Instalación y Configuración de ssh para Cómputo Intercluster

Fernando G. Tinetti* Walter Aróztegui

III-LIDI, Facultad de Informática, UNLP
50 y 115, 1900, La Plata
Argentina

CeTAD, Facultad de Ingeniería, UNLP
48 y 116, 1900, La Plata
Argentina

Reporte Técnico PLA-002-2005[1]
Junio 2005

Resumen

Este reporte está orientado básicamente a documentar la instalación del servicio ssh en el contexto de cómputo paralelo intercluster. La forma de ejecutar programas paralelos en clusters utilizando implementaciones de MPI está relativamente estable utilizando el servicio de rsh estándar y calificado como muy inseguro también. Es de esperar que, en el contexto de cómputo intercluster, cada cluster esté "protegido" por al menos un firewall, donde se filtren la mayoría de los servicios, en particular rsh y sus asociados. La idea entonces es proveer/usar ssh como alternativa válida y más segura, con el requerimiento extra sobre los firewalls de no cancelar este servicio hacia/desde el exterior.

1. Introducción

En un entorno estándar de cómputo paralelo en un cluster, las herramientas también estándares son la utilización de alguna biblioteca de pasaje de mensajes tal como una implementación de MPI junto con los servicios básicos de disparo de procesos remotos: rsh. De hecho, cualquier instalación estándar de las implementaciones de MPI tales como MPICH y LAM/MPI utilizan el comando **mpirun** como herramienta para la ejecución de programas paralelos SPMD (Single program, Multiple Data) la cual, a su vez, utiliza el servicio rsh para el disparo de comandos en otras máquinas (o *máquinas remotas*). En un cluster utilizado para cómputo paralelo se asume que no hay problemas *internos* de seguridad, sino que los problemas de seguridad (si existen) provienen del exterior, en el caso de que el cluster esté conectado a Internet, por ejemplo.

Los sistemas de seguridad actuales consisten en la restricción casi sin discusión de todos o la mayoría de los servicios normalmente usados para interconexión de máquinas. Y **rsh** y sus *asociados* **rlogin** y **rcp** están entre los primeros que se cancelan como servicios al exterior de las redes locales y/o clusters. También como parte del contexto actual de cómputo en clusters es cierto que los servicios llamados "seguros" son los que tienden a ser usados/permitidos, tales como los provistos por **ssh** y sus asociados **scp** y **sftp**.

Aunque a *priori* este tipo de consideraciones (servicios, seguridad, etc.) no es parte del desarrollo de software paralelo ni de software paralelo en clusters, aparece como un problema agregado cuando se piensa en cómputo paralelo interclusters. Cómputo paralelo interclusters puede pensarse como el *paso siguiente* al cómputo paralelo en clusters tal como se viene desarrollando desde hace algunos años. Dado que es relativamente *natural* hoy pensar en distribuir el cómputo a realizar entre múltiples máquinas de un cluster o red local (LAN: Local Area Network), también es posible pensar en distribuir el cómputo a realizar entre múltiples clusters disponibles. Si bien la idea es sencilla, la utilidad práctica todavía está siendo definida en términos de parámetros y/o requerimientos de las aplicaciones y del hardware, básicamente de interconexión. Entre las tareas/problemas que se agregan aparecen, justamente, los de seguridad y de interconexión física de clusters, que en clusters aislados no aparecen. Es por eso que, para comenzar la experimentación y/o utilización de más de un cluster para cómputo paralelo es necesario resolver los problemas mínimos de interconexión de clusters.

La alternativa de solución actual la constituyen las herramientas y el software desarrollados en el contexto de "grid". Sin embargo, al menos inicialmente se descartan las herramientas de grid por varias razones. La razón más importante la constituyen los requerimientos actuales para el funcionamiento de

*Investigador Asistente CICPBA
[1]PLA: sigla de Parallel Linear Algebra

hardware/software en grid, que son extremadamente altos en términos de interconexión de las computadoras que se utilizarán. Por otro lado, adoptar el contexto de grid todavía supone una restricción sobre la paralelización de aplicaciones, que ya cuenta con sus propias restricciones en clusters y con restricciones no totalmente definidas (ni cuanlitativa ni cuantitativamente) en el contexto de cómputo paralelo intercluster.

Por lo tanto, se documentará en este reporte técnico la idea básica de instalación del "servicio" **ssh** y sus asociados, más algunas ideas pensando en la interacción de este servicio con los firewalls, herramientas usadas extensivamente para la seguridad de las redes locales y los clusters dedicados a cómputo paralelo en particular interconectados a Internet.

2. Consideraciones Generales

Una vez instalado los servidores y los clientes del servicio ssh, el comando **ssh** funciona de la misma manera que telnet o rsh/rlogin pero encriptando la información. Se conecta a otro host con:

```
ssh hostname.domain
```

o si se usa otro nombre de usuario sobre el host remoto, con la opción -l se especifica este nombre de usuario:

```
ssh hostname.domain -l otronombredeusuario
```

La primera vez que se conecta a un host, **ssh** reconoce que nunca se definió la clave pública de (o, *la conectividad con*) este otro host y avisa que éste aún no es conocido, con el mensaje (si en la configuración del cliente figura StrictHostKeyChecking ask):

```
Host key not found from the list of know hosts.
Are you sure you want to continue connecting (yes/no)?
```

ssh espera en este caso la respuesta de tres letras yes, en caso de contestar no, n o aún y, la conexión es terminada. Todo lo relacionado con la utilización del servicio **ssh** por parte de un usuario, quedará en el directorio $HOME/.ssh, donde en particular quedará registrado que se ha definido un host remoto y se almacena la clave usada a partir de ahora para ese host remoto. A partir de este momento, la idea es, en cierto modo, *mantener* la conectividad o el servicio como con **rsh**, es decir que se puedan disparar procesos remotos y/o hacer login remoto sin la necesidad de ingresar *passwords* interactivos o aún de enviarlos de alguna manera. No es inmediato, pero se pueden configurar clientes y servidores ssh de forma tal que esto sea relativamente *transparente* o similar a la utilización bien conocida de **rsh** y sus asociados. Es decir que hay un lapso de instalación y configuración relativamente interactivo y máquina-por-máquina a partir del cual (cumplimentado satisfactoriamente) se puede continuar con la utilización de **ssh** y *sus* comandos relacionados como con **rsh** y *sus* comandos relacionados.

3. Resumen de Instalación y Configuración

La idea de esta sección es que tenga una lista de pasos a realizar para la instalación y configuración del servicio ssh como para que se pueda continuar utilizando sin la necesidad de ingresar passwords cada vez que se accede a una computadora remota (o incluso la computadora local, pero usando el servicio ssh).

3.1. Fuentes del Software Asociado a ssh

Todo el software puede obtenerse para compilar de las ssh home page o de OpenSSH home page, pero se incluye en las distribuciones más recientes de la mayoría de las distribuciones de Linux (Red Hat, Slackware, Debian, etc.). En el caso de Red Hat, a partir de 7.0 se incluyen los paquetes:

- openssh: contiene los ficheros básicos necesarios para el servidor y clientes ssh.
- openssh-server: contiene el demonio sshd, que permite a los clientes ssh establecer las conexiones seguras con el sistema.

- openssh-clients: contiene los clientes que permiten establecer conexiones con el servidor.

Previamente a éstos deben estar instalados los paquetes de encriptación openSSL. Pueden funcionar aún en distribuciones de RedHat 6.x pero sin la capacidad de compresión como poseen versiones más avanzadas.

Resumiendo para el caso de RedHat: a partir de la distribución 7.0 no solamente se tiene todo el software necesario sino que las instalaciones estándares dejan sin instalar solamente el servidor de `ssh`, es decir que todo lo demás ya está instalado y disponible.

3.2. Pasos de instalación y Configuración

Como se afirma antes, lo único que resta instalar a partir de una instalación estándar de RedHat es el servidor de `ssh`, el `sshd`. En el caso específico de RedHat, no es más que la instalación del paquete que lo contiene, openssh-server, que en el caso de la distribución 7.2, por ejemplo, es el

```
openssh-server-2.9p2-7.i386.rpm
```

En caso de que haga/n falta otro/s paquete/s, el comando `rpm` se encargará de aclararlo y mantendrá la consistencia. No es el caso de la instalación usando RedHat 7.2, así que simplemente se instala el paquete con `rpm -i ...`, de manera estándar y sin problemas agregados. A partir de este momento, el servicio ssh está disponible en la máquina en la cual se instaló a partir de su activación, que es automática en caso de reiniciar la computadora posteriormente a su instalación. Para evitar el reinicio, suele ser conveniente usar imediatamente después de la instalación del servidor

```
service ssh start
```

A partir de este momento, es decir con el servidor instalado y en funcionamiento en una máquina, ya es posible usar el servicio de ssh vía el cliente de ssh, usando el comando `ssh`. Si se utiliza el comando `ssh` tal como se indica en la sección anterior, *registra* el acceso con el servicio ssh a un servidor remoto por única vez y por lo tanto a partir de este momento solamente resta definir y mantener la conectividad del usuario con el host. El host pasa a ser reconocido *automáticamente* a partir de lo que se almacena en el directorio $HOME/.ssh en el primer uso del comando `ssh`.

Habiendo agregado el servidor de ssh a la instalación estándar y habiendo *registrado* un host remoto, lo que resta es la configuración de las máquinas para que se pueda usar el comando `ssh` para disparar otros comandos y/o hacer login remoto sin ingresar passwords. Los pasos a seguir son:

```
ssh-keygen -t rsa
```

y todos los requerimientos interactivos de este comando se responden simplemente con "enter". La idea en este caso es que se genera una clave con la cual se reconocerá el usuario cada vez que se intente conectar usando el comando `ssh` (y el servidor `sshd`, por supuesto). El comando `ssh-keygen -t rsa`, no hace más que generar la clave de identificación del usuario, que debe estar disponible en todas las computadoras a las que se quiera acceder vía `ssh`. Esto se logra, en la máquina local con

```
cd $HOME/.ssh
cp id_rsa.pub authorized_keys2
```

y, de hecho, se debe lograr que el contenido de `id_rsa.pub` aparezca en $HOME/.ssh/authorized_keys2 de todos los hosts a los cuales se quiere acceder vía `ssh`. Es claro que sin contar con `rsh` ni `rcp` ni estos comandos, el contenido debe ponerse, en cierta forma, manualmente, pero también es claro este procedimiento se debe llevar a cabo por única vez, en lo que se denominará la *configuración* del servicio para el cómputo interclusters. A partir de ahora, teniendo la computadora y el usuario reconocidos, al ejecutar

```
ssh hostname.domain
```

no es necesario ingresar password, se tiene automáticamente un login remoto, donde toda la información que se transfiere entre las computadoras estará encriptada. Pero se debe recordar que la idea de la encriptación no es lo importante en cómputo paralelo intercluster, dado que la mayoría de la información que se transfiere entre las computadoras no será usando el servicio ssh sino la biblioteca de pasaje de mensajes, que tiene poco o nada que ver con el servicio ssh con la excepción del disparo mismo de todos los procesos que componen un programa paralelo. Esto se confirma en el propio esquema de las implementaciones de MPI, que utilizan **ssh** (o **rsh**) al ejecutar el comando **mpirun** y a partir de allí construyen su propia capa de interconexión entre tareas/procesos del programa paralelo, casi independientemente de **mpirun** y de **ssh**.

4. Información Adicional

En el caso de la distribución de Linux RedHat, a partir de la distribución 7.0 se incluyen los paquetes:

- openssh: contiene los ficheros básicos necesarios para el servidor y clientes ssh.
- openssh-server: contiene el poceso sshd, que permite a los clientes ssh establecer las conexiones seguras con el sistema.
- openssh-clients: contiene el cliente que permite establecer conexiones con el servidor.

Previamente a éstos deben estar instalados los paquetes de encriptación openSSL. Pueden funcionar aún en distribuciones de RedHat 6.x pero sin la capacidad de compresión como poseen versiones más avanzadas.

4.1. Servidor ssh, sshd

El proceso sshd es el programa que espera conexiones de red de los clientes ssh, controla la autenticación y ejecuta el comando requerido. El puerto estándar sobre el que espera es el 22 aunque puede ser cambiado, y su fichero de configuración es /etc/ssh/sshd_config. En esta configuración se indica la ruta en la que encontrar las claves que identifican al servidor. Estas son la base de la autenticación mediante clave pública y los valores estándares son:

- /etc/ssh/ssh_host_key
- /etc/ssh/ssh_host_rsa_key
- /etc/ssh/ssh_host_dsa_key

Las opciones de configuración que figuran en el manual (algunas pueden cambiar entre versiones distintas) son, por orden alfabético:

AFSTokenPassing Specifies whether an AFS token may be forwarded to the server. Default is “no”.
AllowGroups This keyword can be followed by a list of group name patterns, separated by spaces. If specified, login is allowed only for users whose primary group or supplementary group list matches one of the patterns. '*ánd "? can be used as wildcards in the patterns. Only group names are valid; a numerical group ID is not recognized. By default, login is allowed for all groups.
AllowTcpForwarding Specifies whether TCP forwarding is permitted. The default is “yes”. Note that disabling TCP forwarding does not improve security unless users are also denied shell access, as they can always install their own forwarders.
AllowUsers This keyword can be followed by a list of user name patterns, separated by spaces. If specified, login is allowed only for users names that match one of the patterns. '*ánd "? can be used as wildcards in the patterns. Only user names are valid; a numerical user ID is not recognized. By default, login is allowed for all users. If the pattern takes the form USER@HOST then USER and HOST are separately checked, restricting logins to particular users from particular hosts.
AuthorizedKeysFile Specifies the file that contains the public keys that can be used for user authentication. AuthorizedKeysFile may contain tokens of the form connection set-up. The following tokens are defined:

%% is replaced by a literal '%',

%h is replaced by the home directory of the user being authenticated and

%u is replaced by the username of that user.

After expansion, AuthorizedKeysFile is taken to be an absolute path or one relative to the user's home directory. The default is “.ssh/authorized_keys”.

Banner In some jurisdictions, sending a warning message before authentication may be relevant for getting legal protection. The contents of the specified file are sent to the remote user before authentication is allowed. This option is only available for protocol version 2. By default, no banner is displayed.
ChallengeResponseAuthentication Specifies whether challenge response authentication is allowed. All authentication styles from login.conf(5) are supported. The default is "yes".
Ciphers Specifies the ciphers allowed for protocol version 2. Multiple ciphers must be comma-separated. The default is "aes128-cbc,3des-cbc,blowfish-cbc,cast128-cbc,arcfour,aes192-cbc,aes256-cbc"
ClientAliveInterval Sets a timeout interval in seconds after which if no data has been received from the client, sshd will send a message through the encrypted channel to request a response from the client. The default is 0, indicating that these messages will not be sent to the client. This option applies to protocol version 2 only.
ClientAliveCountMax Sets the number of client alive messages (see above) which may be sent without sshd receiving any messages back from the client. If this threshold is reached while client alive messages are being sent, sshd will disconnect the client, terminating the session. It is important to note that the use of client alive messages is very different from KeepAlive (below). The client alive messages are sent through the encrypted channel and therefore will not be spoofable. The TCP keepalive option enabled by KeepAlive is spoofable. The client alive mechanism is valuable when the client or server depend on knowing when a connection has become inactive. The default value is 3. If ClientAliveInterval (above) is set to 15, and ClientAliveCountMax is left at the default, unresponsive ssh clients will be disconnected after approximately 45 seconds.
Compression Specifies whether compression is allowed. The argument must be "yes" or "no". The default is "yes".
DenyGroups This keyword can be followed by a list of group name patterns, separated by spaces. Login is disallowed for users whose primary group or supplementary group list matches one of the patterns. '*ánd "? can be used as wildcards in the patterns. Only group names are valid; a numerical group ID is not recognized. By default, login is allowed for all groups.
DenyUsers This keyword can be followed by a list of user name patterns, separated by spaces. Login is disallowed for user names that match one of the patterns. '*ánd "? can be used as wildcards in the patterns. Only user names are valid; a numerical user ID is not recognized. By default, login is allowed for all users. If the pattern takes the form USER@HOST then USER and HOST are separately checked, restricting logins to particular users from particular hosts.
GatewayPorts Specifies whether remote hosts are allowed to connect to ports forwarded for the client. By default, sshd binds remote port forwardings to the loopback address. This prevents other remote hosts from connecting to forwarded ports. GatewayPorts can be used to specify that sshd should bind remote port forwardings to the wildcard address, thus allowing remote hosts to connect to forwarded ports. The argument must be "yes" or "no". The default is "no".
HostbasedAuthentication Specifies whether rhosts or /etc/hosts.equiv authentication together with successful public key client host authentication is allowed (hostbased authentication). This option is similar to RhostsRSAAuthentication and applies to protocol version 2 only. The default is "no".
HostKey Specifies a file containing a private host key used by SSH. The default is /etc/ssh/ssh_host_key for protocol version 1, and /etc/ssh/ssh_host_rsa_key and /etc/ssh/ssh_host_dsa_key for protocol version 2. Note that sshd will refuse to use a file if it is group/world-accessible. It is possible to have multiple host key files. "rsa1" keys are used for version 1 and "dsa" or "rsa" are used for version 2 of the SSH protocol.
IgnoreRhosts Specifies that .rhosts and .shosts files will not be used in RhostsAuthentication, RhostsRSAAuthentication or HostbasedAuthentication. /etc/hosts.equiv and /etc/ssh/shosts.equiv are still used. The default is "yes".
IgnoreUserKnownHosts Specifies whether sshd should ignore the user's $HOME/.ssh/known_hosts during RhostsRSAAuthentication or HostbasedAuthentication. The default is "no".
KeepAlive Specifies whether the system should send TCP keepalive messages to the other side. If they are sent, death of the connection or crash of one of the machines will be properly noticed. However, this means that connections will die if the route is down temporarily, and some people find it annoying. On the other hand, if keepalives are not sent, sessions may hang indefinitely on the server, leaving "ghost" users and consuming server resources. The default is "yes" (to send keepalives), and the server will notice if the network goes down or the client host crashes. This avoids infinitely hanging sessions. To disable keepalives, the value should be set to "no".
KerberosAuthentication Specifies whether Kerberos authentication is allowed. This can be in the form of a Kerberos ticket, or if PasswordAuthentication is yes, the password provided by the user will

be validated through the Kerberos KDC. To use this option, the server needs a Kerberos servtab which allows the verification of the KDC's identity. Default is "no".

KerberosOrLocalPasswd If set then if password authentication through Kerberos fails then the password will be validated via any additional local mechanism such as /etc/passwd. Default is "yes".

KerberosTgtPassing Specifies whether a Kerberos TGT may be forwarded to the server. Default is "no", as this only works when the Kerberos KDC is actually an AFS kaserver.

KerberosTicketCleanup Specifies whether to automatically destroy the user's ticket cache file on logout. Default is "yes".

KeyRegenerationInterval In protocol version 1, the ephemeral server key is automatically regenerated after this many seconds (if it has been used). The purpose of regeneration is to prevent decrypting captured sessions by later breaking into the machine and stealing the keys. The key is never stored anywhere. If the value is 0, the key is never regenerated. The default is 3600 (seconds).

ListenAddress Specifies the local addresses sshd should listen on. The following forms may be used:
ListenAddress [host|IPv4_addr|IPv6_addr ListenAddress host|IPv4_addr]:port
ListenAddress [host|IPv6_addr]:port.

If port is not specified, sshd will listen on the address and all prior Port options specified. The default is to listen on all local addresses. Multiple ListenAddress options are permitted. Additionally, any Port options must precede this option for non port qualified addresses.

LoginGraceTime The server disconnects after this time if the user has not successfully logged in. If the value is 0, there is no time limit. The default is 120 seconds.

LogLevel Gives the verbosity level that is used when logging messages from sshd. The possible values are: QUIET, FATAL, ERROR, INFO, VERBOSE, DEBUG, DEBUG1, DEBUG2 and DEBUG3. The default is INFO. DEBUG and DEBUG1 are equivalent. DEBUG2 and DEBUG3 each specify higher levels of debugging output. Logging with a DEBUG level violates the privacy of users and is not recommended.

MACs Specifies the available MAC (message authentication code) algorithms. The MAC algorithm is used in protocol version 2 for data integrity protection. Multiple algorithms must be comma-separated. The default is "hmac-md5,hmac-sha1,hmac-ripemd160,hmac-sha1-96,hmac-md5-96".

MaxStartups Specifies the maximum number of concurrent unauthenticated connections to the sshd daemon. Additional connections will be dropped until authentication succeeds or the LoginGraceTime expires for a connection. The default is 10. Alternatively, random early drop can be enabled by specifying the three colon separated values "start:rate:full" (e.g., "10:30:60"). sshd will refuse connection attempts with a probability of "rate/100" (30%) if there are currently "start" (10) unauthenticated connections. The probability increases linearly and all connection attempts are refused if the number of unauthenticated connections reaches "full" (60).

PAMAuthenticationViaKbdInt Specifies whether PAM challenge response authentication is allowed. This allows the use of most PAM challenge response authentication modules, but it will allow password authentication regardless of whether PasswordAuthentication is enabled.

PasswordAuthentication Specifies whether password authentication is allowed. The default is "yes".

PermitEmptyPasswords When password authentication is allowed, it specifies whether the server allows login to accounts with empty password strings. The default is "no".

PermitRootLogin Specifies whether root can login using ssh(1). The argument must be "yes", "without-password", "forced-commands-only" or "no". The default is "yes". If this option is set to "without-password" password authentication is disabled for root. If this option is set to "forced-commands-only" root login with public key authentication will be allowed, but only if the command option has been specified (which may be useful for taking remote backups even if root login is normally not allowed). All other authentication methods are disabled for root. If this option is set to "no" root is not allowed to login.

PermitUserEnvironment Specifies whether ~/.ssh/environment and environment = options in ~/.ssh/ authorized_keys are processed by sshd. The default is "no". Enabling environment processing may enable users to bypass access restrictions in some configurations using mechanisms such as LD_PRELOAD.

PidFile Specifies the file that contains the process ID of the sshd daemon. The default is /var/run/ sshd.pid.

Port Specifies the port number that sshd listens on. The default is 22. Multiple options of this type are permitted. See also ListenAddress. PrintLastLog Specifies whether sshd should print the date and time when the user last logged in. The default is "yes".

PrintMotd Specifies whether sshd should print /etc/motd when a user logs in interactively. (On some systems it is also printed by the shell, /etc/profile, or equivalent.) The default is "yes".

Protocol Specifies the protocol versions sshd supports. The possible values are "1" and "2". Multiple versions must be comma-separated. The default is "2,1". Note that the order of the protocol list does not indicate preference, because the client selects among multiple protocol versions offered by the server. Specifying "2,1" is identical to "1,2".

PubkeyAuthentication Specifies whether public key authentication is allowed. The default is "yes". Note that this option applies to protocol version 2 only.

RhostsAuthentication Specifies whether authentication using rhosts or /etc/hosts.equiv files is sufficient. Normally, this method should not be permitted because it is insecure.

RhostsRSAAuthentication should be used instead, because it performs RSA-based host authentication in addition to normal rhosts or /etc/hosts.equiv authentication. The default is "no". This option applies to protocol version 1 only.

RhostsRSAAuthentication Specifies whether rhosts or /etc/hosts.equiv authentication together with successful RSA host authentication is allowed. The default is "no". This option applies to protocol version 1 only.

RSAAuthentication Specifies whether pure RSA authentication is allowed. The default is "yes". This option applies to protocol version 1 only.

ServerKeyBits Defines the number of bits in the ephemeral protocol version 1 server key. The minimum value is 512, and the default is 768.

StrictModes Specifies whether sshd should check file modes and ownership of the user's files and home directory before accepting login. This is normally desirable because novices sometimes accidentally leave their directory or files world-writable. The default is "yes".

Subsystem Configures an external subsystem (e.g., file transfer daemon). Arguments should be a subsystem name and a command to execute upon subsystem request. The command sftp-server(8) implements the "sftp" file transfer subsystem. By default no subsystems are defined. Note that this option applies to protocol version 2 only.

SyslogFacility Gives the facility code that is used when logging messages from sshd. The possible values are: DAEMON, USER, AUTH, LOCAL0, LOCAL1, LOCAL2, LOCAL3, LOCAL4, LOCAL5, LOCAL6, LOCAL7. The default is AUTH.

UseLogin Specifies whether login(1) is used for interactive login sessions. The default is "no". Note that login(1) is never used for remote command execution. Note also, that if this is enabled, X11Forwarding will be disabled because login(1) does not know how to handle xauth(1) cookies. If UsePrivilegeSeparation is specified, it will be disabled after authentication.

UsePrivilegeSeparation Specifies whether sshd separates privileges by creating an unprivileged child process to deal with incoming network traffic. After successful authentication, another process will be created that has the privilege of the authenticated user. The goal of privilege separation is to prevent privilege escalation by containing any corruption within the unprivileged processes. The default is "yes".

VerifyReverseMapping Specifies whether sshd should try to verify the remote host name and check that the resolved host name for the remote IP address maps back to the very same IP address. The default is "no".

X11DisplayOffset Specifies the first display number available for sshd's X11 forwarding. This prevents sshd from interfering with real X11 servers. The default is 10.

X11Forwarding Specifies whether X11 forwarding is permitted. The argument must be "yes" or "no". The default is "no". When X11 forwarding is enabled, there may be additional exposure to the server and to client displays if the sshd proxy display is configured to listen on the wildcard address (see X11UseLocalhost below), however this is not the default. Additionally, the authentication spoofing and authentication data verification and substitution occur on the client side. The security risk of using X11 forwarding is that the client's X11 display server may be exposed to attack when the ssh client requests forwarding (see the warnings for ForwardX11 in ssh_config(5)). A system administrator may have a stance in which they want to protect clients that may expose themselves to attack by unwittingly requesting X11 forwarding, which can warrant a "no" setting. Note that disabling X11 forwarding does not prevent users from forwarding X11 traffic, as users can always install their own forwarders. X11 forwarding is automatically disabled if UseLogin is enabled.

X11UseLocalhost Specifies whether sshd should bind the X11 forwarding server to the loopback address or to the wildcard address. By default, sshd binds the forwarding server to the loopback address and sets the hostname part of the DISPLAY environment variable to "localhost". This prevents remote hosts from connecting to the proxy display. However, some older X11 clients may not function with this configuration. X11UseLocalhost may be set to "no" to specify that the forwarding server should be bound to the wildcard address. The argument must be "yes" or "no". The default is "yes".

XAuthLocation Specifies the full pathname of the xauth(1) program. The default is /usr/X11R6/bin/ xauth.

4.2. Cliente ssh

Los programas que permiten al usuario utilizar el protocolo ssh son **scp**, **sftp** y **ssh**, sus opciones generales pueden configurarse desde el archivo /etc/ssh/ssh_config. Las principales opciones que figuran en el manual son por orden alfabético:
Host Restricts the following declarations (up to the next Host keyword) to be only for those hosts that match one of the patterns given after the keyword. ‘*ánd ‘’? can be used as wildcards in the patterns. A single ‘*ás a pattern can be used to pro-provide vide global defaults for all hosts. The host is the hostname argument given on the command line (i.e., the name is not converted to a canonicalized host name before matching).
AFSTokenPassing Specifies whether to pass AFS tokens to remote host. The argument to this keyword must be “yes” or “no”. This option applies to protocol version 1 only.
BatchMode If set to “yes”, passphrase/password querying will be disabled. This option is useful in scripts and other batch jobs where no user is present to supply the password. The argument must be “yes” or “no”. The default is “no”.
BindAddress Specify the interface to transmit from on machines with multiple interfaces or aliased addresses. Note that this option does not work if UsePrivilegedPort is set to “yes”.
ChallengeResponseAuthentication Specifies whether to use challenge response authentication. The argument to this keyword must be “yes” or “no”. The default is “yes”.
CheckHostIP If this flag is set to “yes”, ssh will additionally check the host IP address in the known_ hosts file. This allows ssh to detect if a host key changed due to DNS spoofing. If the option is set to “no”, the check will not be executed. The default is “yes”.
Cipher Specifies the cipher to use for encrypting the session in protocol version 1. Currently, “blowfish”, “3des”, and “des” are supported. des is only supported in the ssh client for interoperability with legacy protocol 1 implementations that do not support the 3des cipher. Its use is strongly discouraged due to cryptographic weaknesses. The default is “3des”.
Ciphers Specifies the ciphers allowed for protocol version 2 in order of preference. Multiple ciphers must be comma-separated. The default is “aes128-cbc,3des-cbc,blowfish-cbc,cast128-cbc,arcfour, aes192-cbc,aes256-cbc”.
ClearAllForwardings Specifies that all local, remote and dynamic port forwardings specified in the configuration files or on the command line be cleared. This option is primarily useful when used from the ssh command line to clear port forwardings set in configuration files, and is automatically set by scp(1) and sftp(1). The argument must be “yes” or “no”. The default is “no”.
Compression Specifies whether to use compression. The argument must be “yes” or “no”. The default is “no”.
CompressionLevel Specifies the compression level to use if compression is enabled. The argument must be an integer from 1 (fast) to 9 (slow, best). The default level is 6, which is good for most applications. The meaning of the values is the same as in gzip(1). Note that this option applies to protocol version 1 only.
ConnectionAttempts Specifies the number of tries (one per second) to make before exiting. The argument must be an integer. This may be useful in scripts if the connection sometimes fails. The default is 1.
DynamicForward Specifies that a TCP/IP port on the local machine be forwarded over the secure channel, and the application protocol is then used to determine where to connect to from the remote machine. The argument must be a port number. Currently the SOCKS4 protocol is supported, and ssh will act as a SOCKS4 server. Multiple forwardings may be specified, and additional forwardings can be given on the command line. Only the superuser can forward privileged ports.
EscapeChar Sets the escape character (default: ’~’). The escape character can also be set on the command line. The argument should be a single character, ’^ followed by a letter, or “none” to disable the escape character entirely (making the connection transparent for binary data).
ForwardAgent Specifies whether the connection to the authentication agent (if any) will be forwarded to the remote machine. The argument must be “yes” or “no”. The default is “no”. Agent forwarding should be enabled with caution. Users with the ability to bypass file permissions on the remote host (for the agent’s Unix-domain socket) can access the local agent through the forwarded connection. An

attacker cannot obtain key material from the agent, however they can perform operations on the keys that enable them to authenticate using the identities loaded into the agent.

ForwardX11 Specifies whether X11 connections will be automatically redirected over the secure channel and DISPLAY set. The argument must be "yes" or "no". The default is "no". X11 forwarding should be enabled with caution. Users with the ability to bypass file permissions on the remote host (for the user's X authorization database) can access the local X11 display through the forwarded connection. An attacker may then be able to perform activities such as keystroke monitoring.

GatewayPorts Specifies whether remote hosts are allowed to connect to local forwarded ports. By default, ssh binds local port forwardings to the loopback address. This prevents other remote hosts from connecting to forwarded ports. GatewayPorts can be used to specify that ssh should bind local port forwardings to the wildcard address, thus allowing remote hosts to connect to forwarded ports. The argument must be "yes" or "no". The default is "no".

GlobalKnownHostsFile Specifies a file to use for the global host key database instead of /etc/ssh_known_hosts.

HostbasedAuthentication Specifies whether to try rhosts based authentication with public key authentication. The argument must be "yes" or "no". The default is "no". This option applies to protocol version 2 only and is similar to RhostsRSAAuthentication.

HostKeyAlgorithms Specifies the protocol version 2 host key algorithms that the client wants to use in order of preference. The default for this option is: "ssh-rsa,ssh-dss".

HostKeyAlias Specifies an alias that should be used instead of the real host name when looking up or saving the host key in the host key database files. This option is useful for tunneling ssh connections or for multiple servers running on a single host.

HostName Specifies the real host name to log into. This can be used to specify nicknames or abbreviations for hosts. Default is the name given on the command line. Numeric IP addresses are also permitted (both on the command line and in HostName specifications).

IdentityFile Specifies a file from which the user's RSA or DSA authentication identity is read. The default is $HOME/.ssh/identity for protocol version 1, and $HOME/.ssh/id_rsa and $HOME/.ssh/id_dsa for protocol version 2. Additionally, any identities represented by the authentication agent will be used for authentication. The file name may use the tilde syntax to refer to a user's home directory. It is possible to have multiple identity files specified in configuration files; all these identities will be tried in sequence.

KeepAlive Specifies whether the system should send TCP keepalive messages to the other side. If they are sent, death of the connection or crash of one of the machines will be properly noticed. However, this means that connections will die if the route is down temporarily, and some people find it annoying. The default is "yes" (to send keepalives), and the client will notice if the network goes down or the remote host dies. This is important in scripts, and many users want it too. To disable keepalives, the value should be set to "no".

KerberosAuthentication Specifies whether Kerberos authentication will be used. The argument to this keyword must be "yes" or "no".

KerberosTgtPassing Specifies whether a Kerberos TGT will be forwarded to the server. This will only work if the Kerberos server is actually an AFS kaserver. The argument to this keyword must be "yes" or "no".

LocalForward Specifies that a TCP/IP port on the local machine be forwarded over the secure channel to the specified host and port from the remote machine. The first argument must be a port number, and the second must be host:port. IPv6 addresses can be specified with an alternative syntax: host/port. Multiple forwardings may be specified, and additional forwardings can be given on the command line. Only the superuser can forward privileged ports.

LogLevel Gives the verbosity level that is used when logging messages from ssh. The possible values are: QUIET, FATAL, ERROR, INFO, VERVERBOSE, DEBUG, DEBUG1, DEBUG2 and DEBUG3. The default is INFO. DEBUG and DEBUG1 are equivalent. DEBUG2 and DEBUG3 each specify higher levels of verbose output.

MACs Specifies the MAC (message authentication code) algorithms in order of preference. The MAC algorithm is used in protocol version 2 for data integrity protection. Multiple algorithms must be comma-separated. The default is "hmac-md5,hmac-sha1,hmac-ripemd160,hmac-sha1-96,hmac-md5- 96".

NoHostAuthenticationForLocalhost This option can be used if the home directory is shared across machines. In this case localhost will refer to a different machine on each of the machines and the user will get many warn-ings about changed host keys. However, this option disables host authentication for localhost. The argument to this keyword must be "yes" or "no". The default is to check the host key for localhost.

NumberOfPasswordPrompts Specifies the number of password prompts before giving up. The argument to this keyword must be an integer. Default is 3.

PasswordAuthentication Specifies whether to use password authentication. The argument to this keyword must be "yes" or "no". The default is "yes".

Port Specifies the port number to connect on the remote host. Default is 22.

PreferredAuthentications Specifies the order in which the client should try protocol 2 authentication methods. This allows a client to prefer one method (e.g. keyboard-interactive) over another method (e.g. password) The default for this option is: "hostbased,publickey,keyboard-interactive,password".

Protocol Specifies the protocol versions ssh should support in order of preference. The possible values are "1" and "2". Multiple versions must be comma-separated. The default is "2,1". This means that ssh tries version 2 and falls back to version 1 if version 2 is not available.

ProxyCommand Specifies the command to use to connect to the server. The command string extends to the end of the line, and is executed with /bin/sh. In the command string, ' %h'will be substituted by the host name to connect and ' %p'by the port. The command can be basically anything, and should read from its standard input and write to its standard output. It should eventually connect an sshd(8) server running on some machine, or execute sshd -i somewhere. Host key management will be done using the HostName of the host being connected (defaulting to the name typed by the user). Setting the command to "none" disables this option entirely. Note that CheckHostIP is not available for connects with a proxy command.

PubkeyAuthentication Specifies whether to try public key authentication. The argument to this keyword must be "yes" or "no". The default is "yes". This option applies to protocol version 2 only.

RemoteForward Specifies that a TCP/IP port on the remote machine be forwarded over the secure channel to the specified host and port from the local machine. The first argument must be a port number, and the second must be host:port. IPv6 addresses can be specified with an alternative syntax: host/port. Multiple forwardings may be specified, and additional forwardings can be given on the command line. Only the superuser can forward privileged ports.

RhostsAuthentication Specifies whether to try rhosts based authentication. Note that this declaration only affects the client side and has no effect whatsoever on security. Most servers do not permit RhostsAuthentication because it is not secure (see RhostsRSAAuthentication). The argument to this keyword must be "yes" or "no". The default is "no". This option applies to protocol version 1 only and requires ssh to be setuid root and UsePrivilegedPort to be set to "yes".

RhostsRSAAuthentication Specifies whether to try rhosts based authentication with RSA host authentication. The argument must be "yes" or "no". The default is "no". This option applies to protocol version 1 only and requires ssh to be setuid root.

RSAAuthentication Specifies whether to try RSA authentication. The argument to this keyword must be "yes" or "no". RSA authentication will only be attempted if the identity file exists, or an authentication agent is running. The default is "yes". Note that this option applies to protocol version 1 only.

SmartcardDevice Specifies which smartcard device to use. The argument to this keyword is the device ssh should use to communicate with a smartcard used for storing the user's private RSA key. By default, no device is specified and smartcard support is not activated.

StrictHostKeyChecking If this flag is set to "yes", ssh will never automatically add host keys to the $HOME/.ssh/known_hosts file, and refuses to connect to hosts whose host key has changed. This provides maximum protection against trojan horse attacks, however, can be annoying when the /etc/ssh_known_hosts file is poorly maintained, or connections to new hosts are frequently made. This option forces the user to manually add all new hosts. If this flag is set to "no", ssh will automatically add new host keys to the user known hosts files. If this flag is set to "ask", new host keys will be added to the user known host files only after the user has confirmed that is what they really want to do, and ssh will refuse to connect to hosts whose host key has changed. The host keys of known hosts will be verified automatically in all cases. The argument must be "yes", "no" or "ask". The default is "ask".

UsePrivilegedPort Specifies whether to use a privileged port for outgoing connections. The argument must be "yes" or "no". The default is "no". If set to "yes" ssh must be setuid root. Note that this option must be set to "yes" if RhostsAuthentication and RhostsRSAAuthentication authentications are needed with older servers. User Specifies the user to log in as. This can be useful when a different user name is used on different machines. This saves the trouble of having to remember to give the user name on the command line.

UserKnownHostsFile Specifies a file to use for the user host key database instead of $HOME/.ssh/known_hosts.

XAuthLocation Specifies the full pathname of the xauth(1) program. The default is /usr/X11R6/bin/

xauth.

Los valores por defecto de este archivo de configuración son:

```
# ForwardAgent no
# ForwardX11 no
# RhostsAuthentication no
# RhostsRSAAuthentication no
# RSAAuthentication yes
# PasswordAuthentication yes
# BatchMode no
# CheckHostIP yes
# StrictHostKeyChecking ask
# IdentityFile ~/.ssh/identity
# IdentityFile ~/.ssh/id_rsa
# IdentityFile ~/.ssh/id_dsa
# Port 22
# Protocol 2,1
# Cipher 3des
# Ciphers aes128-cbc,3des-cbc,blowfish-cbc,cast128-cbc,arcfour,aes192-cbc,aes256-cbc
```

En estos archivos (*predefinidos*) se indican las rutas para obtener las claves públicas y privadas de cada usuario:

- ~/.ssh/identity
- ~/.ssh/id_rsa
- ~/.ssh/id_dsa

4.3. Cliente Telnet / rsh / rlogin y su Reemplazo por ssh

Como se explicó antes, ssh funciona de la misma manera que telnet o rsh/rlogin pero encriptando la información. A partir de que una computadora es identificada con la primera vez que se ejecuta el comando **ssh** (sección 2), si se usa el rlogin sin password, ssh ofrece un mecanismo similar, pudiendo editar un archivo ~/.shosts agregando la línea:

```
remotehosts remoteusers
```

de la misma manera que se hacía con ~/.rhosts aunque ssh es un poco más complicado con este tipo de logins, es más cuidadoso, ya que si se conecta desde un localhost, comprueba que realmente sea localhost y no un cracker.host.org usando una conexión engañosa. El archivo ~/.ssh/known_hosts contiene la clave pública de host de localhost. Esta es la autenticación de host, a partir de ella y de acuerdo a la configuración de los clientes y servidores se procede a la autenticación de usuario. En general, ssh cuenta con tres métodos distintos para ello:

- Autenticación mediante usuario/contraseña: es la forma básica, no sirve para MPI porque el usuario debe ingresar su password.
- Autenticación basada en host/usuario: de la misma manera que en rsh, se puede configurar el acceso de ssh mediante archivos que especifican desde que usuario y máquina se permite el acceso:
  - /etc/ssh/shosts.equiv con el mismo funcionamiento que /etc/hosts.equiv
  - $HOME/.shost a nivel de usuario,como $HOME/.rhosts
- Autenticación mediante claves: el cliente debe generar sus claves privada y pública, compartiendo esta última con el servidor para poder identificarse. Una vez hecho esto las conexiones se podrán establecer sin la necesidad de ingreso de password por parte del usuario.

Un ejemplo de Autenticación basada en host/usuario se puede dar con la utilización del servicio (se pedirá el ingreso de clave) desde un localhost a un remotehost y desde éste de vuelta a la máquina inicial:

```
juan@maquina1$ ssh maquina2.ing.unlp.edu
juan@maquina2's password: [password acá]
Bienvenido a maquina2.ing.unlp.edu
juan@maquina2$ echo 'maquina1.ing.unlp.edu'>> ~/.shosts
juan@maquina2$ chmod 600 ~/.shosts
juan@maquina2$ ssh maquina1.ing.unlp.edu
juan@maquina1's password: [password acá]
Bienvenido a maquina1.ing.unlp.edu
```

La segunda conexión desde maquina1 a maquina2 deberá ser posible sin necesidad de tipear el password. En caso de no funcionar, se puede usar la opción -v (verbose) durante la conexión, que dará información de lo que pasa durante ésta.

Un mensaje encriptado con la clave pública, sólo puede desencriptarse con la clave privada y hay que tener mucho cuidado con los nombres y permisos de los archivos donde se almacenan éstas, pues el proceso de autenticación puede no encontrar la clave necesaria y abortar la conexión o pasar a otro método de autenticación posible como el pedido de password. Puede haber 3 clases diferentes de claves:

- RSA1 para la versión 1 del protocolo.
- RSA para la versión 1 y 2.
- DSA sólo para la versión 2.

La clave del cliente se genera con el comando ssh-keygen conla opción -t para indicar el tipo clave que se quiere:

```
[usuario1@localhost usuario1]$ ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key
(/home/usuario1/.ssh/id_dsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in
/home/usuario1/.ssh/id_dsa.
Your public key has been saved in
/home/usuario1/.ssh/id_dsa.pub.
The key fingerprint is:
1c:bb:8c:da:c5:a4:db:9f:bb:4d:64:86:a8:29:85:05
usuario1@localhost.localdomain
[usuario1@localhost usuario1]$
```

En el ejemplo se ha utilizado el tipo dsa y se ha guardado en los archivos estándares. La clave privada es a que identifica al usuario, por lo que debe ser accesible únicamente por el usuario propietario. Se debe ahora compartir la clave pública, almacenada en este caso en

```
$HOME/.ssh/id_dsa.pub
```

la que debe ser incluída en el archivo

```
$HOME/.ssh/authorized_keys
```

de cada máquina en la que se necesite utilizar la autenticación por clave pública. En el caso de utilizar una passphrase no vacía se solicitaría al realizar la autenticación:

```
[usuario1@localhost usuario1]$ ssh usuario1@servidor.dominio
Enter passphrase for key '/home/usuario1/.ssh/id_dsa':
[usuario1@servidor usuario1]$
```