritmo de hash criptográfico que produce un resumen de mensaje de 512 bits (64 bytes). Es altamente seguro y ampliamente utilizado en aplicaciones que requieren una alta seguridad en la generación de resúmenes criptográficos.

`openssl dgst -sha512 dd.txt`

`sha512sum dd.txt`

```
(jose@kali)-[~/ssl]
$ openssl dgst -sha512 dd.txt
SHA2-512(dd.txt)= 9ba1301e47d9e1cb3ff377ae9113c347bd7fddf096d796329cfd0f362b4cc08daa7d4f4051497e0a75b6d7a88cd1367a74a8018149f0dae2265f88dea476aea9

(jose@kali)-[~/ssl]
$ echo 9ba1301e47d9e1cb3ff377ae9113c347bd7fddf096d796329cfd0f362b4cc08daa7d4f4051497e0a75b6d7a88cd1367a74a8018149f0dae2265f88dea476aea9 | wc -c
129

(jose@kali)-[~/ssl]
$ sha512sum dd.txt
9ba1301e47d9e1cb3ff377ae9113c347bd7fddf096d796329cfd0f362b4cc08daa7d4f4051497e0a75b6d7a88cd1367a74a8018149f0dae2265f88dea476aea9 dd.txt
```

HMAC

HMAC (Hash-based Message Authentication Code) es un algoritmo de autenticación de mensajes que se utiliza para verificar tanto la integridad como la autenticidad de un mensaje o datos.

`openssl dgst -hmac "12345678" pp.txt`

```
Archivo Acciones Editar Vista Ayuda
(jose@kali)-[~/ssl]
$ openssl dgst -hmac "12345678" pp.txt
HMAC-SHA256(pp.txt)= c361031123540ecf58c4b15f214c0b7ce0a072b2150d7e8f75ae9ef88bb2b055

(jose@kali)-[~/ssl]
```

Podemos aumentar la seguridad generando un número aleatorio hexadecimal de 32 bits.

`openssl rand -hex 32`

```
Archivo Acciones Editar Vista Ayuda
(jose@kali)-[~/ssl]
$ openssl rand -hex 32
dcbad78a4cde409ea0d9d53cbde2f7153fbf35c8b48cff200f064ca7a4c2c5d3

(jose@kali)-[~/ssl]
$ openssl dgst -hmac "dcbad78a4cde409ea0d9d53cbde2f7153fbf35c8b48cff200f064ca7a4c2c5d3" pp.txt
HMAC-SHA256(pp.txt)= d322ca221e6a552f533e278111436fafa33c16053d2d201d462390cc75ed04c5
```

.....

Podemos hacer algo similar, pero en lugar de utilizar la codificación hexadecimal, utilizaremos la codificación Base64.

```
openssl rand -base64 32
```

```
openssl dgst -hmac "AbcMOCWoYpWdst2vefxR9U5NPH98DH9yjc/Mxqdcbee=" pp.txt
```

```
Archivo Acciones Editar Vista Ayuda
(jose@kali)-[~/ssl]
$ openssl rand -base64 32
AbcMOCWoYpWdst2vefxR9U5NPH98DH9yjc/Mxqdcbee=

(jose@kali)-[~/ssl]
$ openssl dgst -hmac "AbcMOCWoYpWdst2vefxR9U5NPH98DH9yjc/Mxqdcbee=" pp.txt
HMAC-SHA256(pp.txt)= c9c6dfdee2bae83328e92a4ea6bb40a2b368255f540587df44cc170e93935833
```

Certificados digitales

Los certificados digitales son archivos utilizados en criptografía para asegurar la identidad de un sitio web o una entidad en línea. Lo primero que necesitamos es generar una clave privada y una publica.

```
(jose@kali)-[~/ssl]
$ openssl genrsa -out clave_privada_jose.pem 2048

(jose@kali)-[~/ssl]
$ openssl rsa -in clave_privada_jose.pem -pubout -out clave_publica_jose.pem
writing RSA key

(jose@kali)-[~/ssl]
$ ls
clave_privada_jose.pem  clave_publica_jose.pem

(jose@kali)-[~/ssl]
```

Una vez generadas las claves, procedemos a crear una nueva solicitud de certificado utilizando la clave privada especificada. Luego, guardamos la CSR resultante en un archivo de salida. Esta CSR puede ser enviada a una autoridad de certificación (CA) para su firma y la emisión del correspondiente certificado digital.

```
openssl req -new -key clave_privada_jose.pem -out certificado_jose_para_firmar.csr
```

.....

```

(jose@kali)-[~/ssl]
$ openssl req -new -key clave_privada_jose.pem -out certificado_jose_para_frimar.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:granada
Locality Name (eg, city) []:granada
Organization Name (eg, company) [Internet Widgits Pty Ltd]:ciber
Organizational Unit Name (eg, section) []:ciber
Common Name (e.g. server FQDN or YOUR name) []:ciber
Email Address []:ciber@gmail.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:1234
An optional company name []:

```

Para analizar y mostrar el contenido de la solicitud de certificado (CSR) en formato ASN.1.

openssl asn1parse -in certificado_jose_para_frimar.csr

```

(jose@kali)-[~/ssl]
$ openssl asn1parse -in certificado_jose_para_frimar.csr
 0:d=0  hl=4  l= 732 cons: SEQUENCE
 4:d=1  hl=4  l= 452 cons: SEQUENCE
 8:d=2  hl=2  l=   1 prim: INTEGER               :00
11:d=2  hl=3  l= 129 cons: SEQUENCE
14:d=3  hl=2  l=  11 cons: SET
16:d=4  hl=2  l=   9 cons: SEQUENCE
18:d=5  hl=2  l=   3 prim: OBJECT                  :countryName
23:d=5  hl=2  l=   2 prim: PRINTABLESTRING         :ES
27:d=3  hl=2  l=  16 cons: SET
29:d=4  hl=2  l=  14 cons: SEQUENCE
31:d=5  hl=2  l=   3 prim: OBJECT                  :stateOrProvinceName
36:d=5  hl=2  l=   7 prim: UTF8STRING              :granada
45:d=3  hl=2  l=  16 cons: SET
47:d=4  hl=2  l=  14 cons: SEQUENCE
49:d=5  hl=2  l=   3 prim: OBJECT                  :localityName
54:d=5  hl=2  l=   7 prim: UTF8STRING              :granada
63:d=3  hl=2  l=  14 cons: SET
65:d=4  hl=2  l=  12 cons: SEQUENCE
67:d=5  hl=2  l=   3 prim: OBJECT                  :organizationName
72:d=5  hl=2  l=   5 prim: UTF8STRING              :ciber
79:d=3  hl=2  l=  14 cons: SET
81:d=4  hl=2  l=  12 cons: SEQUENCE
83:d=5  hl=2  l=   3 prim: OBJECT                  :organizationalUnitName
88:d=5  hl=2  l=   5 prim: UTF8STRING              :ciber

```

Procedemos a generar un certificado X.509 firmado digitalmente a partir de una solicitud de certificado (CSR) y una clave privada.

```
openssl x509 -req -days 365 -in certificado_jose_para_fimar.csr -signkey  
clave_privada_jose.pem -out certificado_final_jose.crt
```

```
Archivo Acciones Editar Vista Ayuda
(jose@kali)-[~/ssl]
$ openssl x509 -req -days 365 -in certificado_jose_para_fimar.csr -signkey clave_privada_jose.pem -out certificado_final_jose.crt
Certificate request self-signature ok
subject=C = ES, ST = granada, L = granada, O = ciber, OU = ciber, CN = ciber, emailAddress = ciber@gmail.com
(jose@kali)-[~/ssl]
```

Para analizar y mostrar el contenido de la solicitud de certificado (CSR) en formato ASN.1.

```
openssl asn1parse -in certificado_final_jose.crt
```

```
(jose@kali)-[~/ssl]
$ openssl asn1parse -in certificado_final_jose.crt
0:d=0 hl=4 l= 907 cons: SEQUENCE
4:d=1 hl=4 l= 627 cons: SEQUENCE
8:d=2 hl=2 l= 20 prim: INTEGER               :4A5BD11CD74364FE19C54A48ECAD3257B9D4A16F
30:d=2 hl=2 l= 13 cons: SEQUENCE
32:d=3 hl=2 l=  9 prim: OBJECT               :sha256WithRSAEncryption
43:d=3 hl=2 l=  0 prim: NULL
45:d=2 hl=3 l= 129 cons: SEQUENCE
48:d=3 hl=2 l= 11 cons: SET
50:d=4 hl=2 l=  9 cons: SEQUENCE
52:d=5 hl=2 l=  3 prim: OBJECT               :countryName
57:d=5 hl=2 l=  2 prim: PRINTABLESTRING     :ES
61:d=3 hl=2 l= 16 cons: SET
63:d=4 hl=2 l= 14 cons: SEQUENCE
65:d=5 hl=2 l=  3 prim: OBJECT               :stateOrProvinceName
70:d=5 hl=2 l=  7 prim: UTF8STRING          :granada
79:d=3 hl=2 l= 16 cons: SET
81:d=4 hl=2 l= 14 cons: SEQUENCE
83:d=5 hl=2 l=  3 prim: OBJECT               :localityName
88:d=5 hl=2 l=  7 prim: UTF8STRING          :granada
97:d=3 hl=2 l= 14 cons: SET
99:d=4 hl=2 l= 12 cons: SEQUENCE
101:d=5 hl=2 l=  3 prim: OBJECT              :organizationName
106:d=5 hl=2 l=  5 prim: UTF8STRING          :ciber
113:d=3 hl=2 l= 14 cons: SET
115:d=4 hl=2 l= 12 cons: SEQUENCE
117:d=5 hl=2 l=  3 prim: OBJECT              :organizationalUnitName
122:d=5 hl=2 l=  5 prim: UTF8STRING          :ciber
```


Firma digitalmente con certificado digital

Ahora, procederemos a firmar digitalmente el archivo pp.txt utilizando la clave privada clave_privada_jose.pem. El resultado será un archivo de salida llamado pp.txt.firmado, que contendrá la firma digital del archivo

`openssl dgst -c -sign clave_privada_jose.pem -out pp.txt.firmado pp.txt`

```
Archivo Acciones Editar Vista Ayuda
(jose@kali)-[~/ssl]
$ openssl dgst -c -sign clave_privada_jose.pem -out pp.txt.firmado pp.txt

(jose@kali)-[~/ssl]
$ ls
certificado_jose_para_firmar.csr  clave_privada_jose.pem  pp.txt  pp.txt.smime_descifrado
certificado_final_jose.crt       clave_publica_jose.pem  pp.txt.firmado  pp.txt.smime
```

Podemos verificar que la firma se ha realizado correctamente utilizando la clave pública correspondiente.

`openssl dgst -c -verify clave_publica_jose.pem -signature pp.txt.firmado pp.txt`

```
(jose@kali)-[~/ssl]
$ openssl dgst -c -verify clave_publica_jose.pem -signature pp.txt.firmado pp.txt
Verified OK
(jose@kali)-[~/ssl]
```

.....