Seguridad 2. Criptografía y protocolos de comunicación.

Fecha de Entrega: 01/12/2020.

Objetivo: Desarrollar una experiencia práctica en seguridad de redes utilizando tecnologías que permitan brindar servicios de seguridad apoyados en la criptografía. En particular a partir de la experimentación con tres casos de estudio relevantes para la asignatura: OpenGPG, HTTPS y SSH.

ISO: FCAP**S**.

Bibliografía

- STALLINGS, W. 2011. *Cryptography and Network Security: Principles and Practice.* (5th ed). Prentice Hall
  - Capítulo 1: "Overview"
  - Capítulo 2: "Classical Encryption Techniques"
  - Capítulo 3. Sección 1: "Block Cipher Principles"
  - Capítulo 9. Sección 1: "Principles of Public-Key Cryptosystems"
  - Capítulo 11: "Cryptographic Hash Functions"
  - Capítulo 13. Sección 1: "Digital Signatures"
  - Capítulo 14: Key Management and Distribution en *Cryptography and Network Security - Principles and Practice (6th ed)*. Pearson Education Inc.
  - Capítulo 17: Transport-Level Security en *Cryptography and Network Security - Principles and Practice (6th ed)*. Pearson Education Inc.
  - Capítulo 18. Sección 1: "Pretty Good Privacy (PGP)"
- KESSLER, G. 2019. *An Overview of Cryptography*. https://www.garykessler.net/library/crypto.html (bib. de consulta)
- ASHLEY, M. 1999. Capítulo 1: "Primeros Pasos". En *Guía de "GNU Privacy Guard".* The Free Software Foundation. https://gnupg.org/gph/es/manual.html#INTRO (bib. de consulta)
- GORALSKI, W. 2017. Capítulo 27: "Securing Sockets with SSL" en *The Illustrated Network: How TCP/IP Works in a Modern Network (2nd ed)*. Morgan Kaufmann. https://www.sciencedirect.com/science/book/9780128110270
- DRISCOLL, M. 2018. The Illustrated TLS Connection - Every byte of a TLS connection explained and reproduced - https://tls.ulfheim.net/
- DRISCOLL, M. 2018. Server Certificate Detail - https://tls.ulfheim.net/certificate.html
- COOPER, D., et al. 2008. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, RFC 5280.

Experiencia de laboratorio

PARTE 1 - OpenGPG: 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. ¿Que 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 quien es el emisor del mensaje.
- **No repudio**: se puede saber quien 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 (*kprivAlicia*) 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 que 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 quien 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:

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

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

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 pude haber cifrado este resumen (la clave pública de Beto es justamente pública). Como no podemos saber quien es el emisor de este mensaje, ni decir si esta íntegro (cualquiera puede modificar y generar su propio resumen cifrado con *kpubBeto*) este mensaje, por más bonito y sofisticado que se vea, no tiene garantizado ningún servicio de seguridad. *Para pensar: ¿Cuál sería una alternativa más lógica?¿Como se llamaría a ese mecanismo?*

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.

3 Descargue el siguiente archivo de texto (o genere un archivo propio de texto plano) y luego cífrelo con un algoritmo *simétrico* de su elección:

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

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

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?*

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.

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

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 entropia 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 ademas 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. *Para pensar: ¿De qué forma cree que se guarda la 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` .

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` .

7 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:

$$\text{Servidor:} \quad k_{pub}Server, k_{priv}Server$$

$$\text{Autoridad de certificación (CA):} \quad k_{pub}CA, k_{privCA}$$

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

$$Hash(CSR) = H$$

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

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

Para pensar: ¿Por qué la CA no necesita la clave privada del servidor web? ¿Porque el servidor web no necesita la clave privada de la CA?

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.

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.

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) ¿Que entidades emitieron tales certificados? ¿Cuál es el orden de jerarquía? ¿Hay alguna coincidencia en la jerarquía de los certificados de los sitios visitados?

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.

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?

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 recomendadatos, 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.

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 pensar: ¿Cuál sería el problema de utilizar certificados autofirmados para usar en la WWW?

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*

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 -plunt` (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` .

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/DVQSJhQ5BqyURE64kxOcJ1QyIlKloY.
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 algortimo ECDSA a partir de la clave pública del servidor. De esta manera si nosostros tenemos otro mecanismo para verficar 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.

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).

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 sessión 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` .

5 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` .

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 “tunel 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.

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.

    Para finalizar responda: ¿Son equivalentes los últimos dos casos? Justifique su respuesta.

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 imágen en su respuesta a este punto.

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

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.

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` . ¿Quien podrá ver el contenido de meme.jpg.gpg"? ¿Con que claves cree que fue cifrado?

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.

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

   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=4510DCB57ED470406
   0C6647630550F7871EF0BA8" --output-document yves.key
   ```

   d. Importe dicha clave pública en GnuPG utilizando el comando

   ```
   gpg --import ARCHIVO_CLAVE.KEY
   ```

   e. ¿Donde se almacenó la clave publica según lo visto en la experiencia de laboratorio?

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

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

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?

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?

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?

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?
5. ¿En qué escenarios pueden resultar útiles los certificados autofirmados?
6. Realice un análisis de la captura *cap-ejer-ssl.pcap* y donde:
    a. Identifique las distintas etapas del protocolo TLS.
    b. Identifique opciones intercambiadas respecto a Cipher Suite y Extensiones soportadas.
    c. Identifique la información del los certificados y valídela contra lo generado en los pasos previos. Indique si el certificado es válido para el dominio/ip accedido y si aún es vigente.
7. Investigue y comente brevemente en que consiste el servicio “Lets Encrypt”.

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?
2. En que escenarios supone que podría ser útil realizar un tunel SSH.
3. ¿Es posible ejecutar aplicaciones que requieran de interfaz gráfica con SSH?
4. ¿Qué reglas de firewall implementaría para que un router acepte conexiones SSH al puerto 2222?

Guia de lectura

1) ¿Cuales son los servicios de seguridad definidos por la recomendación ITU-T X.800?
2) Según el modelo OSI ¿Cuáles son los tres elementos de la arquitectura de la seguridad?
3) ¿Qué mecanismos de seguridad se encuentran definidos en la recomendación X.800 y en que consisten?
4) ¿Qué políticas de seguridad se pueden adoptar?
5) ¿Cuales son los dispositivos o roles que se pueden definir para proveer mecanismos de seguridad de red y cuales son sus funciones?
6) ¿En qué consiste el concepto de “Seguridad como un Servicio” (Security as a Service)?
7) ¿Cuales son los elementos de la arquitectura de un sistema de cifrado simétrico ?
8) ¿Que ventajas y desventajas tienen los algoritmos de cifrado simétricos vs los algoritmos asimétricos?
9) ¿Porqué no se recomienda la seguridad por oscuridad?
10) ¿Cuales son los dos tipos de ataques se pueden pueden realizar a un algritmo de encriptación?
11) ¿Qué mecanismos existen para asegurar la integridad de los datos?
12) ¿Qué es un Certificado? ¿Cuáles son los datos más importantes del mismo?
13) ¿Qué es una autoridad de certificación?

14) ¿Por qué un servidor que implemente HTTPs necesitaría una clave privada?

15) ¿Cómo un navegador verifica que un servidor web tiene un certificado válido?

16) ¿Qué es SSH?

17) ¿Qué utilidades se pueden utilizar con SSH?

18) ¿Qué es un tunel SSH y porqué puede resultar útil?

Anexo OpenPGP

Cifrar un archivo simétricamente

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

ALGORITMO es alguno de los algoritmos simétricos soportados por la aplicación, y *ARCHIVO.EXT* es el nombre que usted dio al archivo elegido.

El archivo resultande se almacena en el directorio de trabajo con el nombre ARCHIVO.EXT.gpg

Descifrar un archivo cifrado (simétrica o asimétricamente)

```
gpg ARCHIVO.EXT.gpg
```

Si en el paso de cifrado se utilizó la opción `-a` debe volver a utilizarse al momento de descifrar.

Generar par de claves asimétricas

```
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)
```

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

Exportar claves públicas propias

```
gpg --export -a SU-EMAIL@DOMINIO.COM
```

Importar claves públicas de un tercero

```
gpg --import ARCHIVO.DE.CLAVE
```

Listar claves conocidas

```
gpg --list-keys
```

Cifrar un archivo asimétricamente

```
gpg --encrypt --recipient DESTINATARIO@DOMINIO.COM ARCHIVO.EXT
```

Fimar digitalmente un archivo de texto plano

```
gpg -u SUEMAIL@DOMINIO.COM --clearsign ARCHIVO.TXT
```

El archivo resultande se almacena en el directorio de trabajo con el nombre ARCHIVO.EXT.ASC

Fimar digitalmente un archivo binario (imagen, pdf, etc)

```
gpg -u SUEMAIL@DOMINIO.COM --sign ARCHIVO.EXT   (-sa para utilizar ASCII de 7 bit)
```

Cifrar y firmar digitalmente un archivo

```
gpg -es -u SUEMAIL@DOMINIO.COM --recipient DESTINATARIO@DOMINIO.COM ARCHIVO.EXT
```

Anexo Apache y HTTPS

Creación de un certificado autofirmado (sólo para proveer Confidencialidad)

1. iGenerar la clave privada (Private Key) del servidor, que será almacenada en el archivo `server.key` . Instale el paquete openssl de ser necesario.

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

2. Generar la solicitud de firma de certificado (Certificate Signing Request), que será almacenada en el archivo `server.csr` . Completar los campos solicitados según el formulario de solicitud. Por ejemplo:

   ```
   openssl req -new -sha256 -key server.key -out server.csr
   -----
   Country Name (2 letter code): AR
   State or Province Name (full name): Buenos Aires
   Locality Name (eg, city): Lujan
   Organization Name (eg, company): Organización Example S.A.
   Organizational Unit Name (eg, section): Gerencia de Sistemas
   Common Name (eg, YOUR name): SU-DIRECCION-IP
   Email Address: SU-DIRECCION-DE-CORREO

   Please enter the following 'extra' attributes to be sent with your cert req.
   A challenge password:
   An optional company name:
   ```

3. Firmar la petición con la propia clave privada como sigue. En este caso se lo denomina “autofirmar”, puesto que estamos firmando la clave pública con la misma clave privada que le corresponde. En los sitios web que operan con TLS, quien firma la petición es una tercera entidad (una Autoridad de Certificación) en la que “todos” confían.

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

4. Para visualizar el contenido del certificado digital, ejecutar:

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

Pasos a seguir para configurar e instalar los certificados en el servidor web

1. Instalar el servidor web Apache 2 o superior.

   ```
   # apt-get install apache2
   ```

2. Activar los módulos `rewrite` y `ssl` , y el sitio `default-ssl` en Apache.

```
# a2enmod rewrite
# a2enmod ssl
# a2ensite default-ssl
```

3. Crear la ubicación `/etc/apache2/certificados` donde se almacenarán los certificados, copiarlos a la misma y asignar los permisos adecuados según la documentación disponible en `/usr/share/doc/apache2.2-common/README.Debian.gz` :

```
#
# mkdir /etc/apache2/certificados
# cd /etc/apache2/certificados
# mv origen/server.crt .
# mv origen/server.key .
# chown root.root server.crt server.key
# chmod 444 server.crt
# chmod 400 server.key
```

4. Editar el archivo `/etc/apache2/sites-enabled/default-ssl.conf` y reemplazar las líneas `SSLCertificateFile` y `SSLCertificateKeyFile` según sigue:

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

5. De ser necesario, habilitar el acceso al puerto 443 en el firewall del host y los nodos que correspondan.

6. Reiniciar el servidor apache (no alcanza sólo con recargar la configuración).

```
# service apache2 restart
```