

# **Seguridad en Apache**

Antonio Espín Herranz

# Contenidos

- Configuración de **SSL/TLS** con Let's Encrypt.
- Control de acceso con **htaccess** y **autenticación básica**.
- Protección contra ataques comunes (**DDoS**, **inyección SQL**, **XSS**).
- **OpenSSL**: Uso de certificados.

# SSL / TLS

# Configuración de SSL/TLS con Let's Encrypt

- La **configuración de SSL/TLS con Let's Encrypt** en Apache es una forma de habilitar **HTTPS gratuito y automatizado** en tu servidor web.
- Let's Encrypt es una **autoridad certificadora (CA)** que emite certificados **válidos y confiables** sin coste, lo que mejora la seguridad de tu sitio web al cifrar la comunicación entre el navegador y el servidor.

# Let's Encrypt

- **Cifrado HTTPS:** Protege los datos transmitidos (contraseñas, formularios, etc.).
- **Autenticación:** Verifica que el sitio web pertenece al dominio que dice tener.
- **Integridad:** Evita que los datos sean modificados en tránsito.
- **Automatización:** Permite renovar certificados automáticamente cada 90 días.

# Let's Encrypt

- Let's Encrypt no tiene soporte oficial directo para Apache en Windows, pero puedes usar herramientas como:
  - **Win-ACME (wacs.exe)**: cliente compatible con Let's Encrypt para Windows.
  - **Certify The Web**: interfaz gráfica para gestionar certificados en IIS y Apache.
- En este caso, hay que:
  - Generar el certificado con Win-ACME.
  - Copiar los archivos **.crt** y **.key** a tu carpeta **conf/ssl/**
  - Editar **httpd-ssl.conf** para apuntar a esos archivos.

# Pasos en Windows

- **Ruta de instalación de Apache.**
- Ir a la carpeta **conf** de Windows
- Crear una carpeta **ssl** dentro de **conf**
- Utilizar el comando **OpenSSL** hay uno que viene con **Apache** en la carpeta **bin** y también se puede instalar:
  - <https://slproweb.com/products/Win32OpenSSL.html>
  - Probar: **openssl -v**

# Pasos en Windows: crear el certificado

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 ^  
-keyout C:\Apache24\conf\ssl\ejemplo.key ^  
-out C:\Apache24\conf\ssl\ejemplo.crt
```

*ejemplo.crt → certificado público*  
*ejemplo.key → clave privada*

^ es solo para partir la línea en la consola

- **Datos que pide:**

- Country Name (2 letter code) [AU]: **ES**
- State or Province Name (full name) [Some-State]: **Madrid**
- Locality Name (eg, city) []: **Madrid**
- Organization Name (eg, company) []: **Policía**
- Organizational Unit Name (eg, section) []: **IT**
- Common Name (e.g. server FQDN or YOUR name) []: **localhost**
- Email Address []: **admin@localhost**



# mod\_ssl: comando Openssl

Fragmento	Significado
<code>openssl</code>	Llama al programa OpenSSL
<code>req</code>	Inicia el proceso de creación de un certificado X.509
<code>-x509</code>	Indica que se generará un certificado autofirmado (no se crea CSR)
<code>-nodes</code>	No cifra la clave privada con contraseña (útil para Apache)
<code>-days 365</code>	El certificado será válido por 365 días
<code>-newkey rsa:2048</code>	Crea una nueva clave RSA de 2048 bits junto con el certificado
<code>^</code>	En Windows, indica que el comando continúa en la siguiente línea
<code>-keyout</code> <code>C:\Apache24\conf\ssl\ejemplo.key</code>	Ruta donde se guardará la clave privada
<code>-out C:\Apache24\conf\ssl\ejemplo.crt</code>	Ruta donde se guardará el certificado público

# Pasos en Windows

- Activar el **mod\_ssl** y el archivo de configuración
- Descomentar en **httpd.conf** las líneas:
  - **LoadModule ssl\_module modules/mod\_ssl.so**
  - **Include conf/extra/httpd-ssl.conf**
- Es possible, que haya que descomentar:
  - **LoadModule socache\_shmcb\_module modules/mod\_socache\_shmcb.so**

# Pasos en Windows: Editar el archivo: conf/extra/httpd-ssl.conf

Listen 443

<VirtualHost \_default\_:443>

ServerName localhost

DocumentRoot "\${SRVROOT}/htdocs"

SSLEngine on

SSLCertificateFile "\${SRVROOT}/conf/ssl/ejemplo.crt"

SSLCertificateKeyFile "\${SRVROOT}/conf/ssl/ejemplo.key"

SSLProtocol all -SSLv3

SSLCipherSuite HIGH:!aNULL:!MD5

SSLHonorCipherOrder on

<Directory "\${SRVROOT}/htdocs">

Options Indexes FollowSymLinks

AllowOverride All

Require all granted

</Directory>

ErrorLog "\${SRVROOT}/logs/error.log"

TransferLog "\${SRVROOT}/logs/access.log"

CustomLog "\${SRVROOT}/logs/ssl\_request.log" \

      "%t %h %{SSL\_PROTOCOL}x %{SSL\_CIPHER}x \"%r\" %b"

</VirtualHost>

**Podemos guardar el original en una copia  
Y añadir este contenido**

**Reiniciar Apache**  
**httpd -k restart**

**Probar:**  
<https://localhost>

Verás una advertencia de seguridad  
(porque el certificado es autofirmado).  
Haz clic en **"Avanzado" → "Continuar de todos modos"**.

# **.htaccess**

# Acceso con .htaccess

- Es un archivo de **texto plano** que se ubica en el directorio raíz del sitio Web o en un subdirectorio.
- El nombre viene de “Hypertext Access”
- Permite modificar la configuración de Apache sin acceder al fichero de configuración **httpd.conf**.

# .htaccess

- Se puede utilizar para:
  - Seguridad.
  - Redirecciones
  - Reescritura de URLs
  - Control de Caché
  - Manejo de Errores
  - Configuración de PHP

# .htaccess - Seguridad

- Acceso restringido a directorios o archivos
- Proteger con contraseña usando autenticación básica
- Bloquear IPs o rangos de IPs.

# .htaccess - Redirecciones

- Redirección de URLs antiguas a nuevas
- Redigir de HTTP a HTTPS
- Redirecciones de www a sin www (o viceversa)



# .htaccess – Redirecciones

- Podemos redirigir de un fichero a otro:
  - **Redirect** /antiguo.html /nuevo.html
- http codes:
  - <https://developer.mozilla.org/es/docs/Web/HTTP/Reference/Status>
- Redirección permanente (301)
  - **301 Moved Permanently**
  - **Redirect 301** /vieja-pagina.html /nueva-pagina.html
- Redirección temporal (302)
  - **302 Found**
  - **Redirect 302** /vieja-pagina.html /nueva-pagina.html

# htaccess – Redirecciones

## Con un Virtual Host: web.curso

```
C:\Users\Anton>curl -I http://web.curso/antiguo2.html
HTTP/1.1 301 Moved Permanently
Date: Mon, 22 Sep 2025 10:03:36 GMT
Server: Apache/2.4.57 (Win64) PHP/8.2.29
Location: http://web.curso/nuevo2.html
Content-Type: text/html; charset=iso-8859-1

C:\Users\Anton>curl -I http://web.curso/antiguo3.html
HTTP/1.1 302 Found
Date: Mon, 22 Sep 2025 10:04:40 GMT
Server: Apache/2.4.57 (Win64) PHP/8.2.29
Location: http://web.curso/nuevo3.html
Content-Type: text/html; charset=iso-8859-1
```

## Sin Virtual Host desde htdocs

Si se implementa desde **htdocs** poner la ruta desde la carpeta **htdocs** de apache.

### Redirect 301

```
/curso_apache/redir/antigua.html
/curso_apache/redir/nueva.html
```

```
C:\
  apache24\
    htdocs\
      curso_apache\
        redir\
          antigua.html
          nueva.html
```

En el comando **curl -I** → solicitar **sólo cabeceras**, es como una petición **HEAD**

# Redirección de HTTP a HTTPS

- Antes de la redirección de HTTP a HTTPS, necesitamos cumplir una serie de requisitos:
  - 1) Apache tiene que tener **activado** el **módulo: mod\_rewrite**
    - En el archivo **httpd.conf** tiene que estar habilitado: `mod_rewrite`
    - **LoadModule rewrite\_module modules/mod\_rewrite.so**
  - 2) En el **VirtualHost** tenemos que permitir el archivo **.htaccess**
    - Hay que añadir **AllowOverride All**, esto indica al servidor Apache que puede sobrescribir cualquier configuración.
    - Si la directiva no se indica, Apache ignora el archivo .htaccess, con lo cual no vamos a poder hacer la redirección.

# Redirección de HTTP a HTTPS

- Más requisitos:
  - 3) **Certificado SSL** configurado, para que HTTPS funcione necesitamos:
    - **Un certificado SSL** (puede ser autofirmado para pruebas).
    - **Un VirtualHost para el puerto 443:**

```
<VirtualHost *:443>
  ServerName web.curso
  DocumentRoot "D:/ruta/proyecto"
  SSLEngine on
  SSLCertificateFile "D:/ruta/certificado.crt"
  SSLCertificateKeyFile "D:/ruta/clave.key"
  <Directory "D:/ruta/proyecto">
    Options Indexes FollowSymLinks
    AllowOverride All
    Require all granted
  </Directory>
</VirtualHost>
```

Añadirlo al fichero **hosts** de Windows  
**C:\Windows\System32\drivers\etc\hosts**  
**127.0.0.1 web.curso**

# Redirección de HTTP a HTTPS

- Requisitos / pruebas
  - 4) **Acceso a la URL por HTTP**
    - Por ejemplo: <http://web.curso/info.php> debería redirigir automáticamente a: <https://web.curso.info.php>
  - 5) Dentro del fichero .htaccess:  
RewriteEngine On  
RewriteCond %{HTTPS} off  
RewriteRule ^(.\*)\$ https://%{HTTP\_HOST}%{REQUEST\_URI} [L,R=301]

***Más delante tenemos el uso y configuración de certificados***

Si tenemos definido un Virtual Host definido **web.curso** en **httpd-ssl.conf** en el puerto **443**  
Necesitamos definir el mismo Virtual Host (**web.curso**) en **httpd-vhosts.conf** en el puerto **80**  
***para poder redirigir!!***

# .htaccess – Reescritura de URLs

- Usar URLs “bonitas o amigables” con mod\_rewrite, paso de parámetros en la URL sin queryString.
- De: `curso.es/productos.php?id=123`
- A: `curso.es/productos/123`

# .htaccess – Control de Caché / Errores

- Control de la Caché:
  - Establecer reglas de expiración para imágenes, CSS, JS mejorando el rendimiento.
- Manejo de Errores:
  - Personalizar páginas de error como 404 Not Found, 403 Forbidden

# .htaccess

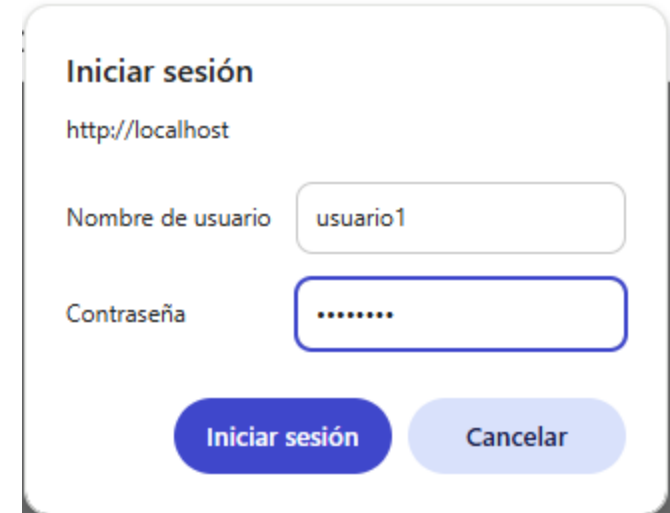
- Configuración de PHP
  - Cambiar directivas de PHP como:
    - upload\_max\_filesize
- Estas configuraciones sólo funcionarán si el servidor Apache tiene habilitado **AllowOverride**.
  - **Si no el servidor Apache ignora el archivo .htaccess**



# **Autenticación básica**

# Autenticación básica

- La **autenticación básica en Apache** es un método sencillo para proteger el acceso a recursos web mediante usuario y contraseña. Aunque no es el más seguro (ya que transmite credenciales en texto plano si no se usa HTTPS), es muy útil para restringir el acceso a directorios o páginas específicas.
- **¿Cómo funciona?**
  - El cliente solicita un recurso protegido.
  - Apache responde con un código **401 Unauthorized** y una cabecera **WWW-Authenticate**.
  - El navegador muestra un cuadro de diálogo solicitando usuario y contraseña.
  - El cliente envía las credenciales codificadas en Base64.
  - Apache verifica las credenciales contra un archivo de usuarios.
  - Si son válidas, se concede el acceso; si no, se rechaza.



A screenshot of a web browser's basic authentication dialog box. The title is "Iniciar sesión". Below the title, the URL "http://localhost" is displayed. There are two input fields: "Nombre de usuario" with the text "usuario1" and "Contraseña" with masked characters "\*\*\*\*\*". At the bottom, there are two buttons: "Iniciar sesión" (highlighted in blue) and "Cancelar" (light blue).

# Configuración

## **.htaccess**

Si lo hacemos dentro de .htaccess NO  
Usar directory, no está permitido.

Quitar directory

- **Crear el archivo de usuarios**
- 1. Usa **htpasswd.exe** (incluido con Apache en Windows):
  - `htpasswd -c "C:\Apache24\htpasswd" usuario1`
- 2. Configurar Apache (httpd.conf)

```
<Directory "C:/Apache24/htdocs/privado">  
    AuthType Basic  
    AuthName "Zona protegida"  
    AuthUserFile "C:/Apache24/.htpasswd"  
    Require valid-user  
</Directory>
```
- 3. Reiniciar Apache

**AuthType Basic:** define el tipo de autenticación.

**AuthName:** texto que aparece en el cuadro de diálogo.

**AuthUserFile:** ruta al archivo de contraseñas.

**Require valid-user:** permite el acceso a cualquier usuario válido.

# Permitir .htaccess en Apache

```
<Directory "D:/apache/Apache24/htdocs/apache_auth">  
    AllowOverride AuthConfig  
    Require all granted  
</Directory>
```

- ***La ruta a nuestra carpeta ...***

# Configuración

- **Consideraciones de seguridad**
- **Usa HTTP:** sin cifrado, las credenciales pueden ser interceptadas.
- **Protege el archivo .htpasswd:** no debe estar accesible desde el navegador.
- **Limita el acceso por IP o grupo si es necesario.**

# Redirección a HTTPs

- La autenticación básica envía el nombre de usuario y contraseña **codificados en Base64**, pero **no cifrados**.
  - Si no usas HTTPS, cualquier atacante en la red puede interceptarlos fácilmente.
- Paso 1:
  - LoadModule ssl\_module modules/mod\_ssl.so
  - Include conf/extra/httpd-ssl.conf

# Redirección a HTTPs

- **Opción A:**

```
<VirtualHost *:80>
```

```
    ServerName ejemplo.local
```

```
    DocumentRoot "D:/apache/Apache24/htdocs/apache_auth"
```

```
    Redirect permanent / https://ejemplo.local/
```

```
</VirtualHost>
```

- **Opción B:**

```
RewriteEngine On
```

```
RewriteCond %{HTTPS} off
```

```
RewriteRule ^(.*)$ https://%{HTTP_HOST}/$1 [R=301,L]
```

# Configuración final

## En .htaccess

AuthType Basic

AuthName "Zona Protegida"

AuthUserFile

"D:/apache/apache24/htdocs/apache\_auth/.htpasswd"

Require valid-user

## # Para redirección a https

RewriteEngine On

RewriteCond %{HTTPS} off

RewriteRule ^(.\*)\$ https://%{HTTP\_HOST}/\$1 [R=301,L]

## En httpd.conf o httpd-vhosts.conf

<Directory "D:/apache/Apache24/htdocs/apache\_auth">

AllowOverride AuthConfig FileInfo

Require all granted

</Directory>

## # Dentro de httpd-vhosts.conf ponerlo todo

<VirtualHost \*:80>

ServerName auth.local

DocumentRoot "D:/apache/Apache24/htdocs/apache\_auth"

<Directory "D:/apache/Apache24/htdocs/apache\_auth">

**AllowOverride AuthConfig FileInfo**

Require all granted

</Directory>

</VirtualHost>

## # El fichero .htaccess quedaría igual



# **Protección contra ataques**

## **Ddos, SQLi, XSS**

# Protección de ataques:

## Ddos, inyección SQL, XSS

- En Apache, puedes aplicar varias capas de protección para mitigar ataques comunes como **DDoS**, **inyección SQL** y **XSS**.
- Apache no puede evitar todos estos ataques por sí solo (especialmente los que dependen del código de la aplicación), sí puede ayudarte a reducir el riesgo y reforzar la seguridad del servidor web.

# Tipos de Ataques DDos

- **Ddos (Distributed Denial of Service)** es un ataque que busca **saturar un servidor o red** enviando una enorme cantidad de peticiones desde múltiples dispositivos (a menudo infectados o controlados remotamente).
- **Objetivo:** Hacer que el sitio web o servicio deje de estar disponible para usuarios legítimos.
- **Ejemplo:** Miles de bots accediendo a tu sitio al mismo tiempo para colapsarlo.

# Ddos en Apache

- **Módulos recomendados:**
- **mod\_evasive:** Detecta y bloquea IPs que hacen muchas peticiones en poco tiempo. ***OJO solo Linux, no está en Windows.***
- **mod\_reqtimeout:** Evita conexiones lentas que agotan recursos.
  - **RequestReadTimeout header=20-40,MinRate=500 body=20,MinRate=500**
- **Limitar tamaño y número de peticiones: A nivel global**
  - **LimitRequestBody 1048576** # 1MB máximo por petición
  - **LimitRequestFields 100** # Máximo 100 cabeceras
  - **LimitRequestFieldSize 8190** # Tamaño máximo por cabecera

# Ddos en Apache

- Las directivas anteriores se pueden colocar en el archivo **httpd.conf** o a nivel de **virtual host**, pero no se permiten en **.htaccess**.
- **LoadModule reqtimeout\_module modules/mod\_reqtimeout.so**

# Si el módulo está activo: en httpd.conf

**<IfModule reqtimeout\_module>**

RequestReadTimeout header=20-40,MinRate=500 body=20,MinRate=500

**</IfModule>**

- **header=20-40,MinRate=500**
  - Apache espera hasta 20 segundos para recibir los encabezados HTTP.
  - Si empieza a recibir datos, puede extender el tiempo hasta 40 segundos.
  - Si la velocidad de recepción es menor a 500 bytes/segundo, la conexión se cancela.
- **body=20,MinRate=500**
  - Apache espera hasta 20 segundos para recibir el cuerpo de la petición (por ejemplo, datos de un formulario).
  - Si la velocidad de recepción es menor a 500 bytes/segundo, también se cancela.

# DDos

- **Directivas:**

Directiva	Función
<code>LimitRequestBody</code>	Limita el tamaño máximo del cuerpo de la petición (por ejemplo, archivos subidos)
<code>LimitRequestFields</code>	Limita el número máximo de cabeceras HTTP que se aceptan en una petición
<code>LimitRequestFieldSize</code>	Limita el tamaño máximo de cada cabecera HTTP individual

# Recomendaciones

- Para proteger todo el servidor: ponlas en **httpd.conf**
- Para proteger **sitios específicos**: ponlas en cada **<VirtualHost>**
- Para proteger **rutas sensibles** (como **/uploads**): usa **<Directory>**

# Ejemplos

**<VirtualHost \*:80>**

ServerName ejemplo.local

DocumentRoot "C:/Apache24/htdocs/ejemplo"

LimitRequestBody 1048576

LimitRequestFields 100

LimitRequestFieldSize 8190

**</VirtualHost>**

**<Directory "C:/Apache24/htdocs/ejemplo/uploads">**

LimitRequestBody 1048576

**</Directory>**



# Tipos de Ataques SQLi

- La **inyección SQL** es una técnica en la que un atacante **inyecta código malicioso en una consulta SQL** a través de formularios o URLs mal protegidas.
- **Objetivo:** Robar, modificar o eliminar datos de la base de datos.
- **Ejemplo:** Enviar admin' OR '1'='1 en un formulario de login para saltarse la autenticación.

# Inyección SQL en Apache

- Desde la parte de PHP utilizar **PDO** con consultas preparadas: **prepare** y **bindParam**.
- No construir SQL mediante concatenación con los campos de un formulario de login.
  - `SELECT * FROM usuarios WHERE usuario = 'admin' OR '1'='1';`

# Tipos de Ataques: XSS

- **XSS** es un ataque que consiste en **inyectar scripts maliciosos en páginas web** que luego se ejecutan en el navegador de otros usuarios.
- **Objetivo:** Robar cookies, secuestrar sesiones o redirigir a sitios falsos.
- **Ejemplo:** Un comentario en un blog que contiene `<script>alert('hackeado')</script>` y se ejecuta al ser leído por otro usuario.

# XSS en Apache

- Dentro de módulo de **mod\_headers** (ya está incluido en Apache)

# Activa protección contra XSS en navegadores antiguos  
**Header always set X-XSS-Protection "1; mode=block"**

# Evita que el navegador interprete tipos MIME incorrectos  
**Header always set X-Content-Type-Options "nosniff"**

# Evita que el sitio se cargue en un iframe (protege contra clickjacking)  
**Header always set X-Frame-Options "DENY"**

# Política de contenido: solo permite recursos del mismo origen  
**Header always set Content-Security-Policy "default-src 'self'"**

# XSS en Apache

- Estas cabeceras **header** se pueden colocar en **httpd.conf**, en un **<VirtualHost>**, o incluso por carpeta usando **<Directory>**.
- **Dentro de la aplicación de PHP:**
  - Apache no puede escapar contenido dinámico, así que tu aplicación debe:
    - Escapar correctamente HTML, JS y atributos
    - No confiar en entradas del usuario sin sanitizar
    - Usar funciones como `htmlspecialchars()` en PHP

# OpenSSL

# OpenSSL: Uso de Certificados

- Se puede descargar e instalar la herramienta OpenSSL, la versión actual es la 3.x
  - La versión light, es suficiente para las pruebas de los certificados.
  - <https://slproweb.com/products/Win32OpenSSL.html>
- *Pero dentro de la instalación de Apache deberíamos tener el comando openssl dentro de **Apache24/bin**, es una versión más antigua, pero sirve para hacer pruebas con la generación y configuración de certificados.*
- Probar: **openssl version** (la carpeta **bin** debe estar en el **PATH**)
- Devuelve: **OpenSSL 1.1.1t 7 Feb 2023**

# OpenSSL: Uso de Certificados

- Generar la clave privada y el certificado:
- `openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout webcurso.key -out webcurso.crt`
- Common Name (CN) debe coincidir con el nombre del dominio local, por ejemplo: web.curso
- Country Name (2 letter code) [AU]: ES
- State or Province Name (full name) [Some-State]: Madrid
- Locality Name (eg, city) []: Madrid
- Organization Name (eg, company) []: PruebasApache
- Common Name (e.g. server FQDN or YOUR name) []: web.curso



# OpenSSL: Uso de Certificados

- Esto generará dos archivos en la carpeta actual:
  - webcurso.key → clave privada
  - webcurso.crt → certificado público
- Mover los ficheros a la carpeta segura de Apache:
  - D:\apache24\certs\webcurso.key
  - D:\apache24\certs\webcurso.crt

# OpenSSL: Uso de Certificados

- En httpd-vhost:

```
<VirtualHost *:443>
```

```
    ServerName web.curso
```

```
    DocumentRoot
```

```
"D:/antonio2/TRABAJO/CURSOS/PHP8_APACHE/repositorio/Apache/soluciones"
```

```
    SSLEngine on
```

```
    SSLCertificateFile "D:/apache24/certs/webcurso.crt"
```

```
    SSLCertificateKeyFile "D:/apache24/certs/webcurso.key"
```

```
    <Directory
```

```
"D:/antonio2/TRABAJO/CURSOS/PHP8_APACHE/repositorio/Apache/soluciones">
```

```
        Options Indexes FollowSymLinks
```

```
        AllowOverride All
```

```
        Require all granted
```

```
    </Directory>
```

```
</VirtualHost>
```

**Options** Indexes FollowSymLinks

**Indexes:** permite mostrar listado de archivos si no hay index.html

**FollowSymLinks:** permite seguir enlaces simbólicos

**AllowOverride All**

Permite que **.htaccess sobrescriba** cualquier configuración

**Require all granted**

Permite acceso a **todos** los **usuarios** (sin restricciones)

# OpenSSL: Uso de Certificados

- Habilitar SSL en Apache:
  - LoadModule ssl\_module modules/mod\_ssl.so
  - Include conf/extra/httpd-ssl.conf
- Reiniciar Apache
  - Hacer una prueba: <https://web.curso/info.php>