Item 06 – protocolo https

Manejo de conexiones seguras

María Victoria Calbet González

Marta Ramírez González

David Romero Esparraga

Jesús Ortiz Calleja

Guillermo Alcalá Gamero

Juan Carlos Utrilla Martín

Índice

[1. Introducción 2](#_Toc512224414)

[2. Protocolo HTTPS 3](#_Toc512224415)

[2.1. Características técnicas 3](#_Toc512224416)

[2.2. Capas de red 4](#_Toc512224417)

[2.2.1. SSL/TLS 5](#_Toc512224418)

[2.2.1.1. Confidencialidad en la comunicación 8](#_Toc512224419)

[2.2.1.2. Integridad en la comunicación 9](#_Toc512224420)

[2.2.1.3. Autenticación de los interlocutores 9](#_Toc512224421)

[2.2.1.4. Ejemplo – Funcionamiento de HTTPS 9](#_Toc512224422)

[2.3. Funcionamiento mediante certificados 9](#_Toc512224423)

[2.4. Inconvenientes 10](#_Toc512224424)

[3. Procedimientos 10](#_Toc512224425)

[3.1. Verificando el uso de HTTPS 10](#_Toc512224426)

[3.2. Configurando el servidor Apache Tomcat 10](#_Toc512224427)

[3.3. Configurando la aplicación web 13](#_Toc512224428)

[4. Fuentes 15](#_Toc512224429)

# Introducción

La LOPD requiere datos personales o datos que estén regulados por un contrato para ser manejados y comunicados de forma segura. Manejar datos de forma segura requiere que nuestra aplicación maneje los datos de cada cliente de forma independiente de los datos y que mantenga seguras las computadoras en las que almacena y procesa. Las comunicaciones seguras requieren la configuración del servicio Tomcat para que utilice el protocolo HTTPS. (Tenga en cuenta que no es la versión de desarrollo que utiliza Eclipse o por medio de Maven en la configuración de desarrollo, sino el servicio Tomcat que utiliza sus configuraciones previas o de producción. Para obtener un A+ debe:

1. Actualizar su plantilla de proyecto a la versión 1.14 para que pueda manejar comunicaciones seguras. Tenga en cuenta que se espera que use esta plantilla para los próximos proyectos, incluso si no opta por ganar un A+ en los próximos entregables.
2. Use una nueva plantilla para producir la nueva versión del proyecto “Acme-Rendezvous” que use el protocolo HTTPS cuando corresponda.
3. Escriba un informe en el que explique a los profesores que tienen que hacer para verificar tu proyecto. Proporcione archivos de configuración previa a la producción y ejecutar su proyecto utilizando comunicaciones seguras.

# Protocolo HTTPS

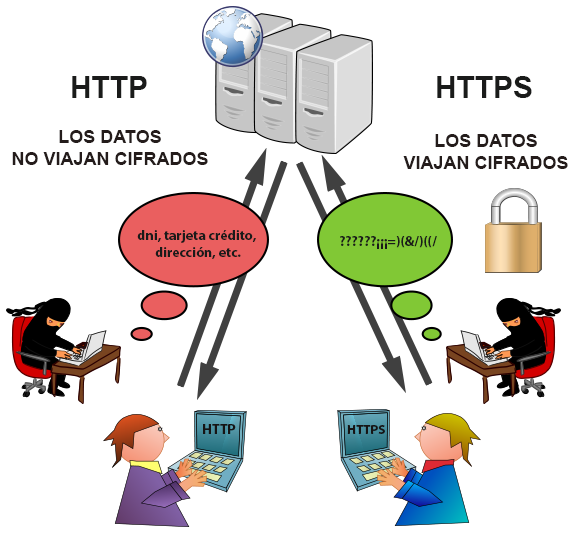
**HTTPS (Hypertext Transfer Protocol Secure)** es un protocolo de la capa de aplicación basado en el protocolo HTTP, destinado a la transferencia segura de datos de datos de Hipertexto, es decir, es la versión segura de HTTP.

* **¿Cómo funciona HTTPS? – Desde cero:** <https://www.youtube.com/watch?v=3nl6_ZOuIps>
* **Protocolos para la comunicación segura**: SSL/TLS (URJCx): <https://www.youtube.com/watch?v=6Y2epibeYYI>

## Características técnicas

El protocolo HTTPS utiliza un cifrado basado en SSL/TLS para crear un canal cifrado (cuyo nivel de cifrado depende del servidor remoto y del navegador utilizado por el cliente) para el tráfico de información sensible que el protocolo HTTP. De esta forma, se consigue que la información sensible no pueda ser usada por un atacante que haya conseguido interceptar la transferencia de datos de la conexión, ya que lo único que obtendrá será un flujo de datos cifrados que le resultará imposible de descifrar.

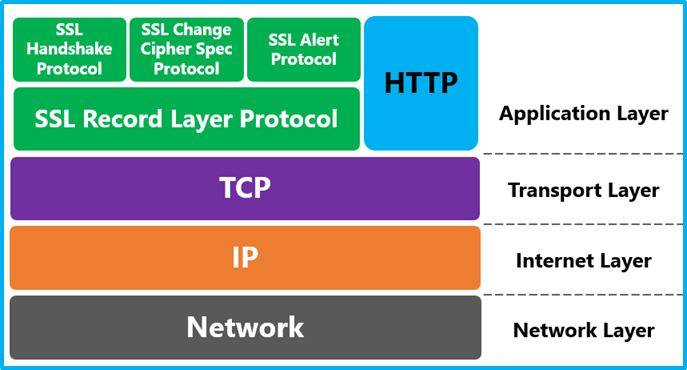
El puerto estándar para este protocolo es el 443.



|  |
| --- |
| HTTP es inseguro y está sujeto a ataques man-in-the-middle y eavesdropping que pueden permitir al atacante obtener acceso a cuentas de un sitio web e información confidencial. HTTPS está diseñado para resistir esos ataques y ser más seguro. |

## Capas de red

HTTP opera en la capa más alta del modelo OSI, la capa de aplicación; pero el protocolo de seguridad opera en una subcapa más baja, cifrando un mensaje HTTP previo a la transmisión y descifrando un mensaje una vez recibido. Estrictamente hablando, HTTPS no es un protocolo separado, pero prefiere el uso del HTTP normal sobre SSL o TLS.



* + 1. SSL/TLS

Introducción

Hoy en día existen multitud de protocolos criptográficos, siendo un poco similar a una sopa de letras – SSL, TLS, ECC, SHA. Todas estas abreviaturas pueden hacer confuso para las personas entender que es lo que necesitan. Una de las preguntas más frecuentes es: ¿Cuál es la diferencia entre SSL (Secure Sockets Layer) y TLS (Transmission Layer Security)? ¿Qué necesitamos, SSL, TLS, o los dos? Veamos un poco para qué sirven.

Comencemos

SSL y TLS son protocolos criptográficos los cuales nos permiten garantizar durante el proceso de comunicación entre dos entidades en las que se intercambian mensajes:

* **Confidencialidad de la comunicación**: mediante el cifrado de la información.
* **Integridad de la comunicación**: mediante algoritmos de Hashing.
* **Autenticación de los interlocutores**: mediante la utilización de certificados.

**TLS es el sucesor de SSL**. Con los años, nuevas versiones de protocolos han sido desarrolladas para enfrentar las vulnerabilidades y para entregar cifrado y algoritmos más fuertes.

SSL fue desarrollado originalmente por Netscape y fue introducido en 1995 con el SSL 2.0 (el 1.0 nunca fue lanzado al público). La versión 2.0 fue rápidamente reemplazada por el SSL 3.0 en 1996 después de que un número de vulnerabilidades fueran encontradas.

|  |
| --- |
| **Nota**: las versiones de SSL 2.0 y 3.0 son algunas veces escritas como SSLv2 y SSLv3. |

**TLS fue introducido en 1999 como una nueva versión del SSL y fue basada en SSL 3.0.**

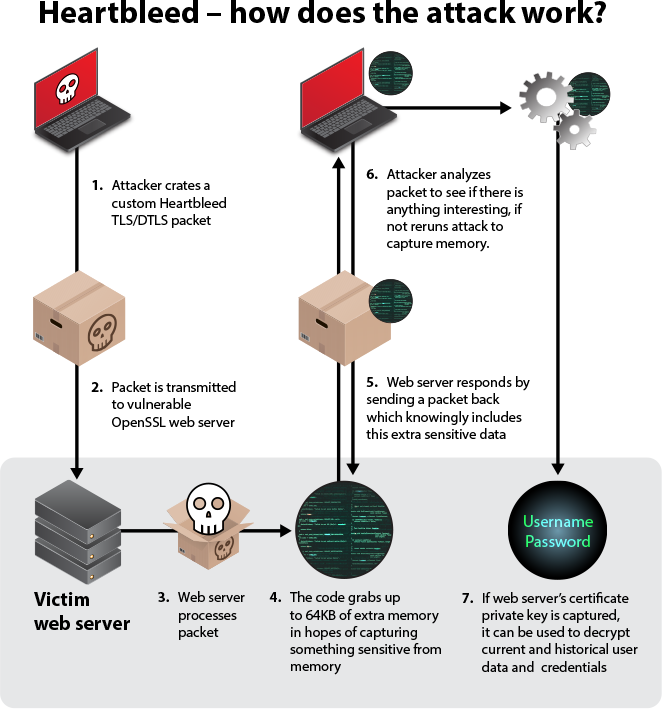
Las diferencias entre TLS y SSL 3.0 no es dramática, pero es lo suficientemente significativa para que el TLS 1.0 y el SSL 3.0 no puedan interoperar ([enlace](https://tools.ietf.org/html/rfc2246))

¿Cuál debería usar? ¿SSL o TLS?

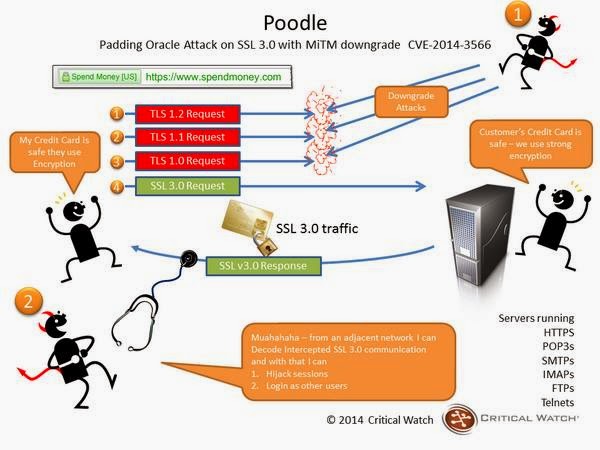
**Tanto SSL 2.0 y SSL 3.0 han sido depreciados por el IETF** (en [2011](https://tools.ietf.org/html/rfc6176) y [2015](https://tools.ietf.org/html/rfc7568), respectivamente). A lo largo de los años vulnerabilidades han sido encontradas en los protocolos SSL. Por estas razones, **usted debe deshabilitar el SSL 2.0 y 3.0 en su configuración del servidor, dejando solo los protocolos TLS habilitados**.

Ataques realizados en los últimos años a HTTPS

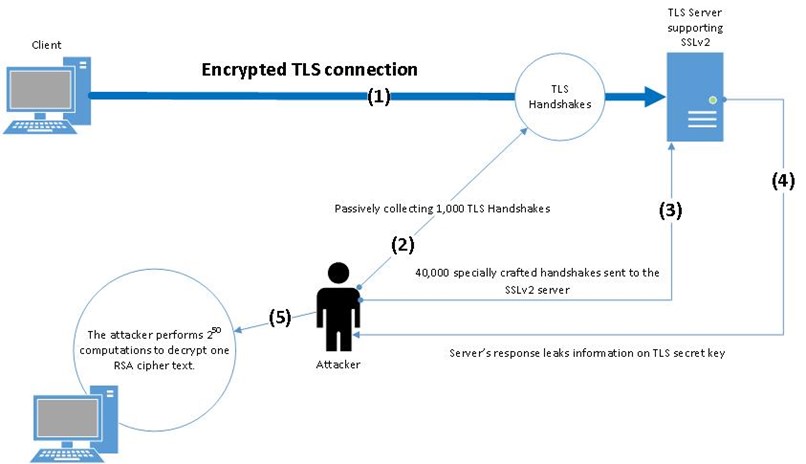
* **Heartbleed (abril 2014)**: es un agujero de seguridad de software en la biblioteca de código abierto OpenSSL, sólo vulnerable en su versión 1.0.1f, que permite a un atacante leer la memoria de un servidor o un cliente, permitiendo, por ejemplo, conseguir las claves privadas SSL del servidor. ([enlace1](https://es.wikipedia.org/wiki/Heartbleed), [enlace2](http://www.elladodelmal.com/2014/04/heartbleed-y-el-caos-de-seguridad-en.html))



* **POODLE (octubre 2014):** exploit man-in-the-middle que aprovecha Internet y la característica del software de clientes de bajar el nivel de seguridad a SSL 3.0 ([enlace1](https://es.wikipedia.org/wiki/Ataque_POODLE), [enlace2](https://www.genbeta.com/seguridad/poodle-asi-es-el-ataque-que-deja-por-fin-obsoleto-a-sslv3), [enlace3](https://www.globalsign.com/en/blog/poodle-vulnerability-in-ssl-30/)).



* **Logjam (mayo 2015)**: vulnerabilidad que aprovecha varias debilidades en la implementación del intercambio de claves Diffie-Hellmnan, un algoritmo criptográfico que permite que dos partes establezcan una clave secreta compartida para crear conexiones seguras, permitiendo a un atacante bajar el nivel de calidad de la seguridad hasta los 512 bits, un nivel de cifrado muy sencillo de romper, y que facilitaría las cosas para romper las conexiones más seguras ([enlace1](https://en.wikipedia.org/wiki/Logjam_(computer_security)), [enlace2](https://www.muyseguridad.net/2015/05/22/logjam-tls/), [enlace3](https://omicrono.elespanol.com/2015/05/todo-lo-que-necesitas-saber-de-logjam-la-nueva-vulnerabilidad-en-ssl/), [enlace4](https://weakdh.org/)).
* **DROWN (marzo 2016):** vulnerabilidad que puede descifrar las sesiones TLS, aunque no se trata de una vulnerabilidad del protocolo TLS en sí misma. Básicamente lo que hacen es aprovecharse de que un servidor permite vulnerabilidades SSL 2.0 inseguras y conexiones TLS para capturar las sesiones TLS y descifrar las claves RSA que se usan en la conexión TLS mediante SSL 2.0 Es decir, tal y como dice el paper, necesitan capturar unas 1.000 sesiones TLS utilizando intercambios de claves RSA, después hacer unas 40.000 conexiones SSL 2.0 y realizar 2^50 operaciones de cifrado simétrico para descifrar una sesión TLS ([enlace1](https://es.wikipedia.org/wiki/Ataque_DROWN), [enlace2](https://www.muyseguridad.net/2016/03/04/drown-ataque-jaque-criptografia/), [enlace3](https://www.globalsign.com/en/blog/drown-attack-sslv2/), [enlace4](http://www.elladodelmal.com/2016/03/drown-attack-descifrar-https-tls-por.html)), [enlace5](https://drownattack.com/)) .



#### Confidencialidad en la comunicación

Para garantizar la confidencialidad entre emisor y receptor, debemos conocer los siguientes conceptos que se desarrollan a continuación.

Tipos de cifrado aplicados a HTTPS

**Gracias a su velocidad para cifrar grandes cantidades de datos,** **HTTPS utiliza cifrado simétrico** para la encriptación de los datos, como por ejemplo AES.

Una vez obtenida esta clave, se emplea un **cifrado asimétrico** para compartirla entre el emisor y el receptor.

¿Cómo se calculan/comparten las claves de cifrado entre cliente y servidor?

Para garantizar la confidencialidad con un cifrado simétrico, debemos tener en emisor y receptor las mismas claves de cifrado. Para obtener esas claves de forma segura hay principalmente dos métodos:

* **Generación de claves Diffie-Hellman**: tanto cliente como servidor obtienen el ***master secret*** compartiendo ciertos datos que pueden ser públicos, de tal forma que cada entidad añade a los cálculos unos parámetros que al compartir resultados intermedios llegarán a la misma clave de encriptación.
* **Intercambio de claves**: tanto cliente como servidor generan una serie de cadenas únicas denominadas ***nonces*** que se intercambian. El cliente genera una clave ***pre-master secret*** que será enviada al servidor de forma segura (por ejemplo, encriptándolo con la clave pública del servidor).

A partir de los **nonces** y el **pre-master secret** y mediante un generador pseudoaleatorio se genera el mismo ***master secret*** tanto para el cliente como para el servidor.

Gracias a la master secret y unos nuevos nonces, tanto cliente como servidor generan las claves de sesión necesarias para el cifrado simétrico de los datos.

Además, es importante destacar que se generan distintas claves para cifrar el tráfico de cliente a servidor que, de servidor a cliente, evitando así la excesiva reutilización de claves.

#### Integridad en la comunicación

Para garantizar la integridad de los mensajes y que otros no puedan modificarlo, se utilizan **algoritmos de hashing**, como por ejemplo SHA256.

#### Autenticación de los interlocutores

Para la autenticación de los interlocutores, nos basaremos en el empleo de **certificados**.

## Funcionamiento mediante certificados

Para que el sistema pueda funcionar, se debe implementar el **esquema de “Certificado”**, teniendo que estar esté debidamente firmado por una autoridad. En el caso de los navegadores web, este método es transparente para el usuario; ya que los certificados necesarios como para poder explorar Internet sin problemas son obtenidos con el navegador.

Un **certificado** es un fichero que usa una firma digital para enlazar una **clave pública**

<https://latam.kaspersky.com/blog/explicativo-certificados-digitales-y-https/615/>

|  |  |
| --- | --- |
| *keytool* nos solicitará dos claves:   * **Clave pública**: cuando se quiere enviar un mensaje, el emisor busca la clave pública con esa clave. * **Clave privada**: una vez que el mensaje cifrado llega al receptor, este se ocupa de descrifrarlo con su clave privada | Esquema de criptografía asimétrica |

## Inconvenientes

**La velocidad en HTTPS es menor que en HTTP**. Esto se debe a que la labor de cifrado y descifrado que ha de hacer con los datos genera un mayor consumo de banda y, además, una mayor lentitud.

# Procedimientos

## Verificando el uso de HTTPS

Para verificar el uso de HTTPS utilizaremos el entorno de Pre-producción. Para verificar que el proyecto funciona correctamente y que utiliza dicho protocolo, introduciremos la siguiente URL en el navegador:

[www.acme.com](http://www.acme.com)

El servidor nos redirigirá de forma automática a la URL <https://www.acme.com:8443>.

## Configurando el servidor Apache Tomcat

Para configurar el servidor Apache Tomcat para el uso de HTTPS debemos de seguir los siguientes pasos:

* **Paso 1**: los sistemas Java usan un almacén de certificados propio o ***keystore*** que es la base de datos de las claves privadas y sus certificados asociados. Para manipular este almacén, Java cuenta con la utiliza ***keytool*** que se encuentra distribuida entre los binarios del **entorno de desarrollo Java**.

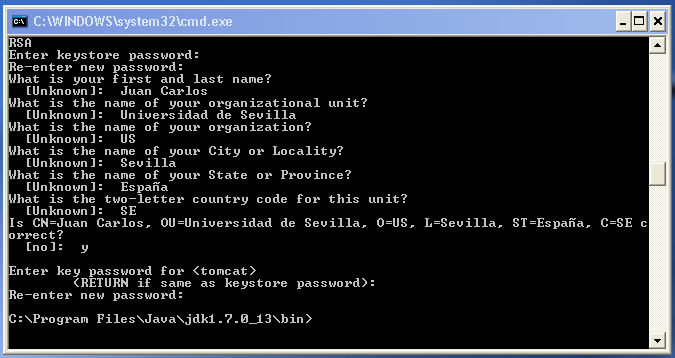
**Utilizaremos *keytool* para la generación de pares de claves**, ya que automáticamente almacena **la clave pública dentro de un certificado autofirmado**. Para generar el *keystore* con el certificado autofirmado, ejecutamos los siguientes comandos:

cd %JAVA\_HOME%/bin

keytool.exe -genkey -alias tomcat -keyalg RSA

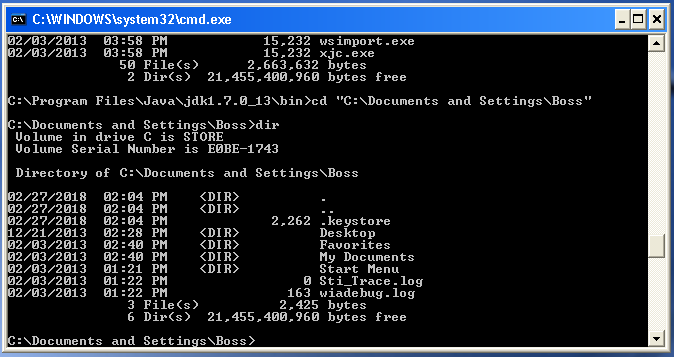
|  |
| --- |
| **Nota**: %JAVA\_HOME% es una variable de entorno de Windows que indica el directorio donde se encuentra instalado Java Development Kit (JDK). En nuestro caso, se encuentra en:  C:/Program Files/Java/jdk1.7.0\_13 |

Acto seguido, la consola de comandos nos realizará una serie de preguntas las cuales debemos ir respondiendo. Si todos los datos son correctos, al final introducimos un ‘y’.



* **Paso 2**: como resultado, se generará un fichero “.keystore” en la ruta:

C:\Documents and Settings\Boss



* **Paso 3**: ahora nos centraremos en configurar el servidor de aplicaciones Apache Tomcat sobre SSL necesaria para el protocolo HTTPS. Para ello, se debe editar la siguiente configuración que habilitará el puerto para que indiquemos para todo el tráfico HTTPS en el fichero *server.xml* ubicado en la siguiente ruta:

%CATALINA\_BASE%/conf/server.xml

|  |
| --- |
| **Nota**: %CATALINA\_BASE % es una variable de entorno de Windows que indica el directorio donde se encuentra instalado el servidor de aplicaciones Apache Tomcat. En nuestro caso, se encuentra en:  C:/Program Files/Apache Software Foundation/Tomcat 7.0 |

Añadimos:

<Connector port="8443"

protocol="org.apache.coyote.http11.Http11NioProtocol"

SSLEnabled="true"

maxThreads="150" scheme="https" secure="true"

disableUploadTimeout="true"

clientAuth="false" sslProtocol="TLS"

enableLookups="false" acceptCount="100"

keystoreFile="C:\Documents and Settings\Boss\.keystore"

keystorePass="nuestraContraseña"/>

* **Paso 4:** para verificar los cambios realizados, reiniciamos el servidor y escribimos:

[**https://localhost:8443/manager**](https://localhost:8443/manager)

|  |
| --- |
| **Nota**: como es lógico la anterior URL <http://localhost/manager> ha dejado de funcionar. |

|  |
| --- |
| **Problema**: dependiendo del navegador, puede mostrar un error de validación debido a que se está accediendo a través de una IP cuando **un certificado debe estar ligado siempre a un dominio**.    **Solución**: esto desaparecerá cuando se configure el dominio a quien pertenece el certificado Apache Tomcat. |

Si hemos seguido todos los pasos, nuestro servidor de aplicaciones estará preparado para realizar conexiones seguras mediante HTTPS.

## Configurando la aplicación web

Para configurar una aplicación web para el uso de HTTPS debemos de seguir los siguientes pasos:

* **Paso 1**: modificaremos el fichero *web.xml* de la aplicación, en este caso, “*Acme-Rendezvous*”:

<security-constraint>

<web-resource-collection>

<web-resource-name>Acme-Rendezvous</web-resource-name>

<url-pattern>/\*</url-pattern>

</web-resource-collection>

<user-data-constraint>

<transport-guarantee>CONFIDENTIAL</transport-guarantee>

</user-data-constraint>

</security-constraint>

|  |
| --- |
| CONFIDENTIAL se utiliza para asegurarnos de que nuestra aplicación web trabajará con SSL. |

|  |
| --- |
| **Nota**: es necesario reiniciar el servidor Apache Tomcat para que los cambios funcionen. |

* **Paso 2**: exportamos el proyecto war, preparamos los scripts y lo subimos a la url:

[**https://localhost:8443/manager**](https://localhost:8443/manager)

# Fuentes

* **HTTPS - así funciona**: <https://www.genbeta.com/web/https-asi-funciona>
* **¿Cómo funciona HTTPS? – Desde cero:** <https://www.youtube.com/watch?v=3nl6_ZOuIps>
* **Protocolos para la comunicación segura**: SSL/TLS (URJCx): <https://www.youtube.com/watch?v=6Y2epibeYYI>
* **Apuntes HTTPS:** <https://www.studocu.com/es/institution/universidad-rey-juan-carlos/8910>
* **Información sobre el protocolo HTTPS**: <https://es.wikipedia.org/wiki/Protocolo_seguro_de_transferencia_de_hipertexto>
* **Transport Layer Security (SSL y TLS)**: <https://es.wikipedia.org/wiki/Transport_Layer_Security>
* **Diferencias entre SSL y TLS:** <https://www.globalsign.com/es/blog/ssl-vs-tls-difference/>
* **The TLS Protocol Versión 1.0**: <https://tools.ietf.org/html/rfc2246>
* **Prohibiting Secure Sockets Layer (SSL) Versión 2.0 (March 2011):** <https://tools.ietf.org/html/rfc6176>
* **Deprecating Secure Sockets Layer Version 3.0 (June 2015)**: <https://tools.ietf.org/html/rfc7568>
* **The DROWN Attack:** <https://drownattack.com/>
* **Ataque POODLE:** <https://es.wikipedia.org/wiki/Ataque_POODLE>
* **POODLE, así es el ataque que deja (por fin) obsoleto a SSLv3:** <https://www.genbeta.com/seguridad/poodle-asi-es-el-ataque-que-deja-por-fin-obsoleto-a-sslv3>
* **POODLE Vulnerability in SSL 3.0**: <https://www.globalsign.com/en/blog/poodle-vulnerability-in-ssl-30/>
* **Ataque DROWN:** <https://es.wikipedia.org/wiki/Ataque_DROWN>
* **DROWN, la vulnerabilidad que ha puesto en jaque a la criptografía**: <https://www.muyseguridad.net/2016/03/04/drown-ataque-jaque-criptografia/>
* **DROWN Attack – Descifrar HTTPs TLS por bugs en SSLv2:** <http://www.elladodelmal.com/2016/03/drown-attack-descifrar-https-tls-por.html>
* **The DROWN Attack Vulnerability and Changing Your Server Configuration:** <https://www.globalsign.com/en/blog/drown-attack-sslv2/>
* **Imagen – HTTP vs HTTPS**: <http://pablitholadera.blogspot.com.es/2015/08/la-diferencia-entre-el-http-y-https.html>
* **Imagen – Capas de red de HTTPS**: <https://blogs.msdn.microsoft.com/kaushal/2013/08/02/ssl-handshake-and-https-bindings-on-iis/>
* **SSL/TLS How to – Apache Tomcat 7.0**: <https://tomcat.apache.org/tomcat-7.0-doc/ssl-howto.html>
* **Criptografía simétrica, asimétrica e híbrida:** <https://www.genbetadev.com/seguridad-informatica/tipos-de-criptografia-simetrica-asimetrica-e-hibrida>
* **Cifrado de correo con GPG para dummies:** <https://victorhckinthefreeworld.com/2017/07/14/explicando-que-es-el-cifrado-de-correo-con-gpg-para-dummies/>
* **Instalar Tomcat e importar el certificado SSL existente**: <https://raiolanetworks.es/blog/instalar-tomcat-e-importar-certificado-ssl-existente/>
* **A Simple Step-By-Step Guide To Apache Tomcat SSL Configuration**: <https://www.mulesoft.com/tcat/tomcat-ssl>
* **RSA:** <https://es.wikipedia.org/wiki/RSA>