SSH2.3

Olivier Hoarau (olivier.hoarau@fnac.net)

V1.0 du 10.9.00

1 Préambule

Ce document présente SSH dans sa version 2.3, SSH est un outil pour établir des connexions entre machine de manière totalement sécurisée.
La dernière version de ce document est téléchargeable à l'URL http://funix.free.fr. Ce document peut être reproduit et distribué librement dès lors qu'il n'est pas modifié et qu'il soit toujours fait mention de son origine et de son auteur, si vous avez l'intention de le modifier ou d'y apporter des rajouts, contactez l'auteur pour en faire profiter tout le monde.

2 Présentation

SSH est une alternative sécurisée à des outils de connexion comme **telnet** (pour lequel le mot de passe circule en clair sur le réseau), mais c'est bien plus que cela puisqu'il permet aussi de lancer des commandes à distance (comme **rsh**, ou **remsh**), mais aussi de transférer des fichiers ou des répertoires entiers (comme **rcp**). Il permet aussi de sécuriser des communications qui à la base ne le sont pas comme **POP** ou **telnet** grâce à des "tunnels" sécurisés.
SSH se présente sous la forme d'un daemon et d'un client, le daemon tourne sur un serveur et attend les requêtes des clients **SSH** pour que les utilisateurs distants puissent se connecter sur le serveur.

OpenSSH2.1.1 et **SSH2.3** ne sont pas pleinement compatibles, si on utilise le hachage **MD5** et l'authentification **DSA** que ce soit dans un sens ou un autre.

3 Avertissement

L'utilisation de **SSH** obéit à des règles en terme de licence, vous devez vous trouver dans les cas suivants:

- Evaluation pour une utilisation à but commercial
- Utilisation dans un but commercial si vous avez acheté la licence
- Utilisation dans un cadre non commercial (universités, utilisation privée...)

Il n'est plus nécessaire de s'enregistrer sur le site.

4 Comment ça marche

4.1 Présentation

On prend comme exemple **olivier** sur **obelix** et **veronique** sur **asterix**. **SSH** repose sur le protocole **SSH**, le protocole a évolué au fil du temps, et on fait la différence entre **SSH1** et **SSH2**, le dernier étant plus récent et quelque peu différent du premier. Avec **SSH** vous pouvez combiner les caractéristiques des deux protocoles. Voilà les différences entre les deux versions.

4.2 SSH1

Il y a trois manières de s'authentifier.

La première méthode va sans doute rappeler à certains les commandes "r" (**rlogin**, **rcp**, **remsh** ou rsh). Si **asterix** est listé dans le fichier /**etc/hosts.equiv** ou **/etc/shots.equiv** d'**obelix**, et qu'il existe un compte **veronique** sur **obelix**, **veronique** pourra se connecter sans problème avec **ssh** sur **obelix**.

Si l'utilisateur **olivier** possède un fichier **~/.shosts** ou **~/.rhosts** contenant:

asterix veronique

veronique pourra se connecter sur le compte d'**olivier** avec **ssh** (en donnant néanmoins le mot de passe d'**olivier**).

Ces deux formes d'authentification ne sont absolument pas sécurisées, Le problème avec cette méthode est que n'importe quelle machine peut se faire passer pour **asterix** (spoofing).

La deuxième méthode est l'authentification toujours sur les fameux fichiers **rhosts** et **hosts.equiv** mais combinée avec une authentification en utilisant les clés **RSA** des machines. C'est à dire qu'on vérifie d'abord **/etc/hosts.equiv**, **/etc/shosts.equiv**,**~/.rhosts** et **~/.shosts** puis ensuite on vérifier si la clé publique du de la machine qui cherche à se connecter se trouve dans le fichier **/etc/ssh_known_hosts** ou **~/.ssh/known_hosts**. Cette méthode est plus sécurisée car elle évite qu'une machine se fasse passer pour une autre, car à la connexion il va y avoir vérification de la clé publique par rapport à la clé privée de la machine cliente.

La troisième méthode est basée sur l'échange des clés utilisateurs en utilisant le protocole de gestion des clés **RSA**, **veronique** donne sa clé publique à **olivier**, quand elle essaye de se connecter sur le compte d'**olivier**, **SSH** vérifie que la clé publique que détient **olivier** correspond bien à la clé privée de **véronique**. Plus précisément du côté d'**olivier** le serveur **SSH** crypte un nombre aléatoire (un "challenge" dans la doc en anglais) en utilisant la clé publique de **véronique** détenue par **olivier**, seule la clé privée de **veronique** pourra décrypter ce "challenge", celui-ci est envoyé au client **SSH** qui va le décrypter avec la clé privée de **véronique** ce qui va authentifier cette dernière complètement.

Voyons le fonctionnement en détail de l'établissement d'une connexion:

Chaque machine possède un couple de clé (publique et privée par défaut de 1024 bits) de type **RSA**, quand le daemon se lance il génére une clé serveur (**server key**) de type **RSA** (par défaut 768bits). Cette clé serveur est normalement détruite et reconstruite à intervalle régulier (toutes les heures par défaut) et n'est jamais stockée sur le disque.

Quand un client se connecte, le daemon répond en envoyant la clé publique de la machine serveur et la clé serveur. La machine cliente va comparer la clé publique de la machine serveur par rapport à celle qui possède (s'il l'a) pour voir si elle n'a pas changée. Le client va générer alors un nombre aléatoire de 256 bits. Il crypte ce nombre en se servant de la clé publique et de la clé serveur de la machine serveur. Ce nombre aléatoire va servir pour générer une clé de session, on l'appelle clé de session car elle est spécifique à la session de connexion, cette clé de session va servir à chiffrer tout ce qui va transiter pendant la connexion. Les algo qui sont choisis pour crypter les données avec la clé de session qui vont circuler sont **Blowfish**, **tripleDES**.

4.3 SSH2

4.3.1 Principe

Avec **SSH2** on utilise aussi l'authentification par clé privée-publique au niveau utilisateur, mais cette fois-ci en utilisant l'algorithme de gestion des clés **DSA** plutôt que **RSA** qui n'est pas dans le domaine public.

Avec ce protocole on a en plus des moyens supplémentaires pour assurer la confidentialité des données, on peut chiffrer les données qui transitent en utilisant le **tripleDES**, **Blowfish**, **CAST128** ou **Arcfour** et l'intégrité avec des algorithmes de hachage comme **SHA1** ou **MD5** ce que ne possède pas la version 1 de **SSH**.

Fonctionnement en détail:

C'est similaire à **SSH1**, chaque machine possède un couple de clé (publique et privée) de type **DSA** par contre. Quand le serveur se lance, aucune clé serveur n'est générée, on utilise une gestion de clé de type **Diffie-Hellman**. Cette gestion débouche sur la création d'une clé de session spécifique à la session de connexion qui servira à chiffrer les données qui vont transiter.

4.3.2 En pratique

Un utilisateur **olivier** sur la machine **obelix** peut autoriser certaines personnes venant de certaines machines à se connecter sous **obelix** en utilisant son compte. Pour cela il doit récupérer la clé publique de chacune de ces personnes. Lors de l'établissement de la communication **SSH** va vérifier que la clé publique d'un utilisateur que possède **olivier** correspond bien à clé privé de l'utilisateur en question. En conséquence si **olivier** ne possède pas votre clé publique, il sera impossible de vous connecter sous son compte, le seul moyen est de "voler" la clé privée d'une personne habilitée (dont **olivier** possède la clé publique).

Si vous n'avez toujours pas compris, voici les différentes étapes pour établir une connexion via **SSH** entre les utilisateurs **veronique** sous **asterix** et **olivier** sous **obelix**.

veronique génére clé publique et privée sur **asterix**
olivier génére clé publique et privée sur **obelix**
veronique donne sa clé publique à **olivier**

veronique veut se connecter sous le compte d'**olivier** sous **obelix** en lançant un client **SSH** sous **asterix**
le daemon **SSH** sous **obelix** vérifie si **olivier** possède bien la clé publique de **veronique** (autorisation)
Le daemon **SSH** sous **obelix** et le client **SSH** sous **asterix** rentrent en communication pour savoir si la clé publique de **veronique** que possède **olivier** correspond bien à la clé privée de **veronique**
veronique doit rentrer une phrase password pour s'identifier auprès du client **SSH** tournant sur **asterix**
veronique doit maintenant rentrer le mot de passe d'**olivier**

Ca y est **veronique** est connecté sous le compte d'**olivier** après être passé par quatre phases d'identification.

5 Installation

Vous pouvez récupérer la dernière version de **SSH** (2.3) sur le site www.ssh.com , plus précisément ftp.ssh.com/pub/ssh .C'est une archive tarball **ssh-2_3_0_tar.tar.gz**. La décompression de l'archive se fait dans un répertoire de travail en tapant:

tar xvfz ssh-2_3_0_tar.tar.gz

Cela va créer le répertoire **ssh-2.3.0**. Avant d'aller plus loin, il faudra installer les sources de **XFree86** si vous voulez pouvoir lancer des commandes X à distance. Pour une mandrake 7.1, à partir du CD d'install, on tapera:

rpm -ivh XFree86-devel-3.3.6-14mdk.i586.rpm

Dans le répertoire de **ssh**, vous devez maintenant taper:

./configure

Normalement tout devrait bien se passer, à présent, tapez

make

Normalement tout devrait bien se dérouler lors de la compil. A présent en tant que root, vous pouvez taper

make install

L'installation va installer le daemon **sshd** sous **/usr/local/sbin**

```
lrwxrwxrwx    1 root     root            5 Apr 18 18:09 sshd -> sshd2*
-rwxr-xr-x   1 root     root      1584323 Apr 18 18:09 sshd2*
```

et les exécutables suivants sous **/usr/local/bin**,

```
lrwxrwxrwx   1 root     root            4 Apr 18 18:09 ssh -> ssh2*
lrwxrwxrwx   1 root     root            8 Apr 18 18:09 ssh-add -> ssh-add2*
-rwxr-xr-x   1 root     root      1127385 Apr 18 18:09 ssh-add2*
lrwxrwxrwx   1 root     root           10 Apr 18 18:09 ssh-agent -> ssh-agent2*
-rwxr-xr-x   1 root     root      1074750 Apr 18 18:09 ssh-agent2*
lrwxrwxrwx   1 root     root           12 Apr 18 18:09 ssh-askpass -> ssh-askpass2*
-rwxr-xr-x   1 root     root       161022 Apr 18 18:09 ssh-askpass2*
-rwxr-xr-x   1 root     root        50853 Apr 18 18:09 ssh-dummy-shell*
lrwxrwxrwx   1 root     root           11 Apr 18 18:09 ssh-keygen -> ssh-keygen2*
-rwxr-xr-x   1 root     root      1042893 Apr 18 18:09 ssh-keygen2*
lrwxrwxrwx   1 root     root           10 Apr 18 18:09 ssh-probe -> ssh-probe2*
-rwxr-xr-x   1 root     root       399355 Apr 18 18:09 ssh-probe2*
lrwxrwxrwx   1 root     root           11 Apr 18 18:09 ssh-signer -> ssh-signer2*
-rws--x--x   1 root     root      1056511 Apr 18 18:09 ssh-signer2*
-rwxr-xr-x   1 root     root      1709052 Apr 18 18:09 ssh2*
```

Les fichiers de config sont sous **/etc/ssh2**:

```
-rw-------   1 root     root          828 Apr 16 08:36 hostkey
-rw-r--r--   1 root     root          697 Apr 16 08:36 hostkey.pub
-rw-r--r--   1 root     root          394 Apr 16 09:35 ssh2_config
```

-rw-r--r-- 1 root root 122 Apr 18 18:09 ssh_dummy_shell.out
-rw-r--r-- 1 root root 1323 Apr 16 10:06 sshd2_config

A noter que pendant le **make install**, des clés public et privée sont créées, si les clés existent déjà (présence du répertoire /etc/ssh2), aucune clé n'est recréée, on reprend celle existante. Voici les messages de création des clés :

```
if test '!' -d /etc/ssh2; then mkdir -p /etc/ssh2; fi
Generating 1024 bit host key.
Generating 1024-bit dsa key pair
  1 oOo.oOo
Key generated.
1024-bit dsa hostkey
Key is stored with NULL passphrase.
(This is not recommended.
 Don't do this unless you know what you're doing.
 If file system protections fail (someone can  access the keyfile),
 or if the super-user is malicious, your key can  be used without
 the deciphering effort.)
Private key saved to /etc/ssh2/hostkey
Public key saved to /etc/ssh2/hostkey.pub
```

Pour mémoire avec la version 2.1 de SSH sur ma Mandrake 7.0, j'avais cette erreur à la compil :

```
sshchx11.c: In function `ssh_channel_x11_send_request':
sshchx11.c:203: warning: unused variable `f'
sshchx11.c:201: warning: unused variable `line'
sshchx11.c: In function `ssh_channel_x11_process_request':
sshchx11.c:491: `XAUTH_PATH' undeclared (first use in this function)
sshchx11.c:491: (Each undeclared identifier is reported only once
sshchx11.c:491: for each function it appears in.)
make[4]: *** [sshchx11.o] Error 1
make[4]: Leaving directory
`/alphonse/linux/securite/ssh-secure-shell-2.1.0-noncommercial/apps/ssh'
make[3]: *** [all-recursive] Error 1
make[3]: Leaving directory
`/alphonse/linux/securite/ssh-secure-shell-2.1.0-noncommercial/apps/ssh'
make[2]: *** [all-recursive] Error 1
make[2]: Leaving directory
`/alphonse/linux/securite/ssh-secure-shell-2.1.0-noncommercial/apps'
make[1]: *** [all-recursive] Error 1
make[1]: Leaving directory
`/alphonse/linux/securite/ssh-secure-shell-2.1.0-noncommercial'
make: *** [all-recursive-am] Error 2
```

C'est qu'il n'a pas détecté **X** lors du **configure**, bien que vous ayez les biblio **X** (**libX11.a**) sous **/usr/X11R6/lib** et les headers dans **/usr/X11R6/include** et qu'il est sensé regarder dedans. Pour corriger cela, on peut a priori forcer les chemins comme arguments avec

configure, mais ça marche pas, tout ce que j'ai trouvé à faire c'est d'aller dans tous les **makefiles** et de définir la variable **XAUTH_PATH** mais **X_CFLAGS** et **X_LIBS**.

Cela concerne les fichiers suivants (en se plaçant à partir du répertoire **ssh-secure-shell-2.1.0-noncommercial**)

./Makefile
./apps/ssh/tests/Makefile
./apps/ssh/Makefile
./apps/Makefile
./include/Makefile
./include/sshconf.h
./sshconf.h

tous les Makefile sous les différents répertoires se trouvant sous.

./lib

Dans ces fichiers vous veillerez à faire en sorte que

XAUTH_PATH = /usr/X11R6/bin/xauth
X_LIBS=-L/usr/X11R6/lib
X_CFLAGS=-I/usr/X11R6/include

Maintenant la compil doit repartir de 0, tapez:

make clean

Et maintenant retentez le **make** et normalement ça devrait passer.

NOTE J'ai eu cette erreur de compil sur ma mandrake 7.0, par contre je n'ai absolument pas eu cette erreur sur ma mandrake 6.0. et encore moins sur la Red Hat 6.2.

6 Configuration

6.1 Présentation

SSH est constitué d'un daemon et d'un client, le daemon tourne sur la machine sur laquelle on peut se connecter à distance, et le client sur la machine distante.

6.1.1.1 Le daemon sshd

Le daemon **SSH** se configure avec le fichier **sshd2_config** qu'on trouve sous **/etc/ssh2**, voici son contenu détaillé:

sshd2_config
SSH 2.0 Server Configuration File
##

```
#  le "*" concerne toutes les machines
*:

## Paramètres généraux

# mode verbeux
   VerboseMode        no

# si vous utilisez un shell de type csh ou tcsh
# vous devez laisser cette variable à no pour ne pas
# avoir de problème (pb d'exécution de commandes dans le
# .cshrc ou le .tcshrc)
   AllowCshrcSourcingWithSubsystems    no

# c'est pour allouer un terminal (TTY) spécifique
# ça sert à rien a priori pour le moment
# (not yet implemented dixit le man)
   ForcePTTYAllocation     no

# type de log à utiliser, en clair les messages de log
# apparaîtront dans /var/log/secure
# car /etc/syslog.conf contient la ligne
# authpriv.*    /var/log/secure
   SyslogFacility         AUTH

## Paramètres réseau

# port par défaut
   Port               22

# adresse IP de l'interface d'où viendront les requêtes
# par défaut 0.0.0.0 c'est à dire toutes les interfaces
# il est judicieux de préciser l'interface d'où viendront les requêtes
# en cas d'adresse dynamique, laissez 0.0.0.0, ou alors
# reconfigurer sshd à chaque connexion
   ListenAddress       192.168.13.11

# Si à yes avant connexion sur une machine, on teste d'abord
# si le DNS la reconnait bien, en cas d'échec, on essaye
# même pas de se connecter
   RequireReverseMapping      no

# nombre de broadcast UDP max par seconde
# aucun par défaut
   MaxBroadcastsPerSecond     0

# on vérifie périodiquement si la machine distante n'est pas
# morte, au cas où, on a droit à un message d'avertissement
   KeepAlive          yes
```

```
# nombre de connexion max que le serveur peut accepter simultanément
# 0=illimité
   MaxConnections      1

## Paramètres de cryptographie

# Type de chiffrement (cipher) à utiliser, vous avez différent choix
# faire man sshd pour voir les autres
   Ciphers             AnyCipher

# méthode de hachage
   MACs                AnyMAC

## Paramètres utilisateur

# est-ce qu'à la connexion on affiche le petit mot contenu dans /etc/motd
   PrintMotd           yes

# est-ce qu'à la connexion on vérifie s'il y a du courrier ou pas
   CheckMail           yes

# répertoire contenant les clés utilisateurs
   UserConfigDirectory     "%D/.ssh2"

# vous pouvez vous authentifier (Paramètre AllowedAuthentifcations)
# avec la méthode "hostbased"
# qui repose sur des fichiers .shosts (équivalent à .rhosts) ou
# /etc/shosts.equiv (équivalent hosts.equiv)
# UserKnownHosts spécifie que le répertoire utilisateur à utiliser est
# ~/.ssh2/knownhosts
   UserKnownHosts          yes

## configuration des clés

# nom du fichier contenant la clé privée
   HostKeyFile         hostkey

# nom du fichier contenant la clé publique
   PublicHostKeyFile       hostkey.pub

# nom du fichier random
   RandomSeedFile          random_seed

# nom du fichier d'identification d'un fichier
   IdentityFile            identification

# nom du fichier contenant les personnes autorisées
# à se connecter
   AuthorizationFile       authorization
```

```
# Cette option spécifie si la méthode d'authentification utilisée
# doit être envoyé vers la machine distante
   AllowAgentForwarding       yes


## Tunneling configurations

# est-ce qu'on permet les commandes X à distance
   AllowX11Forwarding     yes

# Relais de TCP autorisé
   AllowTcpForwarding     yes


## Authentication methods

# nombre d'essai autorisé avec fermeture de la connexion
   PasswordGuesses        3

# type d'authentification utilisée (par clé, puis mot de passe (passphrase))
   AllowedAuthentications     publickey,password

# l'utilisateur sera identifié par la clé publique puis par le mot de passe de login
  RequiredAuthentications     publickey,password

## Définitions des machines autorisées ou pas

# le SHosts correspond aux machines listées dans les fichiers
# .shosts et shosts.equiv

#   AllowHosts         localhost, foobar.com, friendly.org
#   DenyHosts          evil.org, aol.com
#   AllowSHosts         trusted.host.org
#   DenySHosts          not.quite.trusted.org
#   IgnoreRhosts           no
#   IgnoreRootRHosts       no
# (the above, if not set, is defaulted to the value of IgnoreRHosts)


## Définitions des utilisateurs autorisés ou pas

#   AllowUsers         sj*,s[:isdigit:]##,s(jl|amza)
#   DenyUsers          skuuppa,warezdude,31373
#   DenyUsers          don@untrusted.org

# Pour permettre root de se loguer directement ou pas
# il est fortement recommandé de laisser ce paramètre à no
# pour se loguer root, loguer vous en tant que simple utilisateur
# puis su
   PermitRootLogin     no
```

SSH1 compatibility

 # si vous voulez garder la compatibilité vers SSH1
 # Ssh1Compatibility <set by configure by default>
 # Sshd1Path <set by configure by default>

Chrooted environment settings

 # on peut définir ici des utilisateurs ou des groupes qui ne pourront
 # pas à accèder à toute l'arborescence, exemple si je me connecte guest
 # j'aurai accès à /home/guest et puis c'est tout.

 # ChRootUsers ftp,guest
 # ChRootGroups guest

subsystem definitions
 # Définition du serveur ftp sécurisé

subsystem-sftp sftp-server

Maintenant pour lancer automatiquement **sshd** on va se servir du super serveur **inetd**, rajouter dans **/etc/inetd.conf** la ligne suivante:

ssh stream tcp nowait root /usr/sbin/tcpd /usr/local/sbin/sshd -i

Relander **inetd**

killall -HUP inetd

C'est bon le daemon **inetd** est lancé.

ATTENTION Vous devez veiller à ce que les clients devant se connecter ne soient pas listés dans **/etc/hosts.deny**.

NOTE Accessoirement si **SSH** devient votre outil de connexion préféré, désactiver le serveur **telnet** en mettant un **#** devant la ligne **telnet** dans le fichier **/etc/inetd.conf**.

6.2 Le client ssh

Le fichier de config du client **SSH** (se trouvant sur la machine distante et non pas sur le serveur) se trouve sous **/etc/ssh2** et a pour nom s**sh2_config**

ssh2_config
SSH 2.0 Client Configuration File
##

The "*" is used for all hosts, but you can use other hosts as
well.
 # concerne toutes les machines

```
# pour définir une machine en particulier il suffit de taper:
# machine:
*:

## Paramètres généraux

# mode verbeux
   VerboseMode        no

# type de prompt à affichier
   PasswordPrompt         "%U's password: "

# pour avoir le message "Authentification successful"
# si la connexion échoue
   AuthenticationSuccessMsg    yes


## Paramètres réseau

# numéro du port
   Port               22

# Il paraît que mettre ça à Yes permet d'améliorer les performances
# réseau, on peut les croire sur parole :-)
   NoDelay            no

# on vérifie régulièrement si le serveur n'est pas down ou la connexion
# rompue pour avertir à temps l'utilisateur
   KeepAlive          yes

## config cryptographie

# type de chiffrement (cipher) à utiliser
# man ssh2_config pour avoir la liste complète
   Ciphers            AnyStdCipher

# type de hachage
   MACs               AnyMAC

# en mettant à "ask" les machines ne seront archivées
# dans $HOME/.ssh2/hostkeys qu'après votre accord
   StrictHostKeyChecking          ask

## User public key authentication

# nom du fichier d'identification
   IdentityFile           identification

# nom du fichier contenant les personnes autorisées
   AuthorizationFile      authorization
```

```
# nom du fichier random pour générer les clés
   RandomSeedFile         random_seed

## compatibilité SSH1
       Ssh1Compatibility      no
   Ssh1AgentCompatibility     none

# type d'authentification utilisée (par clé, puis mot de passe (passphrase))
   AllowedAuthentications     publickey,password

# For ssh-signer2 (only effective if set in the global configuration
# file, usually /etc/ssh2/ssh2_config)
#   DefaultDomain          foobar.com
#   SshSignerPath          ssh-signer2

# pour forwarder des agents d'authentifcation
ForwardAgent yes

# pour forwarder X11
Forward X11 yes
```

7 Utilisation

7.1 Création et échange des clés

Considérons toujours l'utilisateur **olivier** sur la machine **obelix** qui veut autoriser l'utilisateur **veronique** de la machine **asterix** à se connecter sur son compte. L'utilisateur **olivier** doit d'abord créer son couple de clé:

```
[olivier@obelix olivier]$  ssh-keygen
Generating 1024-bit dsa key pair
  4 .oOo..oOo.oO
Key generated.
1024-bit dsa, olivier@obelix.armoric.bz, Tue Apr 18 2000 20:38:41 +0200
Passphrase :
Again      :
Private key saved to /home/olivier/.ssh2/id_dsa_1024_a
Public key saved to /home/olivier/.ssh2/id_dsa_1024_a.pub
```

Il est nécessaire de rentrer un mot de passe (passphrase), à noter que ce mot de passe peut être une phrase avec des blancs. Les clés ont été sauvegardés sous **/home/olivier/.ssh2**, la clé publique se nonmme **id_dsa_1024_a.pub** et la clé privée **id_dsa_1024_a**. A présent **olivier** doit créer un fichier qui va identifier sa clé privée. Pour cela, tapez:

```
cd ~/.ssh2
echo "IdKey id_dsa_1024_a" > identification
```

Ceci permet éventuellement de changer le nom de la clé privée. L'utilisateur **veronique** doit faire la même chose sur **asterix** (lancement de **ssh-keygen** et création du fichier **identification**), le mot de passe ne doit évidemment pas être le même.

On va procéder maintenant à l'échange des clés, l'utilisateur **veronique** va donner sa clé publique **id_dsa_1024_a.pub** à **olivier** (par email, copie de fichiers, comme vous voulez). L'utilisateur **olivier** va la récupérer, la renommer pour bien l'identifier **veronique-asterix.pub** par exemple, et placer ce fichier dans **/home/olivier/.ssh2**. A présent **olivier** doit créer un fichier **authorization** (à placer sous **/home/olivier/.ssh2**) qui contiendra:

Key veronique-asterix.pub

Voyons par exemple ce que contient le **.ssh2** de l'utilisateur **olivier**:

```
-rw-r--r--   1 olivier  hoarau       21 Apr 18 20:41 authorization
-rw-------   1 olivier  hoarau      891 Apr 18 20:35 id_dsa_1024_a
-rw-r--r--   1 olivier  hoarau      752 Apr 18 20:35 id_dsa_1024_a.pub
-rw-r--r--   1 olivier  hoarau       20 Apr 18 20:38 identification
-rw-------   1 olivier  hoarau      512 Apr 23 08:06 random_seed
-rw-------   1 olivier  hoarau      751 Apr 18 20:39 vero-asterix.pub
```

Et celui de **veronique**:

```
-rw-------   1 veroniqu  hoarau      891 Apr 18 20:35 id_dsa_1024_a
-rw-r--r--   1 veroniqu  hoarau      752 Apr 18 20:35 id_dsa_1024_a.pub
-rw-r--r--   1 veroniqu  hoarau       20 Apr 18 20:38 identification
-rw-------   1 veroniqu  hoarau      512 Apr 23 13:41 random_seed
```

C'est bon maintenant tout est en place. Il ne manque qu'une chose, **olivier** doit communiquer à **veronique** son mot de passe de login sur **obelix**.

NOTE Pour changer de mot de passe au niveau de **ssh-keygen**, il suffit de taper:

ssh-keygen -p

7.2 Connexion simple avec ssh

Une fois l'échange des clés effectuées, **veronique** sur **asterix** va se connecter sur **obelix** en utilisant le compte d'**olivier**:

```
[veronique@asterix]$ ssh2 -l olivier obelix
Host key not found from database.
Key fingerprint:
xuteh-mosec-bomig-vuzib-zabas-roves-mehug-vedez-dobor-zutek-roxax
You can get a public key's fingerprint by running
% ssh-keygen -F publickey.pub
on the keyfile.
Are you sure you want to continue connecting (yes/no)? yes
Host key saved to /home/veronique/.ssh2/hostkeys/key_22_obelix.pub
host key for obelix, accepted by olivier Sat Apr 22 2000 08:05:42 +0200
Passphrase for key "/home/veronique/.ssh2/id_dsa_1024_a" with comment "1024-bit
dsa, veronique@asterix.armoric.bz, Tue Apr 18 2000 20:35:04 +0200":
olivier's password:
Authentification successful
Last login: Sat Apr 22 2000 08:08:22 +0200
```

You have mail.
[olivier@obelix olivier]$

C'est bon on est connecté, à noter:

Host key not found from database.
Key fingerprint:
xuteh-mosec-bomig-vuzib-zabas-roves-mehug-vedez-dobor-zutek-roxax
You can get a public key's fingerprint by running
% ssh-keygen -F publickey.pub
on the keyfile.
Are you sure you want to continue connecting (yes/no)? yes
Host key saved to /home/veronique/.ssh2/hostkeys/key_22_obelix.pub

Ca c'est quand on se connecte la première fois, il va créer un répertoire **hostkeys** contenant la clé publique d'**olivier**.

host key for obelix, accepted by olivier Sat Apr 22 2000 08:05:42 +0200

Phase d'identification réussie, après comparaison entre la clé publique détenue par **olivier** et la clé privée de **véronique**.

Passphrase for key "/home/veronique/.ssh2/id_dsa_1024_a" with comment "1024-bit dsa, veronique@asterix.armoric.bz, Tue Apr 18 2000 20:35:04 +0200":

L'utilisateur **veronique** doit rentrer son password.

olivier's password:

Elle doit rentrer le mot de passe de login d'**olivier** sur **obelix**. C'est bon **veronique** est connectée sur **obelix**. La prochaine fois qu'elle se connectera, elle aura uniquement les messages suivants:

[veronique@asterix veronique]$ ssh2 -l olivier obelix
Passphrase for key "/home/veronique/.ssh2/id_dsa_1024_a" with comment "1024-bit dsa, veronique@asterix.armoric.bz, Tue Apr 18 2000
20:35:04 +0200":
olivier's password:
Last login: Sun Apr 23 2000 07:10:30 +0200
You have mail.
[olivier@obelix olivier]$

NOTE Il est même plus nécessaire de rajouter **-l olivier** lors des connexions suivantes, le client **ssh** considérera par défaut qu'on essaye de se connecter en utilisant le compte **olivier**. Pour spécifier un autre compte vous devez rajouter **-l login**.

7.3 Lancement une commande à distance

Imaginons que **veronique** veuille lancer la commande **df** sur **obelix**, il suffit de taper sur **asterix**:

[veronique@asterix veronique]$ ssh obelix df
Passphrase for key "/home/veronique/.ssh2/id_dsa_1024_a" with comment "1024-bit dsa, veronique@asterix.armoric.bz, Tue Apr 18 2000 20:35:04 +0200":
olivier's password:
Filesystem 1k-blocks Used Available Use% Mounted on
/dev/sdb1 149712 108148 33834 76% /
/dev/sda1 991995 830487 110258 88% /alphonse
/dev/sdb3 246879 98364 135767 42% /roger
/dev/sdb4 99136 64242 29774 68% /home
/dev/sdc1 938312 620276 270372 70% /usr
/dev/sdb5 496627 298672 172305 63% /usr/local
Connection to obelix closed.
[veronique@asterix veronique]$

Après exécution de la commande, la connexion est automatiquement coupée.

7.4 Lancement d'une commande X à distance

Voici ce qu'on devrait avoir pour un fonctionnement normal :

[veronique@asterix .ssh2]$ ssh -l olivier obelix
Passphrase for key "/home/veronique/.ssh2/id_dsa_1024_a" with comment "1024-bit dsa, veronique@asterix.armoric.bz, Tue Apr 18 2000 20:35:04 +0200":
olivier's password:
Last login: Thu May 11 2000 19:02:23 +0200
You have mail.
[olivier@obelix olivier]$ env | grep DISPLAY
DISPLAY=obelix:10.0
[olivier@obelix olivier]$ xauth list
obelix.armoric.bz:0 MIT-MAGIC-COOKIE-1 33d2dc0b5bead74165c894334edb7908
obelix/unix:0 MIT-MAGIC-COOKIE-1 33d2dc0b5bead74165c894334edb7908
obelix.armoric.bz/unix:0 MIT-MAGIC-COOKIE-1 8174acc11b7ab2947ae68285095df932
obelix.armoric.bz:10 MIT-MAGIC-COOKIE-1 f481da2b09cd0935937b07acc7fabf2e
obelix/unix:10 MIT-MAGIC-COOKIE-1 f481da2b09cd0935937b07acc7fabf2e
[olivier@olivier olivier]$

Remarquer bien que sur le client **SSH**, **DISPLAY** est positionné à **nom-serveur-sshd:10.0**, c'est tout à fait normal, et là si vous lancez une commande **X** elle s'affichera bien à l'écran d'**asterix** et non pas d'**obelix**.

xauth permet d'ajouter, éditer les authorisations pour se connecter à un serveur X, concrètement **sshd** va appeler **xauth** pour que les commandes **X** puissent s'afficher sur le client.

7.5 Copier des fichiers

On retrouve une syntaxe complètement équivalente à la commande UNIX **rcp**. Admettons que **veronique** veuille copier le fichier **toto** se trouvant dans sa home directory dans le répertoire **/tmp** d'**obelix**. Il suffit de taper:

[veronique@asterix veronique]$ scp ~/toto obelix:/tmp
Passphrase for key "/home/veronique/.ssh2/id_dsa_1024_a" with comment "1024-bit dsa, veronique@asterix.armoric.bz, Tue Apr 18 2000 20:35:04 +0200":
olivier@obelix's password:
toto | 0B | 0.0 kB/s | TOC: 00:00:01 | 100%

Maintenant on veut récupérer le fichier **titi** se trouvant dans la home directory d'**olivier**, pour le mettre sous le **/tmp** d'**asterix**

[veronique@asterix veronique]$ scp obelix:/home/olivier/titi /tmp
Passphrase for key "/home/veronique/.ssh2/id_dsa_1024_a" with comment "1024-bit dsa, veronique@asterix.armoric.bz, Tue Apr 18 2000 20:35:04 +0200":
olivier@obelix's password:
titi | 0B | 0.0 kB/s | TOC: 00:00:01 | 100%

Maintenant on va récupérer tout le répertoire **php3** d'**olivier** pour le placer dans le répertoire courant (.)

[veronique@asterix veronique]$ scp -r obelix:/home/olivier/php3 .
Passphrase for key "/home/veronique/.ssh2/id_dsa_1024_a" with comment "1024-bit dsa, veronique@asterix.armoric.bz, Tue Apr 18 2000 20:35:04 +0200":
olivier@obelix's password:
php3/toto | 0B | 0.0 kB/s | TOC: 00:00:01 | 100%
php3/ici/bof | 0B | 0.0 kB/s | TOC: 00:00:01 | 100%

Autres options intéressantes de **scp**:

-v verbose pour avoir un max de commentaires, notamment pour l'établissement de la connexion et l'identification.
-p pour préserver les droits (conseillé)
-c pour changer le type de cryptage

7.6 FTP sécurisé

Avec **SSH**, vous disposez même qu'un équivalent **ftp** mais sécurisé, il suffit de taper:

[veronique@asterix veronique]$ sftp obelix
Passphrase for key "/home/veronique/.ssh2/id_dsa_1024_a" with comment "1024-bit dsa, veronique@asterix.armoric.bz, Tue Apr 18 2000 20:35:04 +0200":
olivier@obelix's password:
sftp> help
Secure FTP client Sftp2
Copyright (c) 1999, 2000 SSH Communications Security, Finland.

Type 'help <topic>', where <topic> is one of the following commands:

open localopen close quit cd
lcd pwd lpwd ls lls
get mget put mput rm
lrm mkdir lmkdir rmdir lrmdir
help

sftp>

On dispose des commandes standards **ftp** pour récupérer ou envoyer des fichiers.

8 Authentification basée sur le nom de la machine

En plus de l'authentification de l'utilisateur par clé (**publickey**), et par mot de passe (**password**), on peut rajouter une authentification de la machine qui cherche à se connecter (**hostbased**). Pour cela il est nécessaire d'obtenir la clé publique de la machine, qu'on peut trouver sous **/etc/ssh2** et qui a pour nom **hostkey.pub**.
La première chose à faire est de modifier le fichier de config du daemon **SSH**, à savoir **/etc/ssh2/sshd2_config**, on modifiera les lignes suivantes ainsi:

AllowedAuthentications publickey,password,hostbased
RequiredAuthentications publickey,password,hostbased

Prenons l'exemple d'**olivier** sur la machine **obelix** qui veut autoriser **veronique** à se connecter sur son compte mais à partir uniquement de la machine **asterix**. Pour être sûr que **veronique** se connecte bien d'**asterix** et non pas d'une machine qui se fait passer pour **asterix**, **olivier** doit obtenir la clé publique d'**asterix** (**/etc/ssh2/hostkey.pub** sous **asterix**). Une fois obtenue, cette clé publique doit être placée sous le répertoire d'**olivier ~/.ssh2/knownhosts** (répertoire éventuellement à créer) avec un nom bien particulier, à savoir **xxxxyyyy.pub** avec **xxxx** le nom complet (FQDN) de l'hôte, soit **asterix.armoric.bz**, et **yyyy** le nom de l'algorithme pour générer la **hostkey**, vous avez le choix entre **ssh-dss** (par défaut) et **ssh-rsa**. En clair **olivier** va nommer le fichier **hostkey.pub** d'**asterixasterix.armoric.bz.ssh-dss.pub**. Ce n'est pas fini, maintenant **olivier** doit créer dans sa homedirectory un fichier **.shosts** contenant

asterix.armoric.bz veronique

ATTENTION Dans ce fichier vous devez veiller à ce que le nom de la machine soit complet (avec nom de domaine).

Une autre solution consiste, en tant que **root**, à placer le fichier **asterix asterix.armoric.bz.ssh-dss.pub** sous **/etc/ssh2/knownhosts** (répertoire éventuellement à créer) d'obelix et de rajouter:

asterix.armoric.bz veronique

Au fichier **/etc/shosts.equiv**.

Si l'utilisateur a un fichier **xxxxyyyy.pub** sous **~/.ssh2/knownhosts** et qu'il existe un fichier du même nom sous **/etc/ssh2/knownhosts**, c'est le fichier utilisateur qui sera utilisé par défaut. Si vous voulez au contraire que ce soit le fichier sous **/etc/ssh2/knownhosts** qui soit utilisé, rajoutez (ou modifiez) la ligne suivante au fichier **sshd2_config**

UserKnownHosts no

De même le fichier **~/.shosts** prévaut sur **/etc/shosts.equiv**, pour que les **.shosts** soient ignorés et que seul le fichier **shosts.equiv** soit utilisé, rajoutez (ou modifiez) au fichier **sshd2_config**:

IgnoreRhosts yes

NOTE N'oubliez pas que **SSH** est lancé par **inetd** et donc bénéfie des services des **TCP_WRAPPERS**, par conséquent vous pouvez restreindre l'accès avec les fichiers **/etc/hosts.allow** et **/etc/hosts.deny**

NOTE Si vous autorisez vos utilisateurs à se servir des **.shosts**, vous pouvez cependant interdire certaines machines à se connecter chez vous, en rajoutant dans **sshd2_config**

DenySHosts machine1 machine2

9 Les tunnels sécurisés

La notion de tunnel ("Tunneling" en anglais) est très importante pour **SSH**, elle permet d'établir une connexion sécurisée (tunnel) entre l'hôte et un hôte distant, et dans ce tunnel on va y faire transiter des communications qui à la base ne sont pas sécurisées comme **POP** ou **telnet**. Accessoirement comme les données qui transitent dans le tunnel sont compressés, le transfert est plus rapide. La syntaxe est la suivante :

login = nom d'utilisateur sur la machine distante
remote= nom de la machine distante

ssh -f login@remote -L port local:remote:port distant [commandes supplémentaires]

On va prendre comme exemple un tunnel sécurisé pour faire circuler le courrier récupéré par **POP**, tout le monde sait que le mot de passe circule en clair sur le réseau et les mails sont évidemment non cryptés, ni même compressés.

La contrainte est qu'il faut qu'un serveur **SSH** tourne sur le serveur **POP** pour que ça marche, et qu'il y ait eu échange de clé publique (si authentification basée sur clé), la syntaxe est la suivante:

ssh -C -f login@serveurpop -L 1234:serveurpop:110 sleep 50

-C active la compression des données qui transitent le tunnel

-f Une fois que la connexion est activée, **ssh** bascule en tâche de fond (background ou &) pour qu'un autre programme puisse se lancer

login le nom d'utilisateur sur le serveur **pop**

serveurpop le nom du serveur **pop**

-L 1234:serveurpop:110 le mail va aboutir sur le port 1234 local

sleep 50, le tunnel est ouvert, mais sera fermé au bout de 50 secondes si aucune commande n'a essayer d'emprunter le tunnel, vous pouvez éventuellement augmenter ce délai ou l'abaisser.

Démonstration avec un client **idefix** sur le serveur **asterix** (**breizland.bz** étant le nom du domaine privé) sur lequel tourne un serveur **pop** sur le port 110, en local le mail transitera par le port libre 1234.

[olivier@idefix olivier]$ ssh -f -C olivier@asterix -L 1234:asterix:110 sleep 50
Passphrase for key "/home/olivier/.ssh2/id_dsa_1024_a" with comment "1024-bit dsa, olivier@zoulou.kervao.fr, Tue Aug 29 2000 21:08:08 +0200":
olivier's password:
Authentication successful.
[olivier@idefix olivier]$

Maintenant pour voir si le tunnel est bien en place, dans les 50s on tape :

[olivier@idefix olivier]$ telnet localhost 1234
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
+OK POP3 asterix.breizland.bz v7.64 server ready

Pour automatiser la récupération du mail en passant par un tunnel sécurisé, si on utilise **fetchmail** pour récupérer le courrier, voici la syntaxe du fichier **.fetchmailrc**

poll localhost with protocol pop3 and port 1234:
preconnect "ssh -C -f olivier@asterix -L 1234:asterix:110 sleep 50"
password mot-de-passe;

On ne se connecte pas directement au serveur **POP**, mais en local (**localhost**) en utilisant le port 1234. La directive **preconnect** établit le tunnel sécurisé à chaque fois que fetchmail est appelé, 50 secondes sont largement suffisantes pour que **fetchmail** utilise le tunnel.

A chaque fois que vous lancerez **fetchmail**, on vous demandera donc votre password **SSH** pour authentification, c'est pas très pratique si vous exécutez **fetchmail** en tâche de fond.

10 Le client windows SSHWin-2_3_0

Sur le même site de téléchargement que la version pour UNIX, vous pouvez récupérer un client marchant pour windows en l'occurrence **SSHWin-2_3_0.exe**. L'installation sous windows est très simple il suffit de lancer cet exécutable, laissez vous guider lors de l'installation, par défaut le répertoire d'install est **c:\Program Files\SSH Communications**

Security\SSH Secure Shell, vous pouvez le changer. Vous pouvez aussi choisir d'installer les composants suivants :

- **Desktop Icons** (icônes du bureau),
- **Documentation**,
- **scp2** (commande copie de fichier sécurisée, non installé par défaut),
- **Add scp2 to path** (pour rajouter scp2 au PATH).

Rebooter la machine même si c'est pas indiqué.

Pour lancer le client SSH, dans le menu **Démarrer->SSH Secure Shell** puis **Secure Shell Client**

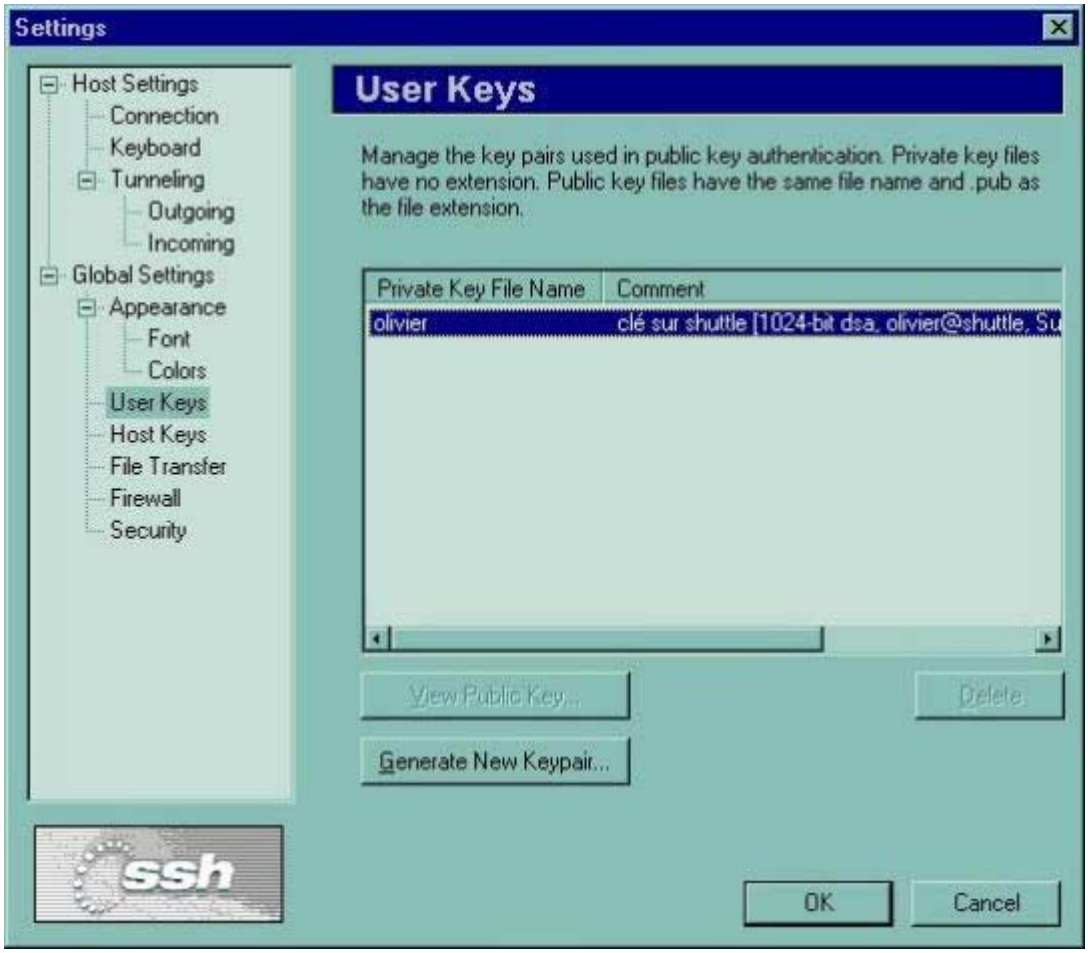
Voilà ce que ça donne au lancement de **SSHWin 2.3**. La première chose à faire est de créer les clés publique et privée, pour cela, on va cliquer sur l'icône avec les engrenages (ou alors **Edit->Settings**)

Pour générer le couple de clé, il suffit de cliquer dans la fenêtre **Settings** sur **User Keys**, puis **Generate New Keypair**, à un moment il vous sera demandé de saisir:

- **File Name** : nom du fichier contenant la clé **cle** par exemple
- **Comment** : commentaire quelconque qui apparaîtra dans le fichier de clé publique que vous allez distribuer
- **Passphrase** : phrase mot de passe (à saisir deux fois)

Vous pouvez ensuite "uploader" votre clé publique, vous pouvez ignorer cette étape.

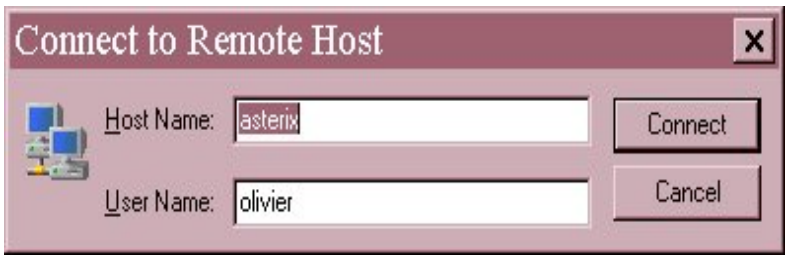
En considérant que vous avez installé **Win SSH** sous **c:\program files\SSH Secure Shell** vous trouverez la clé publique de l'utilisateur **toto** sous c**:\program files\Users\login-de-connexion\UserKeys** avec pour nom **cle** pour la clé privée et **cle.pub** pour la clé publique.



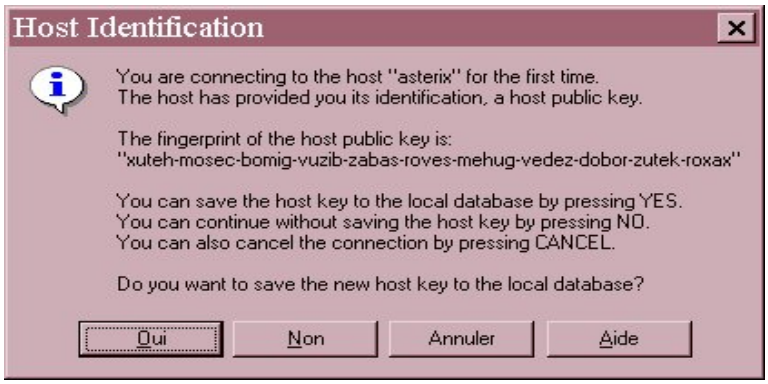
Pour reprendre notre exemple, à présent l'utilisateur windows doit donner sa clé publique à **olivier** sur **asterix**. Pour cela par un moyen ou un autre le **cle.pub** doit se retrouver dans le répertoire **.ssh2** d'**olivier**. L'utilisateur **olivier** sous **asterix** doit maintenant éditer son **~/.ssh2/authorization** et rajouter en fin de fichier:

Key cle.pub

Pour se connecter, il suffit maintenant de cliquer sur l'icône représentant un ordinateur (ou alors **File->Connect**) Saisissez le nom de l'ordinateur distant et le nom de login.



Si on se connecte pour la première fois, on a le message suivant, acquittez en cliquant sur **Oui** (C'est rigolo, le texte est en anglais, mais les boutons en français).

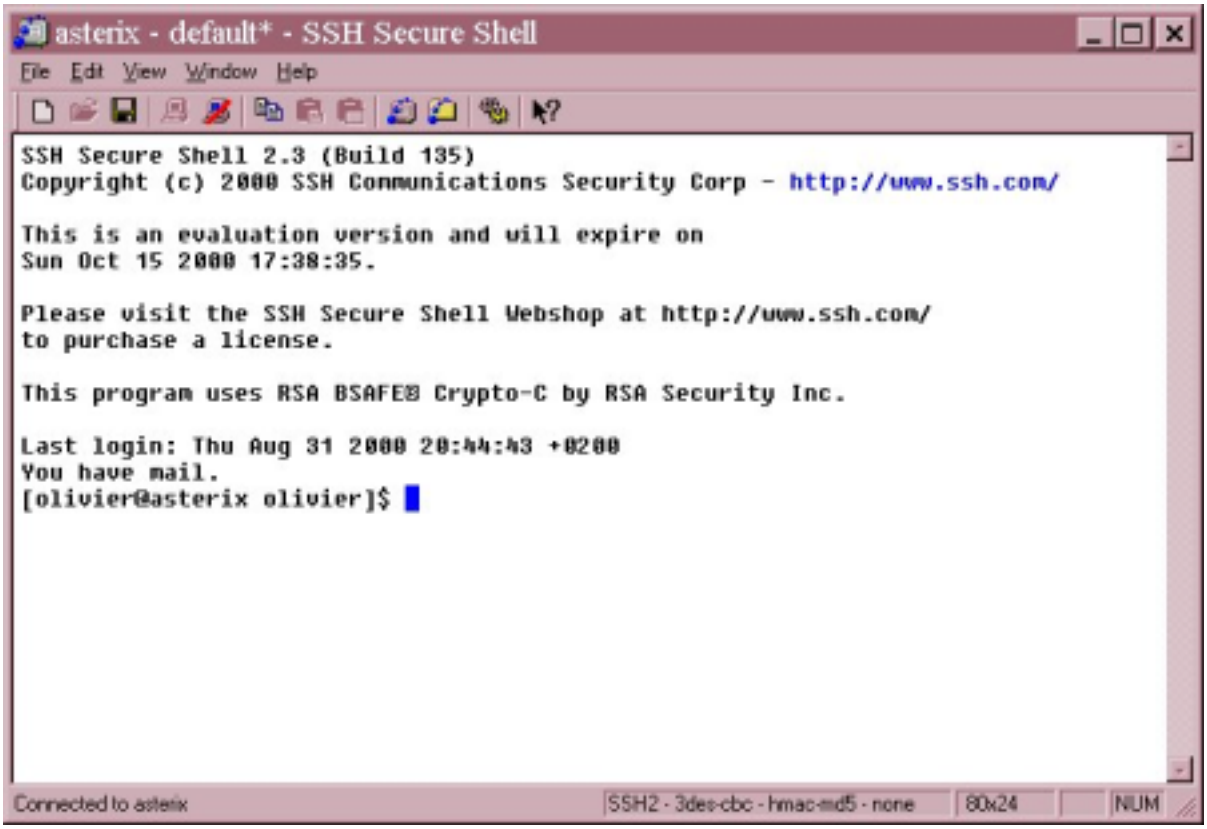


Maintenant rentrez le mot de passe (phrase password) que vous avez défini lors de la génération des clés

Saisissez le mot de passe d'**olivier** sous **asterix**

Ca y est vous êtes connecté. A priori il semblerait qu'il soit possible de lancer des commandes X (**Edit->Settings->Tunneling**), mais je pense qu'il faut disposer d'un serveur X sous Windows pour que ça marche.

Pour ouvrir un nouveau shell, il suffit de cliquer dans **Windows** puis **New Terminal.**



Vous disposez d'une fonction de transfert de fichier sécurisé (équivalent **sftp**), pour cela il suffit de cliquer sur l'icône représentant un dossier (ou alors **Windows->New File Transfer**).

Je ne rentrerai pas dans le détail du fonctionnement de cette fenêtre