

## GUÍA 4: OPENSSL Y JAVA CRYPTOGRAPHY ARCHITECTURE (JCA)

### OpenSSL

OpenSSL es una implementación *open source* de SSL (Secure Socket Layer) que puede descargarse de <http://www.openssl.org>. En algunas distribuciones linux ya viene instalado.<sup>1</sup>

Dispone de una librería para programar aplicaciones y de una utilidad de línea de comandos (`openssl`) que permite realizar tareas criptográficas tales como:

- calcular hashes criptográficos (digestos). Por ejemplo: MD5, RIPEMD-160, SHA, SHA-1
- cifrar con **algoritmos de cifrado en bloque** (AES, CAST, DES, Triple DES, Blowfish, RC2) o en flujo (RC4), y usar aquéllos en múltiples modos de operación (ECB, CBC, CFB, OFB).
- generar **claves asimétricas** para los algoritmos habituales (Diffie-Hellman, DSA, RSA)
- encriptar y desencriptar con algoritmos asimétricos (RSA, DSA)
- Utilizar los algoritmos para **firmar**, **certificar** y revocar claves.
- manejar formatos de **certificados** (X.509, PEM, PKCS7, PKCS8, PKCS12).

Para ver los subcomandos disponibles,

```
$ openssl --help
```

### Rutinas de Hashing en OpenSSL

Con el comando `dgst` se puede obtener el hash (**digesto** o **resumen**) de un mensaje. Luego se puede usar, entre otras cosas, para firma digital. (`man openssl dgst`)

```
openssl dgst|digest [-digest] [-list] [-help] [-c] [-d] [-debug] [-hex]
[-binary] [-xoflen length] [-r] [-out filename] [-sign filename|uri]
[-keyform DER|PEM|P12|ENGINE] [-passin arg] [-verify filename]
[-prverify filename] [-signature filename] [-sigopt nm:v] [-hmac key] [-mac alg]
[-macopt nm:v] [-fips-fingerprint] [-engine id] [-engine_impl id] [-rand files]
[-writerand file] [-provider name] [-provider-path path] [-propquery propq]
[file ...]
```

### Rutinas de Cifrado simétrico en OpenSSL

Las rutinas de cifrado simétrico están asociadas al comando `enc` (`man openssl enc`)

```
openssl enc|cipher [-cipher] [-help] [-list] [-ciphers] [-in filename]
[-out filename] [-pass arg] [-e] [-d] [-a] [-base64] [-A] [-k password]
[-kfile filename] [-K key] [-iv IV] [-S salt] [-salt] [-nosalt] [-z]
[-md digest] [-iter count] [-pbkdf2] [-saltlen size] [-p] [-P] [-bufsize number]
[-nopad] [-v] [-debug] [-none] [-engine id] [-rand files] [-writerand file] [-
provider name] [-provider-path path] [-propquery propq]
```

### Rutinas de Criptografía de Clave Pública en OpenSSL

Para obtener una clave privada RSA se utiliza el comando `genrsa` (`man openssl genrsa`)

```
openssl genrsa [-help] [-out filename] [-passout arg] [-aes128] [-aes192]
[-aes256] [-aria128] [-aria192] [-aria256] [-camellia128] [-camellia192]
[-camellia256] [-des] [-des3] [-idea] [-F4] [-f4] [-3] [-primes num] [-verbose]
[-quiet] [-traditional] [-rand files] [-writerand file] [-engine id]
[-provider name] [-provider-path path] [-propquery propq] [numbits]
```

<sup>1</sup> Tomaremos como referencia para esta guía y para el tp de implementación la plataforma de pampero. En pampero está instalada la versión 3.2.0 Los comandos se pueden investigar en <https://www.openssl.org/docs/man3.2>

## GUÍA 4: OPENSSL Y JAVA CRYPTOGRAPHY ARCHITECTURE (JCA)

Para extraer de la clave privada recientemente obtenida, la clave pública, utilizar el comando `rsa.(man openssl rsa)`

```
openssl rsa [-help] [-inform DER|PEM|P12|ENGINE] [-outform DER|PEM] [-in
filename|uri] [-passin arg] [-out filename] [-passout arg] [-aes128] [-aes192]
[-aes256] [-aria128] [-aria192] [-aria256] [-camellia128] [-camellia192] [-
camellia256] [-des] [-des3] [-idea] [-text] [-noout] [-modulus] [-traditional]
[-check] [-pubin] [-pubout] [-RSAPublicKey_in] [-RSAPublicKey_out] [-pvk-strong]
[-pvk-weak] [-pvk-none] [-engine id] [-provider name] [-provider-path path] [-
propquery propq]
```

Para cifrar un mensaje con rsa, y clave pública, se usa el comando `rsautl.(man openssl rsautl)`

```
openssl rsautl [-help] [-in file] [-passin arg] [-rev] [-out file]
[-inkey filename|uri] [-keyform DER|PEM|P12|ENGINE] [-pubin] [-certin] [-sign]
[-verify] [-encrypt] [-decrypt] [-pkcs] [-x931] [-oaep] [-raw] [-hexdump]
[-asn1parse] [-engine id] [-rand files] [-writerand file] [-provider name]
[-provider-path path] [-propquery propq]
```

### Ejercicio 1. Investiga:

1. ¿Qué algoritmos de cifrado simétrico soporta openssl?
2. ¿Qué modos de cifrado de bloque soporta openssl?
3. ¿Qué modos de cifrado tiene AES en openssl? ¿A qué hace referencia?
4. ¿Cómo se establece el vector de inicialización para comenzar el cifrado?
5. ¿Qué relación hay entre clave, vector de inicialización y password en openssl?
6. ¿Para qué sirve el "SALT" y como se indica en openssl?

### Ejercicio 2:

- a) Cifrar la cadena "basta de mosquitos" con AES 128, modo CBC (es el default) y en modo ECB usando como contraseña "reloj". Buscar una opción para codificar en base64 el resultado y que salga algo legible.<sup>2</sup>
- b) Volver a cifrar y comparar los resultados ¿Se usó el SALT?
- c) Volver a cifrar sin SALT y observar en qué se diferencia la salida.
- d) Cifrar con AES 128 modo CBC y con modo ECB la cadena "buen diabuen diabuen diabuen dia".(NO usar base 64) ¿Qué cambia en cada caso?

### Ejercicio 3:

Descifra la cadena siguiente, sabiendo que está cifrada con AES 128 en modo CBC y la contraseña es "margarita" (observa que cada comando de cifrado tiene una opción de cifrar y otra de descifrar). No se usó Salt.

```
UU32ReW9vbs3MyoniAxJpRVnhg6sQ805FoAJzqGi63E=
```

### Ejercicio 4:

Descifra la cadena siguiente, sabiendo que está cifrada con AES 128, que la clave es "1234A" y el iv es "AAFF"

```
OLhfDpOmKHFkujdSbLGIPrvVnBINKPJRvjdg5S8NxkTuiU8ks30QoVuzAo9otQwD
```

- a) ¿De cuántos bytes es una clave para AES128?
- b) En este caso, ¿Cuál es, en hexa, la clave que realmente se utilizó?

### Ejercicio 5:

Construir un archivo de 1KB de longitud con todos sus bytes en 0. Encriptarlo con AES usando la misma clave pero una vez en modo ECB y otra en modo CBC. Comparar los resultados.

### Ejercicio 6:

<sup>2</sup> Tener en cuenta que -K es el parámetro para poner la clave en hexa, mientras que -k es para poner una password. Cuando se escribe una password, openssl usa un algoritmo para derivar iv y clave. Usar -pbkdf2 como algoritmo para derivar iv y key a partir de la password.

---

## GUÍA 4: OPENSSL Y JAVA CRYPTOGRAPHY ARCHITECTURE (JCA)

---

Genera un clave privada RSA de 1024 bits y guardarla en el archivo "`priv.key`". Abrir el archivo resultante y observar el encabezamiento.

### Ejercicio 7:

Extraer de la clave privada recientemente obtenida, la clave pública. ¿Es la clave privada "más grande" que la clave pública? ¿Por qué?

### Ejercicio 8:

Intercambiar claves públicas con otro compañero. Luego, intercambiar un mensaje usando un directorio compartido de pampero.

### Ejercicio 9:

Generar con `genrsa` un clave privada RSA de 1024 bits y guardarla en el archivo "`priv.pem`". Extraer de la clave privada recientemente obtenida, la clave pública. Guardarla como "`<apellido>pub.pem`" (Por ejemplo, `ariaspub.pem`) Para ello, utilizar el comando `rsa`. Alojar la clave pública en el directorio `/tmp` de pampero.

- ¿Cuántas personas podrían enviarte un mensaje encriptado correctamente?
- ¿Puede alguien enviarte un mensaje haciéndose pasar por otra persona?

### Ejercicio 10:

Si enviaras un mensaje a un compañero que diga "mi nombre es ..." encriptado con tu clave privada.

- ¿Con qué clave deberá desencriptar el mensaje tu compañero?
- ¿Puede tener la duda de que vos se lo enviaste? ¿Por qué?
- Efectúa lo anterior en openssl usando la opción `-sign` al encriptar. Tu compañero deberá usar `-verify` para desencriptar
- ¿Se mantiene la confidencialidad del mensaje? ¿Por qué?
- ¿Qué se aseguró con este procedimiento?

### Ejercicio 11:

Para firmar un documento en openssl, deberemos primero obtener un resumen (hash) del mismo.

- Elige un documento (puede ser este archivo pdf) y obtén su hash con `sha1`. Guarda el hash en un archivo
- Luego, cifrar el hash con tu clave privada, mediante el comando `rsautl` y la opción `-sign`. **Lo que obtuviste es la firma digital del documento.**
- Ahora envía a tu compañero: el documento original (sin cifrar ni resumir), la firma digital y la clave pública.
- ¿Cómo sabrá tu compañero que el archivo que le enviaste fue enviado por vos?

### Ejercicio 12:

Modifica un solo carácter de la firma anterior. Observa que ya no se verifica la autenticidad del documento.

### Ejercicio 13:

También se puede firmar un documento haciendo al mismo tiempo el hash y el cifrado, usando el comando `dgst`. Efectúa el ejercicio 3 en un solo paso con dicho comando. Deberías obtener Verified OK.

## JCA (Java Cryptography Architecture)

JCA es parte del lenguaje Java, como parte del API de seguridad, desde la versión JDK 1.1. Es una especificación del lenguaje que propone interfaces y clases abstractas que sirven de base para las implementaciones concretas de algoritmos criptográficos.

Está diseñada de acuerdo con dos principios:

- independencia e interoperabilidad de las implementaciones
- independencia y extensibilidad de los algoritmos.

La independencia de las implementaciones se consigue empleando una arquitectura basada en proveedores (**providers**). Un proveedor se refiere a un paquete o conjunto de paquetes que proporciona una implementación concreta de funcionalidades criptográficas de la API de seguridad de Java.

***GUÍA 4: OPENSSL Y JAVA CRYPTOGRAPHY ARCHITECTURE (JCA)***

---

La interoperabilidad de las implementaciones permite que cada una de ellas pueda usarse con las demás.

Algunos ejemplos de **providers**:

- SunJCE (Java Cryptography Extension)
- BC (Proyecto Bouncy Castle)

**Ejercicio 1:** Ejecuta los ejemplos del Manual.

**Ejercicio 2:** Repetir los ejercicios realizados en Openssl programándolos en java.