

TP8 - Criptografía y protocolos de comunicación

Fecha de entrega: 17/11/2021

Franco, Juan Martín 149.615

juanmartin_franco@hotmail.com

Experiencia de laboratorio

PARTE 1 - OpenPGP: Cifrado y firma.

1. Gracias a las características de la criptografía asimétrica tenemos a disposición nuevas herramientas para lograr servicios de seguridad en nuestros mensajes. Recordemos que en un sistema criptográfico asimétrico cada participante del sistema tiene dos claves: una clave que se comparte y que denominamos clave pública y otra que se mantiene oculta (y en secreto) y que denominamos clave privada. Sabemos, además, que lo que se cifra con una clave solo se puede descifrar con la otra. Finalmente y, como si fuera poco, también podemos cifrar un mensaje más de una vez y con distintas claves. Siempre que hagamos el camino inverso en orden y con las claves correctas lograremos recuperar el mensaje original.

Vamos a hacer algunos ejercicios conceptuales usando dos interlocutores: Alicia y Beto, para ejercitar estas ideas.

Primer escenario: (a) Si Alicia cifra un mensaje con su clave privada y comparte el mensaje cifrado en Internet. ¿Qué servicio de seguridad se lograría para su mensaje? Recordemos que los servicios que tienen sentido evaluar aquí son:

- Integridad: el mensaje original no ha sido modificado o alterado por un tercero.
- Confidencialidad: solo las partes autorizadas pueden acceder a su contenido.
- Autenticidad: se puede saber quién es el emisor del mensaje.
- No repudio: se puede saber quién es el emisor del mensaje y el emisor no puede negar que lo ha enviado.

Para resolver esta pregunta vamos a traducir primero esto en los términos habituales de un sistema criptográfico:

$$C_{k_{priv}Alicia}(M) = M'$$

Es decir, ciframos con la clave privada de Alicia ($k_{priv}Alicia$) un mensaje M y este mensaje cifrado resultante M' lo compartimos en internet. Vamos a razonar ahora: M' solo puede ser descifrado con la clave pública de Alicia; por otro lado la clave pública de Alicia es compartida así que cualquiera que la tenga podría descifrarlo. Entonces, si cualquiera puede descifrarlo, ¿tenemos garantizado algún servicio de seguridad?

Por lo pronto la confidencialidad no la podemos lograr. Pero, *¿qué pasa con el resto de los servicios?* Como solo Alicia conoce su clave privada, nadie más que ella podría haber generado ese mensaje, por lo que tenemos garantizada la autenticidad así también como el no repudio (aunque Alicia lo niegue, solo ella puede haber generado ese mensaje). ¿Y la integridad? Este es un punto difícil sin el contexto de qué es M. Vamos a desarrollar el por qué:

Si M es un mensaje en texto claro, por ejemplo “**Hola Beto, ¿Como está todo por allá?**”, el mensaje cifrado M' podría, tranquilamente, verse así “**j10asdmfax0i12kj**”. Como Beto no conoce el mensaje, pero sabe que está esperando recibir un mensaje de texto plano, espera que el mensaje descifrado sea de ese tipo. Si alguien modificara el mensaje M y generara un mensaje falso F y se intentara descifrarlo con la clave pública de Alicia es casi imposible que ese mensaje descifrado se parezca a un texto.

Es decir por lo pronto parecería que la *integridad* si es posible lograrla. Pero, *¿qué pasa si el mensaje es en realidad la salida de un sensor que comprende una lista de números?* ¿o si no sabemos qué tipo de mensaje estamos esperando? Se necesita de más herramientas para poder lograr la integridad de este mensaje si no sabemos cuál es el formato de lo que estamos esperando.

Para pensar: ¿Existe otra herramienta que en conjunto podría ayudar a brindar integridad?

Vamos con otro ejemplo: (b) Alicia cifra un mensaje con su clave pública y comparte el mensaje cifrado en Internet. Dicho de otra manera:

$$C_{k_{pub}Alicia}(M) = M'$$

¿Quién puede descifrar este mensaje? Todo aquel que posea la clave privada de Alicia. Pero si Alicia hizo las cosas bien, esa clave solo la conoce ella. Por lo que la única que puede descifrar este mensaje sería ella. Si el destinatario de este mensaje es Beto, y si bien el mensaje M es secreto (tal vez demasiado) no se cumple la confidencialidad ya que el destinatario del mensaje no puede verlo. Igualmente tenemos otro problema más: ¿quién puede generar ese mensaje M'? Todo aquel que posea la clave pública de Alicia que es, por definición, la clave que se pone a disposición de todos. En definitiva este mensaje lo pudo haber generado cualquiera. No tiene mucho sentido a partir de aquí pensar en el resto de los servicios.

Un tercer caso de ejemplo: (c) Alicia cifra un mensaje con la clave pública de Beto y envía el mensaje cifrado a Beto. Dicho de otra manera:

$$C_{k_{pub}Beto}(M) = M'$$

Alicia tiene la clave pública de Beto por lo que este es un escenario posible. Este mensaje solo puede ser descifrado por quien tenga la clave privada de Beto, por lo que solo Beto podría hacerlo. Tenemos garantizada la confidencialidad. Pero, ¿quién pudo haber generado este mensaje? Cualquiera que tenga la clave pública

de Beto, que es pública. Es decir que cualquiera pudo haber generado M' . El resto de los servicios por lo tanto no se pueden cumplir: no sabemos quién lo pudo haber enviado y mucho menos si el mensaje es íntegro.

Vamos con el último caso de ejemplo: (d) Beto genera un mensaje, obtiene un resumen del mismo, cifra el resumen con su clave pública y comparte el mensaje y el resumen cifrado en Internet. Dicho de otro modo:

$$\text{Hash}(M) = H \quad (\text{Hash es una función de resumen})$$

$$C_{k_{\text{pub}}\text{Beto}}(H) = H'$$

Beto publica M y H' en Internet

En este caso Beto genera un mensaje M , y luego genera un resumen del mismo, H . El problema aquí es que Beto cifra el resumen con su clave pública, es decir que estamos en un caso similar a b), donde cualquiera pudo haber cifrado este resumen (la clave pública de Beto es justamente pública).

Como no podemos saber quién es el emisor de este mensaje, ni decir si está íntegro (cualquiera puede modificar y generar su propio resumen cifrado con $k_{\text{pub}}\text{Beto}$) este mensaje, por más bonito y sofisticado que se vea, no tiene garantizado ningún servicio de seguridad.

2. OpenPGP es un proyecto de software libre muy popular para su uso en correo electrónico e implementa el estándar de la IETF RFC 4880. Con este software es posible generar claves públicas y privadas, cifrar mensajes con criptografía simétrica y asimétrica así también como generar y verificar firmas digitales. Para utilizarlo instale el paquete **gnupg**.

En lo que queda de esta parte de la experiencia de laboratorio vamos a descubrir algunas de sus utilidades. Para ello primero vamos a conocer la versión que instalamos del software y los algoritmos de cifrado que puede utilizar: **gpg --version**. Una posible salida de este programa es:

```
gpg (GnuPG) 2.2.4
libgcrypt 1.8.1
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <https://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.

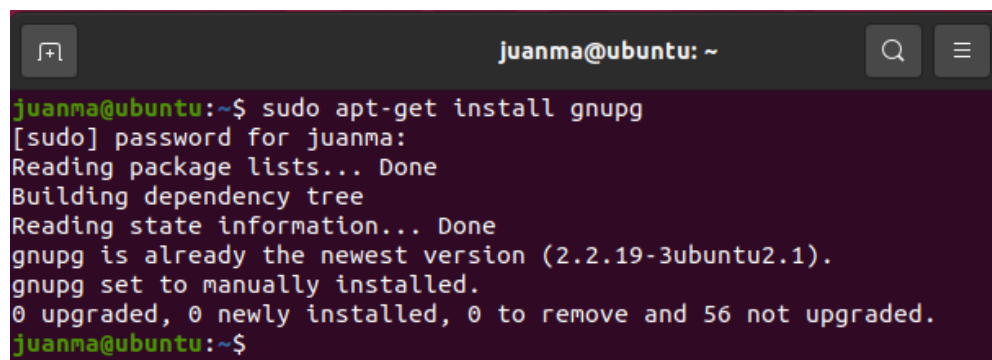
There is NO WARRANTY, to the extent permitted by law.

Home: /home/alejandro/.gnupg
Supported algorithms:
Pubkey: RSA, ELG, DSA, ECDH, ECDSA, EDDSA
Cipher: IDEA, 3DES, CAST5, BLOWFISH, AES, AES192, AES256, TWOFISH,
        CAMELLIA128, CAMELLIA192, CAMELLIA256
Hash: SHA1, RIPEMD160, SHA256, SHA384, SHA512, SHA224
Compression: Uncompressed, ZIP, ZLIB, BZIP2
```

Aquí podemos ver por ejemplo que el programa soporta para clave asimétrica RSA y DSA; para criptografía de clave simétrica IDEA y AES; y finalmente que soporta SHA1 y SHA256 como algoritmo de hash o función resumen entre otros tantos algoritmos.

Para instalar el paquete gnupg, utilizo el comando:

```
sudo apt-get install gnupg
```


A terminal window with a dark background and light text. The prompt is 'juanma@ubuntu: ~'. The user enters 'sudo apt-get install gnupg'. The terminal shows the password prompt, then the package lists, dependency tree, and state information. It confirms that gnupg is already the newest version (2.2.19-3ubuntu2.1) and is set to manually installed. The output shows 0 upgraded, 0 newly installed, 0 to remove, and 56 not upgraded.

```
juanma@ubuntu:~$ sudo apt-get install gnupg
[sudo] password for juanma:
Reading package lists... Done
Building dependency tree
Reading state information... Done
gnupg is already the newest version (2.2.19-3ubuntu2.1).
gnupg set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 56 not upgraded.
juanma@ubuntu:~$
```

En este caso, yo ya lo tenía instalado.

Ahora, ejecuto el comando

```
gpg --version
```

A terminal window showing the output of the 'gpg --version' command. The output lists the version (2.2.19), libgcrypt version (1.8.5), copyright (C) 2019 Free Software Foundation, Inc., and the license (GPLv3+). It also lists supported algorithms: RSA, ELG, DSA, ECDH, ECDSA, EDDSA for public keys; IDEA, 3DES, CAST5, BLOWFISH, AES, AES192, AES256, TWOFISH, CAMELLIA128, CAMELLIA192, CAMELLIA256 for ciphers; and SHA1, RIPEMD160, SHA256, SHA384, SHA512, SHA224 for hashes. Compression methods listed are Uncompressed, ZIP, ZLIB, and BZIP2.

```
juanma@ubuntu:~$ gpg --version
gpg (GnuPG) 2.2.19
libgcrypt 1.8.5
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <https://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Home: /home/juanma/.gnupg
Supported algorithms:
Pubkey: RSA, ELG, DSA, ECDH, ECDSA, EDDSA
Cipher: IDEA, 3DES, CAST5, BLOWFISH, AES, AES192, AES256, TWOFISH,
        CAMELLIA128, CAMELLIA192, CAMELLIA256
Hash: SHA1, RIPEMD160, SHA256, SHA384, SHA512, SHA224
Compression: Uncompressed, ZIP, ZLIB, BZIP2
```

3. Descargue el siguiente [archivo de texto](#) (o genere un archivo propio de texto plano) y luego cifrelo con un algoritmo simétrico de su elección:

gpg --symmetric --cipher-algo ALGORITMO ARCHIVO.EXT

Una vez descargado el archivo de texto provisto por el enunciado, elijo un algoritmo simétrico, por ejemplo: AES.

ATENCIÓN --cipher-algo es el nombre del parámetro completo (no significa que deba remplazar la palabra algo por otra cosa).

Por lo que utilizo el siguiente comando para cifrar dicho archivo:

```
gpg --symmetric --cipher-algo ALGORITMO ARCHIVO.EXT
```

```
juanma@ubuntu: ~/Desktop/TP 8
juanma@ubuntu:~/Desktop/TP 8$ gpg --symmetric --cipher-algo AES pg25525.txt
juanma@ubuntu:~/Desktop/TP 8$
```

Ahora compare ambos archivos, el original y el cifrado. ¿Son iguales? Usando el comando cat vea el contenido de ambos. ¿Puede identificar algo en el archivo que ha sido cifrado?

Ahora, utilizo el comando cat seguido del nombre del archivo para ver el contenido del mismo:

```
cat pg25525.txt
```

```
Section 5. General Information About Project Gutenberg-tm electronic works

Professor Michael S. Hart was the originator of the Project
Gutenberg-tm concept of a library of electronic works that could be
freely shared with anyone. For forty years, he produced and
distributed Project Gutenberg-tm eBooks with only a loose network of
volunteer support.

Project Gutenberg-tm eBooks are often created from several printed
editions, all of which are confirmed as not protected by copyright in
the U.S. unless a copyright notice is included. Thus, we do not
necessarily keep eBooks in compliance with any particular paper
edition.
```

Obviamente, al ejecutar cat sobre el archivo original, se ve el contenido del mismo sin ningún cambio.

Ahora, ejecuto el mismo comando sobre el archivo cifrado:

```
cat pg25525.txt.gpg
```

```
#####Zq<7#####F=====sl
                                X  %*r( :pS
                                bbgj#####.o6Mf0rqyeZl
K~1w&7d_N>8#####2U    cEl%l%o  ##2d*9cp+3^#####0a15Ü.##M#c'#=jnc
t+%wf4徴
    zÊ6l##M_9BSM###[?c#####i(eeA####a_s##F&##?d6,S#
Q=
  oOSQn##*R[##JH
    M'  #
    y+i##hT##0#
    X##{##1###~:F@:#####3###Hbî!###
                                l2<#
                                }h
###c3!#!Rp##)p0##|g/!a(-0##2###\v##NG##^#a}#@<q###Q#
                                I#g4VPJ##o##'~#P2
##hm_###p{#Z###t{%d####`##3#4#0##\pd###:##3|}ASUH##u:
:U%#f~#F##x####
```

Los archivos no son iguales, ni en peso ni en contenido.
En el archivo cifrado no es posible identificar nada.

4. Ahora vamos a descifrar el archivo generado utilizando el comando:

gpg --decrypt ARCHIVO.EXT.gpg > descifrado.txt

ATENCIÓN: Si el programa no le pidió la contraseña es porque el agente gpg la almacenó en memoria. Para forzar al agente a que olvide las contraseñas almacenadas ejecute: **gpgconf --reload gpg-agent** y vuelva a intentar descifrarlo.

Ahora, procedo a descifrar el archivo cifrado previamente, mediante el uso del comando:

```
gpg --decrypt pg25525.txt.gpg > descifrado.txt
```

```
juanma@ubuntu:~/Desktop/TP 8$ gpg --decrypt pg25525.txt.gpg > descifrado.txt
gpg: AES encrypted data
gpg: encrypted with 1 passphrase
juanma@ubuntu:~/Desktop/TP 8$
```

¿Por qué cree que el programa no necesita un parámetro para saber con qué algoritmo fue cifrado?

Sabemos que para brindar seguridad, una buena práctica es que haya transparencia (es decir, hay que evitar basarse en algoritmos secretos). En este caso, elegimos dicho algoritmo de cifrado porque tenemos confianza en él y porque sabemos que es robusto.

Se puede afirmar que la fortaleza del algoritmo está en la clave y no en el método que utiliza dicho algoritmo para cifrarse.

5. Experimentemos ahora con la criptografía asimétrica. Para ello deberá generar un par de claves, ejecutando:

gpg --gen-key

Complete los datos con su nombre y dirección de correo electrónico. Por ejemplo:

Nombre y apellidos: SU NOMBRE

Dirección de correo electrónico: SU-EMAIL@DOMINIO.COM

Frase de contraseña: escriba una frase

(esta clave protege la clave privada y será requerida para firmar o cifrar un documento así que no debe olvidarla)

ATENCIÓN Puede ser que el sistema le solicite generar entropía. ¿Por qué generar entropía o “desorden”? Porque para poder generar una buena clave, se debe poseer suficiente “desorden” para alimentar los algoritmos de números aleatorios (es análogo a agitar el cubo de dados en la generala). Si todo esto demora demasiado, puede cancelar la operación e instalar el siguiente paquete que le ayudará a la generación de entropía: **haveged** . Luego reintente la operación.

Por otro lado, tenga en cuenta que la dirección de correo se utiliza como identificador de la clave, por lo que debe recordarla. El sistema además le solicitará que genere una “phrase”, que es en definitiva una contraseña (más larga) que impedirá que cualquiera pueda ver o utilizar su clave privada.

Las claves de GnuPG se almacenan en el directorio **.gnupg** dentro del directorio del usuario.

El archivo **~/.gnupg/pubring.gpg** (o **pubring.kbx**) contiene la clave pública propia y las de terceros. Las claves secretas se almacenan en el directorio **~/.gnupg/private-keys-v1.d**.

Ahora, procedo a generar un par de claves mediante el comando:

```
gpg --gen-key
```

```
juanma@ubuntu:~/Desktop/TP 8$ gpg --gen-key
gpg (GnuPG) 2.2.19; Copyright (C) 2019 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Note: Use "gpg --full-generate-key" for a full featured key generation dialog.

GnuPG needs to construct a user ID to identify your key.

Real name: Juan Martin Franco
Email address: juanmartin_franco@hotmail.com
You selected this USER-ID:
"Juan Martin Franco <juanmartin_franco@hotmail.com>"
```

6. Este par de claves asimétricas nos va a servir siempre y cuando podamos compartir nuestra clave pública para eso:

```
gpg --export -a SU-EMAIL@DOMINIO.COM > SUNOMBRE.kpub
```

Puede ver el contenido del archivo de clave pública utilizando el comando **cat**.

Ahora, exporto la clave pública a un archivo llamado juanmartin.kpub utilizando el comando:

```
gpg --export -a juanmartin_franco@hotmail.com > juanmartin.kpub
```

```
juanma@ubuntu:~/Desktop/TP 8$ gpg --export -a juanmartin_franco@hotmail.com > juanmartin.kpub
juanma@ubuntu:~/Desktop/TP 8$ ls
descifrado.txt  juanmartin.kpub  pg25525.txt  pg25525.txt.gpg
```

Y visualizo su contenido mediante el comando:

```
cat juanmartin.kpub
```

```
juanma@ubuntu:~/Desktop/TP 8$ cat juanmartin.kpub
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQGNBGGJ+/oBDADWouG8501Qi70RLnkNyLYzGZDlCaMMJsMs260z150y6zKdi1Wt
06es/27sIlBoc3/203qyCuNdKH03Liem9hIML79hmYVbFwew2sbPjiUuwv6lWzaZ
PDX5NZtKtNEJzC7iD2KVXQDn13bPzaWedE7L3LSTPhhw2wRYzQvX71sVPxFnaA3h
QpMKHWq4vFKF5askepQojpyymbvuJ+/bQqUY33DBYdk3BRzr4WdK0sCps/zVomVe
Vv5W2uctVU9ZYRaQd8yn51NB8krC5cr7PNR3AnojxmCEDdXZHKD596XftG3UxVJN
mi7SppFBL04BBMFHxe2NXyO0TF28ZhS0kxwx99eY4LLNvoHGA6fvvudR02uafI22
+fYpwV2U2YcG8AzJk4i8KqR9PLusPFMC3+WglgvH7NX7vy9RQYgX0ueSvdc7tHa4
fHpqrVb60q7h5uW9A202DsVXB9ahXDeYYXW1EYI1uLuRLe6fxMdDgYkI3N1jKrWg
sfZ7ZrAC76Ak/hsAEQEAAbQySnVhbiBNYXJ0aw4GrNjHbmNvIDxqdwFubWFydGlu
X2ZyYW5jb0Bob3RtYwlsLmNvbT6JAdQEEwEKAD4WIQRRC5d/prStQitrqzPORlfw
Y57NRAUCYYn7+gIbAwUJA8JnAAULCQgHAgYVCgkICwIEFgIDAQIeAQIXgAAKCRDO
RlfwY57NRPB8C/9Ms1LjCiHgI8GSij0e4i68aeaxX+okAyBcPjiHFWQlwDRsUMUH
7WdgZu6jjt+TH0VEyIMfZ7QLst2s6HA5LY4a9c+RJDmA5tFRXSjMxFPAbifV/pjT
VKm4ywsP7WAJ/LjmiMfYAJyu77Jjse5RUC0q40ep94kOLfgYvX1vbqoHFaZXX9pK
Z6bCfsxPWL/qqVc58/HSwz/HxOmgNpTTPPf1r3V/6IHmjwHCdC6AKgBwWcx9a9Wz
KZcWcjw80mc6YdEmhNPTh+u6MK7Z84w783drv0FanvNl3Wy+4NUByIEoQdg7TbPf
gkTxRYMnPPf+YpER/mNNVRCpcYoAmCsRX00S6JHf8aaD3u6yIDA34QTnp+upDr2z
Dr0+DxU95jsFl025T+TRCzJ67ZBtNSH94qD7wwqBn5QLUaCLaXtNyejHC6m/SJcN
EUe7QoLwNb+khwxoaPH6q3KYKkAYot2AhEKUCgvI0viDkQo9qiWdxiyCkM0z3J0L
```

Bien hasta aquí hemos llegado con la experiencia de laboratorio con GPG. Hasta el momento ha sido capaz de cifrar y descifrar archivos con clave simétricas y ha generado su propio par de claves. En el trabajo práctico aplicará lo aprendido y experimentará además con la posibilidad de verificar.

PARTE 2 - HTTPS y SSL: TLS en acción.

Más allá de su uso particular y de su aplicación en el uso del correo electrónico, las criptografía asimétrica es de amplia utilidad para brindar servicios de seguridad en la WWW. Como sabe, el protocolo HTTP no cuenta con las herramientas necesarias para brindar ningún servicio de seguridad en sus mensajes. Las peticiones y respuestas del protocolo viajan en texto plano por internet (tal como pudo comprobar en cualquier captura de tráfico web). Por este motivo cualquier intermediario puede ser capaz de leer o alterar las comunicaciones que se producen entre el navegador y un servidor web.

Para lograr dar servicios de seguridad a HTTP se recurre al uso del protocolo TLS y a la criptografía asimétrica, para crear un canal de comunicación seguro (también conocido como Secure Socket Layer o SSL). Así las peticiones y respuestas del protocolo HTTP siguen generándose en texto plano, pero en lugar de encapsularse como “carga” (payload) del protocolo TCP directamente, se encapsula esta información como carga de un paquete TLS, y éste sobre TCP. Por lo tanto es tarea del protocolo TLS que los mensajes HTTP viajen de forma segura hasta su destino. Esta forma de operación, en conjunto, es la que agrega la S al final de HTTP, conformando lo que se conoce como HTTPS.

Todos los servidores web que implementan HTTPS cuentan con un par de claves asimétricas, es decir, con una clave privada y una pública. La clave pública deberá ser enviada al cliente para que este pueda generar un canal seguro de comunicación. De esta manera solo el servidor web con su clave privada será capaz de descifrar los mensajes de establecimiento de este canal que sean enviados por el navegador.

A continuación vamos a seguir, con un alto nivel de abstracción, el escenario en donde un navegador le solicita un recurso web a un servidor utilizando HTTPS.

Lo primero que sucederá es que el browser le solicitará al servidor web iniciar una conexión. El servidor web le contestará con su clave pública. El cliente luego, utilizará la clave pública para enviar, cifrados, los detalles del canal seguro de comunicación que se creará (también conocido como datos de la sesión). Este intercambio de mensajes, de forma resumida, es en definitiva el que corresponde al Handshake de TLS.

```
[Navegador] -> conexión a www.servidor.com      -> [Servidor Web]
[Navegador] <- kpubServer                        <- [Servidor Web]
[Navegador] -> C kpubServer(DATOS_DEL_CANAL)     -> [Servidor Web]
```

¿Ve algún problema en esta configuración? ¡El navegador está recibiendo por un canal no seguro la clave pública del servidor web! ¿Cómo podríamos saber que esa es efectivamente la clave pública del servidor al que nos queremos conectar y que es ESE servidor web quien nos está contestando?

Lo que sucede es que en realidad, el servidor web no le envía solo su clave pública al navegador, sino que le envía algo que se conoce como un certificado SSL o certificado X.509.

Un certificado X.509 contiene (entre otros) los siguientes datos:

- El nombre de dominio para el cual fue hecho el certificado.
- La persona, organización o dispositivo para el cual se hizo el certificado.
- Que autoridad de certificación lo emitió.
- La firma de la autoridad de certificación.
- Los dominios y subdominios para los cuales el certificado es válido.
- La fecha de emisión del certificado.
- La fecha de expiración del certificado.
- La clave pública del servidor.

Como puede observar, no es tan simple como parecía en un principio. Además de la clave pública se necesita de una serie de metadatos que asocian esa clave con una entidad (organismo, empresa, dominio, etc.) y le dan validez a esa clave.

1. Para ver un ejemplo real de certificado X.509, acceda con su navegador web a <https://www.google.com.ar>. Luego si utiliza Firefox, haga click en el ícono del “candadito”, luego en la flecha que dice “Mostrar detalles de la conexión” y finalmente haga click en el botón “Más información”. Allí podrá ver algo de información sobre el certificado y si hace click en “Ver certificado” podrá verlo completo.

Los certificados válidos están firmados digitalmente por un tercero que es de confianza para nuestro navegador web. Existen organizaciones y empresas que brindan servicios de firma de certificados. Estas organizaciones son conocidas como autoridades de certificación o CA (Certification Authority).

Para lograr obtener un certificado, se necesita enviar una solicitud de certificado a una CA. Esta solicitud de certificado (o CSR) contiene la clave pública del servidor junto con algunos metadatos (nombre de solicitante, dominios que valida, fecha de expiración, etc). Esta información debe ser firmada luego por la CA y finalmente todo junto: metadatos, clave pública del servidor, firma e identidad de la CA, es lo que conforma el certificado del servidor web.

Dicho de otra manera y en términos de un sistema criptográfico:

Servidor: $k_{pub}Server, k_{priv}Server$

Autoridad de certificación (CA): $k_{pub}CA, k_{priv}CA$

CSR (solicitud de certificado) = $metadatos + k_{pub}Server$

$Hash(CSR) = H$

$$C_{k_{priv}CA}(H) = H'$$

$$Certificado = metadatos + k_{pub}Server + H'$$

Cuando un navegador web recibe un certificado desde un servidor procede a verificar la validez del mismo. Para hacerlo chequea las fechas presentes en el certificado, los dominios para los cuales fue expedido y, lo más importante, la firma del mismo. El navegador web por lo tanto necesita conocer la clave pública de la CA firmante.

Estamos nuevamente ante el problema de cómo saber fehacientemente cual es la clave pública de alguien. El procedimiento en realidad, es bastante simple (tal vez demasiado). Todos los navegadores web, desde su instalación, tienen una base de datos de certificados con las claves públicas de varias CA. Por lo que si un servidor web envía un certificado que está firmado por alguna de esas CA conocidas, es reconocido por el navegador web (está en esa lista de certificados de autoridades) se puede proceder a verificar la firma utilizando la kPub de la Autoridad de Certificación que firmó el certificado.

Nombre del emisor:

Nombre del emisor	
País	US
Organización	Google Trust Services LLC
Nombre común	GTS CA 1C3

Dominios que valida:

Nombres alternativos del sujeto	
Nombre de la DNS	*.google.com
Nombre de la DNS	*.appengine.google.com
Nombre de la DNS	*.bdn.dev
Nombre de la DNS	*.cloud.google.com
Nombre de la DNS	*.crowdsourcing.google.com
Nombre de la DNS	*.datacompute.google.com
Nombre de la DNS	*.google.ca
Nombre de la DNS	*.google.cl
Nombre de la DNS	*.google.co.in
Nombre de la DNS	*.google.co.jp
Nombre de la DNS	*.google.co.uk
Nombre de la DNS	*.google.com.ar
Nombre de la DNS	*.google.com.au
Nombre de la DNS	*.google.com.br
Nombre de la DNS	*.google.com.co
Nombre de la DNS	*.google.com.mx
Nombre de la DNS	*.google.com.tr
Nombre de la DNS	*.google.com.vn
Nombre de la DNS	*.google.de
Nombre de la DNS	*.google.es

Validez:

Validez	
No antes	Mon, 18 Oct 2021 09:03:24 GMT
No después	Mon, 10 Jan 2022 09:03:23 GMT

Información de la clave pública:

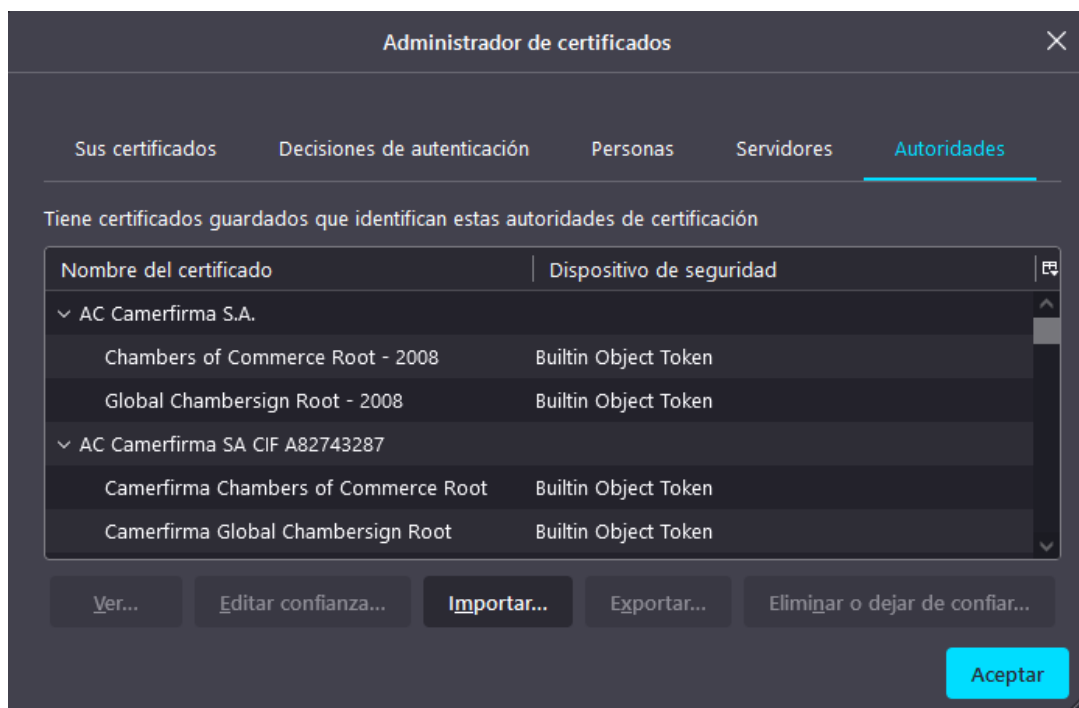
Información de clave pública	
Algoritmo	Elliptic Curve
Tamaño de la clave	256
Curva	P-256
Valor público	04:73:2C:FC:99:16:00:36:F2:F7:C8:85:02:EB:9A:0E:D7:F9:C6:04:BC:D0:CE:28:24:43:1...

2. Para conocer cuales autoridades de certificación conoce su navegador Web, si utiliza Firefox, puede ir a **Preferencias » Privacidad y Seguridad » Certificados » Ver certificados » Autoridades**. Los certificados pueden firmarse “en cadena” por varias **Autoridades de Certificación**. Es decir, la autoridad A firma el certificado de B (una “autoridad intermedia”), y luego B firma el certificado del Servidor. Esta cadena se conoce como **jerarquía de firmas o jerarquía de certificados**.

Ahora, para ver las Autoridades de certificación que conoce mi navegador, me dirijo a:

Preferencias → Privacidad y Seguridad → Certificados → Ver certificados → Autoridades

y obtengo lo siguiente:



3. Acceda a los siguientes sitios: <https://webmail.unlu.edu.ar/>, <https://gitlab.com/help>, <https://www.google.com.ar/>. A través de las herramientas de desarrollador (en Chrome y Firefox) obtenga los certificados y analice la jerarquía de certificados (Certificate Hierarchy). Determine:

- a. ¿Qué entidades emitieron tales certificados? ¿Cuál es el orden de jerarquía? ¿Hay alguna coincidencia en la jerarquía de los certificados de los sitios visitados?

<https://webmail.unlu.edu.ar/>

Las entidades que emitieron los certificados son:

- R3
- Internet Security Research Group

El orden de jerarquía es el siguiente:

ISRG ROOT X1 (Root) → R3 (Intermedio) → *.unlu.edu.ar

<https://gitlab.com/help>

Las entidades que emitieron los certificados son:

- Sectigo RSA Domain Validation Secure Server CA
- USERTrust RSA Certification Authority

El orden de jerarquía es el siguiente:

USERTrust RSA Certification Authority (Root) → Sectigo RSA Domain Validation Secure Server CA (Intermedio) → gitlab.com

<https://www.google.com.ar/>

Las entidades que emitieron los certificados son:

- GTS CA 1C3
- GTS Root R1

El orden de jerarquía es el siguiente:

GTS Root R1 (Root) → GTS CA 1C3 (Intermedio) → *.google.com.ar

- b. Para el certificado de la web de la universidad (*.unlu.edu.ar) detalle: el algoritmo de firma utilizado, el período de validez del certificado, el sujeto (subject), el emisor (issuer) y la copia de la clave pública del servidor.

Certificado de la web de la universidad:

*.unlu.edu.ar

Algoritmo de firma utilizado: RSA

Periodo de validez del certificado: Thu, 09 Sep 2021 13:21:00 - Wed, 08 Dec 2021 13:20:59

Sujeto: *.unlu.edu.ar

Emisor: Let's Encrypt

Copia de la clave pública del servidor:
C7:BB:D9:A8:2D:55:D4:DB:B3:FD:86:44:42:D0:2D:DF:B1:AF:B7:2F:9B:B5:B4:8A:19:0A:A
9:73:30:68:3B:C3:C1:44:BE:28:89:2E:D0:D5:10:D3:DB:FF:C5:14:2A:C4:48:A1:A8:EE:C2:
F9:DE:93:16:D2:09:E0:D7:47:19:BB:DA:F3:5B:EF:85:1F:58:65:D2:7E:FC:A5:22:2E:D4:D2
:42:02:B0:C3:BB:5D:82:75:7C:33:22:A0:A5:BC:52:AC:93:04:DD:42:C2:25:66:56:28:51:C1
:F7:58:23:38:39:F9:86:DE:F9:2B:17:0C:F2:09:77:99:20:C1:BE:51:90:4D:A8:BA:FB:F0:38:
01:B0:3B:5F:59:B8:99:52:79:D2:21:8F:B4:C2:B8:95:20:69:64:7F:21:4D:3C:B7:74:B9:62:3
4:26:FE:A0:B2:51:BE:E3:F4:7F:C0:73:02:56:B0:A0:C9:1C:EE:23:41:FD:28:3F:42:65:69:6
0:07:E9:23:97:67:30:A6:97:F1:FF:73:A3:45:92:E0:F2:93:F7:41:D1:87:43:58:A2:CD:49:F5:
5D:92:D3:B7:69:48:BE:F6:BE:22:33:F1:1F:05:77:69:AE:0F:A2:95:C4:BA:E0:56:F3:7C:25:
08:59:A1:B5:23:84:D1:4C:5F:9A:73:4B:8F

**c. ¿Qué significa la sección “Nombre alternativo del sujeto del certificado”?
¿Para qué puede utilizarse? ¿Qué valores poseen los certificados de UNLu y
de Google?**

La sección Nombre alternativo del sujeto del certificado sirve para detallar todos los dominios a los que aplicará dicho certificado.

Esta sección puede utilizarse para evitar tener que crear un certificado por cada dominio perteneciente registrado.

Los valores de los certificados de UNLu son: *.unlu.edu.ar

Los valores de los certificados de Google son:

- *.google.com
- *.appengine.google.com
- *.bdn.dev
- *.cloud.google.com
- *.crowdsourcing.google.com
- *.datacompute.google.com
- *.google.ca
- *.google.cl

etc. etc.

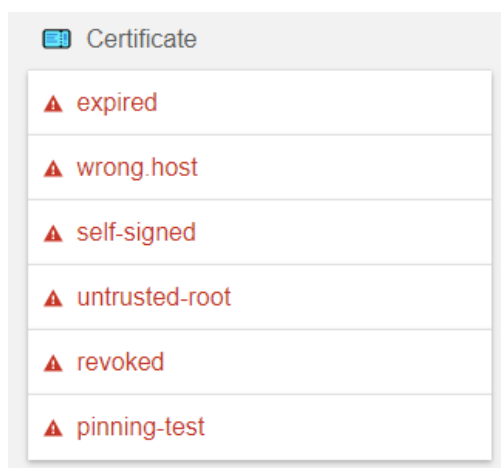
Las razones por las cuales un certificado puede no ser válido son muchas. Entre ellas: certificados vencidos, certificados que no se corresponden con el dominio, utilización de algoritmos de cifrado no recomendados, etc.

- 4. Ingrese al sitio web <https://badssl.com/> donde se recopila una serie de situaciones que pueden suceder en el contexto de HTTPS y los certificados provistos. Examine los diferentes sitios que están enlazados allí y determine los errores que subyacen a las fallas presentadas en el sitio.**

Por lo general conseguir un certificado válido, cuesta dinero. Ya que los servicios de las grandes autoridades de certificación no son gratuitos. Existen dos alternativas a este asunto, uno es utilizar servicios gratuitos como *Lets Encrypt* que emite certificados válidos de corta duración que se pueden renovar; la otra alternativa es firmar uno mismo sus propios certificados. Esto es lo que se conoce como certificado autofirmado.

Una vez que se ingresa al sitio <https://badssl.com/>, dentro de la página se encuentran enlaces a diferentes páginas que cuentan con diversos errores en lo que respecta a certificados (por ejemplo, certificado expirado, certificado que no corresponde al dominio, certificado autofirmado, certificado revocado, etc.).

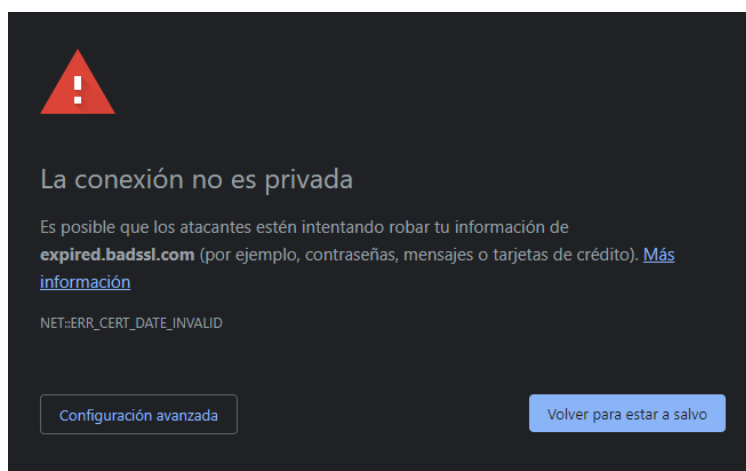
Para cada ejemplo, la página <https://badssl.com> posee una lista como la siguiente:



Por ejemplo, si quiero ver el ejemplo de que error arroja el navegador al acceder a una página que tenga un certificado expirado, accedo al primer enlace, el cual me redirige a la siguiente página:



Y el navegador arroja el siguiente error:



El cual me indica que el certificado efectivamente expiró.

5. Genere un certificado auto-firmado para un sitio web alojado en su propia dirección IP. Configure un servidor web Apache que lo utilice (Ver anexo: Apache y HTTPS).

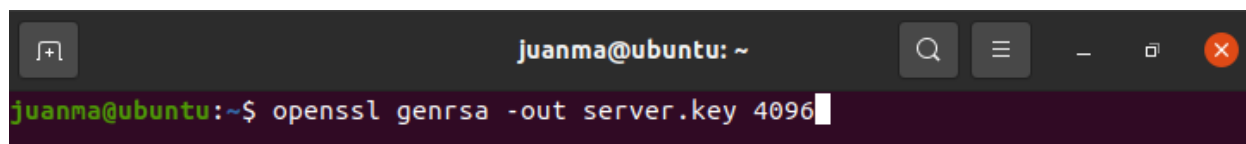
Acceda mediante un navegador a su sitio web utilizando HTTPS. ¿Qué significa el mensaje de error que presenta el navegador?

Como ha podido comprobar, los navegadores web no confían en el certificado autofirmado generado y configurado, ya que no ha sido firmado por una autoridad de certificación reconocida por el mismo.

Sin embargo uno podría, si quisiera, agregar la excepción al navegador y poder navegar de forma segura.

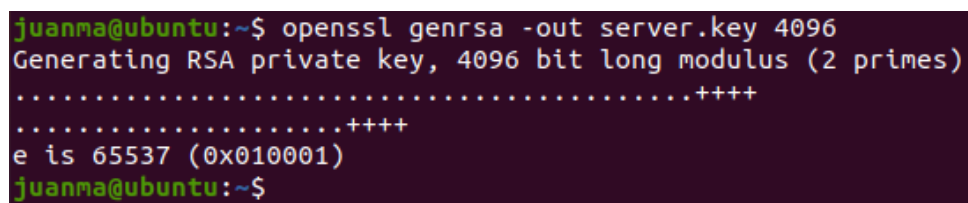
Para comenzar con el ejercicio, utilizo el comando dictado en el anexo relativo a Apache y HTTPS:

```
openssl genrsa -out server.key 4096
```



```
juanma@ubuntu: ~  
juanma@ubuntu:~$ openssl genrsa -out server.key 4096
```

Al ejecutar el comando, obtengo la siguiente salida:

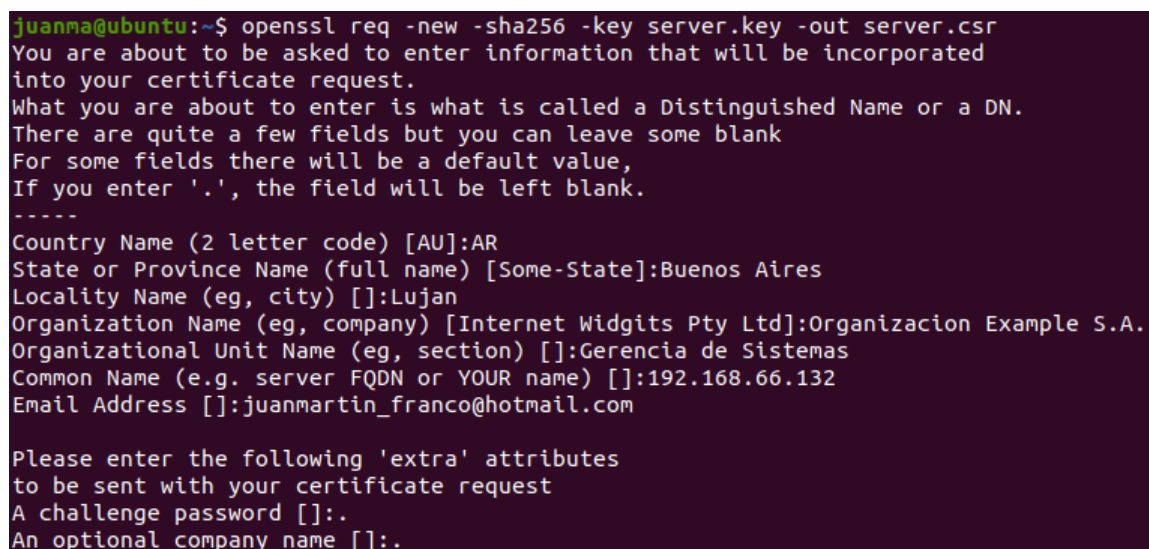


```
juanma@ubuntu:~$ openssl genrsa -out server.key 4096  
Generating RSA private key, 4096 bit long modulus (2 primes)  
.....++++  
.....++++  
e is 65537 (0x010001)  
juanma@ubuntu:~$
```

Ahora, ejecuto el siguiente comando:

```
openssl req -new -sha256 -key server.key -out server.csr
```

Y luego, completo los campos requeridos:



```
juanma@ubuntu:~$ openssl req -new -sha256 -key server.key -out server.csr  
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]:AR  
State or Province Name (full name) [Some-State]:Buenos Aires  
Locality Name (eg, city) []:Lujan  
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Organizacion Example S.A.  
Organizational Unit Name (eg, section) []:Gerencia de Sistemas  
Common Name (e.g. server FQDN or YOUR name) []:192.168.66.132  
Email Address []:juanmartin_franco@hotmail.com  
  
Please enter the following 'extra' attributes  
to be sent with your certificate request  
A challenge password []:.  
An optional company name []:.
```

Ahora, procedo a auto-firmar la petición con la clave privada propia.

Esto lo realizo mediante el uso del comando:

```
openssl x509 -req -days 365 -sha256 -in server.csr -signkey
server.key -out server.crt
```

```
juanma@ubuntu:~$ openssl x509 -req -days 365 -sha256 -in server.csr -signkey server.key -out server.crt
Signature ok
subject=C = AR, ST = Buenos Aires, L = Lujan, O = Organizacion Example S.A., OU = Gerencia de Sistemas, CN
= 192.168.66.132, emailAddress = juanmartin_franco@hotmail.com
Getting Private key
```

Por último, visualizo el contenido del certificado digital, mediante el uso del comando:

```
openssl x509 -text -in server.crt
```

```
juanma@ubuntu:~$ openssl x509 -text -in server.crt
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            26:31:df:48:3f:b8:e6:dc:39:0d:c1:48:f5:de:16:99:b7:96:0a:7d
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = AR, ST = Buenos Aires, L = Lujan, O = Organizacion Example S.A., OU = Gerencia de Sistemas, CN = 192.168.66.132, emailAddr
        ess = juanmartin_franco@hotmail.com
        Validity
            Not Before: Nov 15 04:09:46 2021 GMT
            Not After : Nov 15 04:09:46 2022 GMT
        Subject: C = AR, ST = Buenos Aires, L = Lujan, O = Organizacion Example S.A., OU = Gerencia de Sistemas, CN = 192.168.66.132, emailAddr
        ess = juanmartin_franco@hotmail.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public-Key: (4096 bit)
            Modulus:
                00:c4:dd:0f:8f:04:64:a8:5e:18:75:f7:df:d6:83:
                1f:d8:9f:5c:8d:3f:30:63:82:76:70:e4:55:b5:23:
                18:24:49:93:20:e9:ea:96:99:f3:8b:5c:db:11:42:
```

En dicho fichero, se pueden ver reflejadas la duración del certificado (exactamente 365 días), los datos de quien firma el certificado, el algoritmo de cifrado, y la clave, entre otros datos.

Ahora, hace falta configurar un servidor web para que instale dicho certificado.

Primero, instalo el servidor web Apache 2:

```
apt-get install apache2
```

```
juanma@ubuntu:~$ sudo apt-get install apache2
[sudo] password for juanma:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  apache2-bin apache2-data apache2-utils libapr1 libaprutil1 libaprutil1-dbd-sqlite3 libaprutil1-ldap
Suggested packages:
  apache2-doc apache2-suexec-pristine | apache2-suexec-custom
The following NEW packages will be installed:
  apache2 apache2-bin apache2-data apache2-utils libapr1 libaprutil1 libaprutil1-dbd-sqlite3 libaprutil1-ldap
0 upgraded, 8 newly installed, 0 to remove and 53 not upgraded.
Need to get 1,715 kB of archives.
After this operation, 7,504 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

Ahora, activo los módulos rewrite y ssl, y el sitio default-ssl mediante el uso de los siguientes comandos:

```
a2enmod rewrite
```

```
root@ubuntu:/home/juanma# a2enmod rewrite
Enabling module rewrite.
```

```
a2enmod ssl
```

```
root@ubuntu:/home/juanma# a2enmod ssl
Considering dependency setenvif for ssl:
Module setenvif already enabled
Considering dependency mime for ssl:
Module mime already enabled
Considering dependency socache_shmcb for ssl:
Enabling module socache_shmcb.
Enabling module ssl.
See /usr/share/doc/apache2/README.Debian.gz on how to configure SSL and create self-signed certificates.
```

```
a2ensite default-ssl
```

```
root@ubuntu:/home/juanma# a2ensite default-ssl
Enabling site default-ssl.
```

Ahora, sigo los demás pasos para crear la ubicación donde estarán almacenados los certificados y luego le asigno los permisos correspondientes:

Primero, creo el directorio llamado certificados y me posiciono en el mismo:

```
root@ubuntu:/home/juanma# mkdir /etc/apache2/certificados
root@ubuntu:/home/juanma# cd /etc/apache2/certificados/
```

Luego, muevo los archivos server.crt y server.key a dicho directorio:

```
root@ubuntu:/etc/apache2/certificados# mv /home/juanma/Desktop/server.crt .
root@ubuntu:/etc/apache2/certificados# mv /home/juanma/Desktop/server.key .
```

Por último, modifico los permisos necesarios en dichos archivos.

```
root@ubuntu:/etc/apache2/certificados# chown root.root server.crt server.key
root@ubuntu:/etc/apache2/certificados# chmod 444 server.crt
root@ubuntu:/etc/apache2/certificados# chmod 400 server.key
root@ubuntu:/etc/apache2/certificados#
```

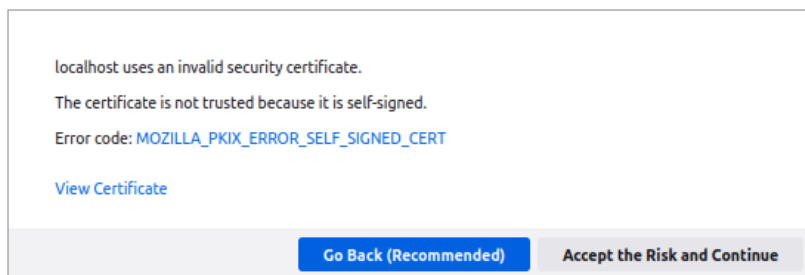
Una vez hecho esto, paso a configurar el archivo ubicado en /etc/apache2/sites-enabled/ llamado default-ssl.conf, utilizando el comando:

```
nano /etc/apache2/sites-enabled/default-ssl.conf
```

```
# A self-signed (snakeoil) certificate can be created by installing
# the ssl-cert package. See
# /usr/share/doc/apache2/README.Debian.gz for more info.
# If both key and certificate are stored in the same file, only the
# SSLCertificateFile directive is needed.
SSLCertificateFile      /etc/apache2/certificados/server.crt
SSLCertificateKeyFile   /etc/apache2/certificados/server.key
```

Por último, guardo los cambios y salgo del editor de texto.

Al ingresar a <https://localhost:443>, obtengo lo siguiente:

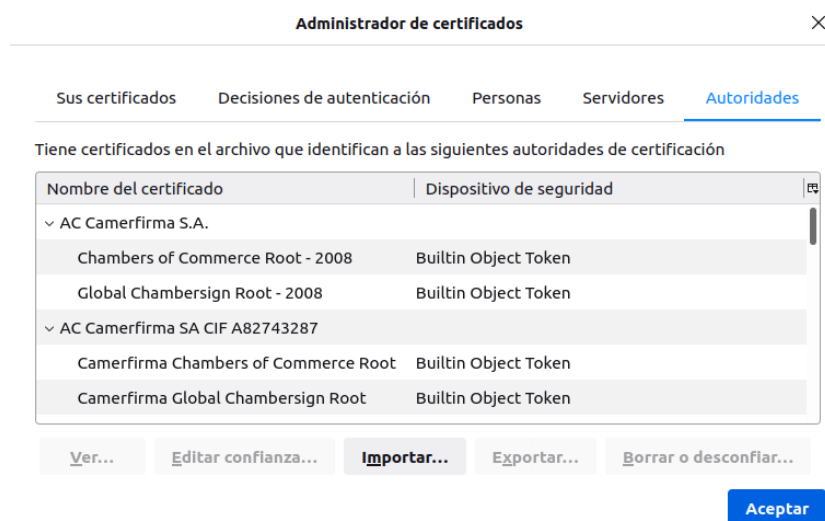


Como se aprecia en el error, se indica que el certificado es autofirmado, lo cual implica un riesgo enorme.

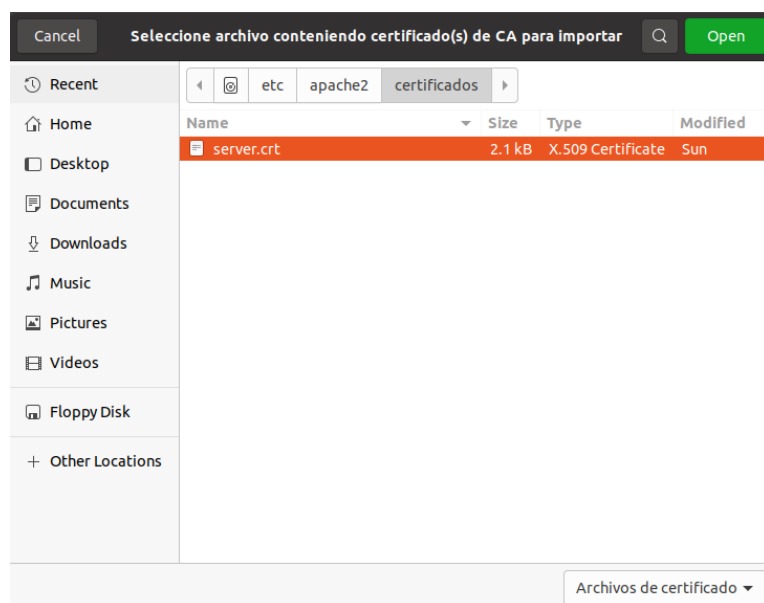
6. Agregue la excepción de seguridad para el servidor configurado, y compruebe que puede navegar correctamente. Luego realice una captura al momento de realizar una consulta al servidor web. Guarde esta captura bajo el nombre cap-ejer-ssl.pcap.

Por último, para agregar el certificado autofirmado a la lista de certificados de confianza, me dirijo a

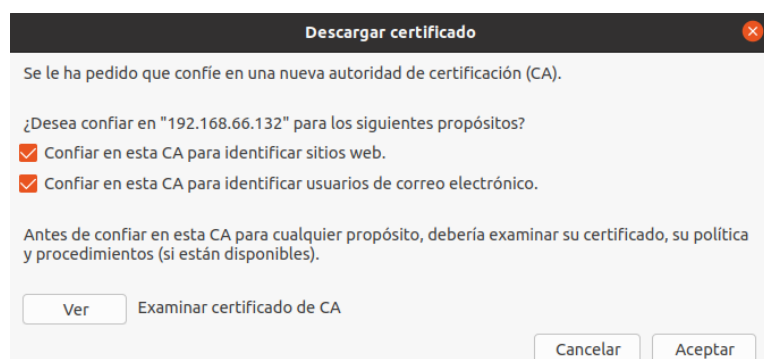
Preferencias → Privacidad y Seguridad → Certificados → Ver certificados → Autoridades



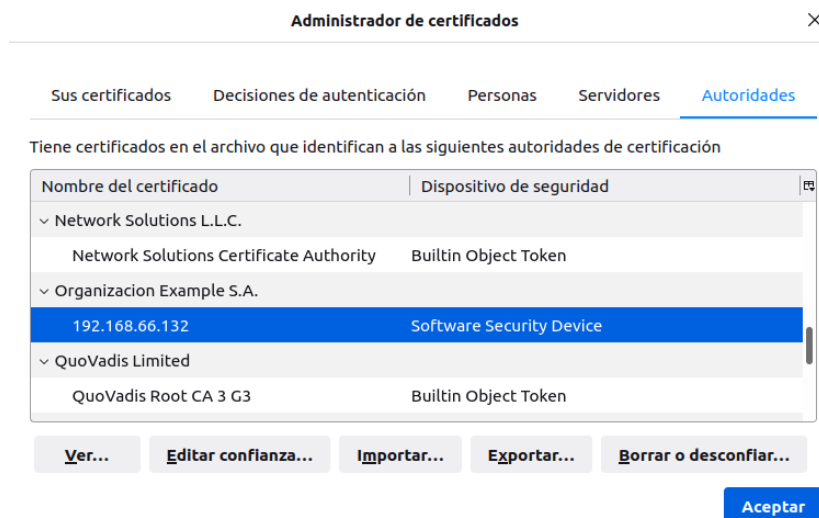
Luego, elijo la opción "Importar..." y localizo el certificado que previamente ubiqué en /etc/apache2/certificados, luego de esto, lo importo.



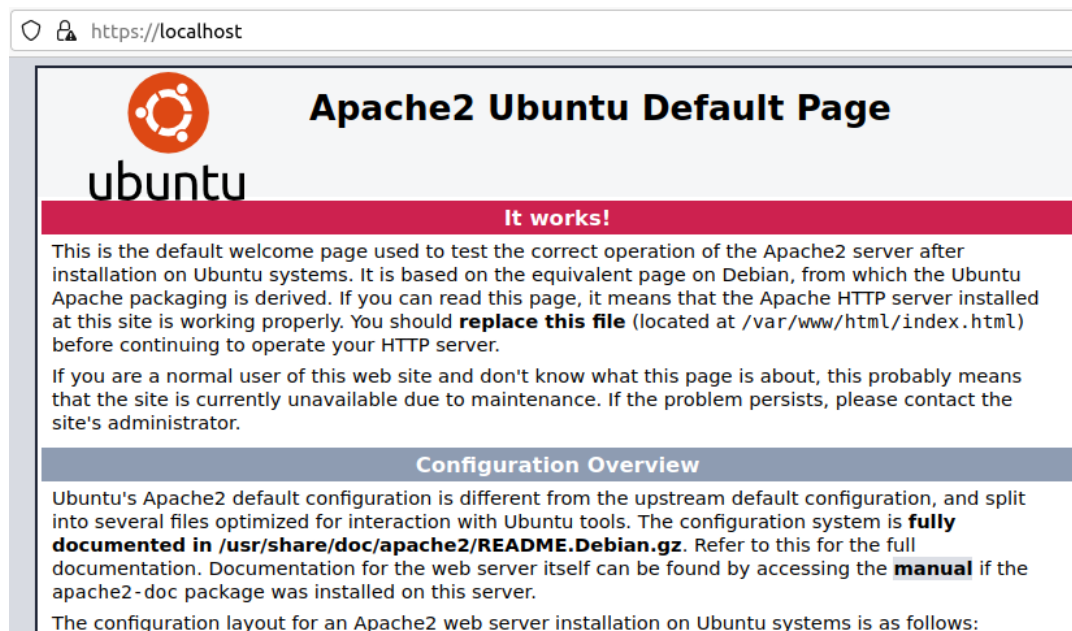
Como es de esperar, una vez seleccionado el certificado, se me solicita que confirme que voy a confiar en la quien firma dicho certificado.



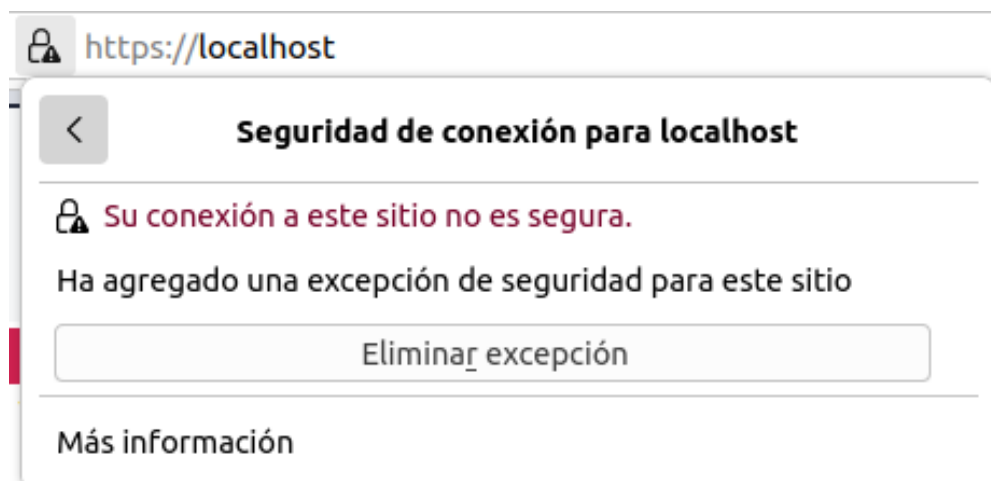
Una vez aceptado, ya se puede visualizar el certificado dentro del administrador de certificados:



Por último, vuelvo a acceder y verifico que el certificado figura como válido:



Se puede visualizar que se ha agregado correctamente la excepción de seguridad:



PARTE 3 - SSH: acceso remoto seguro.

El Secure Shell Protocol, más conocido como SSH, es un esquema de arquitectura cliente/servidor que crea un canal seguro de comunicación entre dos dispositivos sobre una red insegura. Se vale de técnicas de autenticación y cifrado mediante claves tanto simétricas como asimétricas.

El caso de uso más común es cuando un usuario se quiere conectar desde un host a la terminal de un servidor o router en un lugar distante. Para ello inicia una conexión con un cliente SSH que se comunica con el servidor que tiene un servicio de SSH activo. El servidor le solicitará la contraseña para el usuario que se desea comunicar.

Además de la funcionalidad de ejecución de comandos en forma remota (inicio de sesión y login remoto), el protocolo SSH sienta las bases para otros protocolos que operan sobre él, sin requerir servicios ni puertos adicionales en escucha en el sistema.

Sobre la infraestructura que proporciona el protocolo ssh se pueden utilizar diferentes utilidades:

- Acceso a shell (terminal) remota con el comando ssh.
- Copia de archivos desde/hacia host remotos con el comando scp.
- Servicio ftp con el comando sftp.
- Servicio de file system remoto con sshfs.
- Creación de túneles para acceder a puertos en máquinas remotas, con el comando ssh.

Como ha sido mencionado, este protocolo funciona con arquitectura cliente/servidor. Por lo general, todas las distribuciones de linux actuales tienen el cliente ssh instalado. Con este cliente será posible conectarse de forma remota a cualquier servidor, router o computadora que tenga un servidor ssh en escucha.

Elija dos equipos en los que tenga dominio de administración y que se encuentren conectados por una red (puede utilizar máquinas virtuales). El equipo donde usaremos el comando ssh lo llamaremos “local” y al equipo al cual nos conectaremos lo llamaremos “remoto”.

1. El puerto bien conocido de SSH es el TCP 22. Verifique que el equipo remoto tenga un servicio ssh corriendo utilizando el comando **netstat -plnt** (paquete **net-tools**). Este comando le mostrara los puertos y el estado de los mismos en la máquina en que lo ejecute. Por cierto, ¿cómo podría saber si otro equipo que no sea el suyo tiene un servidor ssh corriendo?

Si no encuentra ningún puerto en estado LISTEN en el puerto 22, significa que probablemente no tenga instalado el servidor ssh. Si ese es el caso, instale en el host remoto el paquete **openssh-server**. Verifique el estado del servicio con **sudo systemctl status ssh**.

Primero, procedo a instalar el paquete net-tools mediante el comando:

```
sudo apt install net-tools
```


Luego, ejecuto el comando

```
netstat -plunt
```

```
juanma@ubuntu:~$ netstat -plunt
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 192.168.122.1:53        0.0.0.0:*                LISTEN      -
tcp        0      0 127.0.0.53:53           0.0.0.0:*                LISTEN      -
tcp        0      0 127.0.0.1:631           0.0.0.0:*                LISTEN      -
tcp6       0      0 :::80                   :::*                    LISTEN      -
tcp6       0      0 :::1:631                 :::*                    LISTEN      -
tcp6       0      0 :::443                   :::*                    LISTEN      -
udp        0      0 0.0.0.0:631             0.0.0.0:*                -          -
udp        0      0 192.168.122.1:53        0.0.0.0:*                -          -
udp        0      0 127.0.0.53:53           0.0.0.0:*                -          -
udp        0      0 0.0.0.0:67              0.0.0.0:*                -          -
udp        0      0 0.0.0.0:47287           0.0.0.0:*                -          -
udp        0      0 0.0.0.0:5353            0.0.0.0:*                -          -
udp6       0      0 :::52202                 :::*                    -          -
udp6       0      0 :::5353                  :::*                    -          -
```

Como se puede apreciar, no hay ningún servicio corriendo en el puerto 22, por lo que no tengo ningún servidor SSH instalado y corriendo.

Ahora, procedo a instalar uno utilizando el comando:

```
apt-get install openssh-server
```

```
juanma@ubuntu:~$ sudo apt-get install openssh-server
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  ncurses-term openssh-sftp-server ssh-import-id
Suggested packages:
  molly-guard monkeysphere ssh-askpass
The following NEW packages will be installed:
  ncurses-term openssh-server openssh-sftp-server ssh-import-id
0 upgraded, 4 newly installed, 0 to remove and 53 not upgraded.
Need to get 688 kB of archives.
After this operation, 6,010 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://us.archive.ubuntu.com/ubuntu focal/main amd64 ncurses-term all 6.2-0ubuntu2 [249 kB]
Get:2 http://us.archive.ubuntu.com/ubuntu focal-updates/main amd64 openssh-sftp-server amd64 1:8.2p1-4ubuntu0.3 [51.5 kB]
```

Una vez instalado, verifico si está corriendo el servidor SSH mediante el comando:

```
systemctl status ssh
```

```
juanma@ubuntu:~$ systemctl status ssh
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2021-11-14 23:02:22 PST; 2min 6s ago
     Docs: man:sshd(8)
           man:sshd_config(5)
   Main PID: 4239 (sshd)
    Tasks: 1 (limit: 2256)
   Memory: 1.0M
   CGroup: /system.slice/ssh.service
           └─4239 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups

Nov 14 23:02:22 ubuntu systemd[1]: Starting OpenBSD Secure Shell server...
Nov 14 23:02:22 ubuntu sshd[4239]: Server listening on 0.0.0.0 port 22.
Nov 14 23:02:22 ubuntu sshd[4239]: Server listening on :: port 22.
Nov 14 23:02:22 ubuntu systemd[1]: Started OpenBSD Secure Shell server.
```

Como se aprecia en la imagen, el servidor está corriendo.

Para saber si otro equipo tiene un servidor ssh corriendo, podría utilizar la herramienta nmap, la cual es de gran ayuda cuando se necesita ver que puertos tiene abiertos un host. En este caso, puedo verificar si alguna máquina tiene el puerto 22 abierto, lo cual es muy probable que signifique que hay un servidor ssh corriendo.

2. **Vamos ahora a conectarnos desde el equipo “local” hasta el equipo “remoto”. Para ello escribiremos desde el equipo local ssh USER@IP-REMOTO. USER en este caso corresponde a un usuario válido del equipo remoto (el cual usted tiene su contraseña). Cuando se intente conectar por primera vez aparecerá el siguiente mensaje en consola:**

```
The authenticity of host 'IP-REMOTO(IP-REMOTO)' can't be established.  
ECDSA key fingerprint is SHA256:nCnddHUI5D6a1/DVQSJhQ5BqyURE64kx0cJ1QyIlKl0Y.  
Are you sure you want to continue connecting (yes/no)?
```

El servidor remoto, tiene su propio par de claves para utilizar en criptografía asimétrica que utiliza luego para generar las sesiones SSH. El Fingerprint, es un número que fue calculado con el algoritmo ECDSA a partir de la clave pública del servidor. De esta manera si nosotros tenemos otro mecanismo para verificar el fingerprint podremos saber si este es o no el host al que nos queremos conectar.

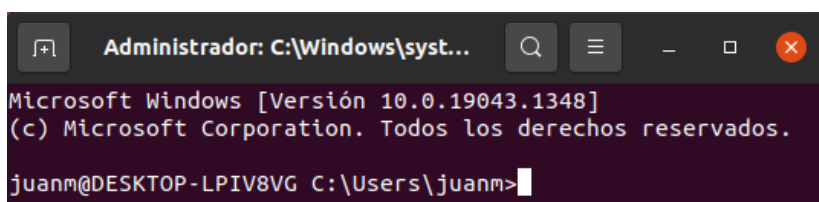
Una vez que decimos que queremos continuar, este fingerprint es almacenado en el archivo `~/.ssh/known_hosts`, por lo que la verificación para los siguientes casos será automática revisando el archivo.

Una vez instalado el servidor SSH en ambos extremos, utilizo el siguiente comando para iniciar una conexión SSH entre “local” (máquina virtual) y remoto (host físico):

```
ssh juanm@192.168.0.197
```

```
juanma@ubuntu:~$ ssh juanm@192.168.0.197  
The authenticity of host '192.168.0.197 (192.168.0.197)' can't be established.  
ECDSA key fingerprint is SHA256:DkEb6Xp58NoPaZEftwDbqoUUZrpEL72KbKa9sJ7I4Pg.  
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

Una vez que acepto continuar, escribo la contraseña de mi equipo host y obtengo acceso a la terminal:



```
Administrador: C:\Windows\syst...  
Microsoft Windows [Versión 10.0.19043.1348]  
(c) Microsoft Corporation. Todos los derechos reservados.  
juanm@DESKTOP-LPIV8VG C:\Users\juanm>
```

3. **Una vez que nos pudimos conectar, tenemos acceso a una terminal. De esta manera vamos a verificar con `netstat -plunt` que el equipo remoto tiene una conexión establecida desde el equipo local (con puerto efímero) al equipo remoto (con puerto 22).**

Ahora, verifico la conexión establecida desde el equipo local hacia el equipo remoto mediante el uso del comando:

```
netstat -pnt
```

```
juanma@ubuntu:~$ netstat -pnt
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 192.168.66.132:47868    192.168.0.197:22       ESTABLISHED 3890/ssh
udp        0      0 192.168.66.132:68      192.168.66.254:67      ESTABLISHED -
```

4. Para crear la sesión segura de comunicación se ha utilizado un algoritmo que permite establecer una contraseña simétrica para cifrar el contenido de toda la conexión. Este algoritmo se conoce con el nombre de “Diffie-Hellman Key Exchange Algorithm” o simplemente Diffie-Hellman. Este algoritmo, de forma somera, funciona de la siguiente manera:

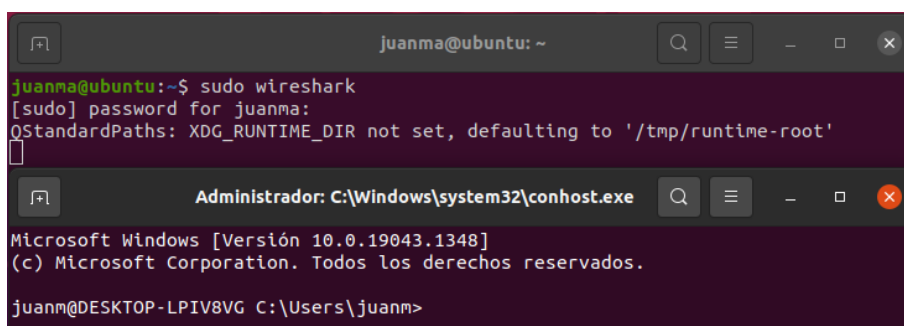
- Tanto el cliente como el servidor acuerdan un número primo muy grande. Este valor también se conoce como valor semilla.
- Luego, las dos partes acuerdan utilizar un mecanismo de cifrado común para generar otro conjunto de valores a partir de los valores semilla de una manera específica. Estos algoritmos, también conocidos como generadores de cifrado, realizan grandes operaciones a partir de la semilla. Un ejemplo de tal tipo de algoritmos es AES (Advanced Encryption Standard).
- Ambas partes generan independientemente otro número primo. Esto se usa como una clave privada secreta para la interacción.
- Esta clave privada recién generada, con el número compartido y el algoritmo de cifrado (por ejemplo, AES), se usa para calcular una clave pública que se distribuye a la otra computadora.
- Luego, las ambos interlocutores usan su clave privada personal, la clave pública compartida de la otra máquina y el número primo original para crear una clave compartida final. Esta clave es calculada independientemente por ambas computadoras, pero creará la misma clave de cifrado en ambos lados.
- Ahora que ambas partes tienen una clave compartida (que ha sido generada en ambos extremos), pueden cifrar simétricamente toda la sesión SSH.

Escriba exit en la terminal de la sesión SSH. Inicie una captura y vuelva a iniciar la conexión SSH.

Analice la captura y asistido por Wireshark observe que partes puede ver del intercambio ¿Algún dato de usuario viaja por la red sin cifrar? Guarde la captura con nombre **conexion-ssh.pncap**.

Para cerrar la conexión, basta con escribir exit en la terminal donde se inició.

Ahora, realizo una captura con Wireshark y luego vuelvo a iniciar la conexión:



En la captura de Wireshark se pueden visualizar: el establecimiento de la conexión (el Three-way Handshake, ya que SSH opera con TCP en la capa de transporte), los mensajes de anuncio de protocolo, los mensajes de intercambio de claves y por último los paquetes encriptados.

Establecimiento de la conexión:

3	8.838104784	192.168.66.132	192.168.0.197	TCP	74	47898 → 22 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 T...
4	8.838868728	192.168.0.197	192.168.66.132	TCP	60	22 → 47898 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
5	8.838939758	192.168.66.132	192.168.0.197	TCP	54	47898 → 22 [ACK] Seq=1 Ack=1 Win=64240 Len=0

Mensajes de anuncio de protocolo:

6	8.839351377	192.168.66.132	192.168.0.197	SSHv2	95	Client: Protocol (SSH-2.0-OpenSSH 8.2p1 Ubuntu-4ubuntu0.3)
8	8.879827804	192.168.0.197	192.168.66.132	SSHv2	87	Server: Protocol (SSH-2.0-OpenSSH_for_Windows_8.1)

Mensajes de intercambio de claves:

10	8.880731917	192.168.66.132	192.168.0.197	SSHv2	1566	Client: Key Exchange Init
13	8.910558544	192.168.0.197	192.168.66.132	SSHv2	1134	Server: Key Exchange Init
15	8.913003595	192.168.66.132	192.168.0.197	SSHv2	102	Client: Diffie-Hellman Key Exchange Init
17	8.919296822	192.168.0.197	192.168.66.132	SSHv2	506	Server: Diffie-Hellman Key Exchange Reply, New Keys, Encrypte...
19	8.922069279	192.168.66.132	192.168.0.197	SSHv2	70	Client: New Keys

Mensajes encriptados:

21	8.923352284	192.168.66.132	192.168.0.197	SSHv2	98	Client: Encrypted packet (len=44)
23	8.923677138	192.168.0.197	192.168.66.132	SSHv2	98	Server: Encrypted packet (len=44)
24	8.923824106	192.168.66.132	192.168.0.197	SSHv2	122	Client: Encrypted packet (len=68)
26	8.935499366	192.168.0.197	192.168.66.132	SSHv2	130	Server: Encrypted packet (len=76)
27	8.935825286	192.168.66.132	192.168.0.197	SSHv2	146	Client: Encrypted packet (len=92)
29	8.947507880	192.168.0.197	192.168.66.132	SSHv2	130	Server: Encrypted packet (len=76)
32	13.175515771	192.168.66.132	192.168.0.197	SSHv2	202	Client: Encrypted packet (len=148)
34	13.248351084	192.168.0.197	192.168.66.132	SSHv2	82	Server: Encrypted packet (len=28)
36	13.248944406	192.168.66.132	192.168.0.197	SSHv2	166	Client: Encrypted packet (len=112)
38	13.283444791	192.168.0.197	192.168.66.132	SSHv2	726	Server: Encrypted packet (len=672)
40	13.283807042	192.168.66.132	192.168.0.197	SSHv2	514	Client: Encrypted packet (len=460)
42	13.287700857	192.168.0.197	192.168.66.132	SSHv2	162	Server: Encrypted packet (len=108)
44	13.310854883	192.168.0.197	192.168.66.132	SSHv2	186	Server: Encrypted packet (len=132)

Los paquetes que viajan sin cifrar son los mensajes de anuncio de protocolo.

- Otra utilidad que aprovecha las conexiones seguras SSH es el comando “scp”. Este comando le permite copiar, de forma segura, archivos entre diferentes equipos. Inicie una captura y pruebe enviar un archivo desde una terminal en el host local hacia el host remoto. Para ello ejecute el siguiente comando: **scp rutaOrigen USER@REMOTO:/rutaDestino**. El cliente scp, realizará una conexión SSH y luego enviará los datos del archivo de usuario utilizando esta sesión. Detenga la captura y compárela con la captura **conexion-ssh.pncap**.

Para comenzar con el ejercicio, procedo a iniciar una captura en Wireshark.

Luego, utilizo el siguiente comando para enviar un archivo desde el host local hacia el host remoto:

```
scp /home/juanma/Desktop/pruebaSSH juanm@192.168.0.197:/users/juanm/
```

```

juanma@ubuntu:~$ scp /home/juanma/Desktop/pruebaSSH juanm@192.168.0.197:/users/juanm/
juanm@192.168.0.197's password:
pruebaSSH                               100% 18   12.7KB/s   00:00
juanma@ubuntu:~$

```

En este caso, el archivo enviado es uno creado con el editor de texto Nano, el cual contiene un mensaje que dice “Mensaje de Prueba”.

Por último, verifico que llegó correctamente en la ruta indicada:

Postman	12/8/2021 19:32	Carpeta de archivos	
Videos	2/11/2021 09:02	Carpeta de archivos	
Vínculos	12/8/2021 16:22	Carpeta de archivos	
WebstormProjects	11/10/2021 17:30	Carpeta de archivos	
.bash_history	4/10/2021 01:41	Archivo BASH_HIS...	2 KB
.gitconfig	27/8/2021 15:39	Archivo GITCONFIG	1 KB
.packettracer	4/10/2021 13:42	Archivo PACKETT...	1 KB
NTUSER.DAT	16/11/2021 05:31	Archivo DAT	6.400 KB
pruebaSSH	16/11/2021 19:45	Archivo	1 KB

6. SSH permite además aprovechar las sesiones seguras de comunicación para transportar datos de otras aplicaciones. Esto es lo que se conoce comúnmente con el nombre de “túnel SSH”. Existen muchas formas y alternativas de conexión, incluso uno puede realizar un túnel a través de más de un servidor SSH. Los túneles son una herramienta muy versátil que permite transportar de forma segura protocolos de aplicación así también como eludir ciertas reglas de firewall para poder comunicarnos con servicios no permitidos. Para entender el funcionamiento de esta herramienta seguiremos el siguiente ejemplo, con un túnel de tipo local (-L).

```
ssh -L 8000:IP-REMOTO:80 USUARIO@IP-REMOTO
```

El resultado de este comando es que cuando nosotros iniciemos un navegador web en el host local y coloquemos allí la dirección: 127.0.0.1:8000 obtendremos como respuesta lo que sea que se esté sirviendo en el puerto 80 de la máquina remota.

Lo que sucede aquí es que el cliente SSH inicia una conexión segura con el servidor remoto. El cliente local ssh, además, pondrá en escucha un servicio en el puerto local 8000 para su ip 127.0.0.1.

Todo tráfico local que esté destinado al puerto 8000 será encapsulado en paquetes SSH y enviados al equipo remoto. Luego en el equipo remoto se tomarán esos paquetes y serán reenviados al puerto 80 de la máquina remota. La respuesta luego realizará el camino inverso.

Como puede observar toda la conexión se realiza al puerto 22 del equipo remoto y ningún intermediario (y esto incluye a los posibles firewalls) podrá ver que en realidad se está realizando una consulta HTTP al servidor remoto. Ejecute el mencionado comando y verifique que el comportamiento sea el esperado.

Para realizar un Túnel SSH, utilizo el comando:

```
ssh -L 8000:IP-REMOTO:80 USUARIO@IP-REMOTO
```

En este caso:

```
ssh -L 8000:192.168.0.197:80 juanm@192.168.0.197
```

```
juanma@ubuntu:~$ ssh -L 8000:192.168.0.197:80 juanm@192.168.0.197
juanm@192.168.0.197's password:
```

Luego, ingreso la contraseña y accedo a la terminal ssh.

Trabajo Práctico

PARTE 1

1. Determine qué servicios de seguridad se provee en cada uno de los siguientes escenarios. Para ello confeccione una tabla donde las columnas sean: integridad, confidencialidad, autenticidad, no repudio.
 - a. Alicia cifra un mensaje con la clave privada de Beto y envía el mensaje cifrado a Beto.
 - b. Beto genera un mensaje, obtiene un resumen criptográfico del mismo, cifra el resumen con su clave privada y publica el mensaje y el resumen cifrado en Internet.
 - c. Alicia cifra un mensaje con su clave privada y luego con la clave pública de Beto, y lo envía a Beto.
 - d. Alicia cifra un mensaje con la clave pública de Beto y luego con su clave privada, y lo envía a Beto.

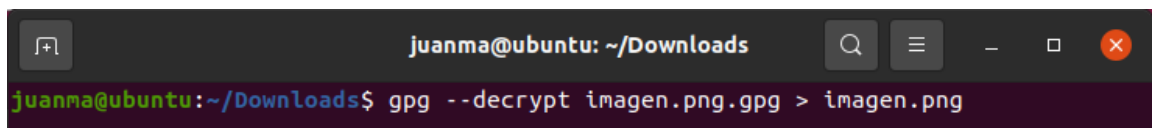
	Integridad	Confidencialidad	Autenticidad	No repudio
Escenario A	No se puede comprobar	Provee	Provee	Provee
Escenario B	No provee	No provee	No provee	No provee
Escenario C	Provee	Provee	Provee	Provee
Escenario D	No Provee	Provee	Provee	Provee

2. Recibirá en su correo electrónico los siguientes archivos:

- a. “imagen.jpg.gpg”: el cual está cifrado simétricamente. La clave de este archivo es: “quieromastps”. Deberá descifrar el archivo y colocar su respuesta a la pregunta que aparece en la imagen en su respuesta a este punto.

Para descifrar el primer archivo, primero lo descargo y luego utilizo el siguiente comando:

```
gpg --decrypt imagen.png.gpg > imagen.png
```



Una vez ejecutado el comando, ingreso la contraseña provista por el enunciado. En este caso, “quieromastps”.

Passphrase:

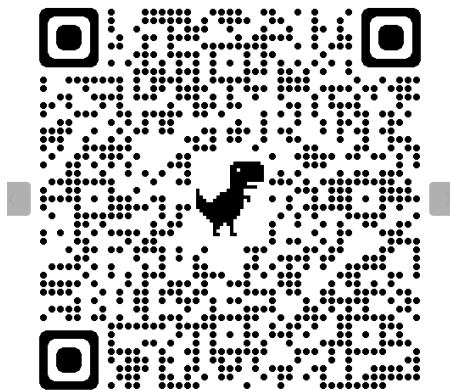
Enter passphrase

quieromastpsl

☐ Save in password manager

Cancel OK

Obteniendo el siguiente resultado:



El cual nos dirige a la siguiente página:

 **SEGURILATAM**

NOTICIA

Códigos QR: qué son, riesgos asociados y consejos de seguridad

Los códigos QR se incorporaron al mundo tecnológico en 2009 y se utilizan para numerosos servicios, algo que es aprovechado por los ciberdelincuentes para llevar a cabo algunas de sus campañas. Por dicho motivo, es importante conocer los riesgos asociados y poner en práctica unos consejos básicos de seguridad.



El código de respuesta rápida (QR, por sus siglas en inglés) es una evolución del código de barras.

17/10/2021 | Por Redacción.

 Los **códigos QR** llevan entre nosotros desde hace muchos años. Concretamente, se incorporaron al mundo tecnológico en 2009 y se han convertido en un **recurso muy empleado** en nuestro día a día. Actualmente, los encontramos en las **cartas de los**

Cuyo link es:

https://www.segurilatam.com/actualidad/codigos-qr-que-son-riesgos-asociados-y-consejos-de-seguridad_20211017.html

- b. “kpubAYGR”: el cual es la clave pública de AYGR. Importe la misma con **gpg --import ARCHIVO.DE.CLAVE** . La necesitará luego.

Ahora, procedo a importar la clave pública de AyGR.

Para realizar esto, utilizo el comando:

```
gpg --import aygr.kpub
```

```
juanma@ubuntu:~/Downloads$ gpg --import aygr.kpub
gpg: key B20F1F771E911277: public key "aygr-pilar <aygr@unlu.edu.ar>" imported
gpg: Total number processed: 1
gpg:             imported: 1
```

3. Deberá exportar su clave pública y enviarla al equipo docente, tal como ha aprendido a hacer en la experiencia de laboratorio. Se le solicita especificar su nombre en el archivo. Puede enviar esta clave junto con la resolución del práctico.

Para exportar la clave pública, utilizo el comando:

```
gpg --export -a juanmartin_franco@hotmail.com > juanmartinfranco.kpub
```

```
juanma@ubuntu:~/Desktop/TP 8$ gpg --export -a juanmartin_franco@hotmail.com > juanmartinfranco.kpub
```

[Adjunto archivo juanmartinfranco.kpub]

4. Descargue una imagen en formato jpg de un “meme” relacionado con la materia (o si está apurado de cualquier otra imagen que usted desee). A esa imagen que llamaremos “meme.jpg” deberá cifrarla asimétricamente para enviarla al equipo docente junto a la resolución del práctico. Para ello `gpg --encrypt --recipient aygr@unlu.edu.ar ARCHIVO.EXT`.
¿Quién podrá ver el contenido de meme.jpg.gpg”? ¿Con que claves cree que fue cifrado?

Una vez descargada la imagen, procedo a cifrarla asimétricamente.

Para realizar esto, utilizo el comando:

```
gpg --encrypt --recipient aygr@unlu.edu.ar meme.jpg
```

```
juanma@ubuntu:~/Desktop/TP 8$ gpg --encrypt --recipient aygr@unlu.edu.ar meme.jpg
gpg: 079D491BE4A8522F: There is no assurance this key belongs to the named user

sub rsa3072/079D491BE4A8522F 2020-06-02 aygr-pilar <aygr@unlu.edu.ar>
Primary key fingerprint: 0C0F A2DB 0F28 6FC6 6A43 B703 B20F 1F77 1E91 1277
Subkey fingerprint: 1194 ACE9 99C1 E723 320D 5A53 079D 491B E4A8 522F

It is NOT certain that the key belongs to the person named
in the user ID. If you *really* know what you are doing,
you may answer the next question with yes.

Use this key anyway? (y/N) y
```

Podrán ver el archivo cifrado aquellos que posean la clave privada de AyGR (ya que el mensaje fue cifrado con la clave pública de AyGR).

5. Descargue y valide la autenticidad de un mensaje de correo. Para ello:

- a. Descargue el mensaje de correo indicado en el enlace siguiente: **w3m -dump <https://lists.debian.org/debian-security-announce/2017/msg00259.html> > mensaje.txt**. Si no tiene disponible el comando w3m deberá instalar el paquete del mismo nombre.

Primero, procedo a instalar el paquete w3m, mediante el uso del comando:

```
sudo apt-get install w3m
```

```
juanma@ubuntu:~$ sudo apt-get install w3m
[sudo] password for juanma:
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  cmigemo dict dict-wn dictd libsixel-bin mpv w3m-el w3m-img xsel
The following NEW packages will be installed:
  w3m
0 upgraded, 1 newly installed, 0 to remove and 55 not upgraded.
Need to get 935 kB of archives.
After this operation, 2,589 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu focal/main amd64 w3m amd64 0.5.3-37
35 kB]
Fetched 935 kB in 3s (292 kB/s)
```

Ahora, ejecuto el comando del enunciado:

```
w3m -dump https://lists.debian.org/debian-security-
announce/2017/msg00259.html > mensaje.txt
```

```
juanma@ubuntu:~/Desktop/TP 8$ w3m -dump https://lists.debian.org/debian-security
-announce/2017/msg00259.html > mensaje.txt
```

- b. Abra el archivo con un editor de texto. ¿Qué parte del archivo corresponde al mensaje y qué parte corresponde a la firma?

Para abrir el mensaje con un editor de texto (en este caso opté por usar nano), utilizo el comando:

```
nano mensaje.txt
```

```
GNU nano 4.8 mensaje.txt
[Report as spam] [Date Prev][Date Next] [Thread Prev][Thread Next] [Date Index]
[Thread Index]
[SECURITY] [DSA 3997-1] wordpress security update

• To: debian-security-announce@lists.debian.org
• Subject: [SECURITY] [DSA 3997-1] wordpress security update
• From: Yves-Alexis Perez <corsac@debian.org>
• Date: Wed, 11 Oct 2017 13:51:21 +0200
• Message-id: <[📧] 59de05b9.a0071.7885b94b@scapa.corsac.net>
• Reply-to: debian-security-announce-request@lists.debian.org

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA512
-----
```

La parte correspondiente al mensaje es la siguiente:

```

-----BEGIN PGP MESSAGE-----
Debian Security Advisory DSA-3997-1                                security@debian.org
https://www.debian.org/security/                                Yves-Alexis Perez
October 10, 2017                                                https://www.debian.org/security/faq
-----BEGIN PGP MESSAGE-----

Package      : wordpress
CVE ID       : CVE-2017-14718 CVE-2017-14719 CVE-2017-14720 CVE-2017-14721
               CVE-2017-14722 CVE-2017-14723 CVE-2017-14724 CVE-2017-14725
               CVE-2017-14726 CVE-2017-14990
Debian Bug   : 876274 877629

Several vulnerabilities were discovered in Wordpress, a web blogging tool.
They would allow remote attackers to exploit path-traversal issues, perform SQL
injections and various cross-site scripting attacks.

For the oldstable distribution (jessie), these problems have been fixed
in version 4.1+dfsg-1+deb8u15.

For the stable distribution (stretch), these problems have been fixed in
version 4.7.5+dfsg-2+deb9u1.

For the testing distribution (buster), these problems have been fixed
in version 4.8.2+dfsg-2.

For the unstable distribution (sid), these problems have been fixed in
version 4.8.2+dfsg-2.

We recommend that you upgrade your wordpress packages.

Further information about Debian Security Advisories, how to apply
these updates to your system and frequently asked questions can be
found at: https://www.debian.org/security/

Mailing list: debian-security-announce@lists.debian.org
-----BEGIN PGP MESSAGE-----

```

La parte correspondiente a la firma es la siguiente:

```

-----BEGIN PGP SIGNATURE-----

iQEzBAEBCgAdFiEE8vi34Qgfo83x35gF3rYcyPpXRFsFALneAZYACgkQ3rYcyPpX
RftLWwgAqlmWIr+gUrKGKGdCCL1bHy/XI6RM3ezTXa3Mbkesu0JPh9L7JvxpInSm
sAPjgPCMjzL89uJvUt8DsFlK7DsZv03NxZAeLoR7NbZPx0z1iHFUogbv32yZfKwI
DNtBF9HjwTLxb8CSZG9XKgxk1RWwsc0e6DqyyYTcVGdQonzPFUvoZxDctrS6R2TZ
el5ycFEpaoNb9hIaZvpYtXmDYeUibvHeV0SC9WfmBCXKpTX/0th/YK2hS+/P1Xcn
xLLM4tVXgftYOnZYVMZugVcJuVWm8edoMOyv+kFYQQvSgwuyJuGNVePGrUod5kmV
w9bAsukRwTjloWNFJQik90zHahuNcw==
=CMTQ
-----END PGP SIGNATURE-----

```

- c. Busque y descargue de la base de claves de Debian la clave pública del desarrollador que redactó el mensaje. Utilice el formulario disponible en <https://db.debian.org/>

wget

"https://db.debian.org/fetchkey.cgi?fingerprint=4510DCB57ED4704060C6647630550F7871EF0BA8" --output-document yves.key

Para descargar desde la base de claves de Debian la clave pública del desarrollador que redactó el mensaje, utilizo el comando:

```
wget
"https://db.debian.org/fetchkey.cgi?fingerprint=4510DCB57ED4704060
C6647630550F7871EF0BA8" --output-document yves.key
```

```

juanma@ubuntu:~/Desktop/TP 8$ wget "https://db.debian.org/fetchkey.cgi?fingerprint=4510DCB57ED4704060C6647630550F7871EF0BA8" --output-document yves.key
--2021-11-15 22:24:21-- https://db.debian.org/fetchkey.cgi?fingerprint=4510DCB57ED4704060C6647630550F7871EF0BA8
Resolving db.debian.org (db.debian.org)... 82.195.75.106, 2001:41b8:202:deb:1a1a:0:52c3:4b6a
Connecting to db.debian.org (db.debian.org)|82.195.75.106|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/plain]
Saving to: 'yves.key'

yves.key          [ <=> ] 39.66K  119KB/s  in 0.3s

2021-11-15 22:24:23 (119 KB/s) - 'yves.key' saved [40615]

```

d. Importe dicha clave pública en GnuPG utilizando el comando

gpg --import ARCHIVO_CLAVE.KEY

Ahora, procedo a importar la clave pública utilizando el comando:

```
gpg --import yves.key
```

```

juanma@ubuntu:~/Desktop/TP 8$ gpg --import yves.key
gpg: key 30550F7871EF0BA8: 44 signatures not checked due to missing keys
gpg: key 30550F7871EF0BA8: public key "Yves-Alexis Perez <corsac@corsac.net>" imported
gpg: Total number processed: 1
gpg:      imported: 1
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid: 2  signed: 0  trust: 0-, 0q, 0n, 0m, 0f, 2u
gpg: next trustdb check due at 2023-11-09

```

e. ¿Dónde se almacenó la clave pública según lo visto en la experiencia de laboratorio?

Según lo visto en la experiencia de laboratorio, la clave pública recientemente importada se almacenó en el directorio /home/juanma/.gnupg (en mi caso, ya que "juanma" es mi nombre de usuario).

```

juanma@ubuntu: ~/gnupg
juanma@ubuntu:~/gnupg$ ls
openpgp-revocs.d  private-keys-v1.d  pubring.kbx  pubring.kbx~  random_seed  trustdb.gpg
juanma@ubuntu:~/gnupg$

```

f. Valide la autenticidad del mensaje e indique si sufrió alguna alteración. Analice y comente la salida del siguiente comando.

gpg --verify mensaje.txt

Por último, procedo a validar la autenticidad del mensaje mediante el uso del comando:

```
gpg --verify mensaje.txt
```

```

juanma@ubuntu:~/Desktop/TP 8$ gpg --verify mensaje.txt
gpg: Signature made Wed 11 Oct 2017 04:33:42 AM PDT
gpg:      using RSA key F2F8B7E1081FA3CDF1DF9805DEB61CC8FA57445B
gpg: Good signature from "Yves-Alexis Perez <corsac@corsac.net>" [unknown]
gpg:      aka "Yves-Alexis Perez (Debian) <corsac@debian.org>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:      There is no indication that the signature belongs to the owner.
Primary key fingerprint: 4510 DCB5 7ED4 7040 60C6  6476 3055 0F78 71EF 0BA8
Subkey fingerprint: F2F8 B7E1 081F A3CD F1DF  9805 DEB6 1CC8 FA57 445B

```

Como se aprecia en la salida del comando, no se puede indicar que la clave pertenezca a “Yves-Alexis Perez”, es decir, no se provee el servicio de autenticidad.

PARTE 2

1. **Acceda a <https://www2.mincyt.gob.ar/> y tome nota del error. ¿Por qué el navegador dice no confiar en el contenido de esa web? (ayuda: haga clic en “Avanzadas”). ¿Qué validación no se cumple en este caso?**

Una vez accedido a la dirección <https://www2.mincyt.gob.ar/>, el navegador arroja el siguiente error:

ERR_CERT_COMMON_NAME_INVALID

El nombre al que hace referencia el error (es decir el COMMON NAME), es el dominio en el que está instalado el certificado SSL.

Este error está indicando que el nombre común dentro del certificado no es válido por alguna razón.

Esto puede deberse a una de las siguientes causas:

- El nombre del certificado no coincide con el dominio en el que está instalado.
- El certificado SSL no tiene en cuenta las variaciones dominios www vs dominios sin www.
- El sitio tiene instalado un certificado SSL auto firmado y el navegador no lo reconoce como válido o seguro.

entre otras, siendo la primera la más frecuente.

2. **¿Cuántas Autoridades de Certificación (CA) son reconocidas por su navegador web? ¿Qué problemas puede ocasionar la adición de nueva autoridad de certificación falsa? ¿Qué problemas puede ocasionar la eliminación de una o más autoridades de la lista?**

En mi navegador web tengo actualmente 51 autoridades de certificación raíz reconocidas, sumado a las 22 autoridades de certificación intermedias.

Agregar una autoridad de certificación falsa puede ocasionar graves problemas, ya que por ejemplo, puedo creer que estoy accediendo a una página segura (bajo el protocolo HTTPS) cuyo certificado sea uno firmado por una autoridad de certificación falsa, y enviar mis credenciales sin enterarme que están expuestas.

Por el contrario, eliminar una o más autoridades de la lista, me estaría limitando la cantidad de sitios en los que confío (aunque en este caso SI sean sitios confiables que están firmados por autoridades de certificación confiables, las cuales fueron removidas). Esto último puede llevar a la situación inversa a la anterior, estaría desconfiando de una página la cual es totalmente confiable (y mi navegador probablemente mostraría una advertencia al ingresar a la misma, ya que no poseería un certificado firmado por una autoridad de certificación confiable).

3. **Según lo realizado en la experiencia de laboratorio ¿A que corresponden las extensiones de archivos “.crt”, “.key” y “.csr” en el contexto de los certificados?**

Los archivos .crt son archivos de certificado, los cuales son utilizados por los sitios web seguros (HTTPS) para verificar la autenticidad.

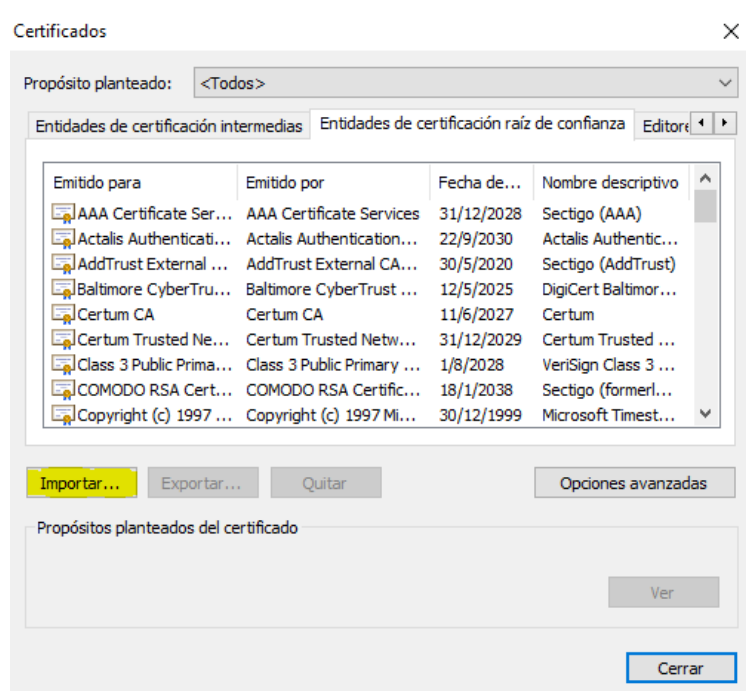
Los archivos .key son archivos que contienen claves. En el ejemplo contenía la clave que luego se utilizaría dentro del archivo .csr.

Los archivos .csr son archivos de solicitud de firma para un certificado digital. Contiene un bloque de texto encriptado que identifica al solicitante del certificado e incluye datos encriptados de país, estado, organización, dominio, dirección de correo electrónico y clave pública. El archivo es utilizado por una Autoridad de Certificación para establecer la prueba de identidad de los sitios web.

4. ¿En qué situación los certificados que son firmados por un tercero pueden aún considerarse no seguros para un navegador? ¿Cómo se puede lograr que un navegador confíe en el certificado para esta situación?

La situación en la que un certificado que es firmado por un tercero puede considerarse no seguro para un navegador es cuando dicho certificado está firmado por una autoridad de certificación desconocida (es decir, no está firmado por ninguna autoridad de certificación de las que posee el navegador en su lista).

Para lograr esto, basta con dirigirse al apartado de certificados de mi navegador de preferencia (en este caso, Brave) y buscar la opción “importar”.



5. ¿En qué escenarios pueden resultar útiles los certificados autofirmados?

Los certificados autofirmados, son aquellos que no han sido validados por una autoridad de certificación (CA). Esto quiere decir que son firmados por uno mismo.

Al no estar firmados por una autoridad de certificación, el navegador mostrará una advertencia cada vez que quiera acceder a un sitio con dicho certificado.

Un escenario en el cual puede resultar útil un certificado autofirmado puede ser, por ejemplo, cuando deba emitir un certificado de manera rápida y fácil para testear algún tipo de configuración.

6. Realice un análisis de la captura cap-ejer-ssl.pcap y donde:

a. Identifique las distintas etapas del protocolo TLS.

1. Cuando el cliente (por ejemplo, un web browser) se contacta con el servidor web, este último le envía primero su certificado, el cual servirá para probar que el servidor es autentico y no está simulando ninguna posible identidad falsa.
2. El cliente verifica la validez del certificado y le envía al servidor un número aleatorio cifrado con la clave pública del servidor.
3. Con ese número, el servidor genera una clave de sesión (session key) con la cual se encriptará la comunicación. Como el número procede del cliente, se puede asegurar que la clave de sesión es originada realmente en el servidor contactado.
4. El server remite la clave de sesión al cliente de forma cifrada. Este cifrado se realiza con el protocolo criptográfico Diffie-Hellmann.
5. Una vez hecho esto, ambas partes pueden enviar sus datos de forma segura con la clave de sesión.

b. Identifique opciones intercambiadas respecto a Cipher Suite y Extensiones soportadas.

Dentro del mensaje Client Hello (de TLS), se encuentra un apartado llamado Cipher Suite, el cual contiene un conjunto de algoritmos que ayudan a mantener segura la conexión a una red.

En dicho mensaje, se encontraban los siguientes algoritmos de cifrado:

```
▼ Cipher Suites (17 suites)
  Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
  Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
  Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
  Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
  Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
  Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
  Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcc8)
  Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
  Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
  Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
  Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
  Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
  Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
  Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
  Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)
  Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
  Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
```

Luego, se encuentran unos apartados referentes a extensiones, como se ven en la siguiente imagen:

- > Extension: server_name (len=35)
- > Extension: extended_master_secret (len=0)
- > Extension: renegotiation_info (len=1)
- > Extension: supported_groups (len=14)
- > Extension: ec_point_formats (len=2)
- > Extension: session_ticket (len=0)
- > Extension: application_layer_protocol_negotiation (len=14)
- > Extension: status_request (len=5)
- > Extension: Unknown type 34 (len=10)
- > Extension: key_share (len=107)
- > Extension: supported_versions (len=5)
- > Extension: signature_algorithms (len=24)
- > Extension: psk_key_exchange_modes (len=2)
- > Extension: record_size_limit (len=2)
- > Extension: padding (len=120)

Dentro de las extensiones, se encuentran por ejemplo, el compartimiento de la clave, los algoritmos de firma, nombre del servidor, información de la renegociación, versiones (de TLS) soportadas, etc.

- c. Identifique la información de los certificados y válidelos contra lo generado en los pasos previos. Indique si el certificado es válido para el dominio/ip accedido y si aún es vigente.**

La información que presenta Wireshark sobre el certificado es la siguiente:

```

Certificates (2948 bytes)
  Certificate Length: 1766
  Certificate: 308206e2308205caa00302010202100ce6b5fd8fb1b07cd4... (id-at-commonName='.telemetry.mozilla.org,id-at-organizationalUnitName=Cloud Services,id-at-organiza
    signedCertificate
      Version: V3 (2)
      serialNumber: 0x0ce6b5fd8fb1b07cd4d54caefe4dbf57
      signature (sha256WithRSAEncryption)
      issuer: rdnSequence (0)
        rdnSequence: 3 items (id-at-commonName=DigiCert SHA2 Secure Server CA,id-at-organizationName=DigiCert Inc,id-at-countryName=US)
          RDNSequence item: 1 item (id-at-countryName=US)
          RDNSequence item: 1 item (id-at-organizationName=DigiCert Inc)
          RDNSequence item: 1 item (id-at-commonName=DigiCert SHA2 Secure Server CA)
      validity
        notBefore: utcTime (0)
          utcTime: 20-08-24 00:00:00 (UTC)
        notAfter: utcTime (0)
          utcTime: 22-10-28 12:00:00 (UTC)
      subject: rdnSequence (0)
      subjectPublicKeyInfo
      extensions: 10 items
      algorithmIdentifier (sha256WithRSAEncryption)
      Padding: 0
      encrypted: d1d426b3908dbabf3237e3b108daa959202671a3cbb7c4b0...

```

El certificado es válido y aun es vigente.

7. Investigue y comente brevemente en que consiste el servicio “Lets Encrypt”.

Let's Encrypt es una autoridad de certificación que provee certificados X.509 gratuitos para el cifrado de seguridad de nivel de transporte (TLS).

Para operar, funciona utilizando el protocolo ACME (Automatic Certificate Management Environment), el cual hace posible la configuración de un servidor HTTPS y que este obtenga, de manera automática, un certificado confiado por el navegador, sin necesidad de que intervenga ningún humano.

PARTE 3

1. ¿Por qué cree que, por defecto, uno no puede conectarse como usuario root? ¿Por qué cree que se recomienda cambiar el puerto por defecto? ¿Qué archivo debe modificar para poder cambiar estas opciones?

Creo que, por defecto, uno no puede conectarse como usuario root ya que eso implicaría un riesgo de seguridad enorme.

Al no permitir el acceso root por defecto, la persona que intente realizar una conexión ssh tiene que conocer el nombre de usuario. En caso de que la persona que intentó realizar la conexión ssh logre acertar el nombre de usuario, no tendría los permisos de root sobre el host.

Para cambiar estas opciones, debo modificar el archivo ubicado en el directorio /etc/ssh/ llamado sshd_config.

Una vez dentro de ese archivo, modifico la línea

PermitRootLogin prohibit-password

Por

PermitRootLogin yes

Y luego reinicio el servicio para aplicar los cambios (service sshd restart).

2. En que escenarios supone que podría ser útil realizar un tunel SSH.

Realizar un Tunel SSH puede ser útil para navegar por la red en forma segura, estando en una red pública (por ejemplo).

Puede darse el caso de que nos conectemos a la red pública de un aeropuerto.

En este caso, todo el tráfico puede ser interceptado por alguien que esté sniffendo dicha red.

Sin ir más lejos, las peticiones HTTP viajan en forma de texto plano, por lo que al realizar una quien esté interceptando el tráfico podría visualizar toda la información de dicha petición.

Para evitar esto, una posible solución podría ser un Tunel SSH, el cual consta de realizar un tunel entre mi dispositivo cliente (el cual está en la red comprometida) y un servidor SSH (el cual tiene que estar fuera de dicha red).

Luego, una vez realizada la petición, viajará cifrada por el Tunel SSH y será el servidor SSH el que realice la petición HTTP.

3. ¿Es posible ejecutar aplicaciones que requieran de interfaz gráfica con SSH?

Si, es posible.

Para realizar esto, debemos agregar el parámetro -X para indicar que vamos a ejecutar un programa con interfaz gráfica.

Por ejemplo:

```
ssh -X -p 22 usuario@servidor aplicación
```

Siendo -p el puerto destino (el puerto por defecto de SSH es el 22) y aplicación la aplicación que se va a ejecutar con interfaz gráfica (por ejemplo, libreoffice).

4. ¿Qué reglas de firewall implementaría para que un router acepte conexiones SSH al puerto 2222?

Para que un router acepte conexiones SSH en el puerto 2222, utilizaría los siguientes comandos:

```
iptables -A INPUT -p tcp --dport 2222 -j ACCEPT  
iptables -A OUTPUT -p tcp --sport 2222 -j ACCEPT
```