Introducción a Secure Shell

Javier Smaldone
http://www.smaldone.com.ar

Versión 0.2 - 20 de enero de 2004

Resumen

Este texto tiene como objetivo servir de guía introductoria al uso de Secure Shell. No se trata de una descripción exhaustiva acerca de sus posibilidades, aunque trataré de explorar algunas de sus características y funciones más útiles. También realizaré una breve descripción de los métodos de cifrado[1], en particular de los sistemas de cifrado asimétrico y de RSA.

No soy experto en cifrado ni en seguridad informática. Sólo he intentado escribir un pequeño texto de ayuda a quienes, como yo, comienzan a utilizar esta herramienta.

Índice

[1]La palabra inglesa "*crypt*" suele traducirse como "*criptografía*" o "*encripción*", vocablos inexistentes en el idioma español. El término correcto es "*cifrado*".

1. Introducción

1.1. ¿Qué es la "seguridad"?

Hace un par de años asistí a un curso de seguridad del que, si algo aprendí, fue una frase muy clarificadora respecto de la seguridad: "*La seguridad no es un producto, es un proceso*". Muchas veces, decimos que "*el software X es seguro*" o que "*el protocolo Y es seguro*", siendo que lo correcto sería decir que "*tal sistema ofrece tal o cual mecanismo que permite asegurar la seguridad en tal o cual sentido*". Es asombroso ver en la actualidad a supuestos "expertos" en seguridad que venden soluciones mágicas ("*seguridad en cajas*"). Para que un sistema sea considerado *seguro* en determinado grado, deben definirse, respetarse y revisarse continuamente políticas y procedimientos tendientes a lograr ese objetivo.

Si a lo largo de este texto usted encuentra que en algún momento digo que algo "*es seguro*", le pido que lo tome como una forma abreviada de decir lo que he expresado en el párrafo precedente.

1.2. ¿Qué es el Secure Shell?

Secure Shell, también llamado SSH, es un protocolo utilizado para login y ejecución de procesos remotos.

Resumiendo (quizás demasiado) SSH nos permite:

- Iniciar sesiones (login) en servidores remotos.
- Ejecutar comandos remotamente.
- Copiar archivos entre distintos hosts.
- Ejecutar aplicaciones X11 remotamente.
- Realizar túneles IP cifrados.
- Divertirnos[2].

SSH no es solo un reemplazo de telnet, rlogin, rexec, ftp, XDMCP y otros. Además de brindar todas estas posibilidades con, básicamente, un único programa, brinda comunicaciones seguras (cifradas) entre el cliente y el servidor.

1.3. Algoritmos de cifrado

Trataré de explicar muy brevemente (dado mis pobres conocimientos al respecto) los principios básicos del cifrado de información.

Las técnicas de cifrado consisten en manipular la información de manera de asegurar las siguientes propiedades:

[2] ¡Nunca debemos olvidarnos de esto!

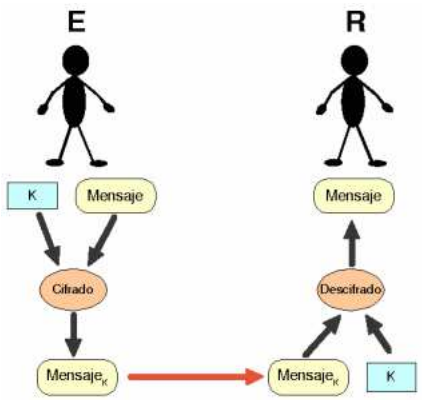

Figura 1: Cifrado simétrico

1. Confidencialidad: No puede acceder a la información nadie que no sea su legítimo destinatario.
2. Integridad: La información no puede ser alterada (sin ser esto detectado) en el tránsito desde el emisor hacia el destinatario.
3. No repudio: El creador o emisor de la información no puede dejar de reconocer su autoría.
4. Autenticación: Tanto el emisor como el receptor pueden confirmar la identidad de la otra parte involucrada en la comunicación.

En particular, para lo que nos interesa, prestaremos atención a tipos de algoritmos: los de cifrado simétrico (con clave compartida) y los de cifrado asimétrico (con clave pública y clave privada).

1.3.1. Cifrado simétrico

Esta técnica consiste en el uso de una clave que es conocida tanto por el emisor como por el receptor (y, se supone, por nadie más). La figura 1 muestra un esquema de este tipo:

E y R conocen la clave K. El emisor, E, desea transmitir el mensaje $Mensaje$ a R. Para ello, utilizando determinado algoritmo de cifrado simétrico y la clave K, genera el mensaje $Mensaje_K$, que es transmitido a R. Éste, aplicando la misma clave y el algoritmo inverso, obtiene nuevamente el mensaje original.

Al respecto del algoritmo utilizado, es deseable que cumpla varias propiedades, entre ellas:

- Que sea muy difícil descifrar el mensaje sin conocer la clave.
- Que la publicidad del algoritmo de cifrado/descifrado no tenga influencia en la seguridad del mismo. (¡Nunca un algoritmo de cifrado puede basarse en la suposición de que nadie conoce exactamente cómo funciona!)
- Que una mínima alteración de la clave, produzca grandes cambios en el mensaje cifrado.

Quizás el ejemplo más simple de cifrado simétrico sea el juego de niños de desplazar las letras del mensaje una determinada cantidad de posiciones: Por ejemplo, cifrar el mensaje "Hola Mundo" como "Ipmb Nvñep", desplazando las letras una posición. En este caso, la clave que ambas partes deben conocer es la cantidad de posiciones que se desplazaron las letras. (De todas formas, esto no es muy difícil de averiguar, ¿no?)[3].

La principal desventaja del cifrado simétrico consiste en que ambas partes deben conocer la clave: Un secreto compartido no es un secreto por mucho tiempo.

Entre los algoritmos de cifrado simétrico más utilizados podemos nombrar 3DES, Blowfish, IDEA, CAST128 y Arcfour

[3] Una variante de este algoritmo, conocida como ROT13 dado que desplaza las letras 13 posiciones, es muy utilizada para cifrar textos de forma simple.

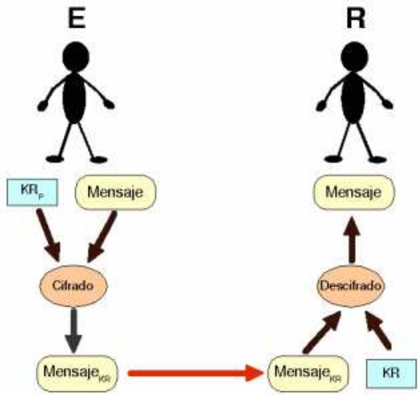

Figura 2: Confidencialidad mediante cifrado asimétrico

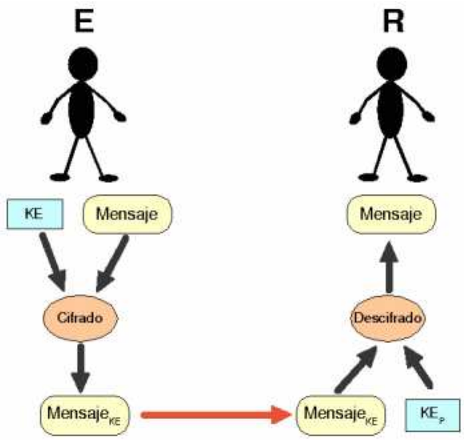

Figura 3: Autenticidad mediante cifrado asimétrico

1.3.2. Cifrado asimétrico

Las técnicas de cifrado asimétrico se basan en el uso de dos claves: una pública y otra privada.

En el ejemplo de la figura 2, R posee dos claves: KR (su clave privada, conocida sólo por él) y KR_P (su clave pública, conocida por cualquiera). En este ejemplo, E desea transmitir el mensaje $Mensaje$ a R, de manera que nadie, excepto este último, pueda conocer su contenido. Para ello, utilizando la clave KR_P y un algoritmo de cifrado asimétrico, genera el mensaje $Mensaje_{KR}$, el cual es transmitido. Luego, R, utilizando el algoritmo inverso y su clave privada KR, reproduce el mensaje original.

El algoritmo debe garantizar que nadie que no conozca la clave KR puede obtener el mensaje original.

La figura 3 muestra un ejemplo en el cual, E cifra el mensaje con su clave privada KE, obteniendo el mensaje $Mensaje_{KE}$, que es transmitido a R quien, usando la clave privada de E (KE_P), obtiene el mensaje original. Dado que KE_P es pública, cualquiera podría haber obtenido el mensaje, pero lo que esto le garantiza a R es que la única persona que pudo haberlo generado es E (ya que solamente él conoce la clave KE con que fue cifrado)[4].

El siguiente ejemplo, la figura 4 muestra los dos casos anteriores combinados para lograr las propiedades de confidencialidad y de autenticación.

La complejidad computacional de los algoritmos de cifrado asimétrico hace que sea muy costosa el uso

[4]Esta es una simplificación del mecanismo utilizado en la realidad, conocido como "*firma digital*", que también garantiza la integridad de la información.

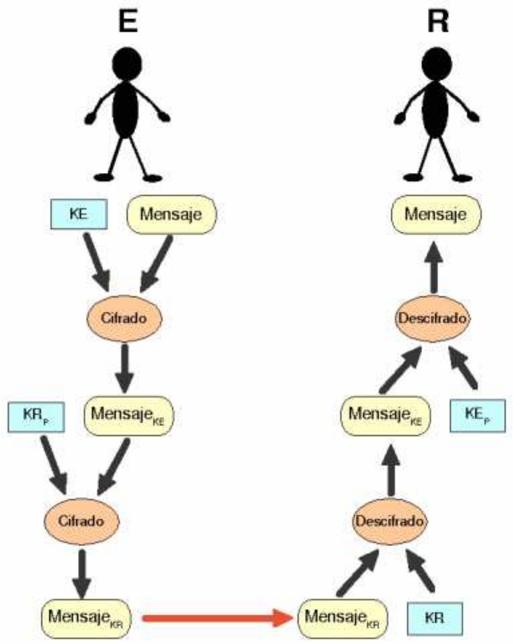

Figura 4: Un ejemplo más complejo

de un esquema de "doble cifrado" como el anterior, por lo cual generalmente se utiliza un esquema mixto, combinando cifrado asimétrico y simétrico.

En una primera etapa de la comunicación, se utiliza cifrado asimétrico para intercambiar, de manera segura, la clave que luego será utilizada junto con un algoritmo de cifrado simétrico para cifrar el resto de la comunicación.

El algoritmo de cifrado asimétrico más utilizado, y el que utilizaremos con SSH, es el llamado RSA (ver apéndice A).

2. Secure Shell. Principios y funcionamiento

SSH es una aplicación cliente servidor que utiliza el puerto 22 de TCP. La implementación más utilizada de SSH es *OpenSSH*, desarrollado por el equipo de *OpenBSD*, y que puede obtenerse de `http://www.openssh.org`[5]. *OpenSSH*, que incluye tanto el cliente como el servidor, está diseñado para plataformas Unix. Si usted tiene la desdicha de utilizar Microsoft Windows[6], puede utilizar un excelente cliente llamado *PuTTY* (`http://www.chiark.greenend.org.uk/~sgtatham/putty/`), aunque no podrá beneficiarse de todas las opciones que ofrece SSH.

A lo largo de este texto, examinaremos algunos detalles de la configuración y el uso de *OpenSSH* en un sistema *GNU/Linux* (la instalación dependerá de la distribución utilizada).

2.1. Descripción general del funcionamiento de SSH

Existen en la actualidad dos versiones: SSH1 y SSH2. Es ampliamente aconsejable el uso de SSH2 (incorpora varias mejoras y es más seguro). Por lo tanto, en el resto del texto nos referiremos siempre a éste último.

Al conectarse un cliente de SSH con el servidor, se realizan los siguientes pasos:

[5] Obviamente, se trata de software libre **;)**

[6] Esto dicho con toda sinceridad.

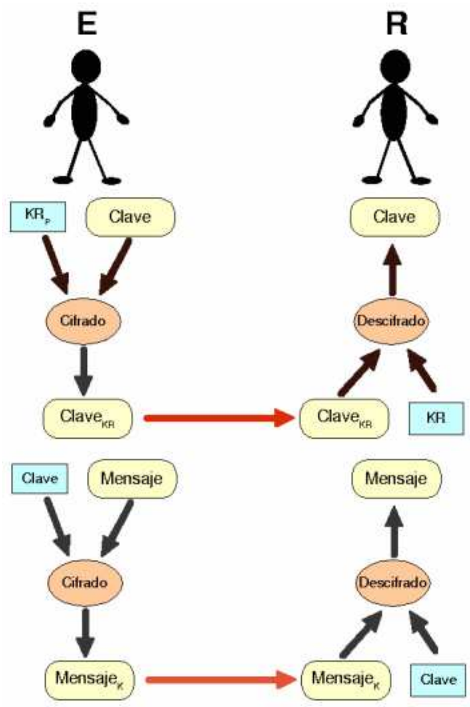

Figura 5: Cifrado asimétrico y simétrico combinados

1. El cliente abre una conexión TCP al puerto 22 del host servidor.
2. El cliente y el servidor acuerdan la versión del protocolo a utilizar, de acuerdo a su configuración y capacidades[7].
3. El servidor posee un par de claves pública/privada de RSA (llamadas "claves de host"). El servidor envía al cliente su clave pública.
4. El cliente compara la clave pública de host recibida con la que tiene almacenada, para verificar su autenticidad. Si no la conociera previamente, pide confirmación al usuario para aceptarla como válida[8].
5. El cliente genera una clave de sesión aleatoria y selecciona un algoritmo de cifrado simétrico.
6. El cliente envía un mensaje conteniendo la clave de sesión y el algoritmo seleccionado, cifrado con la clave pública de host del servidor usando el algoritmos RSA.
7. En adelante, para el resto de la comunicación se utilizará el algoritmo de cifrado simétrico seleccionado y clave compartida de sesión.
8. Luego se realiza la autenticación del usuario. Aquí pueden usarse distintos mecanismos. Más adelante analizaremos los más importantes.
9. Finalmente se inicia la sesión, por lo general, interactiva.

Una de las principales fortalezas de SSH es la seguridad: ni bien se establece la conexión, y antes de la autenticación del usuario, queda establecido un canal cifrado seguro. En adelante, todo el tráfico de la sesión será indescifrable.

[7] Previamente el servidor realiza ciertas comprobaciones, como por ejemplo si el host del cliente tiene permisos para conectar, etc.

[8] Esto implica asumir el riesgo de que alguien, en el camino entre el servidor y el cliente, haya cambiado la clave pública por otra suya (lo que se conoce con el nombre de *ataque "man in the middle"*.

2.2. Métodos de autenticación de usuarios

Existen varios métodos que pueden utilizarse para autenticar usuarios. Aunque son mutuamente excluyentes, tanto el cliente como el servidor pueden soportar varios de ellos. En el momento de la autenticación se aplicarán los métodos que ambos soporten, siguiendo un orden determinado.

Analizaremos los dos métodos más importantes a los fines de este documento:

2.2.1. Autenticación con contraseña:

Es el método más simple (y común) de autenticación. El cliente solicita al usuario el ingreso de una contraseña (password) y la misma es enviada al servidor, el cuál validará la misma utilizando los mecanismos configurados en el sistema (generalmente, utilizará la autenticación del sistema Unix para validar la contraseña contra el contenido del archivo `/etc/shadow`, de la misma manera que lo hace el `login`).

Las desventajas de este sistema son varias:

- El usuario debe tipear su contraseña cada vez que se conecta al servidor.
- La contraseña del usuario es enviada hacia el servidor[9].

2.2.2. Autenticación con clave pública:

Para poder llevarse a cabo, el usuario debe tener un par de claves pública/privada, y la clave pública debe estar almacenada en el servidor.

Luego de establecida la conexión, el servidor genera un número aleatorio llamado "desafío" (*challenge*) que es cifrado con la clave pública del usuario usando RSA o DSA (en los ejemplos nos referiremos siempre a RSA). El texto cifrado es enviado al cliente, que debe descifrarlo con la clave privada correspondiente y devolverlo al servidor, demostrando de esta manera que el usuario es quien dice ser.

Las ventajas de este método respecto del anterior son:

- El usuario no debe tipear (ni recordar) ninguna contraseña.
- Ninguna contraseña secreta (en este caso, la clave privada del usuario) es enviada al servidor.

Importante: No debe confundirse el hecho de que el cliente y el servidor utilicen RSA para establecer la comunicación cifrada, con el hecho de que un usuario pueda utilizar el mismo algoritmo para autenticarse ante el servidor. Son dos etapas bien diferenciadas e independientes (aunque puede utilizarse el mismo algoritmo de cifrado en ambas).

2.3. Configuración de *OpenSSH*

Los archivos de configuración de *OpenSSH* se encuentran en `/etc/ssh` o en `/usr/local/etc/`, dependiendo de cómo se haya instalado (asumiremos `/etc/ssh`). Los más importantes son:

2.3.1. Archivo `sshd_config`

En este archivo se describe la configuración del servidor SSH. Analizaremos las opciones más relevantes:

- **`Port`**: Esta opción permite especificar el puerto TCP que utilizará el servidor. El valor usual es **`22`**.
- **`Protocol`**: Versión del protocolo a utilizar. Usaremos solamente el valor **`2`**.
- **`HostKey`**: Clave privada de RSA o DSA del host. Normalmente las claves de host son generadas en el momento de la instalación de *OpenSSH*, y el valor de esta opción es **`/etc/ssh/ssh_host_rsa_key`**.

[9] Por ejemplo, si estamos conectándonos al servidor a través de su nombre de DNS y alguien ha manipulado el servidor DNS podríamos enviar la contraseña a un servidor distinto al que creemos estar accediendo.

- **PubkeyAuthentication**: Si el valor de esta opción es **yes**, entonces se permite la autenticación de usuarios mediante clave pública.

- **AuthorizedKeysFile**: Mediante esta opción se indica al servidor en donde están almacenadas las claves públicas de los usuarios. El valor por defecto es **%h/.ssh/authorized_keys**, e indica que deben buscarse en el archivo **authorized_keys**, del directorio **.ssh** dentro del directorio *home* de cada usuario.

- **PasswordAuthentication**: Si el valor de esta opción es **yes**, se permite la autenticación de usuarios mediante contraseñas.

- **X11Forwarding**: Si el valor de esta opción es **yes**, se habilita el reenvío de X11 a través de la conexión SSH. Más adelante analizaremos en detalle esta interesante opción.

2.3.2. Archivo ssh_config

En este archivo se describe la configuración del cliente SSH. Las opciones más importantes son:

- **Host**: Esta opción actúa como un divisor de sección. Puede repetirse varias veces en el archivo de configuración. Su valor, que puede ser una lista de patrones, determina que las opciones subsiguientes sean aplicadas a las conexiones realizadas a los hosts en cuestión. El valor * significa "*todos los hosts*".

- **Port**: Es el puerto de TCP que utiliza el servidor (normalmente, **22**).

- **Protocol**: Es la versión del protocolo utilizada (normalmente **2,1**, pero se recomienda solamente **2**) .

- **PubkeyAuthentication**: Autenticación mediante clave pública (se recomienda **yes**).

- **IdentityFile**: Archivo que contiene la clave pública (en caso de usar RSA, lo usual es **~/.ssh/id_rsa**).

- **PasswordAuthentication**: Autenticación mediante contraseñas (**yes** o **no**).

- **StrictHostKeyChecking**: Define que hará el cliente al conectarse a un host del cual no se dispone de su clave pública. El valor **no** hace que sea rechazara la clave del servidor (y por lo tanto, se aborte la conexión), el valor **yes** hace que se acepte automáticamente la clave recibida, y el valor **ask** hace que se pida confirmación al usuario.

- **Ciphers**: Algoritmos de cifrado simétrico soportados para su uso durante la sesión.

- **ForwardX11**: Reenvío de aplicaciones X11 (los posibles valores son **yes** o **no**).

2.3.3. Otros archivos:

Estos son algunos de los demás archivos utilizados por *OpenSSH*. La variable `$HOME` debe ser interpretada como el directorio *home* del usuario en cuestión.

- **/etc/ssh/ssh_host_rsa_key**: Clave privada de RSA del host (con permiso de lectura sólo para el usuario `root`).

- **/etc/ssh/ssh_host_rsa_key.pub**: Clave pública de RSA del host (con permiso de lectura para todos los usuarios).

- **/etc/ssh_known_hosts2**: Claves públicas de hosts conocidos (del sistema).

- **$HOME/.ssh/config**: Configuración del cliente de SSH para cada usuario. Su contenido es similar al archivo de configuración global `/etc/ssh/ssh_config`.

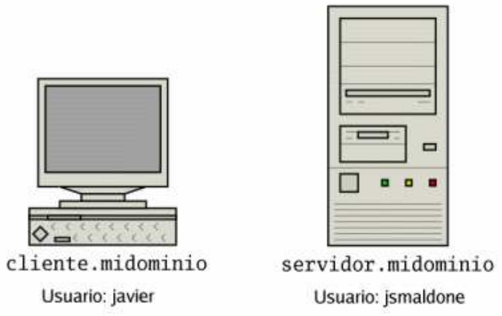

Figura 6: Ejemplo de uso de SSH

- **$HOME/.ssh/id_rsa**: Clave privada de RSA del usuario (con permiso de lectura sólo para el usuario).
- **$HOME/.ssh/id_rsa.pub**: Clave pública de RSA del usuario.
- **$HOME/.ssh/known_hosts2**: Claves públicas de hosts conocidos (del usuario).
- **$HOME/.ssh/authorized_keys2**: Claves públicas del usuario para la autenticación del mismo. Este archivo debe estar en el servidor a conectar.

2.3.4. El servidor de SSH

El servidor o *demonio* incluido en *OpenSSH* se llama `sshd` y usualmente está ubicado en `/usr/sbin/` o en `/usr/local/sbin/`. La configuración más común este se ejecute al iniciar el sistema a través de init.

Para mayor información, como siempre, RTFM (`man sshd`).

2.3.5. El cliente de SSH

El cliente incluido en *OpenSSH* se llama ssh y usualmente está ubicado en `/usr/bin` o en `/usr/local/bin/`. A lo largo de el presente texto examinaremos algunas de sus opciones, pero por el momento nos bastará con conocer su sintaxis básica:

```
ssh [usuario@]host
```

Si el nombre de usuario en el host remoto (servidor) coincide con el nombre de usuario en el host local (cliente), podemos omitirlo. Por ejemplo, el siguiente comando

```
javier@cliente:~>ssh servidor.midominio
```

intentará iniciar una sesión interactiva (shell) en el host remoto `servidor.midominio` usando el usuario `javier`, en tanto que

```
javier@cliente:~>ssh jsmaldone@servidor.midominio
```

intentará iniciar una sesión interactiva en el host remoto `servidor.midominio` usando el usuario `jsmaldone`.

Nuevamente, para mayores detalles: `man ssh`.

3. Usando Secure Shell

Para ejemplificar algunas situaciones de uso utilizaremos una situación como la descripta en la figura 6.

En el cliente `cliente.midominio` existe el usuario `javier`, en tanto que en el servidor `servidor.midominio` existe un usuario `jsmaldone`. Ambas cuentas son utilizadas por la misma persona.

El archivo `/etc/ssh/sshd_config` en `servidor.midominio` contiene lo siguiente[10]:

```
Port 22
Protocol 2
HostKey /etc/ssh/ssh_host_rsa_key
PubkeyAuthentication yes
PasswordAuthentication yes
X11Forwarding yes
```

Como puede verse, el servidor está usando SSH2, su clave privada de RSA es `/etc/ssh/ssh_host_rsa_key`, soporta autenticación con clave pública o contraseña y tiene habilitado el reenvío de aplicaciones X11.

En `cliente.midominio`, el archivo `/etc/ssh/ssh_config` contiene lo siguiente:

```
Host *
Port 22
Protocol 2
StrictHostKeyChecking ask
PasswordAuthentication yes
PubkeyAuthentication yes
IdentityFile ~/.ssh/id_rsa
ForwardX11 yes
```

El cliente está configurado para conectarse al puerto 22, utilizando SSH2, con autenticación mediante contraseña o clave pública (usando RSA y buscando la clave privada en `.ssh/id_rsa` dentro del directorio *home* de cada usuario) y soporta el reenvío de aplicaciones X11.

3.1. Iniciando una sesión remota con contraseña

El primer ejemplo que analizaremos es el inicio de una sesión remota a través de SSH, utilizando autenticación por contraseña. Para ello, el usuario `javier` en `cliente.midominio` debe ingresar el siguiente comando:

```
javier@cliente:~>ssh jsmaldone@servidor.midominio
```

Al ser la primera vez que se conecta al servidor servidor.midominio, si previamente no ha agregado la clave pública del mismo en `~/.ssh/known_hosts2` (o en el archivo global `/etc/ssh/ssh_known_hosts2`), aparecerá el siguiente mensaje:

```
The authenticity of host 'servidor.midominio (192.168.0.1)' can't be established.
RSA key fingerprint is d7:7d:c9:b4:09:56:50:bd:9e:78:b8:93:8c:8d:ed:5c.
Are you sure you want to continue connecting (yes/no)?
```

Si el usuario confía en que esa es la verdadera clave pública de `servidor.midominio` deberá tipear `yes`. Luego el cliente informará:

```
Warning: Permanently added 'servidor.midominio,192.168.0.1' (RSA) to the list
of known hosts.
```

[10]Por favor, no tome esta configuración como la recomendada para SSH. Existen varias otras opciones importantes que omito aquí por razones de simplicidad.

Lo que significa que se ha agregado la clave pública de `servidor.midominio` en `~/.ssh/known_hosts`. Luego el cliente solicitará el ingreso de la contraseña:

```
jsmaldone@servidor.midominio's password:
```

Finalmente, si la contraseña ingresada es correcta, aparecerá el mensaje de bienvenida del servidor:

```
Bienvenido a servidor.midominio (Debian GNU/Linux)
jsmaldone@servidor:~$
```

Con lo cual hemos iniciado una sesión en `servidor.midominio` como el usuario `jsmaldone`.

Al finalizar, tipeando `exit` (o presionando `CTRL+D`), se cerrará la sesión remota:

```
jsmaldone@servidor:~$ exit
logout
Connection to servidor.midominio closed.
javier@cliente:~>
```

Si al intentar una nueva conexión a servidor.midominio el cliente recibe una clave pública diferente a la que tiene almacenada en known_hosts2, nos aparecerá el siguiente mensaje de advertencia y se abortará la conexión:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@     WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!    @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
d7:7d:c9:b4:09:56:50:bd:9e:78:b8:93:8c:8d:ed:5c.
Please contact your system administrator.
Add correct host key in /home/javier/.ssh/known_hosts2 to get rid of this message.
Offending key in /home/javier/.ssh/known_hosts2:1
RSA host key for servidor.midominio has changed and you have requested strict
checking.
```

3.2. Iniciando una sesión remota con clave pública

3.2.1. Generando las claves

El primer paso para utilizar la autenticación mediante clave pública es generar el par de claves de RSA que se utilizarán. Para ello, el usuario `javier` en cliente.midominio ejecutará el siguiente comando:

```
javier@cliente:~>ssh-keygen -t rsa
```

El programa `ssh-keygen` permite generar y administrar claves para SSH. El comando ingresado sirve para crear un par de claves pública/privada de RSA (con una longitud por defecto de 1024 bits). El programa responderá con lo siguiente:

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/javier/.ssh/id_rsa):
```

Nos solicita que ingresemos el nombre del archivo en donde se almacenará la clave privada, proponiéndonos `/home/javier/.ssh/id_rsa`, lo cual concuerda con la configuración del cliente SSH. Presionando `Enter` se acepta el valor por defecto. Luego nos solicitará una frase clave:

```
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
```

Presionando dos veces `Enter` omitimos el uso de una frase clave. Más adelante nos ocuparemos de esto. Finalmente nos informa:

```
Your identification has been saved in /home/javier/.ssh/id_rsa.
Your public key has been saved in /home/javier/.ssh/id_rsa.pub.
The key fingerprint is:
13:8b:23:74:53:e4:0f:b3:16:49:1b:79:64:60:7c:38 javier@cliente
javier@cliente:~>
```

3.2.2. Transfiriendo la clave pública al servidor

Luego, debemos transferir la clave pública del usuario `javier` (`~/.ssh/id_rsa.pub`) al directorio *home* del usuario `jsmaldone` en `servidor.midominio` y añadirla al final del archivo `~/.ssh/authorized_keys`. Para ello, el usuario `jsmaldone` podrá tipear:

```
jsmaldone@servidor:~>cat id_rsa.pub >>.ssh/authorized_keys
```

3.2.3. Iniciando la sesión

El usuario `javier` en cliente.midominio nuevamente ingresa el comando:

```
javier@cliente:~>ssh jsmaldone@servidor.midominio
```

El servidor nuevamente envía su clave pública de RSA, la cual es comparada con la almacenada en `known_hosts2`, y si coincide el proceso continúa.

El cliente de SSH, al encontrar el archivo `~/.ssh/id_rsa`, primero intentará la autenticación con clave pública. El servidor le enviará el *challenge* cifrado con la clave pública encontrada en `~/.ssh/authorized_keys` (en el directorio *home* del usuario `jsmaldone`) y el cliente deberá devolverla descifrada (usando la clave `~/.ssh/id_rsa` en el directorio *home* del usuario `javier`).

Si esto se realiza correctamente, se iniciará la sesión remota[11]:

```
Bienvenido a servidor.midominio (Debian GNU/Linux)
jsmaldone@servidor:~$
```

Si la autenticación con clave pública hubiera fallado, el cliente intentará con la autenticación con contraseña descripta en la sección anterior.

3.2.4. Asegurando la clave privada

Tal como lo hemos hecho en el paso anterior, cualquier persona que tenga acceso a la clave privada del usuario `javier` en `cliente.local` podrá conectarse como `jsmaldone` en `servidor.local`. Esto es aún más grave si tenemos en cuenta que el mismo par de claves podrían ser utilizadas para acceder a varios otros sistemas. Esto impone la necesidad de utilizar algún otro mecanismo para proteger la clave privada.

Para ello, tenemos la posibilidad de cifrar la clave privada utilizando un algoritmo de cifrado simétrico y una contraseña (comúnmente llamada "frase clave" o *passphrase)*. El algoritmo utilizado a tal efecto por SSH se denomina 3DES.

En el ejemplo anterior, cuando creamos el par de claves usando `ssh-keygen`, omitimos especificar la frase clave que se usaría a tal efecto. Usando nuevamente `ssh-keygen` podemos asignar una nueva:

```
javier@cliente:~/.ssh>ssh-keygen -p -f /home/javier/.ssh/id_rsa
```

Si ya estamos usando una frase clave, nos la solicitará, y luego nos pedirá ingresar la nueva

[11] Nótese que esto se realiza automáticamente, sin la intervención del usuario.

```
Enter new passphrase (empty for no passphrase):
Enter same passphrase again:
javier@cliente:~/.ssh>
```

Una vez finalizado, la clave privada se encuentra cifrada con 3DES, razón por la cual cada vez que necesitemos usarla (por ejemplo, en cualquiera de los ejemplos anteriores al intentar conectarnos con el servidor), se nos solicitará la frase clave para descifrarla. Por ejemplo:

```
javier@cliente:~/.ssh>ssh jsmaldone@servidor.midominio
Need passphrase for /home/javier/.ssh/id_rsa
Enter passphrase for /home/javier/.ssh/id_rsa
```

El procedimiento continuará como en los casos anteriores.

Importante: Es altamente recomendable proteger la clave privada con una frase clave. Se recomienda que la misma tenga una longitud de más de 10 caracteres, que contenga letras mayúsculas, letras minúsculas y números, y que no sea fácilmente deducible por otras personas. Recuerde lo dicho en la introducción: la elección de una contraseña "débil" o simple echará por tierra las bondades de SSH, RSA y 3DES reduciendo el grado de seguridad a 0.

3.2.5. ssh-agent

Como acabamos de ver, la utilización de una frase clave implicará que al intentar conectar con el servidor, el cliente la solicitará al usuario, debiendo este tipearla. Hemos perdido una de las grandes ventajas (desde los ojos del usuario inexperto, puede parecer que estamos como al principio).

Para evitar esta molestia, y también para reducir el riesgo de que alguien averigüe nuestra frase clave al tener que tipearla 10 veces por hora[12], se utiliza el `ssh-agent`.

Este programa se ejecuta como un demonio (*daemon*) y permite almacenar las claves privadas (descifradas) para su posterior uso por el cliente de SSH.

3.2.6. Usando ssh-agent en el shell

En una sesión del shell, podemos ejecutar el `ssh-agent` de la siguiente forma:

```
javier@cliente:~>eval `ssh-agent`
Agent pid 3156
javier@cliente:~>
```

Luego debemos agregar nuestra clave privada de RSA. Para ello usamos el comando `ssh-add`:

```
javier@cliente:~>ssh-add .ssh/id_rsa
Need passphrase for .ssh/id_rsa
Enter passphrase for .ssh/id_rsa
Identity added: .ssh/id_rsa (.ssh/id_rsa)
javier@cliente:~>
```

Este procedimiento puede repetirse si tenemos varias claves privadas.

Luego, siempre y cuando nos encontremos en el mismo shell en donde hemos iniciado el `ssh-agent`, al ejecutar `ssh` (o `scp`, como veremos más adelante) este le solicitará al `ssh-agent` nuestra clave privada, con lo cual no deberemos introducir la frase clave.

3.2.7. Usando ssh-agent en X-Window

Completar....

[12] Mediante el ataque ingenioso ataque conocido como "*shoulder surfing*". ;)

3.3. Ejecución de comandos remotos

Completar...

3.4. Transfiriendo archivos con Secure Copy

Completar...

3.5. Aplicaciones X11 a través de SSH

Completar...

3.6. Túneles IP cifrados con SSH

Completar...

4. Acerca de este documento

Este documento fue escrito utilizando LyX, TeX y LaTeX. Los gráficos y diagramas fueron realizados utilizando *OpenOffice*, *Dia* **y** *Gimp*.

La última versión puede encontrarse en:
`http://www.smaldone.com.ar/documentos/misdocumentos.shtml`

4.1. Copyright

A. El algoritmo RSA (Rivest-Shamir-Adleman)

El algoritmo RSA fue inventado en 1977 por Ron Rivest, Adi Shamir y Leonard Adleman en el *Massachusetts Institute of Technology* (MIT) *Laboratory*.

A.1. Descripción del algoritmo

Es realmente muy simple, como puede verse a continuación (¡utiliza solamente aritmética modular!):

1. Encuentre dos números primos muy grandes P y Q (por ejemplo, cuya representación binaria sea de 1024 bits).[13]
2. Elija un número E, tal que:
    a) $E > 1$,
    b) $E < P \cdot Q$,
    c) E y $(P-1)\cdot(Q-1)$ sean co-primos (no tengan factores primos distintos de 1 en común o, lo que es equivalente, su máximo común divisor sea 1)[14].
3. Calcule D tal que $(D \cdot E - 1)$ sea divisible por $(P-1)\cdot(Q-1)$. Para ello basta encontrar un entero X que haga que $\frac{X\cdot(P-1)\cdot(Q-1)+1}{E}$ sea entero, siendo esto último el valor de D.
4. La función de cifrado es $C = (M^E) \bmod P \cdot Q$, donde C es el mensaje cifrado (un entero positivo) y M es el mensaje original (M debe ser menor que $P \cdot Q$).[15]
5. La función de descifrado es $M = (C^D) \bmod P \cdot Q$.

La clave pública es el par $(P \cdot Q, E)$. La clave privada es el número D.

La clave pública puede difundirse libremente, ya que no se conoce una forma simple de calcular D, P o Q dados solamente $P \cdot Q$ y E.

Para calcular D es necesario calcular P y Q. Dado que sólo conocemos $P \cdot Q$, y gracias al teorema fundamental de la aritmética[16], podemos factorizar este número para descubrir P y Q. El inconveniente radica en que la forma más eficiente de factorizar un número n que se conoce en la actualidad[17] es $O(2^n)$. Esto significa que toma una cantidad de tiempo exponencial respecto del tamaño del número a factorizar. Si P y Q son dos enteros cuya representación binaria ocupa 1024 bits, factorizar su producto puede llevarle a cualquier computadora varios milenios, como mínimo (esto sin tener en cuenta el espacio necesario para realizar los cálculos).

A.2. Un ejemplo

Sean $P = 61$, $Q = 53$ (ambos primos)[18]. Resulta $P \cdot Q = 3233$ y $(P-1)\cdot(Q-1) = 3120$. Tenemos que $3120 = 13 \cdot 5 \cdot 3 \cdot 2^4$, por lo cual podemos tomar a $E = 17$.

Debemos encontrar un valor X tal que $\frac{X\cdot 3120+1}{17}$ sea entero. Con $X = 15$, el cociente anterior es igual a 2753. Sea pues $D = 2753$.

La clave pública es $(17, 3233)$ y la clave privada 2753.

Definimos las siguientes funciones: $cifrar(M) = M^{17} \bmod 3233$ y $descifrar(C) = C^{2753} \bmod 3233$.

Luego, para cifrar el mensaje 123, calculamos $cifrar(123) = 123^{17} \bmod 3233 = 855$.

Para descifrar 855, calculamos $descifrar(855) = 855^{2753} \bmod 3233 = 123$.

---

[13] Un número es primo si sólo es divisible por sí mismo y por 1.

[14] E no tiene que ser un número primo, pero debe ser impar. $(P-1)\cdot(Q-1)$ no puede ser primo porque es un número par.

[15] $a \bmod b$ es el resto de la división entera entre a y b (por ejemplo, $5 \bmod 3 = 2$).

[16] Todo número entero positivo admite una única descomposición en factores primos.

[17] Nadie ha demostrado todavía que no pueda hacerse de manera más eficiente.

[18] Tomamos dos números primos pequeños para simplificar, en la medida de lo posible, los cálculos.