



## Anexo T2: PKI con OpenSSL

## HTTPS (HTTP over SSL)

- https → Utiliza la librería SSL (Secure Sockets Layer) para garantizar servicio de Transferencia
  - Puerto 443 por defecto permite acceder a los mismos contenidos que el puerto estándar (80), pero mediante TCP.
  - TLS v1.0 [Transport Layer Security] = SSL 3.1 (última versión: TLS 1.5 (2014))

## Servicios de Seguridad

## Autentificación e $\leftrightarrow$ de claves

- Fase de Establecimiento
  - Autentifica al Servidor (y al Cliente, opcional)
  - certif. digitales X.509 + (modo Asimétrico)
    - RSA + variante RSA + variante

## confidencialidad

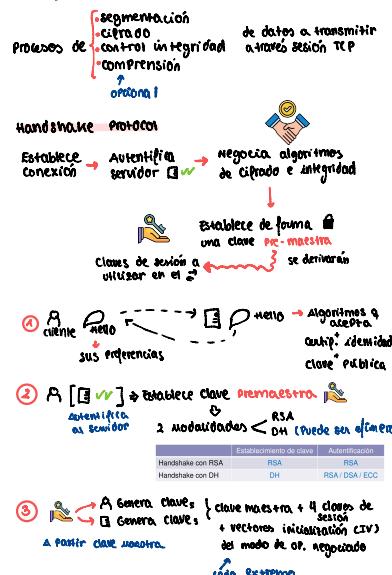
- Cifrar es cliente-servidor  
+  
Algoritmos clave simétrica + modo de operación  
AES, CHACHA20, 3DES, ...      GCM, CBC, CCA, ...  
  
Seguridad  
la garantía en todos los pasos  
Huellas digitales a partir de los datos (canalizadas)  
HMAC y SHA-256, SHA-384, ...

Versiones TLS → implantación & debilidades

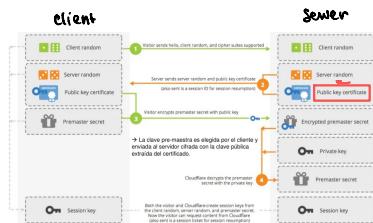
Protocol version	Website support [92]	Security [92][93]
<b>SSL 2.0</b>	0.1%	Insecure
<b>SSL 3.0</b>	1.4%	Insecure [94]
<b>TLS 1.0</b>	27.9%	Deprecated [20][21][22]
<b>TLS 1.1</b>	30.0%	Deprecated [20][21][22]
<b>TLS 1.2</b>	99.9%	Depends on cipher [n 1] and client mitigations [n 2]
<b>TLS 1.3</b>	70.1%	Secure

## **Funcionamiento**

### Record Protocol



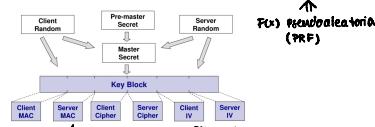
## Handshake Protocol for RSA



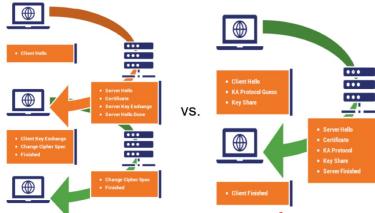
### Obtención de las claves de serie

se extraen de un **key-block** obtenido → valores + clave aleatorios premaster

The diagram illustrates the components of a key-block. It consists of three rectangular boxes arranged horizontally. The first box contains the text "Client Random". The second box contains "Pre-master Secret". The third box contains "Server Random". Above the boxes, the text "se extraen de un **key-block** obtenido → valores + clave aleatorios premaster" is written. An upward-pointing arrow is positioned above the "Pre-master Secret" box.



NOTA: TLS 1.2 vs TLS 1.3



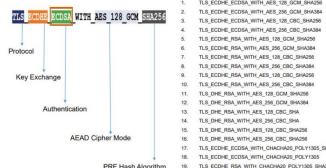
14 AUG 1969 0000 UTC AES-438-0049-3411236 ⇒ 4.2

7.3 AEE 422 - 6044 - SNT 384 → 4.3

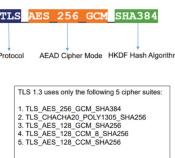
## Handshake Protocol (or DH / DHE)



#### TLS 1.2 – Suites de cifrado recomendadas (20)



TLS 1.3 – Sólo 5 suites de cifrado disponibles



## Configuración HTTPS en Apache

- Utiliza módulo dinámico → mod-ssl
  - Se config. como servidor virtual TCP/443
  - Directivas básicas

- Listen
- Document Root
- sslCertificateFile → del servidor Web
- sslCertificateKeyFile → clave privada
- sslCACertificateFile → CA reconocidas
- sslVerifyClient → requerir certificado en cliente

SSL CA Revocation File → ficheros de certificados Revocados

# Anexo T2: PKI con OpenSSL

## (HTTPS (HTTP over SSL))

### Ejemplo config. HTTPS

```
Listen 0.0.0.0:443
<VirtualHost *:443>
    SSLEngine on
    SSLCertificateFile /etc/pki/tls/certs/servidor.crt
    SSLCertificateKeyFile /etc/pki/tls/private/servidor.key
    SSLVerifyClient require
    SSLVerifyDepth 10
    SSLCACertificateFile /etc/pki/tls/cacert.pem
    SSLCARevocationFile /etc/pki/tls/certs/crl.pem
    SSLCARevocationCheck chain
</VirtualHost>
```

→ Todos misos (las versiones) distintas

```
SSLProtocol all -SSLv3 -TLSv1 -TLSv1.1
SSLCipherSuite ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-
    RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDH-
    ECDHE-CHACHA20-POLY1305:ECDH-ECDSA-AES128-GCM-SHA256:ECDH-
    ECDHE-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-
    AES256-SHA384:CDH-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256
SSLHonorCipherOrder on → Respetar el orden (engañar)
SSLCompression off
SSLSessionTickets off
```

→ cliente puede escuchar las mismas claves de auto de Diffie-Hellman (DHE) DHE off & NO!

## Certificados X.509

- Documento electrónico establece relación fiable entidad ↔ sus ATAs
- Para dar validez → tiene que ser firmado por un CA de confianza
- Para verificar la autenticidad de un certificado clave pública necesaria es necesario verificar la firma digital del mismo
- Requerimientos: Clave Pública CA firmante
  - El certificado
  - CA válido → si el cliente tiene el CA de la CA en la config. de autorización del cliente (ej: navegador)

PKI: Public Key Infrastructure  
Modelo de certificación jerárquico descrito → cert. autorizado → verif. clave revocada → CA

### Certificado X.509

Estandar de certificaciones más extendido

- Versión actual (v3) → formato flexible con campos extensibles adicionales

#### Campos:

- Version (1.2 o 3)
- Número de serie
- Algoritmo de cifrado (RSA, DSS,...)
- CA emisora (Issuer)
- Validad del certificado
- Entidad titulal (Subject) → identif. con su nombre Distintivo o DN
- Clave pública de la entidad
- Firma CA
- Extensiones

↳ uso q. q. va a dar a la clave  
↳ q. tipo de clave va dirigido (servidor, CA, usuario,...)  
↳ Los algoritmos de cifrado simétrico q. utilizan la entidad

#### Nombre Distintivo (CN)

- Formado por un conjunto de campos normalizados q. identifican a la entidad

#### Algunos campos

Geográfico	Organizativo	Personales
(C) Country	(O) organization	(CN) Common Name
(ST) State/province	(OU) organizationalUnit	Identificación del user
(L) location	Unit	Nombre usuario, dirección, nombre sitio web,...

### Certificados múltiples y nombres alternativos

- Certificados wildcard e comodín → todos los subdominios de un dominio dado (+ \*.ca) → se usan para proteger el acceso a los servicios del dominio
- Certificados múltiples (SAN, Subject Alternative Name) → N dominios en 1 certificado
- Certificados con validación extendida (EV) → la identidad ha sido verificada por la CA

## OpenSSL → Infraestructura de clave Pública

- Librería criptográfica open source.

Cifrado simétrico → algoritmos y modos de operación

ofrece Generación + comprobación de huellas de integridad → con ≠ algoritmos (SHA1, SHA256,...)

Cifrado Asimétrico → claves + cifrado + firmas digitales → gestión infra. clave pública,

### USO OpenSSL

#### Versión de la librería

Podemos ver el directorio de configuración (OPENSSLDIR). El archivo openssl.dgt es un enlace simbólico a su ubicación.

En Debian/Ubuntu: /etc/ssl

#### Lista de algoritmos soportados

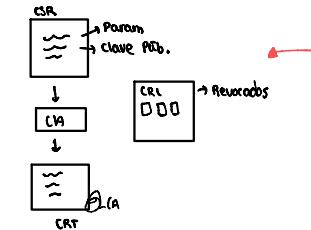
openssl list -cipher-algorithms

openssl list -digest-algorithms

openssl list -public-key-algorithms

#### Generación de valores aleatorios (ej: 16 bytes: 128 bits)

openssl rand -hex 16



### Infraestructura PKI con OpenSSL

- Directorio base → 9 cert./pkcs12 para nuestra CA
- Archivo Config → openssl.cnf

El fichero openssl.cnf viene configurado con una CA de ejemplo (CA-default). Se requieren los siguientes directorios y ficheros:

- Directorio certs → contendrá los certificados reconocidos
- Directorio crt → contendrá los certificados revocados
- Directorio newcerts → contendrá los certificados pendientes
- Directorio private → contendrá la clave privada de la CA (ca.key.pem).
- Fichero cacert.pem → contendrá el certificado de la CA (cacert.pem).
- Fichero serial → número de serie que se asignará al siguiente certificado
- Fichero crlnumber → número de secuencia para los certificados revocados
- Fichero index.txt → listado de certificados creados

### Opciones de OpenSSL

#### Obtención de huellas (fingerprints), digest

openssl dgst -sha256 /etc/pki/passwd

find /etc -type f | xargs openssl dgst -sha1 > huellas.txt

#### Cifrado simétrico, enc

openssl enc -e -aes-128-cbc -in ejem.txt -out ejem.enc

openssl enc -d -aes-128-cbc -in ejem.txt -out ejem.txt2

#### Obtención de claves asimétricas, genrsa

openssl genrsa -des3 -out servidor.key 2048

#### Visualiza el contenido de un archivo de claves, rsa → otras opciones ↗

openssl rsa -nodes -text -in servidor.key

#### Obtención de un requerimiento de certificado (csr), req

openssl req -new -key servidor.key -out servidor.csr

#### Creación de un certificado firmado por la CA, ca

openssl ca -in servidor.csr -out servidor.crt -days 730

#### Exportación de un certificado y sus claves, pkcs12

openssl pkcs12 -export -in juan.crt -inkey juan.key -out juan.p12

\* ↗ cert reg (csr)

Error más común:  
crear archivo utlito (CA, siempre lo haré)

### creación estructura necesaria:

Note: Es importante sincronizar la fecha y hora del equipo

```

mkdir -p /etc/pki/tls
cd /etc/pki/tls
mkdir certs
mkdir crt
mkdir newcerts
mkdir private
touch index.txt
echo 01 > serial
echo 01 > crlnumber
  
```

En el ejemplo, suponemos que la PKI se ubica en /etc/pki/tls

Se recomienda hacer una copia de seguridad de openssl.cnf

# Anexo T2: PKI con OpenSSL

(OpenSSL → Infraestructura de clave Pública)

## Fichero `openssl.cnf`

Contexto [ca] → establece el nombre de la CA

Contexto [chambre de la ca] → Establece ubicaciones (directorios) de los archivos q utilizan la CA + valores por defecto  
+ política de generación (policy)

define política para la emisión de nuevos certificados

Contexto [`<Policy>`] → Indicar si los parámtos del SIR deben coincidir en los de CA (Mandatory)  
Deben ser introducidos por el usuario (`mandatory`) o son opcionales (`optional`)

Contexto [req\_distinguished\_name] → Permite establecer los valores por defecto q se presentarán para los campos qv de un nuevo certificado

Contexto [v3\_ca] → Establece las extensiones de los certificados raíz

Ejemplo directiva `x509_extensions` de [req]

```
[v3_ca]
basicConstraints = critical, CA:TRUE, pathlen:3
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always, issuer
keyUsage = critical, cRLSign, keyCertSign
nsCertType = sslCA, emailCA
```

Contexto [server] → Establece extensiones para los certificados de servidor (⚠️ obsoleto sin SAN)

```
[server]
basicConstraints = CA:FALSE
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:issuer:always
keyUsage = digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth
nsCertType = server
```

Para más información:  
man x509\_v3\_config

Contexto [user] & [user-cert] → Ext. para certif. de usuario

```
[user]
basicConstraints = CA:FALSE
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:issuer:always
keyUsage = digitalSignature
extendedKeyUsage = clientAuth
```

Contexto [server\_SAN] → Ext para los certif. del servidor con nombres Alternativos  
SAN Subject Alternative Names

```
[server_SAN]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth
subjectAltName = @alt_names
```

```
[alt_names]
DNS.1 = www.efrel.com
DNS.2 = efrel.com
```

NOTA Importante:  
Las últimas versiones de los navegadores sólo admiten certificados SAN

## Ejemplo: Creación de una CA

1. Creación de la estructura de directorios y archivos necesarios
2. Edición de openssl.cnf: valores por defecto para los parámetros DN de nuestra CA y nuevos contextos a usar (`server, user...`)
3. Creación de las claves RSA de la CA y emisión de certificado autofirmado. Opciones: genrsa y req. Ficheros obtenidos: cakey.pem y cacert.pem. Extensiones: v3\_ca
4. Creación de la lista de certificados revocados. Comando: ca -gencrl. Fichero: crt.crl.pem (initialmente vacío)

```
openssl genrsa -out private/cakey.pem 2048
openssl req -new -x509 -key cakey.pem -out ca-cert.pem
openssl req -new -x509 -extensions v3_ca -in ca-csr.pem -out cacert.pem -key private/cakey.pem -days 3652
```

```
openssl ca -gencrl -out crt.crl
```

## Ejemplo: Emisión de Certificados y Operación

1. Generación de claves y certificado para nuestro servidor https. CN = www.efrel.com  
Opciones: genrsa, req y ca.  
Ficheros: <servidor>.key.pem y <servidor>.crt.pem  
Extension: server SAN
2. Instalamos nuestra CA como raíz de confianza en los navegadores de los clientes (exportando el certificado caclient.pem)
  - En el SC Windows ubiar los certificados de la práctica en el directorio %SystemRoot%\pki\pub\certs\root.
  - Si protegemos con frase de paso la clave del servidor, nos pedirá la frase cada vez que arrancaremos el servicio (aunque no aparecerá el prompt).

```
openssl genrsa -out efrel-key.pem 2048
openssl req -new -key efrel-key.pem -out efrel-csr.pem
openssl ca -extensions server SAN -in efrel-csr.pem -out efrel-crt.pem -days 730
```

3. Creación de claves y certificados para nuestros clientes y exportando en un contenedor pkcs12.  
Opciones: genrsa, req, ca y pkcs12  
Ficheros: usuario-key.pem, usuario-crt.pem y usuario\_crt.p12  
Extension: user

```
openssl req -new -keyfile Juan-key.pem -out Juan-csr.pem -nodes
openssl ca -extensions user -in Juan-csr.pem -out Juan-crt.pem -days 365
openssl pkcs12 -export -in Juan-crt.pem -inkey Juan-key.pem -out Juan-crt.p12
```

4. Activar el requerimiento de certificado de cliente en el servidor (`SSLVerifyClient require`). Configurar a nuestra propia CA como raíz de confianza del servidor (`SSLCACertificateFile`). Limitar el acceso a zonas del servidor (`SSLRequire`):

```
SSLVerifyClient require
SSLCACertificateFile /etc/ssl/certs/ca.pem
SSLRequire ( !(SSL_CLIENT_FINGERPRINT_md5 ) eq "00:00:00:00" )
SSLRequire ( !(SSL_CLIENT_FINGERPRINT_md5 ) eq "00:00:00:00" )
/Location
```

4. Revocar certificados de usuario (directivas `SSLCACertificateCheck` y `SSLCARevocationFile`). Revocar:

```
openssl ca -revoke Juan-crt.pem
openssl ca -gencrl -out crt.crl
```

NOTA: TLS 1.2+ permite la conexión entre cliente y servidor al inicio de la conexión. Para disponibilidad Autorizada: Se requiere el certificado cliente dentro de un contexto de conexión (use SSLVerifyClient require). El contexto de conexión debe ser el mismo que el contexto de conexión del cliente. El contexto de conexión debe ser el mismo que el contexto de conexión del cliente.

## SNI (Server Name Indication) y HTTPS

En las primeras implm de SSL/TLS → solo 1 puerto HTTPS en cada direct pública  
(muy costoso)

Nowadays se realizaba antes del 2º de cabecera,  
el 1º no sabía a q dominio se dirigía la petición

Ahora → SNI, extensión TLS → Permite ↗ el nombre de dominio. Nowdays  
a partir 2010-2012 en navegadores y servidores  
web clientHello / ServerHello

Config Apache mediante ➡> & servidorname , &serveralias