# SSH – SCP - SFTP

## Présentation

Nous allons voir rapidement quelques modes d'utilisation de **ssh, scp et sftp**.
Qu'est ce que ssh ?
En fait ssh **S**ecure **Sh**ell, propose un shell sécurisé pour les connexions à distance et se présente dans ce domaine comme le standard « de fait ». Mais ce n'est pas que cela. Nous allons essayer de voir quelques modes d'utilisation de ce produit.
L'objectif est de pouvoir supprimer d'un serveur toute application utilisant des protocoles « non sûrs » (ftp, telnet, rsh...), pour les remplacer par des applications plus sûres, ou alors, si'il n'est pas possible de les désactiver, de les faire passer dans un tunnel crypté.

Des programmes de la même famille comme « **scp** » ou « **sftp** » remplacent les commandes ftp ou rcp.
Avec ssh la totalité de la transaction entre un client et le serveur est cryptée. Le client à la possibilité d'utiliser des applications X distantes (X11 forwarding) à partir d'une invite shell dans un environnement sécurisé.
On se sert également de SSH pour sécuriser des transactions non sûres comme pop3 ou imap par exemple. De plus en plus, ces applications supportent maintenant SSL aussi bein pour les clients que pour les serveurs.

SSH sur GNU/Linux est généralement composé de 3 packages :
OpenSSH général, (openssh),
le serveur OpenSSH (openssh-server)
le client (openssh-clients).
Les packages OpenSSH requièrent le paquetage OpenSSL (openssl).

Chaque fois que cela est possible vous devez utiliser une solution qui chiffre vos transactions.

## Mode de fonctionnement de SSH

L'établissement du dialogue entre le client et le serveur suit un protocole particulier :
1 – établissement d'une couche transport sécurisée
2 – chiffremment des données à l'aide de clefs symétriques pendant la transaction

Le client peut s'authentifier en toute sécurité, et accéder aux applications conformes aux spécifications du protocole.

## Mode de fonctionnement de la couche transport SSH

La couche transport assure le chiffrement et le déchiffrement des données. Elle assure également la compression pour améliorer le transfert.

Le client et le serveur négocient plusieurs éléments afin que la session puisse s'établir.
- l'échange des clés
- l'algorithme de clé publique à utiliser
- l'algorithme de chiffrement symétrique à utiliser
- l'algorithme d'authentification de message à utiliser

– l'algorithme repère (hash) à utiliser

Lors du premier échange, le client ne connaît pas le serveur. Le serveur propose alors une clé hôte qui servira par la suite au client de moyen d'identification du serveur.

```
[mly@neptune mly]$ ssh -l amascret linux-france.org
Warning: Permanently added 'linux-france.org,80.247.225.35' (DSA) to the list of known hosts.
amascret@linux-france.org's password:
Last login: Sat Nov  2 11:37:32 2002 from 212.47.248.114
Linux 2.2.19.
amascret@tuxinette:~$
```

Le risque de ce procesus, vient donc surtout de la première transaction, où le client n'a pas le moyen d'identifier de façon fiable le serveur avec qui il communique. Un pirate peut dans certains cas, tenter de détourner cette opération.
Au cours d'un échange, le client et le serveur modifient régulièrement leurs clés. À chaque opération de renouvellement de clé, un pirate qui aurait réussit à décrypter les clés, devrait refaire toute l'opération.

## Authentification

Une fois le tunnel sécurisé mis en place, le serveur envoie au client les différentes méthodes d'authentification qu'il supporte.
Dans le cadre d'une authentification par mot de passe, celui-ci peut être envoyé en toute sécurité puisqu'il est chiffré.

## Connexion

Une fois l'authentification réalisée, le tunnel SSH peut multiplexer plusieurs canaux en délégant la tâche à des agents.

```
[mlx@pegase X11]$ pstree -l 24245
sshd---bash-+-drakfw
            |-pstree
            |-2*[xclock]
            `-xlogo
```

Ici on voit dans la même session ssh, 2 canaux pour xclock, 1 pour xlogo et 1 pour la commande pstree.

Chaque canal est numéroté. Le client peut fermer un canal sans pour autant fermer toute la session.

## Fichiers de configuration d'OpenSSH

OpenSSH est constitué de deux ensembles de fichiers de configuration, un pour les programmes client (ssh, scp et sftp) et l'autre pour le service serveur(sshd).

Les informations de configuration SSH qui s'appliquent à l'ensemble du système sont stockées dans le répertoire **/etc/ssh** :

Voici les principaux fichier de configuration.

**ssh_config** - fichier de configuration client SSH pour l'ensemble du système. Il est écrasé si un même fichier est présent dans le répertoire personnel de l'utilisateur (~/.ssh/config).
**sshd_config** - fichier de configuration pour sshd.
**ssh_host_dsa_key** - clé DSA privée utilisée par sshd.
**ssh_host_dsa_key.pub** - clé DSA publique utilisée par sshd.
**ssh_host_key** - clé RSA privée utilisée par sshd pour la version 1 du protocole SSH.
**ssh_host_key.pub** - clé RSA publique utilisée par sshd pour la version 1 du protocole SSH.
**ssh_host_rsa_key** - clé RSA privée utilisée par sshd pour la version 2 du protocole SSH.
**ssh_host_rsa_key.pub** - clé RSA publique utilisée par sshd pour la version 2 du protocole SSH.

Les informations spécifiques à un l'utilisateur sont stockées dans son répertoire personnel à l'intérieur du répertoire **~/.ssh/:**

**authorized_keys** - ce fichier contient une liste de clés publiques "autorisées". Si un utilisateur se connecte et prouve qu'il connaît la clé privée correspondant à l'une de ces clés, il obtient l'authentification. Notez qu'il ne s'agit que d'une méthode d'authentification facultative.

**id_dsa** - contient l'identité d'authentification DSA de l'utilisateur.
**id_dsa.pub** - la clé DSA publique de l'utilisateur.
**id_rsa** - la clé RSA publique utilisée par sshd pour la version 2 du protocole SSH.
**identity** - la clé RSA privée utilisée par sshd pour la version 1 du protocole SSH.
**known_hosts** - Ce fichier contient les clés hôte DSA des serveurs SSH auxquels l'utilisateur s'est connecté.

## Configurer et utiliser SSH

### *Premiers pas*

L'idée est de mettre en place une procédure entre un client et un serveur qui garantit des transactions sécurisées. À la fin, vous pourrez utiliser les ressources du serveur distant dans un tunnel, sans avoir à vous authentifier à chaque fois.

Pour ceux qui ont déjà utilisé les commandes rlogin, rsh, rcp... et les fichiers .rhosts, le principe est identique. La différence fondamentale est que avec ssh, tout est encrypté.

Pour cela vous devez metter en place les clés qui serviront à ssh pour vous authentifier.

La première chose à faire est de vous créer une clé. Voyons comment réaliser cela.

Tout d'abord allez dans votre répertoire personnel.
**$ cd**
**$ ssh-keygen -t dsa**
**[mlx@neptune mlx]$ ssh-keygen -t dsa**
**Generating public/private dsa key pair.**
**Enter file in which to save the key (/home/mlx/.ssh/id_dsa):**
**Enter passphrase (empty for no passphrase):**
**Enter same passphrase again:**
**Your identification has been saved in /home/mlx/.ssh/id_dsa.**
**Your public key has been saved in /home/mlx/.ssh/id_dsa.pub.**
**The key fingerprint is:**
**5c:d9:d8:c5:3d:8d:0b:10:33:42:9c:83:45:a1:d0:61 mlx@neptune.beaupeyrat.com**

Vérification de **.ssh** Attention il y a un « . » devant ssh

```
[mlx@neptune mlx]$ ls -alR .ssh
.ssh:
total 20
drwx------    2 mlx      mlx          4096 nov 11 18:19 ./
drwx--x--x   37 mlx      mlx          4096 nov 11 18:09 ../
-rw-------    1 mlx      mlx           736 nov 11 18:19 id_dsa
-rw-r--r--    1 mlx      mlx           616 nov 11 18:19 id_dsa.pub
-rw-r--r--    1 mlx      mlx          1956 nov 10 19:38 known_hosts
```

Cette commande a généré une clé DSA par défaut de 1024 bits. La clé privée sera stockée dans **~/.ssh/id_dsa** et la clé publique dans **~/.ssh/id_dsa.pub**.
Si vous voulez générer une clé RSA2, utilisez l'option "**-t rsa**" et pour du RSA1 "**-t rsa1**".
Vous devrez entrer une « **pass phrase** ». Entre 10 et 30 caractères. Mélangez majuscules, minuscules et chiffres. La clé privée doit ensuite être mise en lecture seule pour le propriétaire et aucun accès pour les autres.

Pour modifier votre « **pass phrase** » sur une clé privée DSA, utilisez la commande :

**mlx@neptune mlx]$ ssh-keygen -p -f ~/.ssh/id_dsa**
**Enter old passphrase:**
**Key has comment '/home/mlx/.ssh/id_dsa'**
**Enter new passphrase (empty for no passphrase):**
**Enter same passphrase again:**
**Your identification has been saved with the new passphrase.**

**Attention, on oublie pas une « pass phrase ».**
Il va falloir maintenant copier la clé publique vers le ou les serveurs distants. Il est préférable que vous ayez un compte, sinon vous devrez demander à l'administrateur distant de réaliser la procédure.

La clé publique, doit être copiée sur le serveur distant dans **~/.ssh/authorized_keys**. La clé privée reste sur votre client. Vous pouvez mettre plusieurs clés publiques sur le serveur, si vous le souhaitez ou si vous accédez au serveur depuis plusieurs machines clientes.

Je vais copier ma clé avec scp sur un compte que j'ai sur une machine distante :

```
[mlx@neptune mlx]$ scp .ssh/id_dsa.pub mlx@pegase.beaupeyrat.com:/home/mlx/.ssh/authorized_keys
Warning: Permanently added 'pegase.beaupeyrat.com' (RSA) to the list of known hosts.
mlx@pegase.beaupeyrat.com's password:
id_dsa.pub          100% *********************************************|   616      00:00
```

Le système m'a demandé mon mot de passe.
Elle est transférée. On doit pouvoir maintenant réaliser des opérations (commandes) comme **scp** sur le serveur distant, sans avoir à saisir de mot de passe. Ici on va faire un « **ls** », sans ouvir de session sur la machine distante.

[mlx@neptune mlx]$ ssh mlx@pegase.beaupeyrat.com ls
Enter passphrase for key '/home/mlx/.ssh/id_dsa':
d1
d2
d3
d4

Ça marche, le système distant ne me demande plus mon mot de passe, par contre il me demande la « pass phrase ». Il va falloir aussi essayer de se passer de ça, car s'il est fastidieux de saisir son mot de passe, il est encore plus embêtant de saisir une « pass phrase ». Nous verrons comment se passer de ça avec un agent.

**Remarque :**
Envoyer une clé par mail n'est pas un système sûr. L'administrateur distant peut demander à ce que l'envoyeur justifie qu'il est bien celui qui a envoyé la clé. Il suffit pour cela de téléphoner à l'administrateur et de communiquer « la signature ou empreinte » (finger print) de la clé. On utilise le même procédé pour identifier et vérifier la validité des clés gpg.

Pour obtenir le « finger print » d'une clé utiliser la commande :
**mlx@neptune mlx]$ ssh-keygen -l**
**Enter file in which the key is (/home/mlx/.ssh/id_rsa): .ssh/id_dsa**
**1024 5c:d9:d8:c5:3d:8d:0b:10:33:42:9c:83:45:a1:d0:61 .ssh/id_dsa.pub**
**[mlx@neptune mlx]$ ssh-keygen -l**
**Enter file in which the key is (/home/mlx/.ssh/id_rsa): .ssh/id_dsa.pub**
**1024 5c:d9:d8:c5:3d:8d:0b:10:33:42:9c:83:45:a1:d0:61 .ssh/id_dsa.pub**

## *Utiliser un agent ssh*

L'utilisation d'un agent, évite d'avoir à retaper la « pass phrase » à chaque fois que l'on sollicite l'utilisation de la clé privée. Un agent stocke en mémoire les clés privées. Voici comment activer un agent :

**$ ssh-agent**
La commande met sur la sortie standard des variables environnement à déclarer et à exporter. Faites le.
**$ SSH_AUTH_SOCK=/tmp/ssh-XXXB76f4/agent.2888; export SSH_AUTH_SOCK;**
**$ SSH_AGENT_PID=2889; export SSH_AGENT_PID;**
**$ echo Agent pid 2889;**

On va maintenant exporter les clés. Cela consiste à les mettre dans le cache de l'agent avec la commande **ssh-add**. La commande demandera la « pass phrase »..

**$ ssh-add**
**Enter passphrase for /home/mlx/.ssh/id_dsa:**
**Identity added: /home/mlx/.ssh/id_dsa (/home/mlx/.ssh/id_dsa)**

On vérifie maintenant la connexion.

**$ ssh mlx@pegase.beaupeyrat.com ls**
**d1**
**d2**
**d3**
**d4**

Ça marche, nous n'avons plus besoin de taper le mot de passe, ni la « pass phrase ».

Pour supprimer une clé (ici RSA) de l'agent, utilisez l'option « -d »
$ ssh-add -d ~/.ssh/id_rsa

Utilisez l'option -D pour supprimer toutes les clés de l'agent.
Utilisez l'option -l pour savoir quelles clés sont chargées par l'agent.

### *Automatisation dans X*

Cela vous semble sûrement un peu compliqué et encore trop lourd. Cependant pour une nouvelle session, vous n'avez plus besoin que d'exporter les variables d'environnement $SSH_AGENT_PID $SSH_AUTH_SOCK et d'utiliser la commande ssh-add. Tout cela peut être mis éventuellement dans un script.

Pour généraliser le mécanisme à l'ensemble de votre session X window, il est nécessaire que ces variables d'environnement soient définies avant le lancement du serveur X, qui les exportera par défaut à l'ensemble de vos applications graphiques (lancées sous X).

Cela peut être fait dans le fichier startx ou bien ~/.xinitrc ou bien ~/.xsession, en résumé avant le lancement de votre gestionnaire de fenêtres. Par exemple, si c'est le fichier ~/.xinitrc qui lance le gestionnaire de fenêtres, il ressemblera à ceci:

```
#!/bin/sh

ssh-agent -s > /tmp/ssh.keys              # pour y mettre les variables environnement
.  /tmp/ssh.keys                          # Exporter les variabeles
rm  /tmp/ssh.keys
startx
```

Ainsi, au prochain lancement d'un serveur X, vous n'aurez plus qu'à ouvrir une xterm et taper la commande:  **ssh-add**

## X and Port Forwarding

SSH permet de « tunneler » des protocoles applicatifs via la retransmission de port. Lorsque vous utilisez cette technique, le serveur SSH devient un conduit crypté vers le client SSH.

Cela représente un des plus importants intérêt de SSH. Permettre la création de tunnels « sûrs » pour des protocoles de transports « non sûrs ». Ce principe est utilisé pour des applications comme pop, imap, mais également pour des applications X Window.

La retransmission de port mappe (redirige) un port local du client vers un port distant du serveur. SSH permet de mapper (rediriger) tous les ports du serveur vers tous les ports du client.

Pour créer un canal de retransmission de port TCP/IP qui attend les connexions sur l'hôte local, utilisez la commande suivante :

**ssh -L port-local:nomhôte-distant:port-distant nomutilisateur@nomhôte**

Une fois que le canal de retransmission de port est en place entre les deux ordinateurs, vous pouvez diriger votre client (POP par exemple) pour qu'il utilise le port local redirigé et non plus vers le port distant avec une transmission en clair.

Voyons comment créer un tunnel pour le service d'émulation VT par exemple. On va utiliser un port local compris entre 1024 et 65535 qui n'est sont pas réservé pour des applications utilisateurs.

On va prendre par exemple le port local 1023. Pour créer un tunnel on va utiliser la commande :
**$ ssh -L2323:localhost:23 mlx@pegase.beaupeyrat.com**
**Enter passphrase for key '/home/mlx/.ssh/id_dsa':**

On crée ici un tunnel entre le port local 2323 et le port distant 23.
Le tunnel étant créé il ne reste plus qu'à utiliser le port local 2323 avec un telnet adresse_de_la_machine 2323

Le fait de quitter ssh ferme la session tunnelée.

Il est également possible d'ouvrir une session temporaire pour un temps donné. Cela évite d'avoir à fermer les connexions.

Dans la commande :
**$ ssh – f -L2323:localhost:23 mlx@pegase.beaupeyrat.com sleep 10**

L'option **-f**, met ssh en tâche de fond. La session sera fermé automatiquement au bout du temps déterminé, seulement si aucun processus n'utilise le canal à cemoment.

Le déport d'application graphique fonctionne sur le même principe. Le client (**/etc/ssh/ssh_config**), doit avoir l'option « **ForwardX11** » activée pour les systèmes distants. Le serveur (**/etc/ssh/sshd_config**), doit également avoir l'option « **X11Forwarding** » d'activée. Il est possible de tunneler des applications graphiques dans ssh.

Voyons comment utiliser une application graphique distante :

**$ ssh -C mlx@pegase.beaupeyrat.com**
**Enter passphrase for key '/home/mlx/.ssh/id_dsa':**
**$ xclock&**
**[1] 7771**

L'horloge distante s'affiche. L'option « **-C** », active la compression.

Vous pouvez tout mettre sur la même ligne de commande, avec l'option « **-f** », qui « **forke** » l'application demandée.
**$ ssh -C -f mlx@pegase.beaupeyrat.com xclock**

Notez le prompt, ici que l'ouverture de session a été effectuée en tâche de fond. Pour fermer le canal ssh, il suffit de fermer l'application.

# Automatisation de tâches SSH

Dans la mesure où il est possible de passer des commandes à distances sur un serveur, sans avoir à saisir de mot de passe ou de « **pass phrase** », on peut envisager l'automatisation de tâches entre machines et dans des tunnels, comme par exemple les sauvegardes ou la synchronisation de fichiers. Il suffira pour cela de créer un compte associé à la tâche, lui créer une clé privé et exporter sa clé publique sur les différentes machines.

Attention toutefois, les manipulations sur les serveurs requièrent un accès root. Cela signifie que votre compte opérateur, devra avoir un accès root sur le serveur ce qui n'est jamais très bon.

Vous aurez intérêt à réfléchir à la façon de réaliser le traitement sans que ce compte opérateur ait besoin du compte « super utilisateur ».

Si vous avez un serveur ayant un dm (Desktop Manager) comme gdm par exemple, disponible sur le réseau vous pouvez lancer les émulations X Window des clients en appelant le serveur. Par exemple sur le client faire :

X -query x.y.z.t :u
ou encore
X -broadcast :u

avec u pouvant varierr de 0 à 5, suivant que vous voulez avoir la session graphique sur F7...F12. Pour le système vc7 que vous utilisez par défaut avec CTRL ALT F7 correspond au premier « display » :0.

# Utilisation de SCP et de SFTP

L'utilisation de ces commandes est relativement simple. SCP permet de faire de la copie de fichier. SFTP est utilisable en mode intéractif ou en mode batch et ressemble plus au FTP.

## *Utilisation de scp*

La commande
[mlx@neptune mlx]$ ssh mlx@pegase.beaupeyrat.com "ls -al psionic"

```
-rw-r--r--   1 root   root    64562 Nov 23 23:05 hostsentry-0.02-4.noarch.rpm
-rw-r--r--   1 root   root    26955 Nov 23 23:05 logsentry-1.1.1-1.i386.rpm
-rw-r--r--   1 root   root    48804 Nov 23 23:06 portsentry-1.1-fr7.i386.rpm
-rw-r--r--   1 root   root    48804 Nov 23 23:13 portsentry-1.1-fr7.i386.rpm.1
```

La commande : **ssh mlx@pegase.beaupeyrat.com "ls -al"**
me donne la liste des fichiers distants qui sont dans le répertoire « psionic ». Pour copier ces fichiers localement dans un répertoire psionic on va utiliser :
cd && mkdir psionic
scp mlx@pegase.beaupeyrat.com:/home/mlx/psionic/* ~/psionic

Ou pour envoyer les fichiers du répertoire local psionic, vers le répertoire tmp qui est dans /home/mlx de la machine pegase.beaupeyrat.com :
scp ~/psionic/* mlx@beaupeyrat.com:/home/mlx/tmp

## Utilisation de sftp

sftp peut être utilisé pour du transfert de fichier en mode sécurisé.
sftp  mlx@pegase.beaupeyrat.com
sftp>
On obtient un prompt, ici le système ne m'a pas demandé de m'authentifier. Pour avoir une liste des commande, utiliser « help ».

sftp> help
Available commands:

| | |
|---|---|
| cd path | Change remote directory to 'path' |
| lcd path | Change local directory to 'path' |
| chgrp grp path | Change group of file 'path' to 'grp' |
| chmod mode path | Change permissions of file 'path' to 'mode' |
| chown own path | Change owner of file 'path' to 'own' |
| help | Display this help text |
| get remote-path [local-path] | Download file |
| lls [ls-options [path]] | Display local directory listing |
| ln oldpath newpath | Symlink remote file |
| lmkdir path | Create local directory |
| lpwd | Print local working directory |
| ls [path] | Display remote directory listing |
| lumask umask | Set local umask to 'umask' |
| mkdir path | Create remote directory |
| put local-path [remote-path] | Upload file |
| pwd | Display remote working directory |
| exit | Quit sftp |
| quit | Quit sftp |
| rename oldpath newpath | Rename remote file |
| rmdir path | Remove remote directory |
| rm path | Delete remote file |
| symlink oldpath newpath | Symlink remote file |
| version | Show SFTP version |
| !command | Execute 'command' in local shell |
| ! | Escape to local shell |
| ? | Synonym for help |