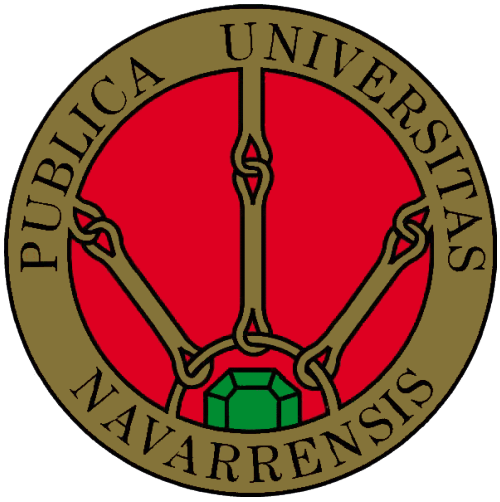
UNIVERSIDAD PÚBLICA DE NAVARRA
NAFARROAKO UNIBERTSITATE PUBLIKOA

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

INDUSTRI ETA TELECOMUNIKAZIO INGENIARIEN GOI ESKOLA TEKNIKOA



Máster en Comunicaciones

"Caracterización y medida pasiva del rendimiento para conexiones Web seguras HTTPS"

Autor del trabajo Fin de Máster: Mikel López Fernández
Director: Dr. Eduardo Magaña Lizarrondo
Pamplona, 18 de septiembre de 2015

Abstract

Secure Sockets Layer (SSL) and Transport Layer Security (TLS) are the most popular protocol used in the World Wide Web to enable secure communications. These protocols provide the essential ingredients to implement privacy, integrity and authentication. Though it is well understood that security always comes at the cost of performance, this cost depends on the cryptographic algorithms. The objective of this project is to characterize the impact of SSL on HTTP connections. To characterize a secure connection, different parameters such as the total number of packets, TCP data exchanged and the time required to download an object will be analyzed. Furthermore, they will be compared with the results obtained from a non secure connection. This will make possible to know the data and processing overhead generated by the cryptographic algorithms. With the conclusions, we will provide a or rule function to characterize any HTTPS connection.

Índice

Índice de figuras

Índice de tablas

1 Introducción

En sus inicios, Internet no fue concebida como una red intrínsecamente segura. Su función principal era lograr comunicaciones eficaces, sin preocuparse por si los datos que se intercambiaban podían ser interceptados por terceras personas. Rápidamente, la red fue creciendo y evolucionando tecnológicamente, llegando hasta el punto de ofrecer servicios como el comercio electrónico o la banca online los cuales necesitan de cierta seguridad. En un principio, esta seguridad se ofreció mediante soluciones a nivel de aplicación, aunque más tarde se vio la necesidad de incorporar mecanismos de seguridad a más bajo nivel.

Con el objetivo de poder establecer un canal seguro más apropiado para el intercambio de la información sensible, Netscape Communications desarrolló SSL (Secure Sockets Layer) [1] y posteriormente el IETF (Internet Engineering Task Force – Grupo de Trabajo de Ingeniería de Internet) se basó en SSL para definir TLS (Transport Layer Security) [2]. Ambas soluciones operan entre la capa de transporte y la de sesión del modelo OSI (Open System Interconnection) [3] o entre la capa de transporte y aplicación en el modelo TCP/IP [4]. Por otro lado, también se ha desarrollado otra solución denominada IPsec (Internet Protocol security) [5] la cual a pesar de operar sobre la capa de red dispone de los mismos mecanismos de seguridad que SSL/TLS.

El hecho de utilizar SSL/TLS para establecer una comunicación segura no solamente provee de confidencialidad sino que también proporciona integridad y autenticación de la identidad (como mínimo por parte del servidor) todo ello a costa de una sobrecarga tanto de datos como de procesado.

Actualmente, existen diferentes estudios acerca de la sobrecarga de procesado que genera SSL/TLS en los servidores Web y en los que se proponen diferentes optimizaciones. Concretamente, K.Kant [6] estudia el rendimiento de los servidores Web cuando se usa HTTPS (Hipertext Transfer Protocol Secure) [7] frente HTTP (Hipertext Transfer Protocol) [8] pero sin detallar de donde viene la sobrecarga. Otros estudios [9][10][11] se han centrado en la criptografía y en buscar mejoras que permitan acelerar las operaciones criptográficas. Con la realización de este trabajo se

pretende obtener toda la información posible de una conexión HTTPS con el fin de caracterizarla y observar el rendimiento que ofrece el servidor Web. Se han probado distintos sistemas de cifrado con el objetivo de comprobar la sobrecarga de procesado y especialmente la sobrecarga de datos que genera cada uno de ellos.

Para llevar a cabo este trabajo, se ha desplegado una maqueta red compuesta por dos equipos, uno de ellos actúa como servidor Web y otro como cliente. Desde el cliente utilizando el navegador Web Mozilla Firefox se han realizado peticiones de objetos con diferentes tamaño y utilizando distintos sistemas de cifrado. Además, también se han realizado peticiones sin utilizar ningún sistema criptográfico. De esta manera a sido posible comparar ambos resultados pudiendo caracterizar dicha conexión y establecer una regla general que permita caracterizar todas las conexiones HTTPS.

En los siguientes puntos se detallan los objetivos y las motivaciones de este trabajo, un estado del arte sobre SSL/TLS, como se ha desarrollado el trabajo, los resultados obtenidos y un análisis de estos resultados.

2 Objetivos y motivaciones

Normalmente, cuando se navega por la World Wide Web se utiliza el protocolo HTTP, que simplemente establece unas directrices acerca de cómo se va a comunicar nuestro equipo con el servidor donde está alojada la página Web. En este caso, no se hace uso de ningún mecanismo de seguridad y por lo tanto los datos pueden ser leídos y modificados por agentes intrusos. Si se quisiera establecer un canal cifrado sería necesario utilizar el protocolo HTTPS el cual emplea SSL/TLS como protocolo criptográfico para el establecimiento de conexiones seguras.

El hecho de establecer una comunicación segura implica una mayor dificultad en identificarla y caracterizarla. Con la realización de este proyecto se pretende obtener la mayor información del tráfico HTTPS. Por ello, el objetivo principal de este proyecto consiste en analizar el tráfico de una comunicación HTTPS. Identificando previamente la sesión SSL/TLS, se han evaluado distintos parámetros como el número total de paquetes, el número total de Bytes intercambiados a nivel de TCP y el tiempo empleado para la descarga del objeto. Los resultados obtenidos se han comparado con los resultados de una comunicación no segura HTTP logrando de esta forma identificar la sobrecarga de procesado y especialmente la sobrecarga de datos que introduce la utilización de SSL/TLS.

Otro de los objetivos que se ha planteado con el desarrollo de este proyecto, ha sido establecer una regla o una fórmula que permita caracterizar en todo momento cualquier conexión HTTPS independientemente del servidor Web.

3 Estado del arte

Hoy en día, la mayoría de comunicaciones seguras se establecen mediante SSL/TLS. La gran difusión de estos protocolos, la facilidad de implantación por parte de los administradores de sistemas y la transparencia para el usuario han convertido a estos protocolos en los dominantes en un terreno donde las soluciones que existen no han avanzado o son más difíciles de implementar.

En 1994 Netscape Communications [12] desarrolló la primera versión de SSL. Esta versión jamás fue implementada de forma pública debido su cantidad de fallos de seguridad. Tan sólo unos pocos meses después, concretamente en febrero de 1995, se liberó una actualización importante que vino a llamarse SSL 2.0 y a pesar de no haber subsanado ciertos errores de diseño, esta versión sí que tuvo una implementación real. Posteriormente, en noviembre de 1995, Netscape publicó la especificación de SSL 3.0, la cual se ha convertido a día de hoy en el estándar 'de hecho' para las comunicaciones seguras entre clientes y servidores en Internet.

El objetivo principal de Netscape Communications era crear un canal de comunicaciones seguro entre cliente y servidor, que fuese independiente del sistema operativo y además, se beneficiará de forma dinámica y flexible de los nuevos adelantos en materia de cifrado a medida que estos estuvieran disponibles.

Años más tarde, en 1999, el IETF se basó en las especificaciones previas de SSL desarrolladas por Netscape Communications para definir TLS. La primera versión de este protocolo, TLS 1.0 [13] es simplemente una actualización de SSL 3.0 sin ninguna diferencia importante. Tal y como se indica en su RFC (RFC 4346) [13] simplemente se impide la interoperabilidad entre TLS 1.0 y SSL 3.0. Posteriormente, en abril de 2006, se define la versión TLS 1.1 [14] la cual implementa diferentes mejoras como la protección contra ataques cuando se utilizan sistemas de cifrado basados en bloque [15] y el soporte para registro de parámetros IANA (Internet Assigned Numbers Authority) [16]. Dos años después, en agosto del 2008, se define la versión TLS 1.2 [17] siendo esta la solución más utilizada en la actualizad para el establecimiento de una conexión segura HTTPS. La principal diferencia frente a las demás versiones es que

introduce el soporte de MD5 [18] para el cálculo del código de autenticación del mensaje. Además, esta versión fue redefinida en marzo de 2011 para evitar que cliente y servidor negocien el uso de la ya extinta versión SSL 2.0 [19].

Ya en otro ámbito, se definió IPsec que a pesar de operar sobre la capa de red incorpora mecanismos de confidencialidad, integridad y autenticación al igual que SSL/TLS. Además, por el hecho de operar sobre la capa de red posee protecciones efectivas contra la repetición de tramas y es capaz de trabajar con distintos protocolos de la capa de transporte. Por otro lado, la complejidad en cuanto a implantación y el mantenimiento han reducido su ámbito de actuación, al menos por el momento. Aunque también es cierto, que algunos autores insisten que IPsec y SSL/TLS pueden coexistir juntos ya que ofrecen soluciones óptimas para diferentes problemas.

Para la realización de este trabajo se ha empleado SSL 3.0 ya que es protocolo de seguridad “padre” de todas las versiones posteriores de TLS. En los siguientes apartados se detallan los servicios que proporcionan, su arquitectura y funcionamiento.

3.1 Secure Sockets Layer, SSL

3.1.1 Servicios de seguridad

Tanto las versiones de SSL como las versiones de TLS proporcionan ciertos servicios de seguridad, estos son la confidencialidad, la autenticación de la entidad y la integridad del mensaje. Además, estos protocolos se diseñaron con criterios adicionales como la eficiencia y extensibilidad.

3.1.1.1 Confidencialidad

El flujo normal en una conexión SSL/TLS consiste en intercambiar paquetes con los datos cifrados por un sistema de claves simétricas (o de clave privada) seleccionado en el establecimiento de la sesión.

Por otro lado, para evitar que terceras personas escuchen el dialogo inicial identificando la clave acordada por los dos extremos de la comunicación, se sigue un mecanismo de intercambio de claves basado en criptografía de clave asimétrica (o de clave pública). El sistema criptográfico para este intercambio también se negocia durante el establecimiento de la conexión.

3.1.1.2 Autenticación de la entidad

Mediante un protocolo de reto-respuesta basado en firmas digitales el cliente puede confirmar la identidad del servidor al cual quiere conectarse. Para validar las firmas el cliente necesita conocer la clave pública del servidor la cual la obtiene a través del certificado digital. Además el cliente puede conocer si el certificado ha sido validado por una autoridad certificadora (CA – Certication Authority) para garantizar que el poseedor del certificado digital es quien dice ser.

Por otra parte, SSL/TLS también dispone de la autenticación del cliente frente al servidor, aunque no se suele utilizar ya que en vez de realizar una autenticación del cliente a nivel de transporte son las propias aplicaciones las que utilizan su propio método de autenticación.

3.1.1.3 Integridad del mensaje

Cada paquete enviado en una conexión SSL/TLS además de ir cifrado incorpora un código de autenticación de mensaje (MAC – Message Authentication Code) que permite al destinatario comprobar que el paquete no ha sido modificado. Las claves secretas para el cálculo de los códigos MAC, una para cada sentido de la comunicación, también se acuerda de forma segura en el establecimiento de la sesión.

3.1.1.4 Eficiencia

Son dos las características del protocolo SSL las que permiten mejorar la eficiencia de la comunicación, una está relacionada con la definición de sesiones y la otra con la posibilidad de comprimir los datos intercambiados.

Si el cliente necesita establecer dos o más conexiones simultáneas o muy seguidas, en lugar de repetir el proceso de autenticación y el intercambio de claves (operaciones computacionalmente costosas ya que intervienen sistemas de cifrado de clave pública), se reutilizan los parámetros previamente acordados. Si se hace uso de esta opción, se considera que la nueva conexión pertenece a la misma sesión anterior. Por lo tanto, existe un identificador de sesión, que permite conocer si la conexión empieza con una sesión nueva o es continuación de otra.

Por otra parte, debido a la sobrecarga de tráfico que introduce el establecimiento de la sesión y el cifrado de los datos para cada conexión SSL permite la negociación de algoritmos de compresión.

3.1.1.5 Extensibilidad

En el establecimiento de la sesión entre cliente y servidor se negocian los algoritmos que se utilizarán para la autenticación y el cifrado de los datos. Las especificaciones del protocolo SSL ya incluye varias combinaciones predefinidas pero es posible añadir nuevos algoritmos de cifrado que sean más eficientes o seguros.

3.1.2 Arquitectura de SSL

El protocolo de seguridad SSL está diseñado para trabajar por encima del protocolo TCP y por debajo de protocolos como HTTP, IMAP y LADP entre muchos otros, pudiendo ser utilizado por todos ellos de forma transparente para el usuario. Opera entre la capa de transporte y la de sesión del modelo OSI, o entre la capa de transporte y de aplicación del modelo TCP/IP, y a su vez, está formado por dos capas y cuatro componentes bien definidos.

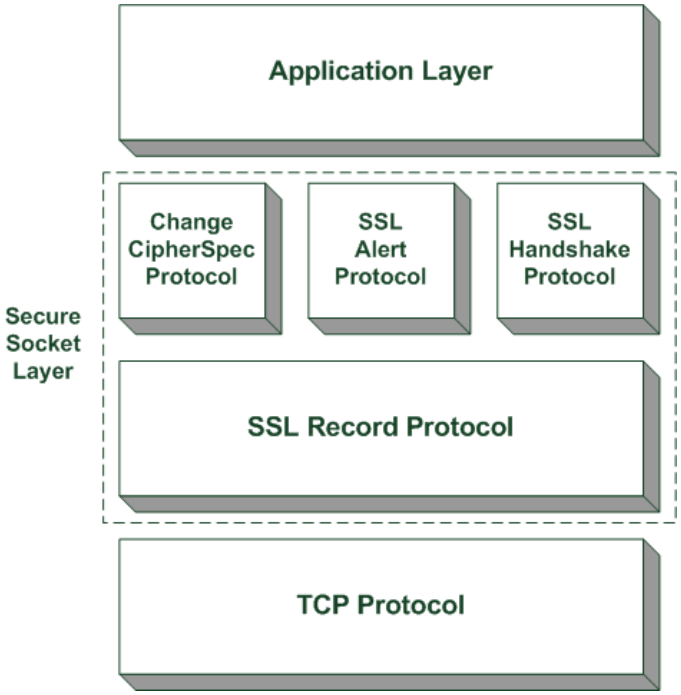


Figura 1. Arquitectura de SSL [32]

En la capa inferior, se define el protocolo SSL Record Protocol, el cual proporciona servicios de seguridad básicos a diversos protocolos de capas superiores, en particular al protocolo HTTP. Los tres protocolos de las capas superiores se definen como la parte lógica de SSL: Handshake Protocol, ChangeCipherSpec Protocol y Alert Protocol. Estos protocolos específicos se utilizan para la gestión de la comunicación segura entre cliente y servidor.

Antes de continuar con la explicación sobre el protocolo SSL, es necesario aclarar dos conceptos muy importantes en SSL que son la sesión y la conexión. La sesión se establece mediante el Handshake Protocol y es una asociación entre cliente y servidor en la cual se definen un conjunto de parámetros de seguridad criptográfica. Por otro

lado, una conexión es un enlace lógico entre cliente y servidor en el que se utiliza el sistema de cifrado seleccionado por los dos extremos para el intercambio de la información cifrada.

Por lo tanto, para que cliente y servidor puedan establecer un canal de comunicación seguro primeramente han de asociarse mediante una sesión y a continuación ya es posible establecer múltiples conexiones seguras. En teoría, también es posible establecer varias sesiones simultáneas pero supondría una mayor sobrecarga de procesado por lo que no se utiliza en la práctica.

3.1.3 SSL Record Protocol

El SSL Record Protocol proporciona dos servicios para las conexiones SSL. La confidencialidad, mediante el cifrado de los datos, y la integridad del mensaje mediante el uso de un código de autentificación de mensaje MAC.

El funcionamiento normal de este protocolo consiste en fragmentar los datos que le llegan desde las capas superiores, comprimirlos y cifrarlos con el algoritmo seleccionado por el cliente y servidor en el establecimiento de la sesión Fig. 2.

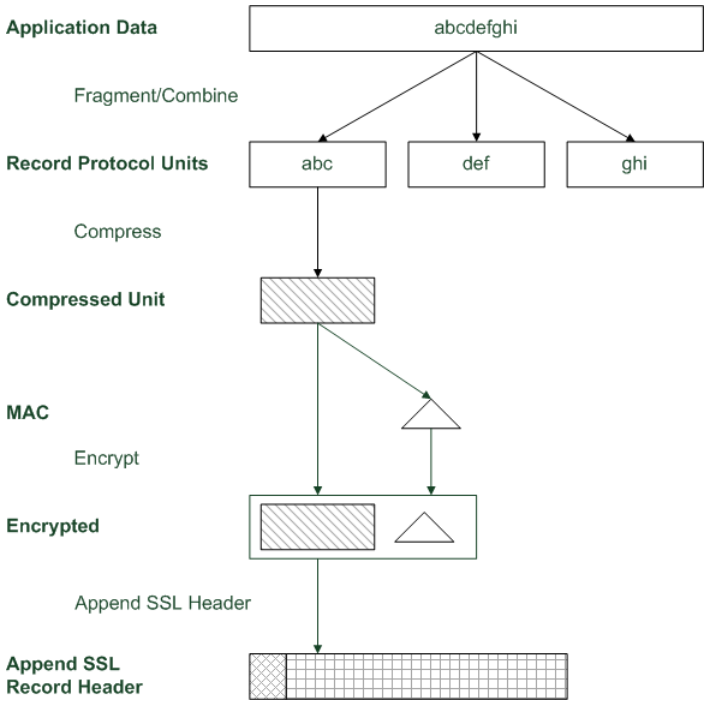


Figura 2. Modo de operación de SSL Record Protocol [33]

El primer paso es la fragmentación, se recogen los datos de la capa de aplicación y se fragmentan en bloques más manejables, concretamente en bloques de un tamaño máximo de 2^{14} bytes (16384 bytes). Estos bloques reciben el nombre de registros. A continuación, se aplica opcionalmente una compresión, ya que en SSL por defecto no se especifica ningún algoritmo de compresión predeterminado. Sin embargo, existen implementaciones específicas las cuales pueden incluir un algoritmo de compresión.

En el siguiente paso, se calcula y se añade el código de autenticación de mensaje. De esta forma el receptor puede comprobar que la información no ha sido alterada realizando el mismo cálculo y comparando con el código introducido por el cliente. Por último, se cifra el mensaje con el sistema de cifrado simétrico seleccionado por los dos extremos de la comunicación y se le añade una cabecera compuesta por los siguientes campos:

- **Content Type (1 Byte):** Indica el tipo de contenido de los datos. Puede ser un mensaje del protocolo de negociación, una notificación de cambio de cifrado, un mensaje de error, o datos de aplicación.
- **Version (2 Byte):** Está compuesto por dos bytes que indican la mayor y la menor versión soportada del protocolo SSL/TLS.
- **Compressed Length (1 byte):** Indica la longitud del resto del registro. Por lo tanto, es igual a la suma de L_d y L_{MAC}, y si los datos están cifrados con un algoritmo de bloque habría que sumarle L_p + 1 Byte.

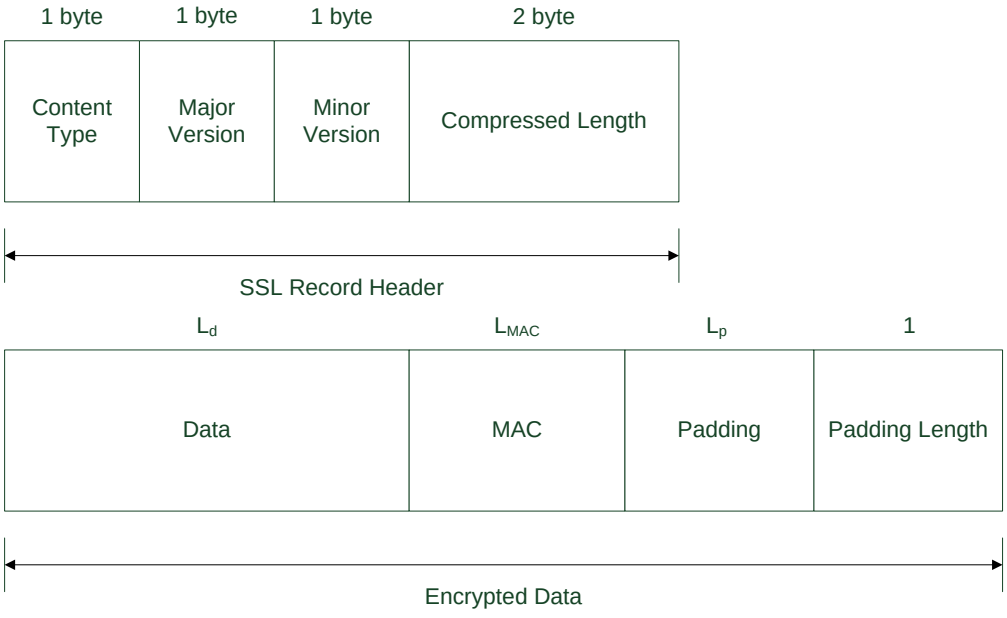


Figura 3. Formato de los registros SSL [34]

3.1.4 ChangeCipherSpec Protocol

El protocolo ChangeCipherSpec es uno de los tres protocolos de la capa superior, además del más simple. Está formado por un único mensaje que consiste en un único byte de valor 1 y que se utiliza para notificar un cambio en la estrategia de cifrado, es decir, del cipher suite.

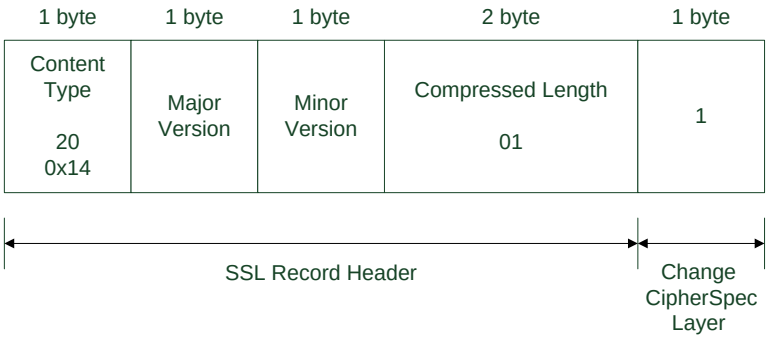


Figura 4. Formato de los mensajes del protocolo ChangeCipherSpec [34]

El mensaje es enviado por ambos extremos de la comunicación, es decir del cliente al servidor y del servidor al cliente para que después todo los siguientes mensajes se envíen utilizando el sistema de cifrado y las claves pactadas. Por último, indicar que este mensaje se envía normalmente al final de handshake de SSL.

3.1.5 SSL Alert Protocol

El protocolo de alerta se utiliza para señalar problemas con una sesión SSL a la entidad par. Del mismo modo que con los datos de la capa de aplicación, a estos mensajes se les añade un código de autenticación y se cifran en función del sistema de cifrado seleccionado en el establecimiento de la sesión de SSL.

Cada mensaje de este protocolo se compone de 2 bytes. El primer byte hace referencia a la gravedad de la alerta; puede tomar el valor '1' para indicar si el nivel de la alerta es una advertencia o '2' para señalar que la alerta es fatal. Si el nivel es fatal, ambas partes cierran inmediatamente la conexión evitando de esta manera cualquier ataque de truncamiento. Por otro lado, no se podrán crear nuevas conexiones en esta sesión aunque todas las existentes antes de la alerta podrán seguir establecidas.

El segundo byte contiene un código que indica una alerta específica, por ejemplo “ilegal_parameter” (un campo en un mensaje de negociación no es soportado o incompatible con otros campos). Otro ejemplo muy típico es el mensaje de advertencia “close_notify”, que notifica al destinatario que el remitente no enviará más mensajes en esta conexión. Es necesario que cada extremo envíe esta alerta antes de cerrar una conexión.

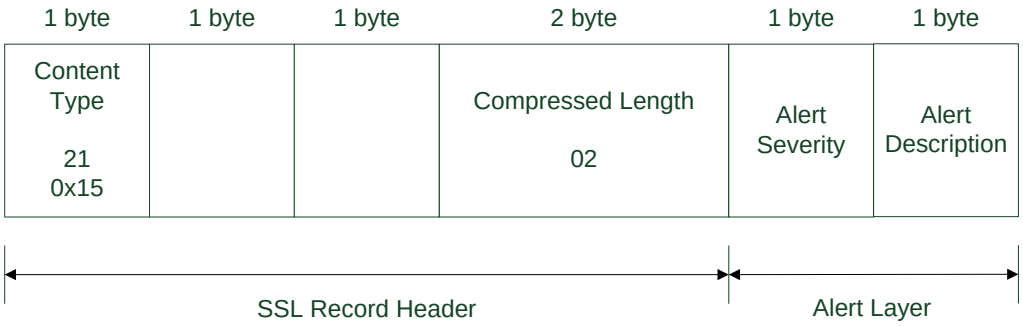


Figura 5. Formato de los mensajes del protocolo de alerta [34]

3.1.6 SSL Handshake Protocol

El protocolo de negociación SSL utilizado para el establecimiento de la sesión, también denominado protocolo de apretón de manos (Handshake Protocol), tiene como finalidad autenticar el servidor y/o el cliente y acordar los algoritmos criptográficos y las claves que se utilizarán para establecer una comunicación segura. De esta manera es posible garantizar la confidencialidad e integridad de la información a intercambiar.

Del mismo modo que con todos los mensajes del protocolo SSL, los mensajes del protocolo de negociación se incluyen dentro del campo de datos de los registros para ser transmitidos al destinatario.

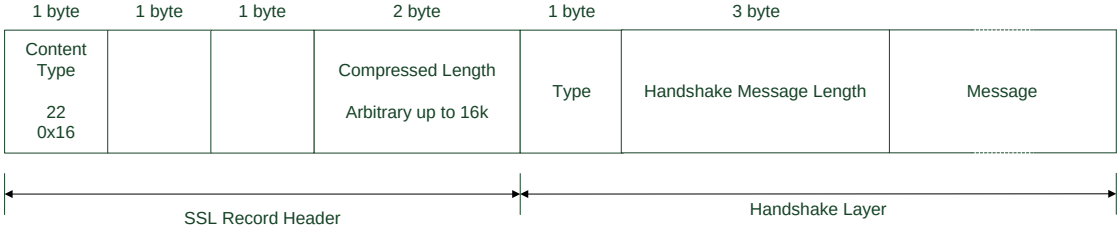


Figura 6. Formato de los mensajes del protocolo de negociación [34]

El contenido del mensaje tendrá unos determinados campos dependiendo del tipo de mensaje que se trate. Concretamente hay 10 tipos distintos, que se detallan a continuación en el orden que se envían por los dos extremos de la comunicación.

1) Petición de saludo *(Hello Request)*

No se utiliza muy a menudo ya que normalmente el servidor espera que el cliente inicie la negociación. Aunque alternativamente, puede optar por enviar este mensaje indicando al cliente que está preparado para iniciar la negociación. Por otro lado, si durante el tiempo de vida de la sesión el servidor quiere iniciar una renegocación se lo indica al cliente enviando el mensaje *Hello Request*.

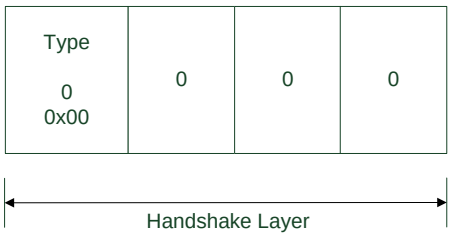


Figura 7. Formato del mensaje *Hello Request* [34]

2) Saludo del cliente *(Client Hello)*

El cliente envía el mensaje *Client Hello* para iniciar la negociación o como respuesta a un *Hello Request.* Este mensaje contiene la siguiente información:

- La versión del protocolo que el cliente desea utilizar.
- Una cadena de 32 bytes aleatorios.
- Opcionalmente, el identificador de una sesión anterior por si el cliente quiere volver a utilizar los parámetros anteriormente acordados.
- Una lista de las combinaciones de algoritmos criptográficos que soporta el cliente ordenada por orden de preferencia. Cada combinación incluye el algoritmo de cifrado, el algoritmo MAC y el método de intercambio de claves.
- Una lista con los algoritmos de compresión ofrecidos.

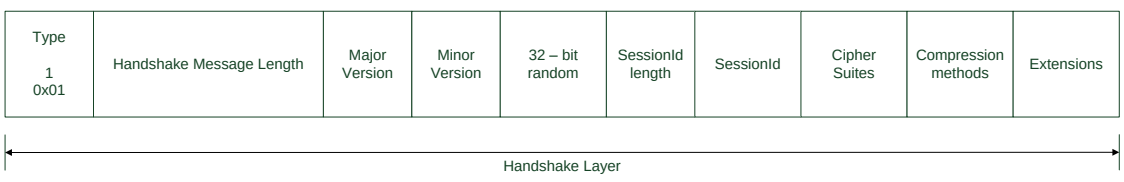


Figura 8. Formato del mensaje *Client Hello* [34]

3) Saludo del servidor *(Server Hello)*

Como respuesta, el servidor envía el mensaje *Server Hello,* el cual contiene la siguiente información:

- La versión del protocolo que se usará en la conexión, siendo esta igual a la que envió el cliente, o inferior si no está soportada por el servidor.
- Una cadena de 32 bytes aleatorios.
- El identificador de la sesión actual. Si el cliente envió uno y el servidor puede volver a utilizar los parámetros de configuración que definen esa sesión debe responder con el mismo identificador. Por el contrario, si el servidor no puede volver a usar la dicha configuración criptográfica el identificador enviado será diferente.
- Una única combinación de algoritmos criptográficos seleccionada por el servidor de entre la lista enviada por el cliente. Para esta selección el servidor dispone de otra lista ordenada por orden de preferencia que comparará con la lista recibida y seleccionará la opción más prioritaria que coincida entre ambas. Si vuelve a utilizar una sesión anterior, la combinación seleccionada debe ser la misma que se utilizó entonces.
- El algoritmo de compresión escogido por el servidor, o el que se utilizó en la sesión que se reemprende.

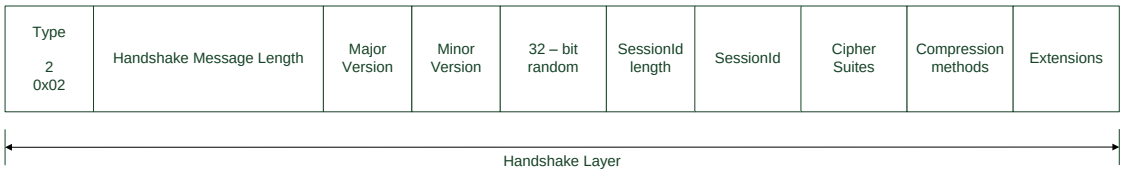


Figura 9. Formato del mensaje *Server Hello* [34]

Si se da la casualidad que el servidor puede continuar con una sesión anterior, ambos extremos pueden empezar a utilizar los algoritmos y claves previamente acordados saltándose los mensajes que se explican a continuación.

4) Certificado del servidor *(Certificate)* **o intercambio de claves del servidor** *(Server Key Exchange)*

Si el servidor puede autenticarse frente al cliente, que es el caso más habitual, este envía el mensaje *Certificate.* Este mensaje contiene normalmente el certificado X.509 [20] del servidor o una cadena de certificados.

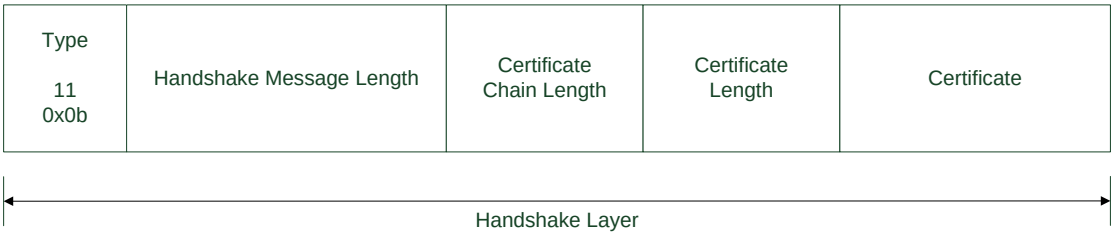


Figura 10. Formato del mensaje *Certificate* [34]

Si por casualidad el servidor no tiene certificado, o se ha acordado otro método de intercambio de claves, debe mandar un mensaje *Server Key Exchange* indicando el proceso a seguir.

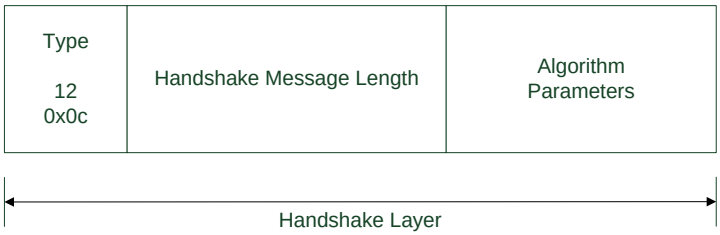


Figura 11. Formato del mensaje *Server Key Exchange* [34]

5) Petición de certificado *(Certificate Request)*

Aunque no se suele hacer normalmente, si el cliente debe realizar también el proceso de autenticación, el servidor envía el mensaje *Certificate Request*. Este mensaje contiene una lista con los tipos de certificado que el servidor puede admitir, ordenados por orden de preferencia y una lista con las autoridades de certificación que el servidor considera dignas de confianza.

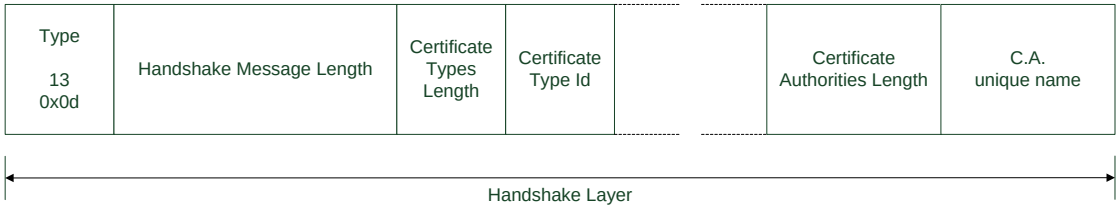


Figura 12. Formato del mensaje *Certificate Request* [34]

6) Fin del saludo del servidor *(Server Hello Done)*

Para terminar con la primera fase del diálogo, servidor envía el mensaje *Server Hello Done*. Este mensaje no lleva ninguna información adicional.

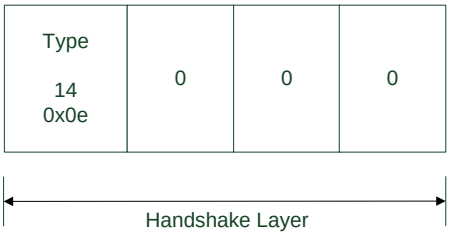


Figura 13. Formato del mensaje *Sever Hello Done* [34]

7) Certificado del cliente *(Certificate)*

Después de que el servidor ha terminado de mandar sus mensajes iniciales, el cliente ya sabe cómo continuar. En primer lugar, si el servidor ha solicitado autenticación por parte del cliente y éste dispone de algún certificado con las características requeridas por el servidor, debe enviar un mensaje *Certificate*. El formato de este mensaje es igual al que envía el servidor Fig. 10.

8) **Intercambio de claves del cliente** *(Client Key Exchange)*

El cliente envía un mensaje *Client Key Exchange*, cuyo contenido depende el método de intercambio de claves acordado. En caso de utilizar el método RSA, como es este el caso, este mensaje contendrá una cadena de 48 Bytes que se usará como "PreMasterSecret" y se cifrará con la clave pública del servidor.

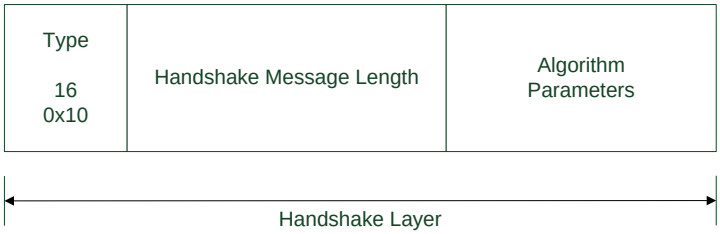


Figura 14. Formato del mensaje *Client Key Exchange* [34]

A continuación, el cliente y el servidor calculan el "MasterSecret", que es otra cadena de 48 Bytes. Se obtiene aplicando funciones hash [21] al "PreMasterSecret" y las cadenas aleatorias enviadas en los mensajes *ClientHello* y *ServerHello*.

A partir del "MasterSecret" y las cadenas aleatorias cliente y servidor obtiene:

- Las dos claves que usarán los sistemas de cifrado simétrico, una para cada sentido, es decir, del cliente al servidor y del servidor al cliente.
- Las dos claves que permitirán obtener los códigos MAC (una para cada sentido).
- Los dos vectores de inicialización si se utilizan algoritmos de cifrado en bloque.

9) Verificación del certificado *(Certificate Verify)*

Este mensaje es utilizado por el cliente para probar que el servidor posee la clave privada correspondiente a la clave pública del certificado. Este mensaje contiene una firma genera con la clave privada del cliente de todos los mensajes de negociación intercambiados hasta el momento.

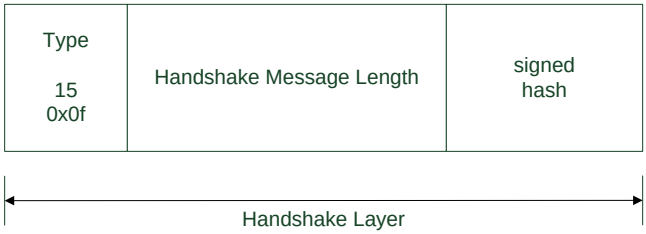


Figura 15. Formato del mensaje *Certificate Verify* [34]

10) Finalización *(Finished)*

Este mensaje indica que la negociación se ha completado y que a partir de este momento cliente y servidor pueden empezar a enviar los datos de aplicación utilizando los algoritmos y claves acordados. Este mensaje debe enviarse encriptado por lo que antes de enviarlo es necesario notificar de un cambio en la estrategia de cifrado mediante el mensaje ChangeCipherSpec.

El contenido de este mensaje se obtiene aplicando funciones hash a la "MasterSecret" y a la concatenación de todos los mensajes de negociación.

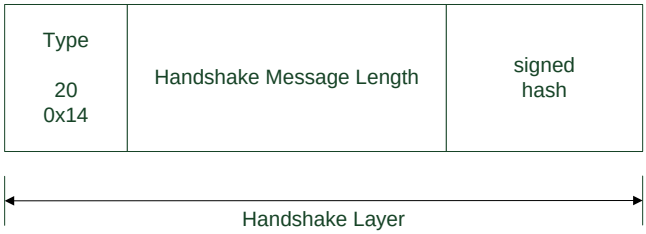
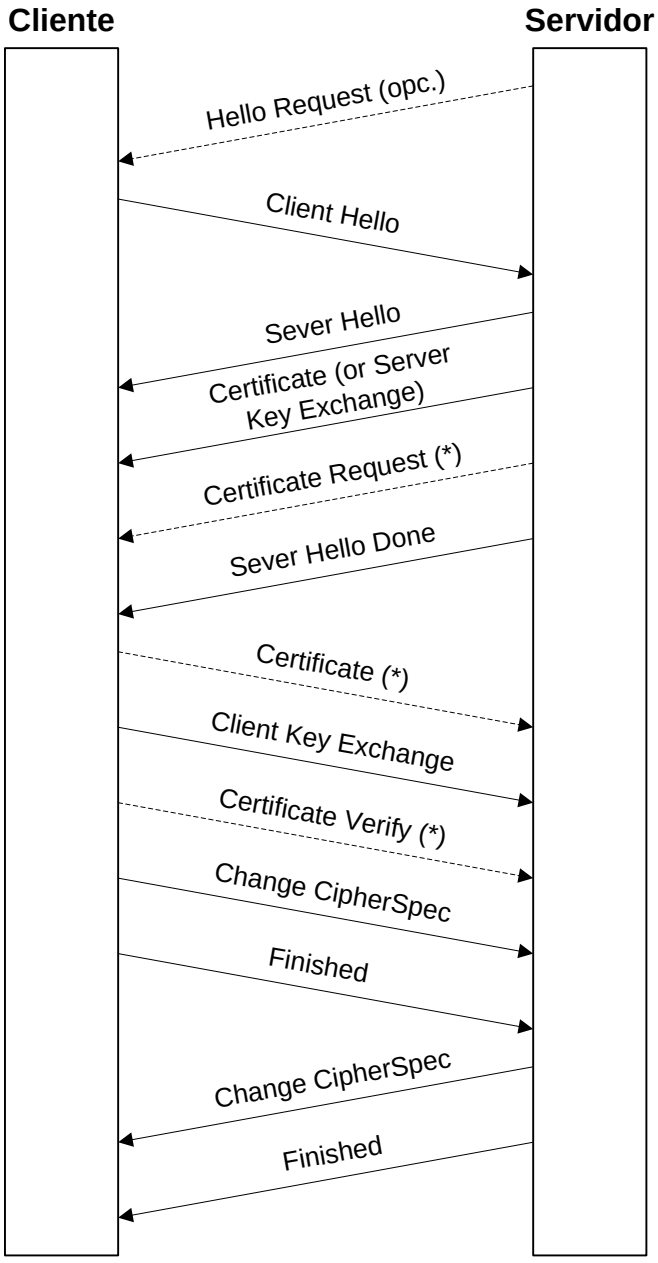


Figura 16. Formato del mensaje *Finished* [34]

En el siguiente diagrama se muestran los mensajes intercambiados durante la fase de negociación de SSL:



(*) → Solamente si se realiza autenticación del cliente

Figura 17. Establecimiento de una sesión SSL nueva [1]

Por otro lado, si se quiere reemprender una sesión ya establecida anteriormente el diagrama de los mensajes intercambiados sería el siguiente:

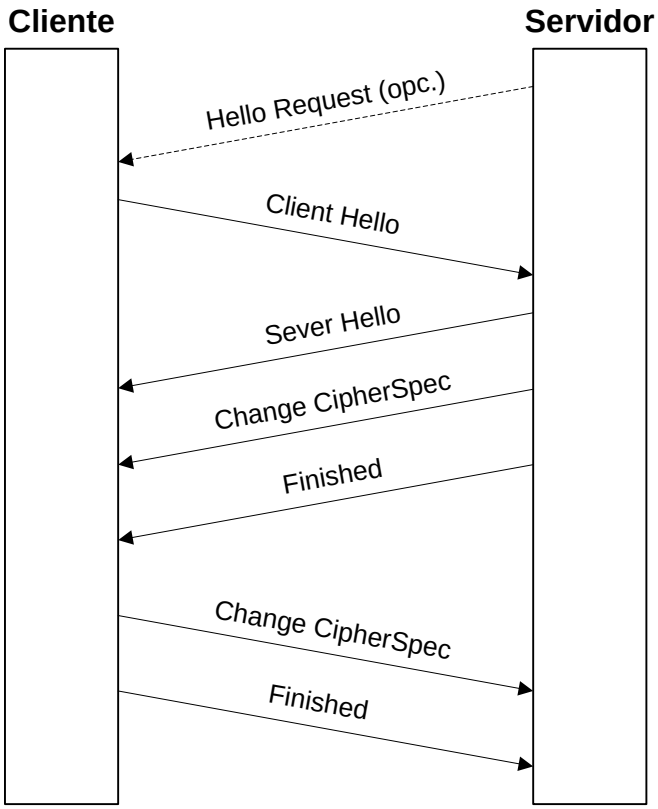


Figura 18. Establecimiento de una sesión SSL reemprendida [1]

En la siguiente captura puede verse los mensajes intercambiados por el cliente y servidor en el establecimiento de la sesión SSL.

23	5.29122700	192.168.1.35	192.168.1.37	TCP	66 49448→443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
24	5.29161700	192.168.1.37	192.168.1.35	TCP	66 443→49448 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
25	5.29166800	192.168.1.35	192.168.1.37	TCP	54 49448→443 [ACK] Seq=1 Ack=1 Win=65700 Len=0
26	5.29178100	192.168.1.35	192.168.1.37	SSLv3	122 Client Hello
27	5.29210800	192.168.1.37	192.168.1.35	TCP	60 443→49448 [ACK] Seq=1 Ack=69 Win=29312 Len=0
28	5.29323200	192.168.1.37	192.168.1.35	SSLv3	660 Server Hello, Certificate, Server Hello Done
29	5.29350700	192.168.1.35	192.168.1.37	SSLv3	202 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
30	5.29481000	192.168.1.37	192.168.1.35	SSLv3	129 Change Cipher Spec, Encrypted Handshake Message
31	5.29499000	192.168.1.35	192.168.1.37	SSLv3	432 Application Data, Application Data

Figura 19. Ejemplo real del establecimiento de una sesión SSL

4 Desarrollo del TFM

En este apartado se resume el desarrollo del TFM, concretamente el montaje de la maqueta de red desplegada en el laboratorio y el plan de pruebas ideado y ejecutado sobre esta misma.

La maqueta de red está compuesta por dos equipos; uno que actúa como servidor en el que se ha instalado Apache y otro que actúa como cliente. Un punto importante a destacar, es que en el cliente se ha utilizado Mozilla Firefox como navegador Web ya que permite una configuración avanzada. Gracias a tener acceso a esta configuración es posible seleccionar que versión de SSL/TLS utilizar y que sistema de cifrado emplear, tanto para el handshake inicial de SSL/TSL como para el posterior cifrado de los datos. Indicar que se ha utilizado la versión SSL 3.0 a pesar de existir versiones superiores como TLS 1.1 y TLS 1.2 y el sistema de cifrado de clave pública RSA (Rivest, Shamir y Adleman) [22][23] para el handshake inicial de SSL.

Las pruebas que se han llevado a cabo pueden dividirse en función del tamaño de la clave RSA utilizada para firmar el certificado digital y obtener la clave secreta para el posterior cifrado de los datos, en función del tamaño del objeto descargado por el cliente y en función del sistema de cifrado simétrico utilizado para dicho objeto. Concretamente, se han utilizado claves RSA de tamaño de 512, 1024 y 2048 bits, objetos de tamaño de 10, 30, 50, 70 y 100 KBytes y los sistemas de cifrado de clave privada basados en bloque AES (Advanced Encrypting Standard) [24], 3DES (Triple Data Encryption Standard) [25], Camellia [26] y SEED [27] y el cifrado de flujo RC4 [28].

Para caracterizar y medir el rendimiento de las conexiones HTTPS, se ha realizado una batería de pruebas en las que se ha cuantificado el número total de paquetes, la cantidad de Bytes intercambiada a nivel de TCP y el tiempo empleado para la descarga del objeto desde el cliente al servidor. Además, para conocer cuál es la sobrecarga de datos y de procesado introducida por la utilización de estos sistemas criptográficos se han realizado medidas sin la utilización de ninguno de estos. De esta manera ha sido posible establecer una regla que permita conocer la sobrecarga de datos para cualquier tamaño de objeto. Por otro lado, en cuando a la sobrecarga de procesado, es

muy difícil establecer una pauta ya que depende mucho del tipo de contenido que se está sirviendo y de las características del servidor. Por ello, simplemente se ha comprobado como aumenta o disminuye en función del tamaño objeto servido y del tipo de cifrado seleccionado.

Un aspecto muy importante que se ha tenido en cuenta a la hora al analizar la sobrecargar de datos es si se podía obviar la sobrecarga que introducía el propio handshake de SSL. De esta forma podría conocerse cuál es la sobrecarga real que introduce el propio cifrado de datos y poder realizar una comparación con el tamaño real del fichero.

Para medir los parámetros anteriormente especificados se ha utilizado en el cliente el analizador de protocolos Wireshark. Mediante las herramientas que incorpora este software es muy sencillo caracterizar las conexiones HTTPS pero aun y todo es algo costoso ir recogiendo los resultados parámetro a parámetro. Es por esto, que se ha desarrollado un programa en lenguaje C el cual permite agilizar la obtención de estos resultados. Ha sido necesaria la utilización de la herramienta *"tshark"* que no es más que un programa de línea de comandos basado en Wireshark que sirve para la captura y visualización de paquetes.

El funcionamiento de este programa es muy sencillo, simplemente indicándole la ruta absoluta del fichero .pcapng y la dirección IP del cliente y servidor devuelve los parámetros por los que se va caracterizar dicha conexión HTTPS.

En los siguientes apartados se muestra cómo se ha configurado el servidor Web Apache para soportar conexiones seguras HTTPS así como el navegador Web Mozilla Firefox para emplear SSL 3.0 y los distintos sistemas de cifrado nombrados anteriormente. Una vez explicado esto se presentan los resultados obtenidos en las distintas pruebas realizadas.

4.1 Configuración del servidor Web Apache

Como se ha indicado anteriormente, en el equipo que actúa como servidor se ha instalado el servicio Web Apache. Por defecto no viene configurado para soportar conexiones seguras HTTPS por ello ha sido necesario seguir los siguientes pasos.

Lo primero de todo, es habilitar el módulo SSL. Para ello hay que utilizar el comando *a2enmod* el cual crea un enlace simbólico de dicho módulo en el directorio *"etc/apache2/mods-enabled"*.

```
sudo a2enmod ssl
```

A continuación, hay que revisar que no esté comentada la entrada que hace referencia al puerto 443 en el fichero *"etc/apache2/ports.conf"* ya que es el puerto que utilizado en las conexiones HTTPS.

```
NameVirtualHost *:80
NameVirtualHost *:8080
Listen 80
Listen 8080
<IfModule mod_ssl.c>
    Listen 443
</IfModule>

<IfModule mod_gnutls.c>
    Listen 443
</IfModule>
```

Seguidamente, hay que habilitar la configuración por defecto de SSL mediante el comando *a2ensite* el cual crea un enlace simbólico de esta configuración en el directorio *"etc/apache2/sites-enabled"*.

```
sudo a2enstie default-ssl
```

Por último, es necesario reiniciar el servidor para que todos los cambios realizados surtan efecto.

```
sudo /etc/init.d/apache2 restart
```

Tras haber realizado estos pasos ya es posible establecer una comunicación segura, ahora lo que queda es generar el certificado digital.

Una vez habilitadas las conexiones seguras en el servidor Web hay que generar un certificado digital. Este tiene como función principal autenticar al servidor frente a los usuarios que quieran establecer una comunicación. Además, la clave privada utilizada para firmarlo se utiliza para cifrar los mensajes del handshake de SSL.

El primer paso es instalar el paquete *openssl* el cual proporciona un conjunto de herramientas de administración y bibliotecas relacionadas con la criptografía.

```
sudo apt-get install openssl
```

A continuación, teniendo en cuenta que para el establecimiento de la sesión de SSL se utilizará el sistema criptográfico RSA hay que generar una clave privada, la cual se utilizará para firmar el certificado digital. Esta clave se creará usando 3DES y su tamaño será una de los valores que se irá variando para la realización de las pruebas.

```
sudo openssl genrsa –des3 –out server.key 1024
```

Al generar esta clave se pide introducir una clave de paso, es decir, una contraseña, la cual será necesario introducirla siempre que se reinicie Apache. Para evitar repetir este proceso es posible que realizar el siguiente paso.

```
mv server.key server.key.old
sudo openssl rsa -in server.key.old -out server.key

Enter pass phrase for servidor.old.key: <clave de paso>
writing RSA key
```

Tras generar la clave privada crearemos el certificado y lo firmaremos utilizando esta misma.

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

Durante este proceso será posible rellenar los distintos campos por los que está compuesto el certificado como son el código del país, el estado o la provincia, el nombre de la localidad, el nombre de la organización, el nombre del equipo, el correo electrónico, y su llave pública.

```
Enter pass phrase for server.key:
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: Navarra
Locality Name (eg, city) []: Pamplona
Organization Name (eg, company) Internet Widgits Pty Ltd: UPNA
Organizational Unit Name (eg, section) []: Los Pinos
Common Name (eg, YOUR name) []: <IP del servidor>
Email Address []: mikelopezfer@gmail.com
#No es necesario completar los siguientes campos.
Please enter the following 'extra' attributes to be sent with your
certificate request
A challenge password []: <frase de paso>
An optional company name []:
```

El siguiente paso debería ser realizado por una autoridad certificadora, aunque en este caso se generará un certificado autofirmado [29]. Por ello, los navegadores no reconocerán este certificado y mostraran un aviso de advertencia al usuario.

```
openssl x509 -req -days 365 -in server.csr -signkey server.key -out
server.crt
```

Una vez firmado el certificado, debería almacenarse en un directorio junto a la clave privada del servidor ya que posteriormente se necesitará añadir esta ruta en el fichero de configuración *"/etc/apache2/sites-enabled/default-ssl"*.

```
sudo mkdir /etc/apache2/ssl
sudo mv server.key server.crt /etc/apache2/ssl
```

Despues, hay que modificar el fichero anteriormente citado indicando las rutas tanto del certificado digital como de la clave privada.

```
SSLCertificateFile /etc/apache2/ssl/server.crt
SSLCertificateKeyFile /etc/apache2/ssl/server.key
```

Por último, habrá que reiniciar el servidor para que todos los cambios realizados surtan efecto.

```
sudo /etc/init.d/apache2 restart
```

4.2 Configuración del navegador Web Mozilla Firefox

El hecho de que el navegador Web Mozilla Firefox permita una configuración avanzada ha permitido realizar el plan de pruebas de una forma más rápida y sencilla. Simplemente escribiendo en la barra de navegación *"about:config"* se accede a esta configuración donde ha sido posible seleccionar el protocolo de seguridad a utilizar, en este caso SSL3.0.

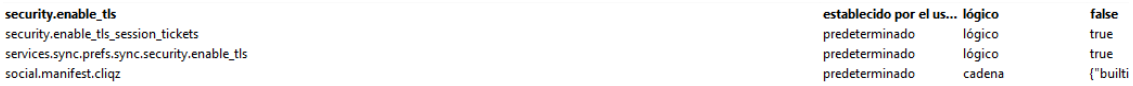


Figura 20. Deshabilitando TLS desde Mozilla Firefox

Disponer de acceso a esta configuración permite no tener que modificar la configuración del servidor Web Apache, puesto que en él también es posible seleccionar los protocolos de seguridad soportados. En concreto en el fichero *"/etc/apache2/mods-enabled/ssl.conf"* se indican los protocolos de seguridad soportados.

```
#   The protocols to enable.
#   Available values: all, SSLv3, TLSv1, TLSv1.1, TLSv1.2
#   SSL v2  is no longer supported
SSLProtocol all
```

Figura 21. Protocolos de seguridad ssl.conf

Por otro lado, dentro de la configuración avanazada del Mozilla Firefox también es posible seleccionar el sistema de cifrado para los datos. Como ya se ha especificado anteriormente se utilizarán los sistemas de cifrado AES, RC4, 3DES, CAMELLIA y SEED.

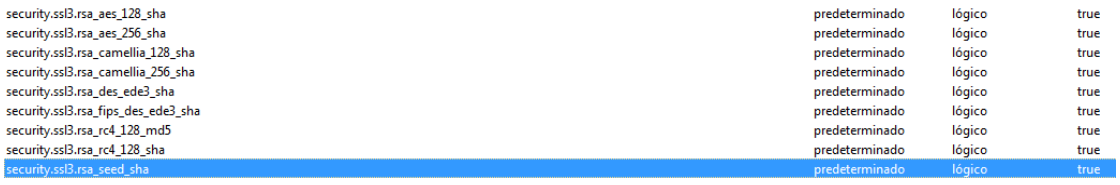


Figura 22. Sistemas de cifrado simétrico Mozilla Firefox

Del mismo modo que con el protocolo de seguridad, en el servidor Web Apache se puede configurar que sistemas de cifrado utilizar e incluso un orden de prioridad. Por lo tanto, cuando el cliente envié el mensaje "ClientHello" donde se especifican los sistemas de cifrado que soporta, el servidor seleccionará de todos ellos el más

prioritario. Sin embargo, para el desarrollo de las pruebas se ha comentado esta opción y se ha dejado disponible un único sistema de cifrado en el navegador Web.

```
#   SSL Cipher Suite:
#   List the ciphers that the client is permitted to negotiate. See the
#   ciphers(1) man page from the openssl package for list of all available
#   options.
#   Enable only secure ciphers:
SSLCipherSuite HIGH:MEDIUM:!aNULL:!MD5

#   Speed-optimized SSL Cipher configuration:
#   If speed is your main concern (on busy HTTPS servers e.g.),
#   you might want to force clients to specific, performance
#   optimized ciphers. In this case, prepend those ciphers
#   to the SSLCipherSuite list, and enable SSLHonorCipherOrder.
#   Caveat: by giving precedence to RC4-SHA and AES128-SHA
#   (as in the example below), most connections will no longer
#   have perfect forward secrecy - if the server's key is
#   compromised, captures of past or future traffic must be
#   considered compromised, too.
#SSLCipherSuite RC4-SHA:AES128-SHA:HIGH:MEDIUM:!aNULL:!MD5
#SSLHonorCipherOrder on

#   The protocols to enable.
#   Available values: all, SSLv3, TLSv1, TLSv1.1, TLSv1.2
#   SSL v2  is no longer supported
SSLProtocol all
```

Figura 23. Sistemas de cifrado simétrico ssl.conf

4.3 Caracterización de una conexión HTTPS

Tras configurar el servidor Web para soportar conexiones seguras HTTPS y el navegador Web para soportar únicamente la versión SSL 3.0 con los distintos sistemas de cifrado simétricos, se ha llevado a cabo el plan de pruebas ideado.

Para realizar las pruebas, en el equipo cliente se ha dejado capturando tráfico al analizador de protocolos Wireshark. Mientras desde el navegador Web se han realizado las peticiones para cada objeto con los distintos tamaños de clave RSA y los distintos sistemas de cifrado indicados anteriormente. Por lo tanto, se ha obtenido un fichero "captura.pcapng" para cada tamaño de objeto con los distintos tamaños de clave RSA y para sistema de cifrado simétrico empleado.

Después de obtener todas capturas tráfico se ha utilizado el programa diseñado en lenguaje C para conseguir los resultados que caracterizan a dicha comunicación. Este programa utiliza la herramienta "tshark" que permite visualizar en un terminal los distintos paquetes capturados pudiendo aplicar distintos filtros. El resultado de ejecutar la herramienta "tshark" se ha volcado a un fichero y se ha ido leyendo línea a línea hasta encontrar los parámetros que caracterizarán la comunicación.

El comando que se ha utilizado para obtener todos los datos a nivel de TCP incluyendo los datos del handshake de SSL ha sido el siguiente:

```
tshark –r fichero.pcapng –z io,stat,0,"SUM(tcp.len)tcp.len and tcp
and ((ip.src==ip_origen and ip.dst==ip_destino) or
(ip.src==ip_destino and ip.dst==origen))"
```

Por el contrario, para evitar contabilizar los datos intercambiados en el handshake de SSL se ha utilizado el siguiente comando:

```
tshark –r fichero.pcapng –z io,stat,0,"SUM(tcp.len)tcp.len and tcp
and ((ip.src==ip_origen and ip.dst==ip_destino) or
(ip.src==ip_destino and ip.dst==origen) and ! ssl.handshake)"
```

Para obtener el tiempo desde que se inicia conexión TCP hasta que se cierra se ha utilizado el siguiente comando:

```
tshark -r fichero.pcapng -z -o tcp.calculate_tiemstamps:true -T fields -e tcp.time_relative "tcp and ((ip.src==ip_origen and ip.dst==ip_destino) or (ip.src==ip_destino and ip.dst==origen))"
```

Para conocer el número de paquetes intercambiados durante la comunicación se ha empleado el siguiente comando:

```
tshark -r fichero.pcapng -z -o tcp.calculate_tiemstamps:true -T fields -e frame.number "tcp and ((ip.src==ip_origen and ip.dst==ip_destino) or (ip.src==ip_destino and ip.dst==origen))"
```

Un ejemplo del funcionamiento de este programa se muestra en la siguiente captura:

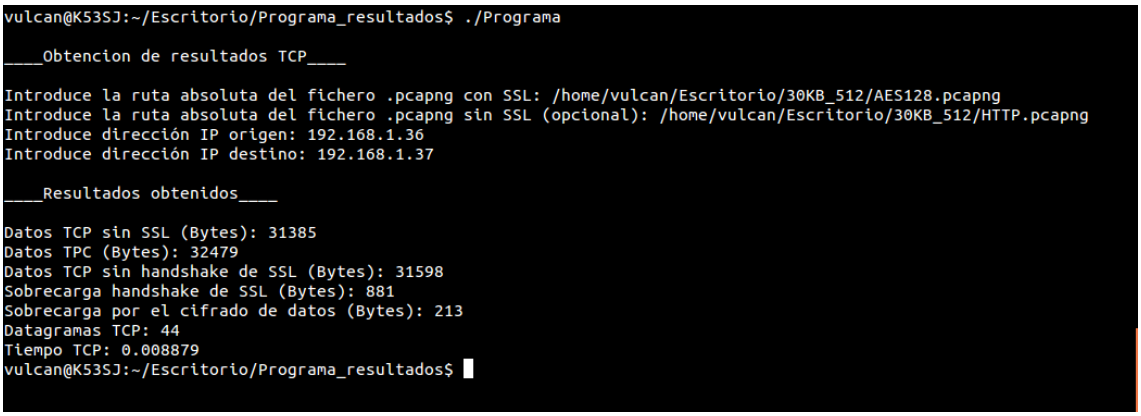


Figura 24. Obtención de los parámetros para caracterizar una conexión HTTPS

5 Resultados obtenidos

Los resultados obtenidos del plan de pruebas descrito anteriormente se muestran en el Anexo I. Del mismo modo que se ha dividido el plan de pruebas, este anexo se divide en varios subapartados, concretamente en función del tamaño de la clave privada o clave RSA utilizada y en función del tamaño del objeto descargado.

Dentro de cada subapartado se indicarán los resultados de los parámetros medidos para cada sistema de cifrado simétrico, siendo estos los sistemas AES_128, AES_256, RC4_128, 3DES, CAMELLIA_128, CAMELLIA_256 y SEED. Los parámetros medidos serán los paquetes totales enviados entre cliente y servidor, el número total de bytes intercambiados, y el tiempo empleado, todo ello desde el inicio de la conexión TCP hasta el final. Por otra parte, también se especificará la sobrecarga de datos que introduce cada sistema de cifrado comparándola con resultados obtenidos de las peticiones realizadas sin seguridad.

Como los resultados que se obtienen en la descarga de los objetos utilizando conexiones HTTP, no dependen del tamaño de la clave RSA y los datos no son cifrados, se ha realizado una única prueba para cada objeto.

HTTP			
Fichero (Bytes)	**Paquetes**	**TCP (Bytes)**	**Tiempo (Segundos)**
9838	19	10518	0,002832
30663	39	31345	0,004914
51459	60	52142	0,006342
71749	79	72431	0,008452
103089	112	103774	0,010449

Tabla 1. Resultados – Objetos de 10, 30, 50, 70, 100 KB utilizando HTTP

Como se observa en estos resultados el número de Bytes cuantificados a nivel de TCP son más que los del propio objeto. Esto es debido a que incluyen las cabeceras HTTP, las cuales también variaran su tamaño dependiendo del servidor Web. Por otra parte indicar que se han realizado 30 pruebas para cada obtener un tiempo medio de la descarga de cada objeto.

6 Análisis de los resultados

El objetivo principal de este trabajo es caracterizar y evaluar el rendimiento de las conexiones seguras HTTPS atendiendo a la sobrecarga de procesado y principalmente a la sobrecarga de datos debida a la utilización de los distintos sistemas criptográficos. Tras realizar las distintas pruebas y observar los resultados obtenidos ha sido posible extraer varias conclusiones.

6.1 Sobrecarga de datos

Durante el desarrollo de las pruebas se ha podido diferenciar entre la sobrecarga introducida por el sistemas de cifrado asimétrico RSA utilizado en el establecimiento de la sesión SSL y la sobrecarga generada por los sistemas de cifrado simétricos empleados en el cifrado de los datos. Gracias a esto, ha sido posible comprobar cómo la sobrecarga introducida por los sistemas de cifrado simétricos disminuye a medida que el tamaño del objeto aumenta, todo ello independientemente del tamaño de la clave RSA utilizada. Para verificar esto se ha medido la sobrecarga generada por estos sistemas como el porcentaje de Bytes que se envían de más al realizar una comunicación segura.

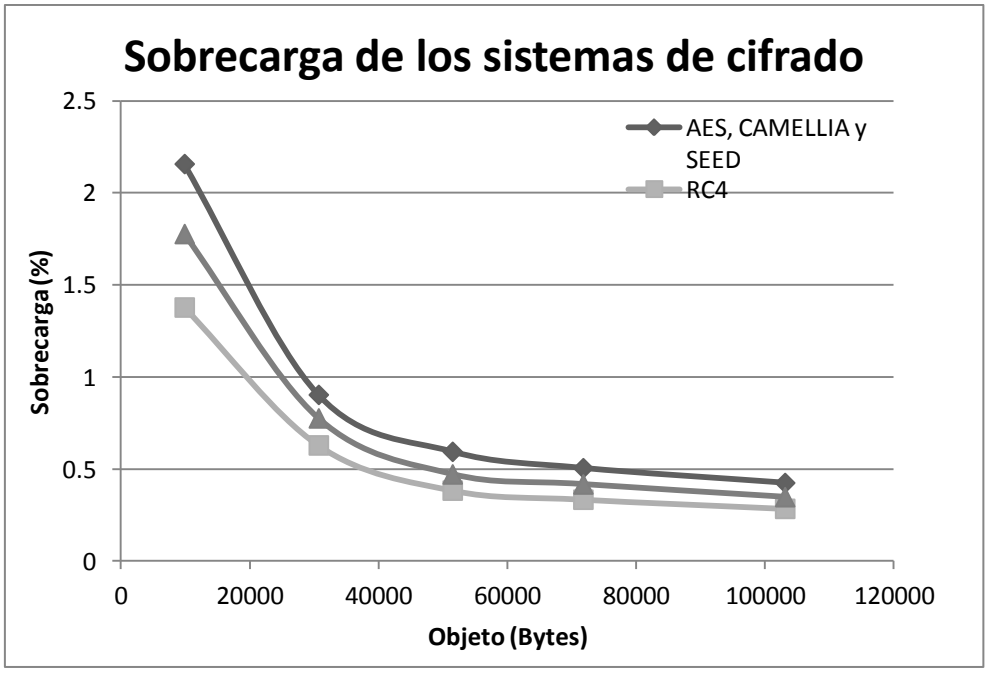


Gráfico 1. Sobrecarga de datos introducida por los sistemas de cifrado

Por otra parte, se ha comprobado que si se aumenta el tamaño de la clave RSA aumentará el tamaño de la sobrecarga introducida por el handshake de SSL ya que aumenta el tamaño del certificado y de la clave secreta que utilizan los sistemas de cifrado simétrico. En las siguientes imágenes se puede observar como aumenta el tamaño del certificado *"Certificate"* y de la clave secreta *"Client Key Exchange"*.

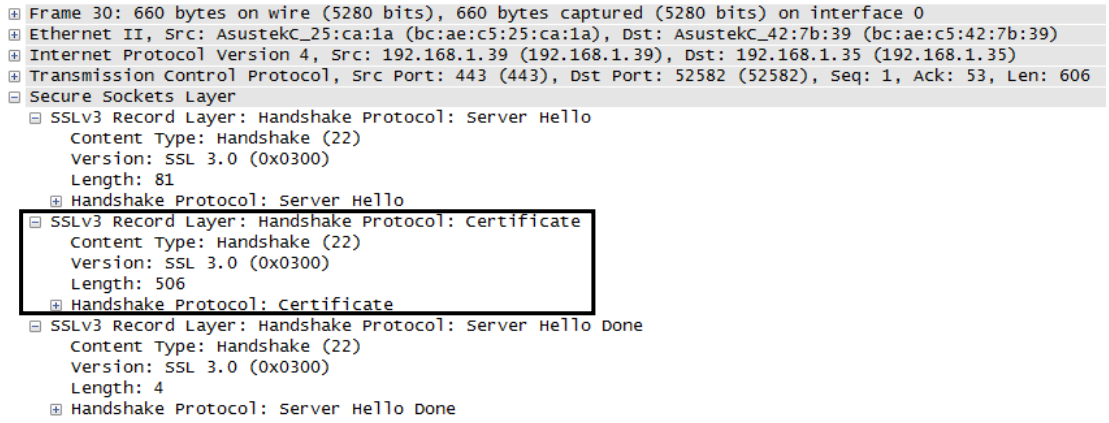


Figura 25. Certificado con clave RSA de 512 bits

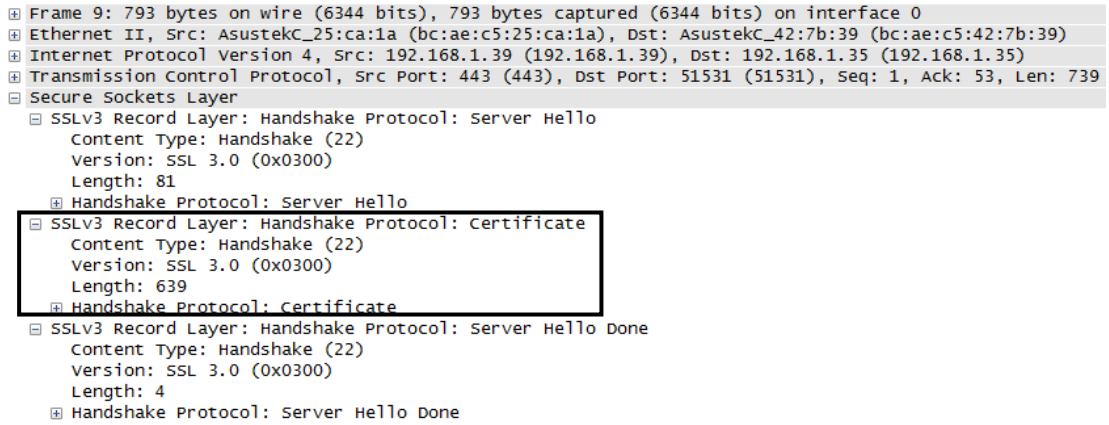


Figura 26. Certificado con clave RSA de 1024 bits

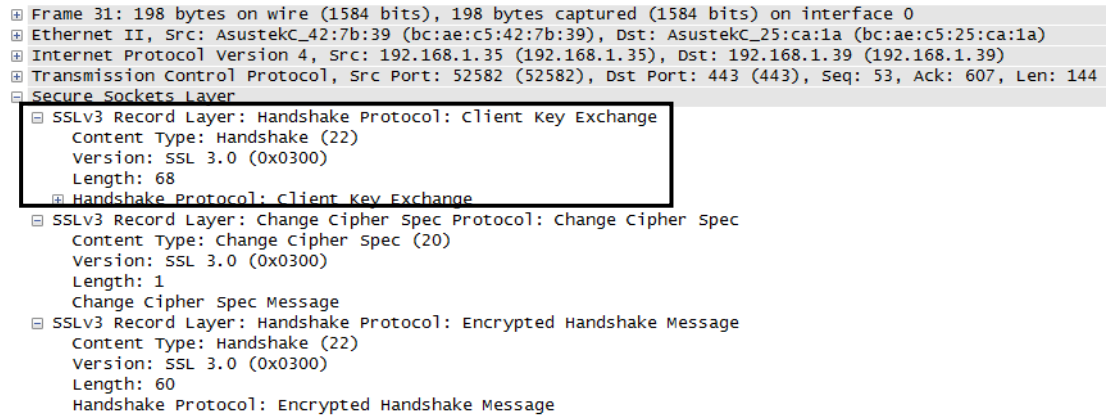


Figura 27. Clave secreta con RSA de 512 bits

```
⊞ Frame 10: 262 bytes on wire (2096 bits), 262 bytes captured (2096 bits) on interface 0
⊞ Ethernet II, Src: AsustekC_42:7b:39 (bc:ae:c5:42:7b:39), Dst: AsustekC_25:ca:1a (bc:ae:c5:25:ca:1a)
⊞ Internet Protocol Version 4, Src: 192.168.1.35 (192.168.1.35), Dst: 192.168.1.39 (192.168.1.39)
⊞ Transmission Control Protocol, Src Port: 51531 (51531), Dst Port: 443 (443), Seq: 53, Ack: 740, Len: 208
⊟ Secure Sockets Layer
  ⊟ SSLv3 Record Layer: Handshake Protocol: Client Key Exchange
      Content Type: Handshake (22)
      Version: SSL 3.0 (0x0300)
      Length: 132
    ⊞ Handshake Protocol: Client Key Exchange
  ⊟ SSLv3 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
      Content Type: Change Cipher Spec (20)
      Version: SSL 3.0 (0x0300)
      Length: 1
      Change Cipher Spec Message
  ⊟ SSLv3 Record Layer: Handshake Protocol: Encrypted Handshake Message
      Content Type: Handshake (22)
      Version: SSL 3.0 (0x0300)
      Length: 60
      Handshake Protocol: Encrypted Handshake Message
```

Figura 28. Clave secreta con RSA de 1024 bits

Si se tiene en cuenta tanto la sobrecarga que introduce el sistema de cifrado asimétrico RSA como los distintos sistemas de cifrado simétricos empleados es posible obtener una función que permita conocer la sobrecarga para cualquier tamaño de objeto. Para ello en una gráfica donde el eje horizontal represente el tamaño de los objetos y el eje vertical la sobrecarga se pueden marcar los resultados obtenidos y utilizar el método de mínimos cuadrados para encontrar una función continua con el mínimo error.

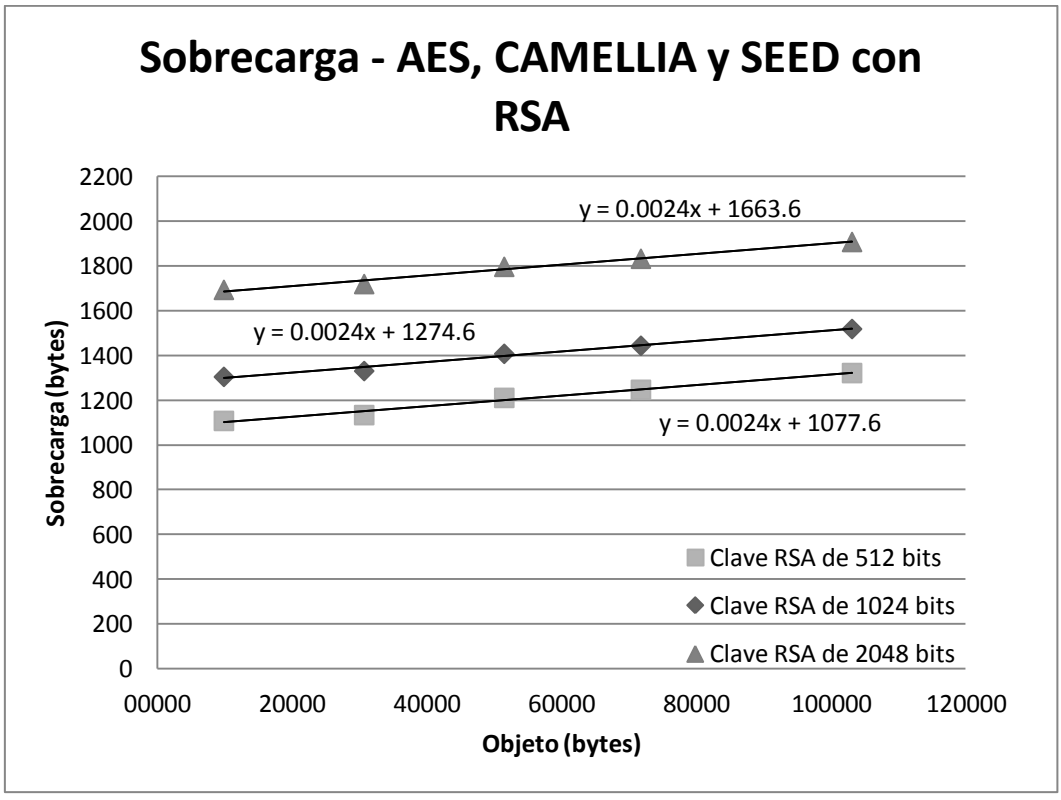


Gráfico 2. Sobrecarga de datos sistemas de cifrado AES, CAMELLIA y SEED con RSA

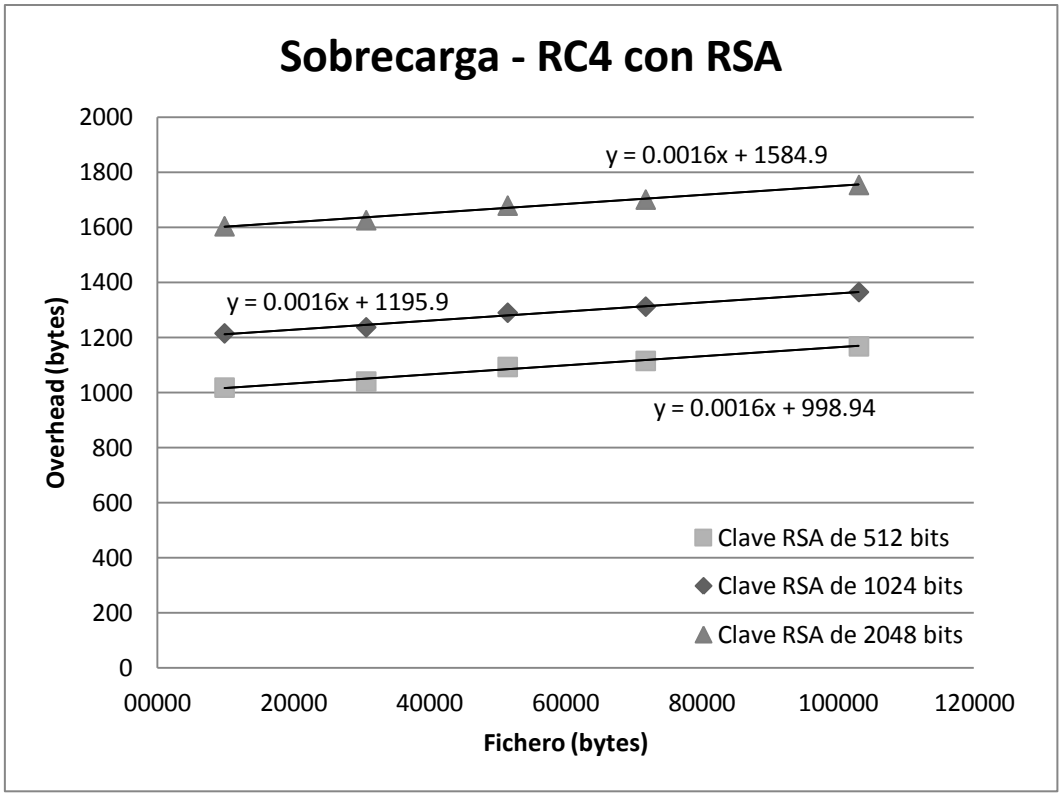


Gráfico 3. Sobrecarga de datos con sistema de cifrado RC4 con RSA

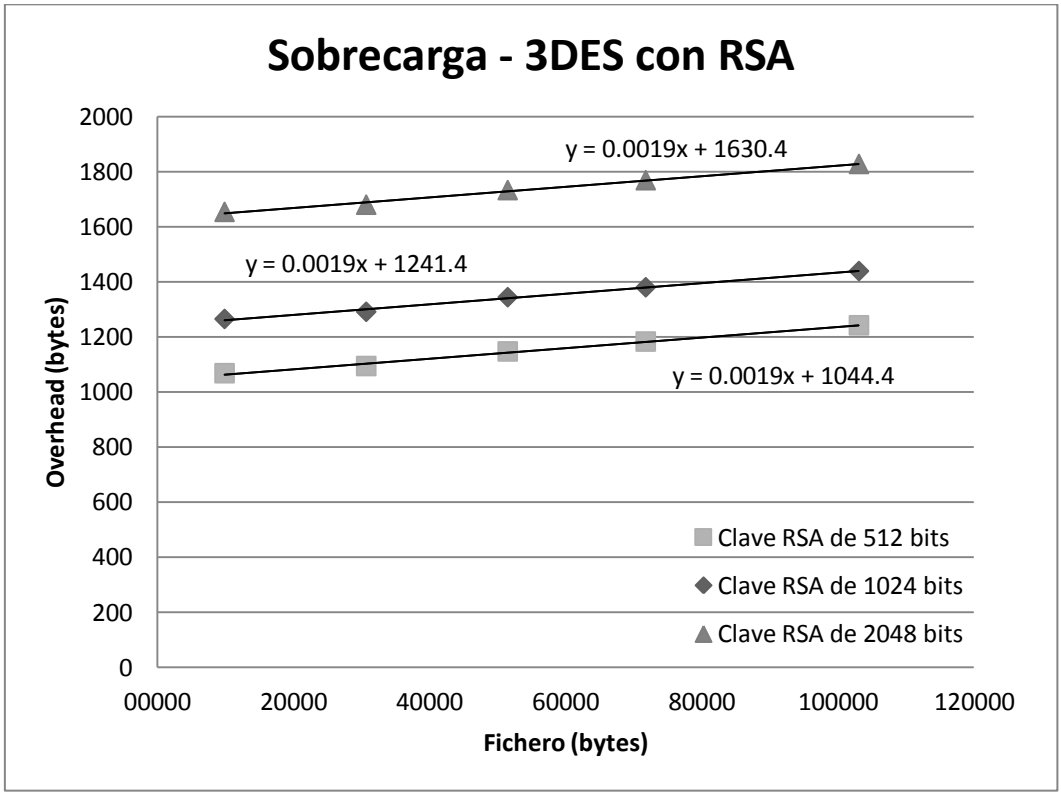


Gráfico 4. Sobrecarga de datos con sistema de cifrado 3DES con RSA

Por otro lado, si obviamos la sobrecarga introducida por el handshake de SSL es posible conocer la sobrecarga que introduce cada sistema de cifrado simétrico. Del mismo modo que se ha realizado anteriormente es posible graficarlo y obtener una función continua.

Como los resultados de los sistemas AES, Camellia y SEED son los mismos se ha aprovechado la misma gráfica para representarlos.

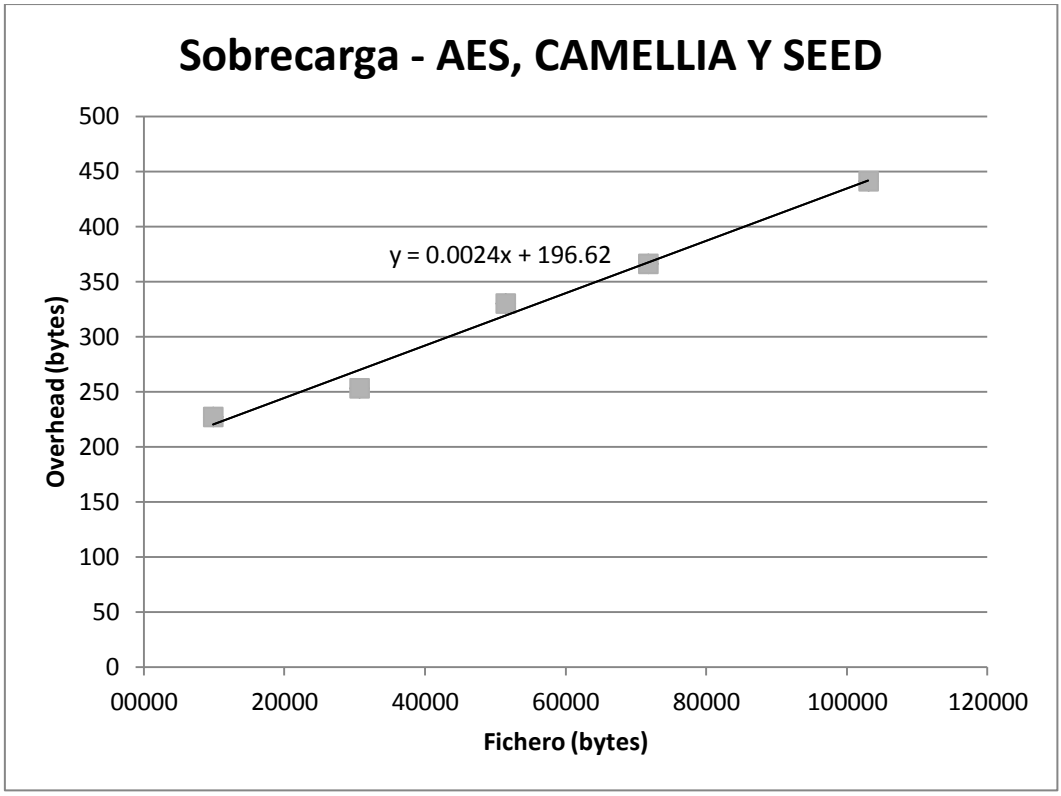


Gráfico 5. Sobrecarga de datos con sistemas de cifrados AES, Camellia y SEED

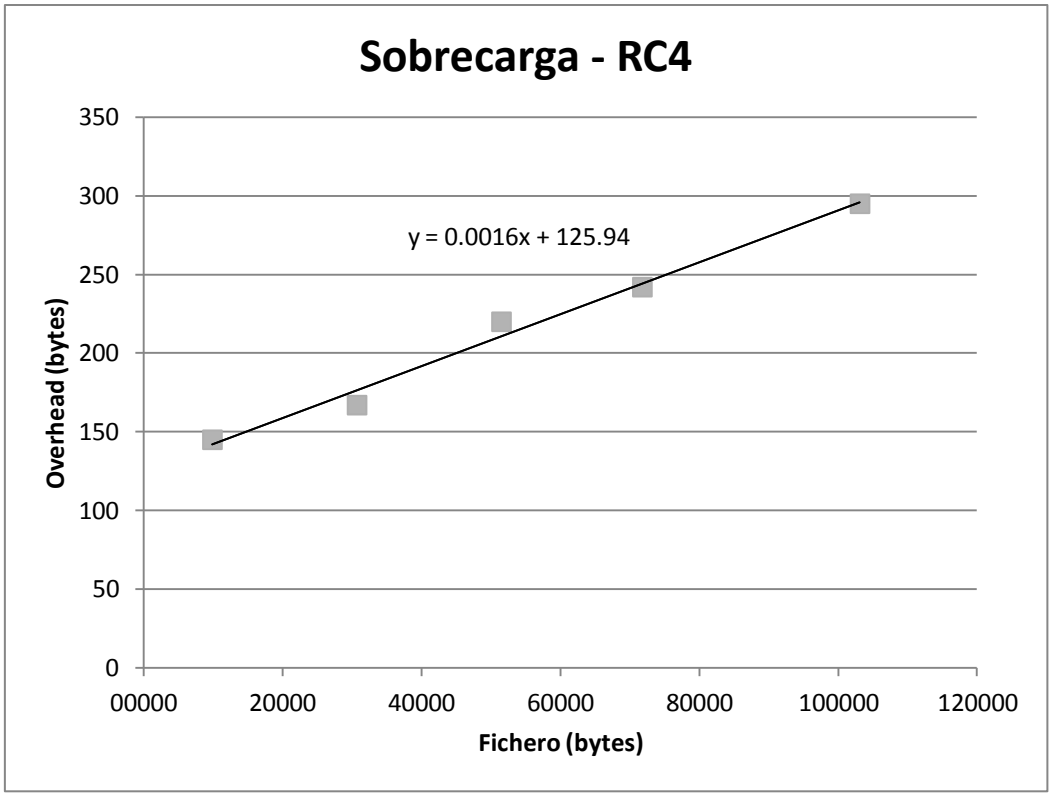


Gráfico 6. Sobrecarga de datos con sistema de cifrado RC4

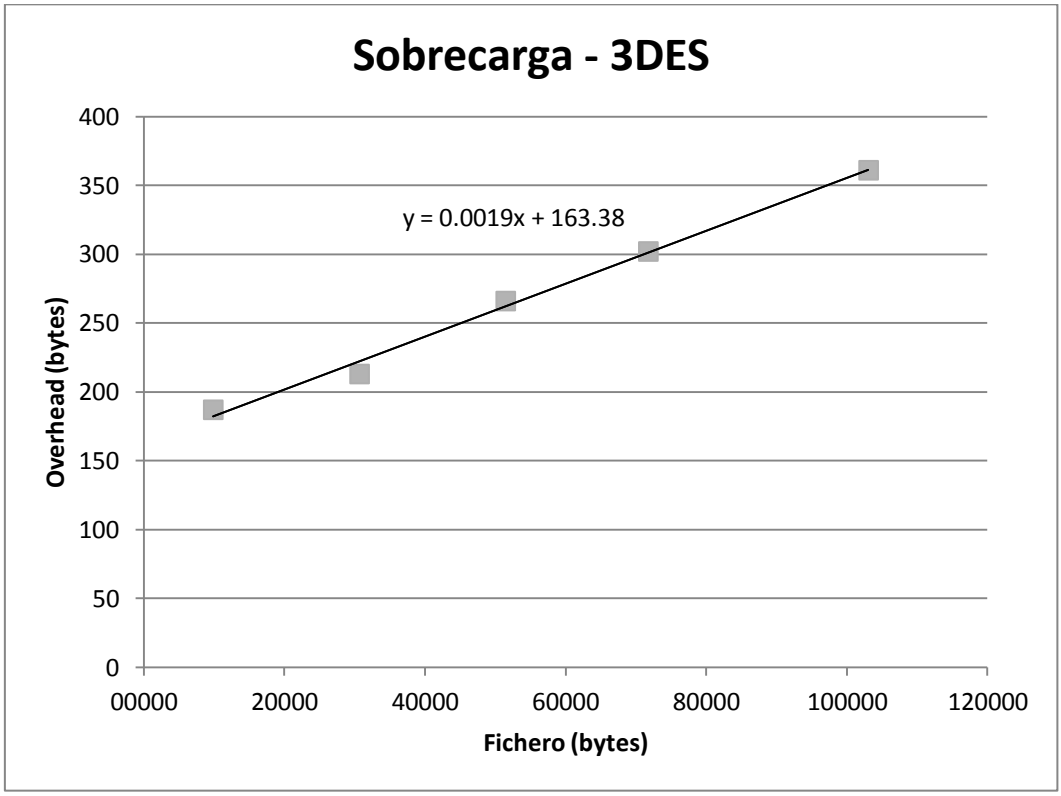


Gráfico 7. Sobrecarga de datos con sistema de cifrado 3DES

Estas pruebas se han realizado con el mismo servidor Web y con el mismo certificado; independientemente del tamaño de clave RSA utilizada para firmarlo. Es por ello que los resultados de la sobrecarga introducida por los sistemas de cifrado simetricos obtenidos se mantienen constantes. Tras haber realizado las mismas pruebas con otro servidor Web se ha podido verificar que los resultados de las sobrecargas no eran los mismos aunque no habia mucha diferencia entre ellos. Además, también cambiaban el tamaño de las cabeceras HTTP. Es por esta razón que se ha optado por establecer una regla o una formula que permita caracterizar en todo momento cualquier conexión HTTPS.

En la siguiente imagen se puede observar como los datos medidos a nivel de TCP en una comunicación segura son iguales a la sobrecarga generada por el handshake de SSL más los datos cifrados. Estos datos cifrados son la suma de los datos TCP sin cifrar mas la sobrecarga que generan los sistemas de cifrado simétrico. Por último, los datos TCP sin cifrar serían el tamaño del objeto a descargar mas las cabeceras HTTP.

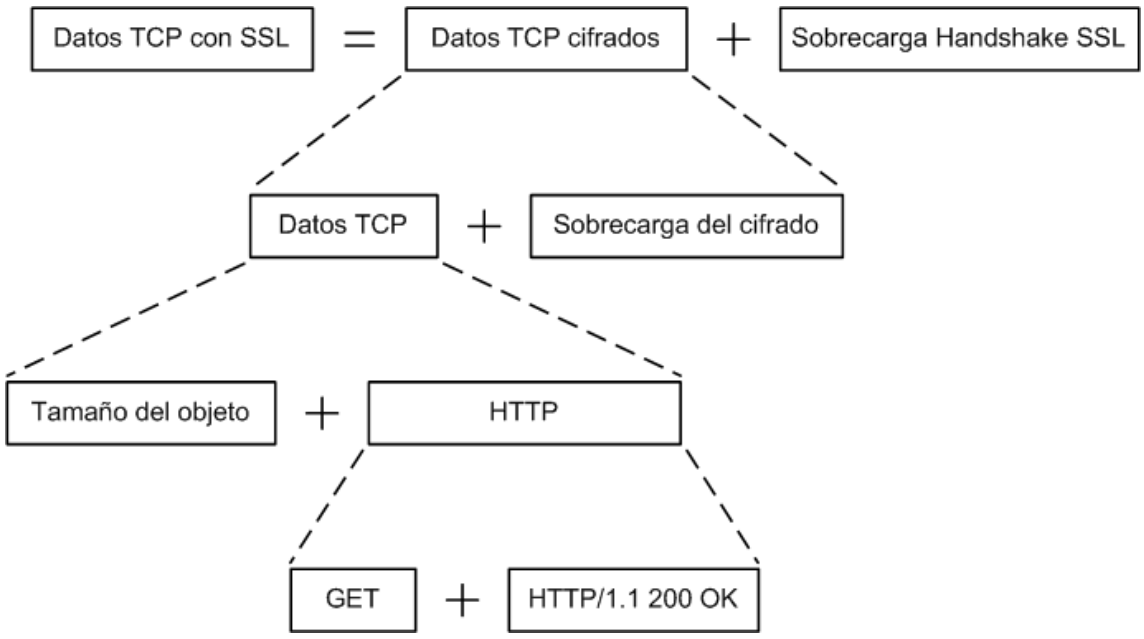


Figura 29. Regla para caracterizar conexiones HTTPS

6.2 Sobrecarga de procesado

En cuanto a la sobrecarga de procesado aunque es menos relevante, se ha comprobado como aumenta en función del tamaño de la clave RSA o el sistema de cifrado simetríco utilizado. En las siguientes gráficas puede observarse una comparativa del objeto de 10 KBytes con los distintos tamaños de la clave RSA y los distintos sistemas de cifrado simétricos utilizados.

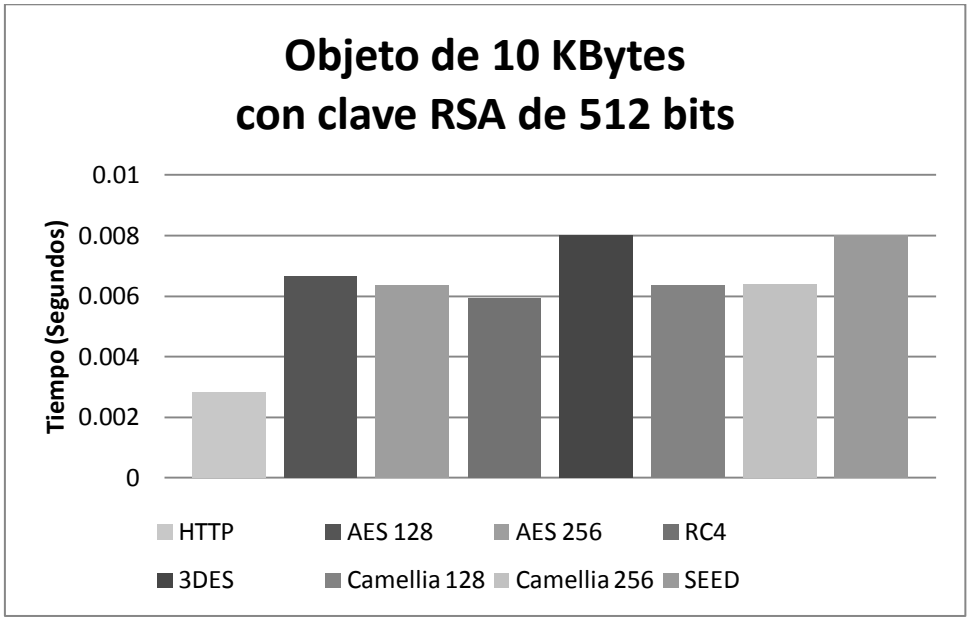


Gráfico 8. Sobrecarga de procesado del objeto de 10 KB con clave RSA de 512 bits

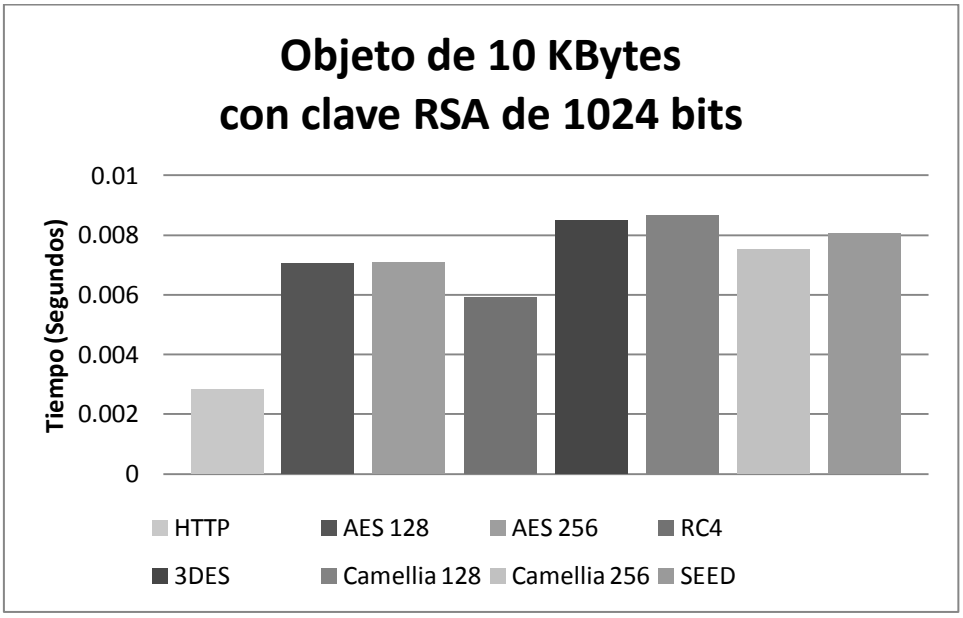


Gráfico 9. Sobrecarga de procesado del objeto de 10 KB con clave RSA de 1024 bits

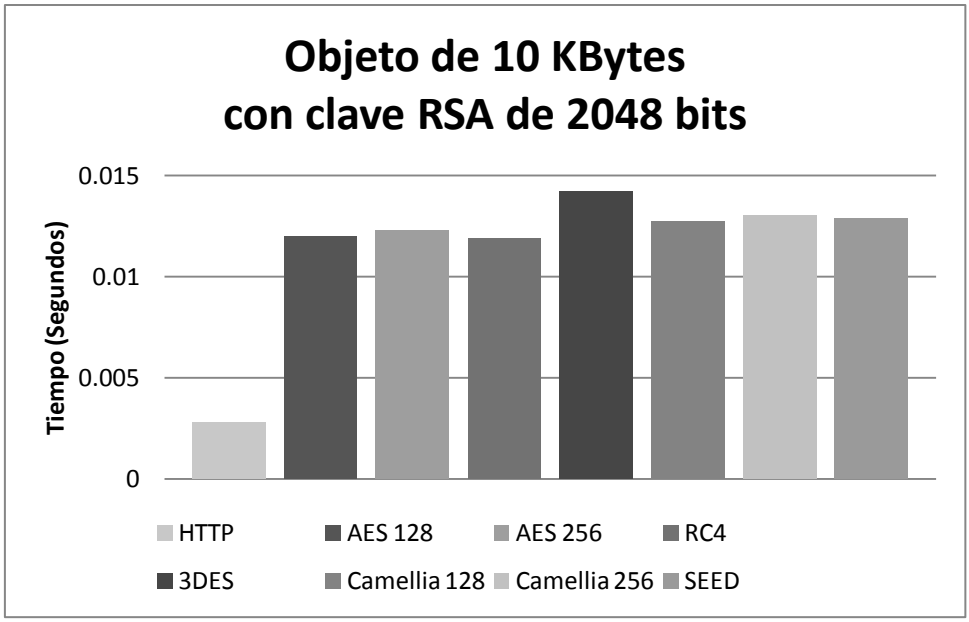


Gráfico 10. Sobrecarga de procesado del objeto de 10 KB con clave RSA de 2048 bits

Se puede apreciar como con los sistemas de cifrado RC4 y AES obtienen mejores resultados. Esto en parte es debido a que las transformaciones que utilizan para cifrar los datos están más optimizadas. Es por esta razón que el propio servidor Web Apache recomienda la utilización de estos dos sistemas si se buscar obtener una sobrecarga de procesado mínima Fig. 4.

Por otro lado, al aumentar el tamaño de la clave privada también aumenta la sobrecarga y esto se debe a que la clave secreta utilizada para el cifrado de los datos es de mayor tamaño Fig. 6 y Fig. 8.

Del mismo modo que se han analizado los resultados obtenidos con el objeto de tamaño de 10 KBytes podría hacer con los demás objetos y llegarían a obtenerse las mismas conclusiones.

7 Conclusiones

Es un hecho evidente, que en el futuro cada vez habrá una mayor necesidad de establecer comunicaciones seguras. Aunque, también es un hecho, que para establecer una comunicación segura es necesario sacrificar parte del rendimiento disponible. Tal y como se ha comprobado con la realización de este trabajo, el coste de establecer una conexión segura HTTPS depende de los algoritmos criptográficos utilizados.

El objetivo principal de este trabajo era obtener la mayor información posible de una conexión segura HTTPS con el fin de caracterizarla y medir su rendimiento. Tras un periodo de recopilación de información no se encontró nada parecido a lo qu se ha realizado en este proyecto. Sí que es cierto, que se encontraron artículos [35][36] que utilizaban los campos no cifrados de los registros SSL Fig. 3 como el campo "Compressed Length" para caracterizar una conexión HTTPS. Leyendo este campo de un flujo constante de tráfico podían distinguir qué tipo de objeto y de qué pagina Web se estaba descargando. Aunque para ello, previamente, se creaba una base de datos donde almacenaban cuantos objetos y el tamaño que tenía cada página Web.

Por otro lado, también se han analizado artículos cuyo único propósito era identificar si el tráfico que se estaba capturando estaba basando en SSL/TLS. Uno de ellos, utilizaba un método denominado DRPSD (Double Record Protocol Structure Detection) el cual comprobaba si los paquetes capturados tenían el mismo formato que los registros SSL Fig. 3 [37]. Otro utilizaba diferentes métodos de aprendizaje automático como AdaBoost, C4.5, RIPPER y Naieve Bayesing con el fin de que una maquina aprendiera a diferenciar el tráfico SSL/TLS [38].

En definitiva, mediante el desarrollo de este trabajo se pretendía ir un paso más allá, se quería conocer cómo y que cambiaba de una comunicación segura al utilizar distintos sistemas criptográficos. Se ha conseguido identificar la sobrecarga de datos y de procesado que generan los algoritmos criptográficos más utilizados actualmente con el objetivo de establecer una regla o una función que permita caracterizar cualquier conexión HTTPS.

8 Referencias

[1] “The Secure Sockets Layer (SSL) Protocol Version 3.0”, A. Freier, P. Karlton, P. Kocher, Netscape Communications, Agosto 2011, https://tools.ietf.org/html/rfc6101

[2] “What is TLS/SSL?”, TechNet, Marzo 2003, https://technet.microsoft.com/en-us/library/cc784450(v=ws.10).aspx

[3] “OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection”, H. Zimmermann, IEEE Transactions on Communications, Abril 1980

[4] “The TCP/IP model”, TechNet, Enero 2005, https://technet.microsoft.com/en-us/library/cc786900(v=ws.10).aspx

[5] IPsec, IPsec-Howto: http://www.ipsec-howto.org/spanish/x161.html

[6] “Architectural Impact of Secure Socket Layer on Internet Servers”, K.Kant, R.Iyer, P.Mahapatra, Proc. Int. Con fon Computer Design, Septiembre 2000

[7] “HTTP over SSL/TLS”, E.Rescorla, Mayo 200 https://tools.ietf.org/html/rfc2818

[8] “HyperText Transfer Protocol” B.Lee, World Wide Web Consortium, Agosto 2010

[9] “Architectural Support for Fast Symmetric-Key Cryptography”, J.Burke, J.McDonald, T.Austing, Proc. Int. Conf. Architectural Support for Programming Languages and Operating Systems, Noviembre 2000

[10] “Efficient Permutation Instructions for Fast Software Cryptography”, R.Lee, Z.Shi, X.Yang, IEEE Micro Vol. 21, Diciembre 2001

[11] “CryptoManiac: A Fast Flexible Architecture for Secure Communication”, L.Wu, C.Weaver, T.Austin, Int. Symposium on Computer Architecture, 2001

[12] “Netscape Time: The Making of the Billion-Dollar Start-Up Took on Microsoft”, J.Clark, Junio 1999

[13] “The TLS Protocol Version 1.0”, T.Dierks, C.Allen, Enero 1999, https://tools.ietf.org/html/rfc2246

[14] “The TLS Protocol Version 1.1” T.Dierks, E.Rescolar, Abril 2006, https://tools.ietf.org/html/rfc4346

[15] “Cryptography Engineering: Design Principles and Practical Applications”, N.Ferguson, B.Schneier, T.Kohno, ISBN: 978-0-470-47424-2, Marzo 2010

[16] Internet Assigned Numbers Authority – IANA, https://www.iana.org/about

[17] "The TLS Protocol Version 1.2", T.Dierks, E.Rescolar, Agosto 2008, https://tools.ietf.org/html/rfc5246

[18] "The MD5 Message-Digest Algorithm", R.Rivest, MIT Laboratory for Compurter Science and RSA Data Security, Abril 1992, http://www.ietf.org/rfc/rfc1321.txt

[19] "Prohibitin Secure Sockets Layer (SSL) Version 2.0", S.Turnet, T.Polk, Marzo 2011, https://tools.ietf.org/html/rfc6176

[20] "Introducción a los certificados digitales", S.Talens-Oliag, http://www.uv.es/sto/articulos/BEI-2003-11/certificados_digitales.html

[21] "Cryptographic Hash Function: An Elevated View", H.Tiwari, K.Asawa, European Journal of Scientific Research, 2010

[22] "The RSA Public Key Cryptosystem", W.Wardlaw, U. S Naval Academy, http://www.usna.edu/Users/math/wdj/_files/documents/papers/cryptoday/wardlaw_rsa.pdf

[23] "Internet X.509 Public Key Infrastructure Certificate Management Protocols", C.Adams, S.Farrell, Marzo 1999, https://tools.ietf.org/html/rfc2510

[24] "Advanced Encryptation Standard", Noviembre 2001, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[25] "Differential Cryptanalysis of the Data Encryption Standard", E.Biham, A.Shamir, 1993, ISBN 978-1-4613-9314-6

[26] "A Description of the Camellia Encrytion Algorithm", M.Matsui, J.Nakajima, S.Moriai, Mitsubishi Electric Corporation, Abril 2004, https://tools.ietf.org/html/rfc3713

[27] "SEED 128 Algorithm Specification", Korea Internet and Security Agency, http://seed.kisa.or.kr/html/egovframework/iwt/ds/ko/ref/%5B2%5D_SEED+128_Specification_english_M.pdf

[28] "Statistical Analysis of the Alleged RC4 Keystream Generator", S.Fluhrer and D.McGrew, Cisco Systems, Inc., http://www.mindspring.com/~dmcgrew/rc4-03.pdf

[29] "Cómo usar un certificado SSL autofirmado de forma segura", F.Lavín, Julio 2014, http://www.yukei.net/2014/07/certificado-ssl-autofirmado-seguro/

[30] "Performance Analysis of TLS Web Servers", C.Coarfa, P.Druschel, S.Wallach, Network and Distributed System Security Symposium, Febrero 2002

[31] "Anatomy and Performance of SSL Processing", L.Zhao, R.Iyer, S.Makineni, L.Bhuyan, Performance Analysis of Systems Software, IEEE Symposium, Marzo 2005

[32] “SSL, Secure Sockets Layer y Otros Protocolos para el Comercio Electrónico”, J.Morales, Universidad Politécnica de Madrid, Febrero 2002

[33] “SSL: Foundation for Web Security”, W.Stallings, The Internet Protocol Journal http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_1-1/ssl.html

[34] “Traffic Analysis of an SSL/TLS Session”, A.Castro-Castilla, Diciembre 2014, http://blog.fourthbit.com/2014/12/23/traffic-analysis-of-an-ssl-slash-tls-session

[35] “Statistical Indentification of Encrypted Web Browsing Traffic”, Q.Sun, D.Simon, Y.Wang, W.Russel, V.Pandmanabhan, L.Qiu, Derpatmen of Computer Sciene of Stanford University and Microsoft Research, 2002

[36] “Inferring the Source of Encrypted HTTP Connections”, M.Liberatore, B.Levine, Conference on Computer and Communications Security, 2006

[37] “DRPSD: An Novel Method of Identifying SSL/TLS Traffic”, C.Lu, G.Sun, Y.Xue, Conference on Dependable Computing, Noviembre 2010

[38] “An Investigation on Idenfifying SSL Traffic”, C.McCarthy, A.Nur Zincir-Heywood, Dalhousie University Canada, 2011

9 Anexo I – Resultados Obtenidos

9.1 Pruebas con clave RSA de 512 bits

Los resultados que se muestran a continuación corresponden a cuando el servidor Web utiliza una clave RSA de 512 bits.

9.1.1 Fichero de 10 KBytes

En la siguiente tabla se especifican los resultados obtenidos por el cliente al realizar una petición de un objeto de 10 KBytes.

HTTPS - con handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Tiempo (Segundos)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	25	11626	0,0066481	1108	10,5343
RSA_WITH_AES_256	25	11626	0,0063461	1108	10,5343
RSA_WITH_RC4	25	11536	0,0059131	1018	9,6786
RSA_WITH_3DES_EDE	25	11586	0,0079801	1068	10,1540
RSA_WITH_CAMELLIA_128	25	11626	0,006369	1108	10,5343
RSA_WITH_CAMELLIA_256	25	11626	0,0063718	1108	10,5343
RSA_WITH_SEED	25	11626	0,0079788	1108	10,5343

Tabla 2. Resultados – Objeto 10 KB y clave RSA 512 bits

Por otro lado, si obviamos los paquetes intercambiados en el establecimiento de la sesión de SSL, obtenemos los siguientes resultados.

HTTPS - sin handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Sobrecarga Handshake SSL (Bytes)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	21	10745	881	227	2,1582
RSA_WITH_AES_256	21	10745	881	227	2,1582
RSA_WITH_RC4	21	10663	873	145	1,3785
RSA_WITH_3DES_EDE	21	10705	881	187	1,7779
RSA_WITH_CAMELLIA_128	21	10745	881	227	2,1582
RSA_WITH_CAMELLIA_256	21	10745	881	227	2,1582
RSA_WITH_SEED	21	10745	881	227	2,1582

Tabla 3. Resultados – Objeto 10 KB, clave RSA 512 bits y sin handshake de SSL

9.1.2 Fichero de 30 KBytes

En las siguientes tablas se especifican los resultados obtenidos por el cliente al realizar una petición de un objeto de 30 KBytes.

HTTPS - con handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Tiempo (Segundos)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	44	32479	0,0088025	1134	3,6178
RSA_WITH_AES_256	44	32479	0,0093581	1134	3,6178
RSA_WITH_RC4	44	32385	0,0093084	1040	3,3179
RSA_WITH_3DES_EDE	44	32439	0,0185878	1094	3,4901
RSA_WITH_CAMELLIA_128	44	32479	0,0080006	1134	3,6178
RSA_WITH_CAMELLIA_256	44	32479	0,0085236	1134	3,6178
RSA_WITH_SEED	44	32479	0,0104524	1134	3,6178

Tabla 4. Resultados – Objeto 30 KB y clave RSA 512 bits

HTTPS - sin handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Sobrecarga Handshake SSL (Bytes)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	40	31598	881	253	0,8071
RSA_WITH_AES_256	40	31598	881	253	0,8071
RSA_WITH_RC4	40	31512	873	167	0,5327
RSA_WITH_3DES_EDE	40	31558	881	213	0,6795
RSA_WITH_CAMELLIA_128	40	31598	881	253	0,8071
RSA_WITH_CAMELLIA_256	40	31598	881	253	0,8071
RSA_WITH_SEED	40	31598	881	253	0,8071

Tabla 5. Resultados – Objeto 30 KB, clave RSA 512 bits y sin handshake de SSL

9.1.3 Fichero de 50 KBytes

En las siguientes tablas se especifican los resultados obtenidos por el cliente al realizar una petición de un objeto de 50 KBytes.

HTTPS - con handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Tiempo (Segundos)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	66	53353	0,0121909	1211	2,3225
RSA_WITH_AES_256	66	53353	0,0102747	1211	2,3225
RSA_WITH_RC4	66	53235	0,0104422	1093	2,0961
RSA_WITH_3DES_EDE	66	53289	0,0219967	1147	2,1997
RSA_WITH_CAMELLIA_128	64	53353	0,0105927	1211	2,3225
RSA_WITH_CAMELLIA_256	64	53353	0,0121632	1211	2,3225
RSA_WITH_SEED	66	53353	0,2216901	1211	2,3225

Tabla 6. Resultados – Objeto 50 KB y clave RSA 512 bits

HTTPS - sin handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Sobrecarga Handshake SSL (Bytes)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	62	52472	881	330	0,6328
RSA_WITH_AES_256	62	52472	881	330	0,6328
RSA_WITH_RC4	62	52362	873	220	0,4219
RSA_WITH_3DES_EDE	62	52408	881	266	0,5101
RSA_WITH_CAMELLIA_128	60	52472	881	330	0,6328
RSA_WITH_CAMELLIA_256	60	52472	881	330	0,6328
RSA_WITH_SEED	62	52472	881	330	0,6328

Tabla 7. Resultados – Objeto 50 KB, clave RSA 512 bits y sin handshake de SSL

9.1.4 Fichero de 70KBytes

En las siguientes tablas se especifican los resultados obtenidos por el cliente al realizar una petición de un objeto de 70 KBytes.

HTTPS - con handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Tiempo (Segundos)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	86	73678	0,0115418	1247	1,7216
RSA_WITH_AES_256	86	73678	0,0121775	1247	1,7216
RSA_WITH_RC4	86	73546	0,014638	1115	1,5393
RSA_WITH_3DES_EDE	86	73614	0,0251159	1183	1,6332
RSA_WITH_CAMELLIA_128	86	73678	0,0117624	1247	1,7216
RSA_WITH_CAMELLIA_256	86	73678	0,0120173	1247	1,7216
RSA_WITH_SEED	86	73678	0,0127815	1247	1,7216

Tabla 8. Resultados – Objeto 70 KB y clave RSA 512 bits

HTTPS - sin handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Sobrecarga Handshake SSL (Bytes)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	82	72797	881	366	0,5053
RSA_WITH_AES_256	82	72797	881	366	0,5053
RSA_WITH_RC4	82	72673	873	242	0,3341
RSA_WITH_3DES_EDE	82	72733	881	302	0,4169
RSA_WITH_CAMELLIA_128	82	72797	881	366	0,5053
RSA_WITH_CAMELLIA_256	82	72797	881	366	0,5053
RSA_WITH_SEED	82	72797	881	366	0,5053

Tabla 9. Resultados – Objeto 70 KB, clave RSA 512 bits y sin handshake de SSL

9.1.5 Fichero de 100 KBytes

En las siguientes tablas se especifican los resultados obtenidos por el cliente al realizar una petición de un objeto de 100 KBytes.

HTTPS - con handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Tiempo (Segundos)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	118	105096	0,0143575	1322	1,2739
RSA_WITH_AES_256	118	105096	0,0154835	1322	1,2739
RSA_WITH_RC4	116	104942	0,0144810	1168	1,1255
RSA_WITH_3DES_EDE	116	105016	0,0273576	1242	1,1968
RSA_WITH_CAMELLIA_128	118	105096	0,0145918	1322	1,2739
RSA_WITH_CAMELLIA_256	119	105096	0,0165426	1322	1,2739
RSA_WITH_SEED	118	105096	0,2156697	1322	1,2739

Tabla 10. Resultados – Objeto 100 KB y clave RSA 512 bits

HTTPS - sin handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Sobrecarga Handshake SSL (Bytes)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	114	104215	881	441	0,4296
RSA_WITH_AES_256	114	104215	881	441	0,4296
RSA_WITH_RC4	112	104069	873	295	0,2842
RSA_WITH_3DES_EDE	112	104135	881	361	0,3478
RSA_WITH_CAMELLIA_128	114	104215	881	441	0,4296
RSA_WITH_CAMELLIA_256	115	104215	881	441	0,4296
RSA_WITH_SEED	114	104215	881	441	0,4296

Tabla 11. Resultados – Objeto 100 KB, clave RSA 512 bits y sin handshake de SSL

9.2 Pruebas con clave RSA de 1024 bits

Los resultados que se muestran a continuación corresponden a cuando el servidor Web utiliza un certificado digital de 1024 bits.

9.2.1 Fichero de 10 KBytes

En las siguientes tablas se especifican los resultados obtenidos por el cliente al realizar una petición de un objeto de 10 KBytes.

HTTPS - con handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Tiempo (Segundos)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	25	11823	0,0070653	1305	12,4073
RSA_WITH_AES_256	25	11823	0,0070984	1305	12,4073
RSA_WITH_RC4	25	11733	0,0059222	1215	9,6786
RSA_WITH_3DES_EDE	25	11783	0,0085012	1265	10,1540
RSA_WITH_CAMELLIA_128	25	11823	0,0086424	1305	10,5343
RSA_WITH_CAMELLIA_256	25	11823	0,0075275	1305	10,5343
RSA_WITH_SEED	25	11823	0,0080508	1305	10,5343

Tabla 12. Resultados – Objeto 10 KB y clave RSA 1024 bits

HTTPS - sin handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Sobrecarga Handshake SSL (Bytes)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	21	10745	1078	227	2,1582
RSA_WITH_AES_256	21	10745	1078	227	2,1582
RSA_WITH_RC4	21	10663	1070	145	1,3785
RSA_WITH_3DES_EDE	21	10705	1078	187	1,7779
RSA_WITH_CAMELLIA_128	21	10745	1078	227	2,1582
RSA_WITH_CAMELLIA_256	21	10745	1078	227	2,1582
RSA_WITH_SEED	21	10745	1078	227	2,1582

Tabla 13. Resultados – Objeto 10 KB, clave RSA 1024 bits y sin handshake de SSL

9.2.2 Fichero de 30 KBytes

En las siguientes tablas se especifican los resultados obtenidos por el cliente al realizar una petición de un objeto de 30 KBytes.

HTTPS - con handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Tiempo (Segundos)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	44	32676	0,0088025	1331	4,2462
RSA_WITH_AES_256	44	32676	0,0093581	1331	4,2462
RSA_WITH_RC4	44	32582	0,0093084	1237	3,9464
RSA_WITH_3DES_EDE	44	32636	0,0185878	1291	4,1186
RSA_WITH_CAMELLIA_128	44	32676	0,0080006	1331	4,2462
RSA_WITH_CAMELLIA_256	44	32676	0,0085236	1331	4,2462
RSA_WITH_SEED	44	32676	0,0104524	1331	4,2462

Tabla 14. Resultados – Objeto 30 KB y clave RSA 1024 bits

HTTPS - sin handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Sobrecarga Handshake SSL (Bytes)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	40	31598	1078	253	0,8071
RSA_WITH_AES_256	40	31598	1078	253	0,8071
RSA_WITH_RC4	40	31512	1070	167	0,5327
RSA_WITH_3DES_EDE	40	31558	1078	213	0,6795
RSA_WITH_CAMELLIA_128	40	31598	1078	253	0,8071
RSA_WITH_CAMELLIA_256	40	31598	1078	253	0,8071
RSA_WITH_SEED	40	31598	1078	253	0,8071

Tabla 15. Resultados – Objeto 30 KB, clave RSA 1024 bits y sin handshake de SSL

9.2.3 Fichero de 50 KBytes

En las siguientes tablas se especifican los resultados obtenidos por el cliente al realizar una petición de un objeto de 50 KBytes.

HTTPS - con handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Tiempo (Segundos)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	66	53550	0,0104554	1408	2,7003
RSA_WITH_AES_256	66	53550	0,0114309	1408	2,7003
RSA_WITH_RC4	66	53432	0,0112121	1290	2,4740
RSA_WITH_3DES_EDE	66	53486	0,017644	1344	2,5775
RSA_WITH_CAMELLIA_128	66	53550	0,011567	1408	2,7003
RSA_WITH_CAMELLIA_256	64	53550	0,012589	1408	2,7003
RSA_WITH_SEED	66	53550	0,216062	1408	2,7003

Tabla 16. Resultados – Objeto 50 KB y clave RSA 1024 bits

HTTPS - sin handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Sobrecarga Handshake SSL (Bytes)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	62	52472	1078	330	0,6328
RSA_WITH_AES_256	62	52472	1078	330	0,6328
RSA_WITH_RC4	62	52362	1070	220	0,4219
RSA_WITH_3DES_EDE	62	52408	1078	266	0,5101
RSA_WITH_CAMELLIA_128	62	52472	1078	330	0,6328
RSA_WITH_CAMELLIA_256	60	52472	1078	330	0,6328
RSA_WITH_SEED	62	52472	1078	330	0,6328

Tabla 17. Resultados – Objeto 50 KB, clave RSA 1024 bits y sin handshake de SSL

9.2.4 Fichero de 70 KBytes

En las siguientes tablas se especifican los resultados obtenidos por el cliente al realizar una petición de un objeto de 70 KBytes.

HTTPS - con handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Tiempo (Segundos)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	86	73875	0,0132149	1444	1,9936
RSA_WITH_AES_256	86	73875	0,0164209	1444	1,9936
RSA_WITH_RC4	86	73743	0,0160128	1312	1,8113
RSA_WITH_3DES_EDE	86	73811	0,0225603	1380	1,6332
RSA_WITH_CAMELLIA_128	86	73875	0,0125274	1444	1,7216
RSA_WITH_CAMELLIA_256	86	73875	0,0127466	1444	1,7216
RSA_WITH_SEED	86	73875	0,0138406	1444	1,7216

Tabla 18. Resultados – Objeto 70 KB y clave RSA 1024 bits

HTTPS - sin handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Sobrecarga Handshake SSL (Bytes)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	82	72797	1078	366	0,5053
RSA_WITH_AES_256	82	72797	1078	366	0,5053
RSA_WITH_RC4	82	72673	1070	242	0,3341
RSA_WITH_3DES_EDE	82	72733	1078	302	0,4169
RSA_WITH_CAMELLIA_128	82	72797	1078	366	0,5053
RSA_WITH_CAMELLIA_256	82	72797	1078	366	0,5053
RSA_WITH_SEED	82	72797	1078	366	0,5053

Tabla 19. Resultados – Objeto 70 KB, clave RSA 1024 bits y sin handshake de SSL

9.2.5 Fichero de 100 KBytes

En las siguientes tablas se especifican los resultados obtenidos por el cliente al realizar una petición de un objeto de 100 KBytes.

HTTPS - con handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Tiempo (Segundos)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	116	105293	0,0151633	1519	1,4637
RSA_WITH_AES_256	116	105293	0,0153139	1519	1,4637
RSA_WITH_RC4	116	104139	0,0153724	1365	1,3153
RSA_WITH_3DES_EDE	116	105213	0,02736	1439	1,3866
RSA_WITH_CAMELLIA_128	118	105293	0,0152264	1519	1,4637
RSA_WITH_CAMELLIA_256	118	105293	0,0153692	1519	1,4637
RSA_WITH_SEED	118	105293	0,224224	1519	1,4637

Tabla 20. Resultados – Objeto 100 KB y clave RSA 1024 bits

HTTPS - sin handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Sobrecarga Handshake SSL (Bytes)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	112	104215	1078	441	0,4296
RSA_WITH_AES_256	112	104215	1078	441	0,4296
RSA_WITH_RC4	112	104069	1070	295	0,2842
RSA_WITH_3DES_EDE	112	104135	1078	361	0,3478
RSA_WITH_CAMELLIA_128	114	104215	1078	441	0,4296
RSA_WITH_CAMELLIA_256	114	104215	1078	441	0,4296
RSA_WITH_SEED	114	104215	1078	441	0,4296

Tabla 21. Resultados – Objeto 100 KB, clave RSA 1024 bits y sin handshake de SSL

9.3 Pruebas con clave RSA de 2048 bits

Los resultados que se muestran a continuación corresponden a cuando el servidor Web utiliza un certificado digital de 2048 bits.

9.3.1 Fichero de 10 KBytes

En las siguientes tablas se especifican los resultados obtenidos por el cliente al realizar una petición de un objeto de 10 KBytes.

HTTPS - con handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Tiempo (Segundos)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	25	12212	0,0120128	1694	16,1057
RSA_WITH_AES_256	25	12212	0,0123136	1694	16,1057
RSA_WITH_RC4	25	12122	0,0119101	1604	15,2500
RSA_WITH_3DES_EDE	25	12172	0,0142215	1654	15,7254
RSA_WITH_CAMELLIA_128	25	12212	0,0127337	1694	16,1057
RSA_WITH_CAMELLIA_256	25	12212	0,0130668	1694	16,1057
RSA_WITH_SEED	25	12212	0,0129113	1694	16,1057

Tabla 22. Resultados – Objeto 10 KB y clave RSA 2048 bits

HTTPS - sin handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Sobrecarga Handshake SSL (Bytes)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	21	10745	1467	227	2,1582
RSA_WITH_AES_256	21	10745	1467	227	2,1582
RSA_WITH_RC4	21	10663	1459	145	1,3785
RSA_WITH_3DES_EDE	21	10705	1467	187	1,7779
RSA_WITH_CAMELLIA_128	21	10745	1467	227	2,1582
RSA_WITH_CAMELLIA_256	21	10745	1467	227	2,1582
RSA_WITH_SEED	21	10745	1467	227	2,1582

Tabla 23. Resultados – Objeto 10 KB, clave RSA 2048 bits y sin handshake de SSL

9.3.2 Fichero de 30 KBytes

En las siguientes tablas se especifican los resultados obtenidos por el cliente al realizar una petición de un objeto de 30 KBytes.

HTTPS - con handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Tiempo (Segundos)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	44	33065	0,0206354	1720	5,4873
RSA_WITH_AES_256	44	33065	0,0146231	1720	5,4873
RSA_WITH_RC4	44	32971	0,0140606	1626	5,1874
RSA_WITH_3DES_EDE	44	33025	0,0186227	1680	5,3597
RSA_WITH_CAMELLIA_128	44	33065	0,0215654	1720	5,4873
RSA_WITH_CAMELLIA_256	44	33065	0,0143781	1720	5,4873
RSA_WITH_SEED	44	33065	0,0150438	1720	5,4873

Tabla 24. Resultados – Objeto 30 KB y clave RSA 2048 bits

HTTPS - sin handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Sobrecarga Handshake SSL (Bytes)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	40	31598	1467	253	0,8071
RSA_WITH_AES_256	40	31598	1467	253	0,8071
RSA_WITH_RC4	40	31512	1459	167	0,5327
RSA_WITH_3DES_EDE	40	31558	1467	213	0,6795
RSA_WITH_CAMELLIA_128	40	31598	1467	253	0,8071
RSA_WITH_CAMELLIA_256	40	31598	1467	253	0,8071
RSA_WITH_SEED	40	31598	1467	253	0,8071

Tabla 25. Resultados – Objeto 30 KB, clave RSA 2048 bits y sin handshake de SSL

9.3.3 Fichero de 50 KBytes

En las siguientes tablas se especifican los resultados obtenidos por el cliente al realizar una petición de un objeto de 50 KBytes.

HTTPS - con handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Tiempo (Segundos)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	66	53939	0,022602	1797	3,4463
RSA_WITH_AES_256	66	53939	0,0159492	1797	3,4463
RSA_WITH_RC4	66	53821	0,021014	1679	3,22
RSA_WITH_3DES_EDE	66	53875	0,023641	1733	3,3236
RSA_WITH_CAMELLIA_128	66	53939	0,0158419	1797	3,4463
RSA_WITH_CAMELLIA_256	64	53939	0,0157284	1797	3,4463
RSA_WITH_SEED	66	53939	0,0155818	1797	3,4463

Tabla 26. Resultados – Objeto 50 KB y clave RSA 2048 bits

HTTPS - sin handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Sobrecarga Handshake SSL (Bytes)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	62	52472	1467	330	0,6328
RSA_WITH_AES_256	62	52472	1467	330	0,6328
RSA_WITH_RC4	62	52362	1459	220	0,4219
RSA_WITH_3DES_EDE	62	52408	1467	266	0,5101
RSA_WITH_CAMELLIA_128	62	52472	1467	330	0,6328
RSA_WITH_CAMELLIA_256	60	52472	1467	330	0,6328
RSA_WITH_SEED	62	52472	1467	330	0,6328

Tabla 27. Resultados – Objeto 50 KB, clave RSA 2048 bits y sin handshake de SSL

9.3.4 Fichero de 70 KBytes

En las siguientes tablas se especifican los resultados obtenidos por el cliente al realizar una petición de un objeto de 70 KBytes.

HTTPS - con handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Tiempo (Segundos)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	85	74264	0,0278089	1833	2,5306
RSA_WITH_AES_256	86	74264	0,0174102	1833	2,5306
RSA_WITH_RC4	85	74132	0,0204937	1701	2,3484
RSA_WITH_3DES_EDE	86	74200	0,0279552	1769	2,4423
RSA_WITH_CAMELLIA_128	86	74264	0,0178473	1833	2,5306
RSA_WITH_CAMELLIA_256	86	74264	0,0183212	1833	2,5306
RSA_WITH_SEED	86	74264	0,0182544	1833	2,5306

Tabla 28. Resultados – Objeto 70 KB y clave RSA 2048 bits

HTTPS - sin handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Sobrecarga Handshake SSL (Bytes)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	81	72797	1467	366	0,5053
RSA_WITH_AES_256	82	72797	1467	366	0,5053
RSA_WITH_RC4	81	72673	1459	242	0,3341
RSA_WITH_3DES_EDE	82	72733	1467	302	0,4169
RSA_WITH_CAMELLIA_128	82	72797	1467	366	0,5053
RSA_WITH_CAMELLIA_256	82	72797	1467	366	0,5053
RSA_WITH_SEED	82	72797	1467	366	0,5053

Tabla 29. Resultados – Objeto 70 KB, clave RSA 2048 bits y sin handshake de SSL

9.3.5 Fichero de 100 KBytes

En las siguientes tablas se especifican los resultados obtenidos por el cliente al realizar una petición de un objeto de 100 KBytes.

HTTPS - con handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Tiempo (Segundos)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	118	105682	0,0270501	1908	1,8386
RSA_WITH_AES_256	116	105682	0,0205471	1908	1,8386
RSA_WITH_RC4	116	105528	0,0200512	1754	1,6902
RSA_WITH_3DES_EDE	117	105602	0,0300139	1828	1,7615
RSA_WITH_CAMELLIA_128	118	105682	0,0202596	1908	1,8386
RSA_WITH_CAMELLIA_256	118	105682	0,0223023	1908	1,8386
RSA_WITH_SEED	118	105682	0,2264705	1908	1,8386

Tabla 30. Resultados – Objeto 100 KB y clave RSA 2048 bits

HTTPS - sin handshake de SSL					
Cipher Suite	**Paquetes**	**TCP (Bytes)**	**Sobrecarga Handshake SSL (Bytes)**	**Sobrecarga (Bytes)**	**Sobrecarga (%)**
RSA_WITH_AES_128	114	104215	1467	441	0,4296
RSA_WITH_AES_256	112	104215	1467	441	0,4296
RSA_WITH_RC4	112	104069	1459	295	0,2842
RSA_WITH_3DES_EDE	113	104135	1467	361	0,3478
RSA_WITH_CAMELLIA_128	114	104215	1467	441	0,4296
RSA_WITH_CAMELLIA_256	114	104215	1467	441	0,4296
RSA_WITH_SEED	114	104215	1467	441	0,4296

Tabla 31. Resultados – Objeto 100 KB, clave RSA 2048 bits y sin handshake de SSL