

Sécuriser ses connexions avec OpenSSH4.3p2

Olivier Hoarau (olivier.hoarau@funix.org)

V2.3 du 26 juillet 2006

1	Historique.....	2
2	Préambule.....	2
3	Présentation.....	2
4	Comment ça marche.....	3
5	Installation.....	6
5.1	Installation OpenSSL.....	6
5.2	Installation d'OpenSSH.....	7
6	Configuration.....	10
6.1	Configuration du serveur.....	10
6.2	Configuration du client.....	13
7	Lancement du daemon sshd.....	15
7.1	Présentation.....	15
7.2	Lancement par xinetd.....	15
7.3	Lancement en standalone.....	16
7.3.1	Lancement en standalone avec Mandriva.....	16
7.3.2	Lancement en standalone avec ubuntu	17
8	Création et échange des clés.....	18
9	Utilisation.....	20
9.1	Connexion simple avec ssh.....	20
9.2	Lancement une commande à distance.....	21
9.3	Lancement d'une commande X à distance.....	21
9.4	Copier des fichiers.....	22
10	Les clients windows.....	23
10.1	Présentation.....	23
10.2	OpenSSH.....	23
10.3	PuTTY.....	24
10.3.1	Présentation.....	24
10.3.2	Installation.....	24
10.3.3	Utilisation.....	24

1 Historique

V2.3	26.07.06	Installation sous ubuntu, passage à OpenSSL 0.9.8b
V2.2	30.04.06	Passage à OpenSSH 4.3p2 et OpenSSL 0.9.8
V1.9	27.11.04	Passage à OpenSSH 3.9p1 et OpenSSL 0.9.7e
V1.7	06.01.04	Passage à OpenSSH 3.7.1p2 et OpenSSL 0.9.7c
V1.6	15.09.02	Passage à OpenSSH 3.4p1
V1.5	07.07.02	Passage à OpenSSH 3.2.3p1, OpenSSL 0.9.6d
V1.4	16.12.01	passage à 3.0.1p1, modification de certains paramètres des fichiers de conf, mise à jour des clients windows
V1.3	11.3.01	Passage à la version 2.5.1p11, rajout paragraphe client SSH2 pour windows
V1.2	10.12.00	Correction du problème de Xforwarding
V1.1	3.12.00	Passage à 2.3.0p11
V1.0	20.8.00	Création du document

2 Préambule

Ce document présente **OpenSSH** qui est une alternative sécurisée à des outils de connexion comme **telnet**.

La dernière version de ce document est téléchargeable à l'URL <http://www.funix.org>. 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.

Ce document ne peut pas être utilisé dans un but commercial sans le consentement de son auteur. Ce document vous est fourni "dans l'état" sans aucune garantie de toute sorte, l'auteur ne saurait être tenu responsable des quelconques misères qui pourraient vous arriver lors des manipulations décrites dans ce document.

3 Présentation

OpenSSH 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**).

OpenSSH 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.

4 Comment ça marche

On prend comme exemple **olivier** sur **obelix** et **veronique** sur **asterix**. **OpenSSH** 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 **OpenSSH** vous pouvez combiner les caractéristiques des deux protocoles. Voilà les différences entre les deux versions.

SSH1

Il y a quatre 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/shosts.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 vérifie si la clé publique 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.

Et la dernière méthode consiste à donner le mot de passe du compte sur lequel on se connecte, le mot de passe circule crypté sur le réseau.

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 génère 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 changé. 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**.

SSH2

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** (tombé dans le domaine public en septembre 2000).

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, entre autres, 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.

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 génère clé publique et privée sur **asterix**

olivier génère 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** (éventuellement)

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

Dans cette page on présentera **OpenSSH** pour un fonctionnement uniquement en utilisant **SSH2**, on passera sous silence tout ce qui concerne **SSH1**, notamment pour l'authentification en utilisant les fichiers **.rhosts**, **.shosts**, **hosts.equiv** et **shosts.equiv**.

5 Installation

5.1 Installation OpenSSL

OpenSSL est un projet qui a pour but de proposer des outils logiciels contenus dans une bibliothèque qui implémente les protocoles Secure Sockets Layer (**SSL** v2 et v3) et Transport Layer Security (**TLS** v1).

Assurez vous d'abord qu'**OpenSSL** n'est pas déjà installé sur votre système, en tapant:

```
rpm -qa | grep -i openssl
```

Vous pouvez éventuellement supprimer les packages obtenus pour installer la version tarball généralement plus récente, avec la commande (**rpm -e nom-du-package**).

S'il y a une tonne de dépendances, ce n'est pas grave vous n'êtes pas obligé de supprimer le package, ça marchera très bien en faisant cohabiter la version mdk et la version tarball. Par contre pour pouvoir utilisé la dernière version d'**OpenSSH** vous devez utiliser aussi la dernière version d'**OpenSSL** qu'on peut récupérer à l'URL www.openssl.org sous la forme d'une archive tarball **openssl-0.9.8b.tar.gz**, qu'on va décompresser en tapant:

```
tar xvfz openssl-0.9.8b.tar.gz
```

La commande se termine par le message

```
tar: Un bloc zéro solitaire repéré à 31800
```

N'en tenez pas compte. Cela va nous créer un répertoire **openssl-0.9.8** dans lequel vous taperez

```
./config
```

Puis

```
make
```

Il est nécessaire avant d'aller plus loin de tester la biblio, pour cela préalablement si vous ne l'avez pas vous devez installer la commande **bc** (calculatrice en ligne), contenue dans une mandrake dans le package **bc**.

Tapons à présent:

make test

Vous ne devriez pas avoir d'erreur. Et enfin, en tant que root

make install

Cela va installer les fichiers (bibliothèques, binaires, man, doc, ...) de **SSL** dans le répertoire **/usr/local/ssl**.

si vous tenez absolument à installer les packages de votre distrib, pour info sous ubuntu les voici

libssl-dev - SSL development libraries, header files and documentation

libssl0.9.8 - SSL shared libraries

5.2 Installation d'OpenSSH

OpenSSH est intégré dans la Mandriva, on va utiliser la dernière version disponible sur le site officiel d'**OpenSSH**. Vous devez d'abord vérifier qu'**OpenSSH** n'est pas déjà installé sur votre système :

rpm -qa | grep -i openssh

Vous supprimez avec la commande **rpm -e nomdupackage**.

Vous pouvez récupérer la dernière version de **OpenSSH** sur le site www.openssh.com. C'est une archive tarball **openssh-4.3p2.tar.gz** . La décompression de l'archive se fait dans un répertoire de travail en tapant:

tar xvfz openssh-4.3p2.tar.gz

Cela va créer le répertoire **openssh-4.3p2**. Vous devez maintenant installer le package **XFree86-devel** (ou **libxorg-x11-devel** ou bien encore **xorg-dev** sous ubuntu) pour pouvoir exporter X, ainsi que le package **zlib1-devel** (**zlib1g-dev** sous ubuntu). Maintenant dans le répertoire d'**OpenSSH**, vous devez maintenant taper:

./configure --with-ssl-dir=/usr/local/linux/openssl-0.9.8 --with-ldflags=-ldl

Vous devez indiquer le chemin des sources **openssl** conforme à votre système. Dans le cas où vous utilisez le package **openssl** de la Mandriva ou de la ubuntu il faudra installer le package **openssl-devel** et tapez simplement **./configure**

On obtient en fin de commande un récapitulatif des options de compilation

OpenSSH has been configured with the following options:

User binaries: /usr/local/bin

System binaries: /usr/local/sbin

Configuration files: /usr/local/etc

Askpass program: /usr/local/libexec/ssh-askpass

Manual pages: /usr/local/man/manX

PID file: /var/run

Privilege separation chroot path: /var/empty

sshd default user PATH: /usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin

Manpage format: doc

PAM support: no

KerberosV support: no

Smartcard support: no

S/KEY support: no

TCP Wrappers support: no

MD5 password support: no

libedit support: no

IP address in \$DISPLAY hack: no

Translate v4 in v6 hack: yes

BSD Auth support: no

Random number source: OpenSSL internal ONLY

Host: i686-pc-linux-gnu

Compiler: gcc

Compiler flags: -g -O2 -Wall -Wpointer-arith -Wno-uninitialized

Preprocessor flags: -I/usr/local/linux/openssl-0.9.8

Linker flags: -L/usr/local/linux/openssl-0.9.8

Libraries: -lresolv -lcrypto -lutil -lz -lnsl -lcrypt

Puis

make

A présent en tant que **root**, vous pouvez taper

On doit maintenant créer un utilisateur **sshd**, on tapera les commandes suivantes en tant que root

mkdir /var/empty

chown root:sys /var/empty

chmod 755 /var/empty

groupadd sshd

useradd -g sshd -c 'sshd privsep' -d /var/empty -s /bin/false sshd

/var/empty ne doit contenir aucun fichier. Ceci permet de réaliser certaines opérations sans être root.

make install

A la fin des commandes les clés de votre machine sont générées dans le cas où celles ci n'existent pas sous **/usr/local/etc**

Generating public/private rsa1 key pair.

Your identification has been saved in /usr/local/etc/ssh_host_key.

Your public key has been saved in /usr/local/etc/ssh_host_key.pub.

The key fingerprint is:

46:06:b1:dd:a1:84:fa:36:d0:77:b1:d8:75:d6:bf:03 root@asterix.kervao.fr

Generating public/private dsa key pair.

Your identification has been saved in /usr/local/etc/ssh_host_dsa_key.

Your public key has been saved in /usr/local/etc/ssh_host_dsa_key.pub.

The key fingerprint is:

0e:6f:19:83:12:96:a6:d1:4d:2b:d6:40:93:39:05:98 root@asterix.kervao.fr

Generating public/private rsa key pair.

Your identification has been saved in /usr/local/etc/ssh_host_rsa_key.

Your public key has been saved in /usr/local/etc/ssh_host_rsa_key.pub.

The key fingerprint is:

6e:1a:ef:b6:2b:92:26:ea:6f:34:72:e4:59:5c:70:4f root@asterix.kervao.fr

/usr/local/sbin/sshd -t -f /usr/local/etc/sshd_config

Cela va créer les clés publiques et privées de la machine qui seront placées par défaut dans **/usr/local/etc** . Plus précisément on trouve la clé privée **ssh_host_key** si on utilise la gestion de clés type **RSA (SSH1)** et la publique associée **ssh_host_key.pub** et la clé privée **ssh_host_dsa_key** si on utilise la gestion de clés type **DSA (SSH2)** et la clé publique associée **ssh_host_dsa_key.pub**.

NOTES - En cas d'upgrade les clés de la machine ne seront pas écrasées de même que les fichiers de conf.

- Dans le cas où les clés de la machine sont réinitialisées, pensez à supprimer les fichiers **~/.ssh/known_hosts** qui contiennent les identifiants des anciennes versions, sous peine d'obtenir le warning suivant

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

```
@  WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!  @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!

Someone could be eavesdropping on you right now (man-in-the-middle attack)!

It is also possible that the DSA host key has just been changed.

The fingerprint for the DSA key sent by the remote host is

0e:6f:19:83:12:96:a6:d1:4d:2b:d6:40:93:39:05:98.

Please contact your system administrator.

Add correct host key in /home/olivier/.ssh/known_hosts to get rid of this message.

Offending key in /home/olivier/.ssh/known_hosts:1

Password authentication is disabled to avoid man-in-the-middle attacks.

Keyboard-interactive authentication is disabled to avoid man-in-the-middle attacks.

X11 forwarding is disabled to avoid man-in-the-middle attacks.

L'installation va installer les binaires utilisateurs sous **/usr/local/bin**, le daemon sous **/usr/local/sbin** et le serveur ftp sécurisé sous **/usr/local/libexec**

Les fichiers de config sont sous **/usr/local/etc** (**ssh_config** pour le client **sshd_config** pour le serveur). Si ces chemins ne vous plaisent pas, lancer **configure** avec les arguments qui vont bien donnés par:

configure --help

A présent si vous utilisez **PAM** (par défaut sur une mandrake) vous devez prendre le fichier **sshd.pam** se trouvant sous **./openssh-4.3p2/contrib/redhat** et le placer sous **/etc/pam.d** en le renommant **sshd** tout court. En vous plaçant donc dans le répertoire de travail d'**OpenSSH**:

cp ./contrib/redhat/sshd.pam /etc/pam.d/sshd

NOTE Comment savoir si vous utilisez **PAM**, jetez un coup d'oeil dans **/var/log/messages** si vous voyez des mentions à **PAM** (**PAM_pwd** par exemple quand vous vous loguez ou faites un **su**), c'est qu'il est actif sur votre système.

Vous noterez que dans le même répertoire on trouve aussi le fichier de démarrage de **sshd** en cas de démarrage standalone.

6 Configuration

6.1 Configuration du serveur

Le daemon **SSH** se configure avec le fichier **sshd_config** se trouvant sous **/usr/local/etc** voici son contenu détaillé:

This is ssh server systemwide configuration file.

port par défaut utilisé par SSH

voir /etc/services

Port 22

2 si on utilise que SSH2

si vous utilisez SSH1 et 2

vous mettrez Protocol 2,1

Protocol 2

adresse IP de l'interface d'où vont

arriver les requêtes

mettez l'adresse IP de votre carte réseau

ListenAddress 192.168.13.11

```
# fichier contenant la clé privée RSA si vous utilisez
# SSH1
# en commentaire ici, car on se sert que de SSH2
# HostKey /usr/local/etc/ssh_host_key

# fichier contenant la clé privée DSA si on utilise
# SSH2
# attention le fichier doit avoir pour droit 600
#HostKey /usr/local/etc/ssh_host_rsa_key
HostKey /usr/local/etc/ssh_host_dsa_key

# taille en bits et durée de validité de la clé serveur
# ne sert à rien pour SSH2
# Lifetime and size of ephemeral version 1 server key
#KeyRegenerationInterval 3600
#ServerKeyBits 768

# information de log
# on sauvegarde les logs dans /var/log/messages
# et /var/log/secure
SyslogFacility AUTH

# niveau de détail des logs
# pour plus de détails DEBUG
LogLevel INFO

# le serveur se déconnecte au bout de 600s
# si l'utilisateur n'a pas réussi à se loguer
LoginGraceTime 600

# est ce qu'on permet root de se loguer
# conseil: laisser à no
# il est préférable de se loguer en tant que simple utilisateur
# puis de faire su, ça laisse une trace au moins
PermitRootLogin no

# le serveur vérifier si les droits et le proprio de la home directory
# et des fichiers qui y contenus sont corrects avant d'accepter la connexion
StrictModes yes

# tentative de connexion limitée à 3
MaxAuthTries 3

# en cas d'authentification RSA
#RSAAuthentication yes

# si protocole SSH2 mettre à yes
PubkeyAuthentication yes
# fichier contenant les clés publiques
AuthorizedKeysFile .ssh/authorized_keys
```

concerne l'authentification basée sur les fichiers .rhosts...
concerne SSH1
on en veut pas ici
RhostsAuthentication no
IgnoreRhosts yes
RhostsRSAAuthentication no
HostbasedAuthentication no
IgnoreUserKnownHosts yes

pour que l'utilisateur ait à donner le mot de passe
du compte sur lequel il se connecte après avoir donné son passphrase
#PasswordAuthentication no

en cas de mot de passe vide, la connexion
est refusée systématiquement
sert à rien si PasswordAuthentication à no
#PermitEmptyPasswords no

mettre à no pour désactiver les mots de passe s/key (?)
#ChallengeResponseAuthentication yes

Kerberos options
#KerberosAuthentication no
#KerberosOrLocalPasswd yes
#KerberosTicketCleanup yes

#AFSTokenPassing no

Kerberos TGT Passing only works with the AFS kserver
#KerberosTgtPassing no

on forward X
X11Forwarding yes

le DISPLAY sera fixé à serveur-ssh:10.0
X11DisplayOffset 10

on peut afficher à la connexion un petit message
contenu dans /etc/motd
PrintMotd no

pour afficher la dernière date et heure de login
PrintLastLog yes

lors d'une connexion, vérification régulière pour voir si le serveur
n'est pas down pour pouvoir avertir l'utilisateur à temps.
TCPKeepAlive yes

Laisser à no sinon le X forwarding ne marche pas
UseLogin no

pour que certaines opérations soient réaliser en tant que non root
UsePrivilegeSeparation yes

nombre de connexions non authentifiées max au serveur sshd
#MaxStartups 10

On peut spécifier ici le path d'une bannière
#Banner /some/path

sshd peut vérifier que l'hôte distant a le nom qui correspond bien
à son adresse IP
#VerifyReverseMapping no

Pour avoir le ftp sécurisé
Subsystem sftp /usr/local/libexec/sftp-server

Les autres paramètres intéressants sont:

AllowUsers et **AllowGroups** pour spécifier les utilisateurs et groupes d'utilisateurs habilités à se connecter, ceux non listés n'ont pas le droit de se connecter. Dans le même style **DenyGroups** et **DenyUsers** qui a l'effet inverse.

ClientAliveInterval pour que **sshd** envoie un message si le client n'a pas d'activité au bout d'un certain temps. Marche avec **ClientAliveCountMax** qui fixe le nombre de message sans réponse à partir duquel la connexion est stoppée.

Ciphers type de cryptage utiliser pour chiffrer le message, on a le choix entre **3des-cbc**, **blowfish-cbc**, **arcfour** et **cast128-cbc**, par défaut on a **3des-cbc** qui correspond au **triple DES**.

6.2 Configuration du client

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

On peut spécifier des paramètres différents suivant
le serveur avec le paramètre suivant
voir exemple plus bas

Host *

Est ce qu 'on forwarde aussi l'agent
d'authentification
ForwardAgent no

est ce qu'on forwarde X
ForwardX11 yes

concerne l'authentification basée sur les fichiers
concerne SSH1
on ignore
RhostsRSAAuthentication no

ça s'applique uniquement à SSH1
RSAAuthentication yes

on utilise SSH2 donc à yes
PubkeyAuthentication yes

pour rentrer le mot de passe du compte
sur lequel on se connecte
PasswordAuthentication no

si connexion par ssh échoue
on tente par rsh
FallBackToRsh no

est ce qu'on utilise rsh
UseRsh no

si vous ne voulez ni rentrer le passphrase
ni le mot de passe, on met à yes
utilise pour lancer une connexion ssh
en mode batch (dans un script)
BatchMode no

#Vérification de l'adresse IP du serveur
CheckHostIP yes

Avant de rajouter la clé publique d'un serveur
dans la liste de serveurs reconnus
ssh va le demander à l'utilisateur
StrictHostKeyChecking ask

fichier pour SSH1 seulement
#IdentityFile ~/.ssh/identity

clés DSA pour SSH2
#Identity File ~/.ssh/id_rsa
IdentityFile ~/.ssh/id_dsa

port par défaut du serveur
Port 22

on utilise le protocole SSH2
Protocol 2

type de chiffrement pour SSH1
Cipher 3des

type de chiffrement pour SSH2 dans l'ordre de préférence
Ciphers aes128-cbc,3des-cbc,blowfish-cbc,cast128-cbc,arcfour,aes192-cbc,aes256-cbc

caractère d'échappement par défaut
EscapeChar ~

Les autres paramètres intéressants sont:

Compression pour en plus de crypter, compresser les données, pour info on utilise **gzip**. Marche avec **CompressionLevel** qui précise le niveau de compression (1 rapide mais compression moyenne à 9 lent mais bonne compression).

LogLevel pour avoir le niveau de détail quand on lance **ssh** en mode verbeux (**-v**), par défaut **INFO** mais on peut mettre **DEBUG** en cas de problèmes.

SmartcardDevice pour accéder à la carte à puce de l'utilisateur contenant la clé RSA !

Pour la variable Host vous pouvez spécifier des paramètres différents suivant le serveur, exemple avec le serveur **tetepa** (utilise SSH2) et **atopa** (utilise SSH1).

Host tetepa

port 22

Protocol 2

Pubkeyauthentication yes

PasswordAuthentication yes

Host atopa

port 22

Protocol 1

RSAAuthentication yes

PasswordAuthentication yes

7 Lancement du daemon sshd

7.1 Présentation

On a le choix entre le lancer par le super serveur **xinetd** ou en standalone, si **SSH** est utilisé occasionnellement préférez le lancement par **xinetd** moins gourmand en ressource (**sshd** est lancé que quand on l'utilise et non pas en permanence).

Il est cependant précisé dans la documentation de préférer le lancement en standalone.

NOTE Accessoirement si **SSH** devient votre outil de connexion préféré, supprimer le fichier **telnet** se trouvant sous **/etc/xinetd.d**.

7.2 Lancement par xinetd

Créez le fichier **ssh** dans le répertoire **/etc/xinetd.d**

service ssh

{

socket_type = stream

```

protocol    = tcp
port        = 22
wait        = no
user        = root
server      = /usr/local/sbin/sshd
log_on_success    += USERID
log_on_failure    += USERID
server_args    = -i
}

```

Puis relancez le daemon

/etc/rc.d/init.d/xinetd restart

C'est bon **xinetd** est relancé, et le daemon **sshd** est prêt à recevoir les requêtes des clients.

ATTENTION En lançant **sshd** par **xinetd**, vous utilisez donc les **TCP_WRAPPERS**. Vous devez veiller à ce que les clients devant se connecter ne soient pas listés dans **/etc/hosts.deny** .

NOTE Il faut savoir que le lancement par **xinetd** n'est pas conseillé, car il génère des temps d'attente trop long (voir **man sshd**).

7.3 Lancement en standalone

7.3.1 Lancement en standalone avec Mandriva

On prendra le fichier **sshd.init** se trouvant sous **./openssh-4.3p2/contrib/redhat** et on le placera sous **/etc/rc.d/init.d** en le renommant **sshd** . En se plaçant donc sous **openssh-4.3p2**, on tapera:

```
cp ./contrib/redhat/sshd.init /etc/rc.d/init.d/sshd
```

On veillera à modifier les chemins présents dans ce fichier (notamment celui de **ssh-keygen**, des clés, de préciser le chemin absolu de **sshd**). Maintenant pour lancer **sshd** aux niveaux de marche 3, 4 et 5, on tapera:

```
chkconfig --level 345 sshd on
```

Pour l'arrêter aux niveaux de marche 0, 1, 2 et 6

```
chkconfig --level 0126 sshd off
```

Pour lancer **sshd**, il suffit maintenant de taper:

/etc/rc.d/init.d/sshd start

Manip qui sera effectué dès lors automatiquement au démarrage et que vous n'aurez plus à refaire.

7.3.2 Lancement en standalone avec ubuntu

Voilà un fichier de lancement à placer sous **/etc/init.d** et à nommer sshd

```
#!/bin/sh
set -e

# /etc/init.d/sshd: start and stop the OpenBSD "secure shell(tm)" daemon

test -x /usr/local/sbin/sshd || exit 0
( /usr/local/sbin/sshd -\? 2>&1 | grep -q OpenSSH ) 2>/dev/null || exit 0

if test -f /etc/default/sshd; then
    . /etc/default/sshd
fi

. /lib/lsb/init-functions

check_for_no_start() {
    # forget it if we're trying to start, and /etc/ssh/sshd_not_to_be_run exists
    if [ -e /usr/local/etc/ssh/sshd_not_to_be_run ]; then
        log_end_msg 0
        log_warning_msg "OpenBSD Secure Shell server not in use (/usr/local/etc/ssh/sshd_not_to_be_run)"
        exit 0
    fi
}

check_privsep_dir() {
    # Create the PrivSep empty dir if necessary
    if [ ! -d /var/empty ]; then
        mkdir /var/empty
        chmod 0755 /var/empty
    fi
}

check_config() {
    if [ ! -e /usr/local/etc/ssh/sshd_not_to_be_run ]; then
        /usr/local/sbin/sshd -t || exit 1
    fi
}

export PATH="${PATH:+$PATH:}/usr/local/sbin:/sbin:/usr/sbin"

case "$1" in
    start)
        log_begin_msg "Starting OpenBSD Secure Shell server..."
        check_for_no_start
        check_privsep_dir
```

```

        start-stop-daemon --start --quiet --pidfile /var/run/sshd.pid --exec /usr/local/sbin/sshd --
$SSH_OPTS || log_end_msg 1
    log_end_msg 0
    ;;
stop)
    log_begin_msg "Stopping OpenBSD Secure Shell server..."
    start-stop-daemon --stop --quiet --oknodo --pidfile /var/run/sshd.pid || log_end_msg 1
    log_end_msg 0
    ;;

reload|force-reload)
    log_begin_msg "Reloading OpenBSD Secure Shell server's configuration"
    check_for_no_start
    check_config
    start-stop-daemon --stop --signal 1 --quiet --oknodo --pidfile /var/run/sshd.pid --exec
/usr/local/sbin/sshd || log_end_msg 1
    log_end_msg 0
    ;;

restart)
    log_begin_msg "Restarting OpenBSD Secure Shell server..."
    check_privsep_dir
    check_config
    start-stop-daemon --stop --quiet --oknodo --retry 30 --pidfile /var/run/sshd.pid
    check_for_no_start
    start-stop-daemon --start --quiet --pidfile /var/run/sshd.pid --exec /usr/local/sbin/sshd --
$SSH_OPTS || log_end_msg 1
    log_end_msg 0
    ;;

*)
    log_success_msg "Usage: /etc/init.d/ssh {start|stop|reload|force-reload|restart}"
    exit 1
esac

exit 0

```

Pour un lancement automatique à l'état de marche 2, 3, 4 et 5 et un arrêt aux états de marche 0, 1 et 6 on tapera

```
sudo update-rc.d sshd start 20 2 3 4 5 . stop 20 0 1 6 .
```

8 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 -d
Generating public/private dsa key pair.
```

Enter file in which to save the key (/home/olivier/.ssh/id_dsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/olivier/.ssh/id_dsa.
Your public key has been saved in /home/olivier/.ssh/id_dsa.pub.
The key fingerprint is:
4c:33:be:2e:b2:5e:4d:fa:52:50:bd:56:c6:95:7a:b9 olivier@obelix.kervao.fr

NOTE: l'option **-d** correspond à **SSH2** pour la gestion de clés en utilisant **DSA**.

Les clés sera sauvegardée par défaut dans **~/.ssh** , la clé privée a pour nom **id_dsa** et la clé publique **id_dsa.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.

Maintenant **veronique** doit faire de même sous **asterix** , elle doit ensuite par un moyen ou un autre faire parvenir sa clé publique (**id_dsa.pub**) à **olivier**, ce dernier va mettre le contenu de ce fichier dans le fichier (éventuellement à créer) **~/.ssh/authorized_keys**.

ATTENTION il est primordial que la clé publique de **veronique** tienne sur **UNE** seule ligne dans le fichier **authorized_keys**, il ne doit pas y avoir de retour chariot.

NOTE Si **olivier** autorise plusieurs personnes à se connecter sur son compte par **ssh**, la syntaxe de **authorized_keys** deviendra:

utilisateur veronique
contenu de id_dsa.pub de veronique sur une seule ligne

utilisateur lambda
contenu de id_dsa.pub de lambda sur une seule ligne

ainsi de suite

est le début de commentaires, et les lignes vides ne sont pas prises en compte.

NOTE Pour changer de pass-phrase au niveau de **ssh-keygen** , il suffit de taper:

```
ssh-keygen -p
```

9 Utilisation

9.1 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 .ssh]$ ssh -l olivier obelix
The authenticity of host 'obelix (192.168.13.11)' can't be established.
DSA key fingerprint is bb:aa:0a:92:66:a8:6c:08:ab:6b:35:18:4d:ab:19:13.
Are you sure you want to continue connecting (yes/no)?yes
Warning: Permanently added 'obelix,192.168.13.11' (DSA) to the list of known hosts.
Enter passphrase for key '/home/veronique/.ssh/id_dsa':
Last login: Sat Jun 22 14:11:43 2002
[olivier@obelix olivier]$
```

C'est bon on est connecté, à noter:

Warning: Permanently added 'obelix,192.168.13.11' (DSA) to the list of known hosts.

La machine **obelix** va être rajouté à la liste des machines connues, y aura plus ce message par la suite, si vous voulez que la machine **obelix** ne soit pas rajoutée de manière permanente mettez le paramètre **StrictHostKeyChecking** à **yes** dans **ssh_config** mais dans ce cas il faudra récupérer la clé publique d'**obelix** pour la mettre manuellement dans le fichier **~/.ssh/known_hosts**

A noter aussi que la prochaine fois que vous essayerez de vous connecter sur **obelix**, il ne sera plus nécessaire de rajouter **-l olivier** , ça sera ajouté automatiquement par défaut.

NOTE Les infos sur la machine seront sauvegardées dans le fichier **~/.ssh/known_hosts**

9.2 Lancement une commande à distance

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

```
[veronique@asterix .ssh]$ ssh obelix df
Enter passphrase for DSA key '/home/veronique/.ssh/id_dsa':
Filesystem      1k-blocks    Used Available Use% Mounted on
/dev/sdb1        149712    75053   66929  53% /
/dev/sda1        991995    806267  134478  86% /alphonse
/dev/sdb3        239979    77414   150175  34% /roger
/dev/sdb4         99136    42591   51425  45% /home
/dev/sdc5        938296    497756  392876  56% /usr
/dev/sdb5        496627    334344  136633  71% /usr/local
[veronique@asterix .ssh]$
```

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

9.3 Lancement d'une commande X à distance

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

```
[veronique@asterix .ssh]$ ssh obelix
Enter passphrase for DSA key '/home/veronique/.ssh/id_dsa':
Last login: Fri May 26 12:14:52 2000 from asterix.armoric.bz
[olivier@obelix olivier]$ env | grep DISPLAY
DISPLAY=localhost:10.0
[olivier@obelix olivier]$ xauth list
obelix.armoric.bz:0 MIT-MAGIC-COOKIE-1 1f2c455765476a6d5c0a1225383d4f52
obelix.armoric.bz/unix:0 MIT-MAGIC-COOKIE-1
1f2c455765476a6d5c0a1225383d4f52
obelix.armoric.bz:10 MIT-MAGIC-COOKIE-1 599555deac54a6b3f338147f9bb56eb8
obelix.armoric.bz:1 MIT-MAGIC-COOKIE-1 545a095522c5fbe32bbb1d357ab52227
obelix.armoric.bz/unix:1 MIT-MAGIC-COOKIE-1
545a095522c5fbe32bbb1d357ab52227
obelix.armoric.bz/unix:10 MIT-MAGIC-COOKIE-1
9434717920d84aa2287b76974f723fe0
[olivier@obelix olivier]$
```

Remarquer bien que sur le client **SSH**, **DISPLAY** est positionné à **localhost** qui correspond au serveur 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 autorisations pour se connecter à un serveur X, concrètement **sshd** va appeler **xauth** pour que les commandes X puissent s'afficher sur le client.

Bon par contre, si vous avez l'erreur suivante

```
[olivier@obelix olivier]$ xmms &
[1] 1036
[olivier@obelix olivier]$ Gdk-ERROR **: X connection to obelix.armoric.bz:10.0
broken (explicit kill or server shutdown).
```

```
[1]+  Exit 1          xmms
```

Il faudra modifier le fichier **.bashrc** de l'utilisateur **olivier** au lieu de:

```
export XAUTHORITY=$HOME/.Xauthority
```

On mettra

```
[ -z $XAUTHORITY ] && export XAUTHORITY=$HOME/.Xauthority
```

Et là magique ça marche !!

9.4 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
Enter passphrase for DSA key '/home/veronique/.ssh/id_dsa':
toto          100% |*****| 27    00:00
```

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
Enter passphrase for DSA key '/home/veronique/.ssh/id_dsa':
olivier@obelix's password:
titi          100% |*****| 15    00:00
```

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 .
Enter passphrase for DSA key '/home/veronique/.ssh/id_dsa':
fichier1      100% |*****| 5      00:00
repertoire1   100% |*****| 5      00:00
fichier2      100% |*****| 5      00:00
```

A noter que l'option **-r** n'est ni documentée dans le man, ni en faisant un **scp -help** (oubli ?).
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
- B mode batch pour mettre **scp** dans un script (plus besoin de rentrer le passphrase et le mot de passe)
- C pour compresser

10 Les clients windows

10.1 Présentation

Ne sont présentés dans ce paragraphe que les clients compatibles SSH2 opensource.

10.2 OpenSSH

Cygwin par défaut dispose d'un client **SSH2**, les commandes, à taper dans un shell **cygwin**, sont strictement identiques à celles d'**OpenSSH**, et c'est tout à fait normal puisque le client **SSH** de cygwin est la version d'**OpenSSH** compilée sous windows avec **cygwin**.

La syntaxe des commandes est strictement identique à celle d'**OpenSSH** sous linux.

10.3 PuTTY

10.3.1 Présentation

PuTTY est un client **SSH** placé sous licence **MIT**, proche de la licence **BSD**, si ça ne vous dit pas plus, sachez que le logiciel est certifié "Open source" et qu'il répond aux exigences des logiciels libres de la **Debian**. Vous pouvez le récupérer à www.chiark.greenend.org.uk/~sgtatham/putty. C'est à mon avis la meilleure alternative pour un client **SSH** pour windows.

10.3.2 Installation

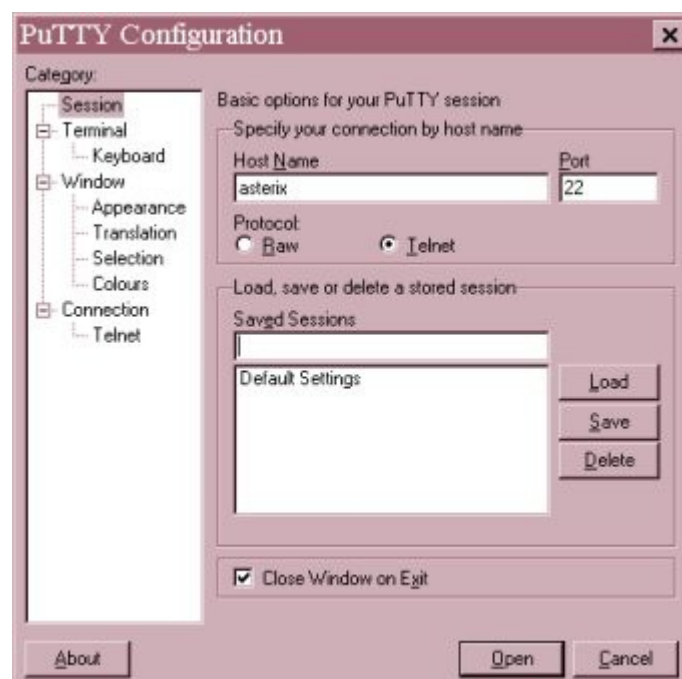
L'installation est on ne peut plus basique il suffit de récupérer les exécutables suivants sur le site de **PuTTY** :

pageant.exe plink.exe pscp.exe putty.exe puttygen.exe puttytel.exe

10.3.3 Utilisation

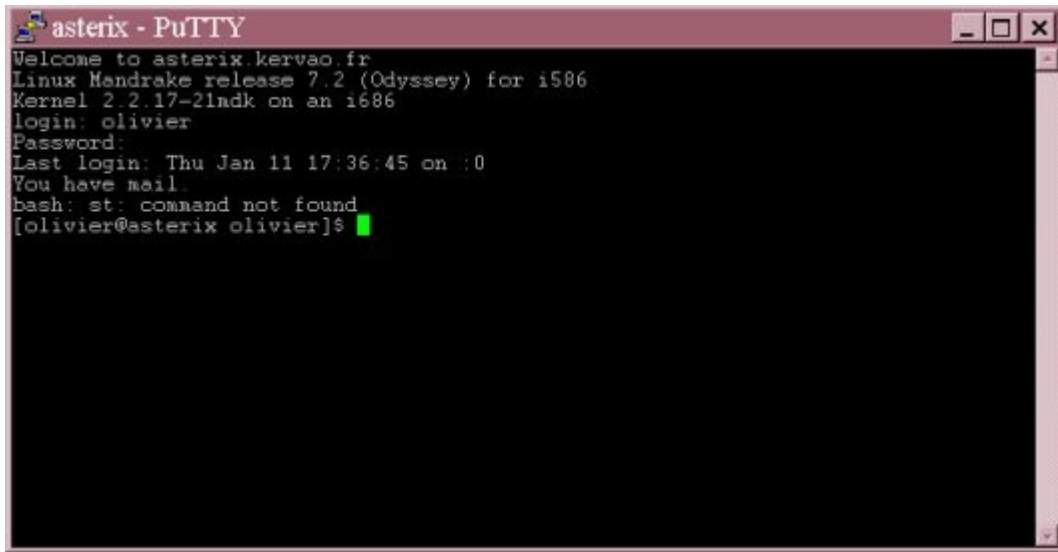
puttytel.exe

puttytel.exe se limite à la fonction **telnet**, voilà ce que ça donne en l'exécutant :



Vous pouvez sauvegarder les sessions (connexions réussies) et de nombreux paramètres de configuration sont disponibles.

Une fois les paramètres de connexion saisis, on clique sur **Open** :



A noter que la complétion marche (**TAB** pour compléter le nom des commandes), pour quitter il suffit de taper **logout**.

puttygen.exe

puttygen.exe permet de générer un couple de clés (privée et publique).



On fixe le nombre d'octets puis on clique sur **Generate** :



Pendant la génération il est nécessaire de bouger un peu partout avec la souris, les mouvements de la souris rentrent en compte pour générer la clé.



Une fois les clés générés, il faut saisir un mot de passe sous forme de phrase mot de passe "**passphrase**", dans le champ **Key comment**, vous pouvez très bien mettre de que vous voulez. Cliquez sur **Save** pour sauvegarder la clé privée à un endroit spécifique.

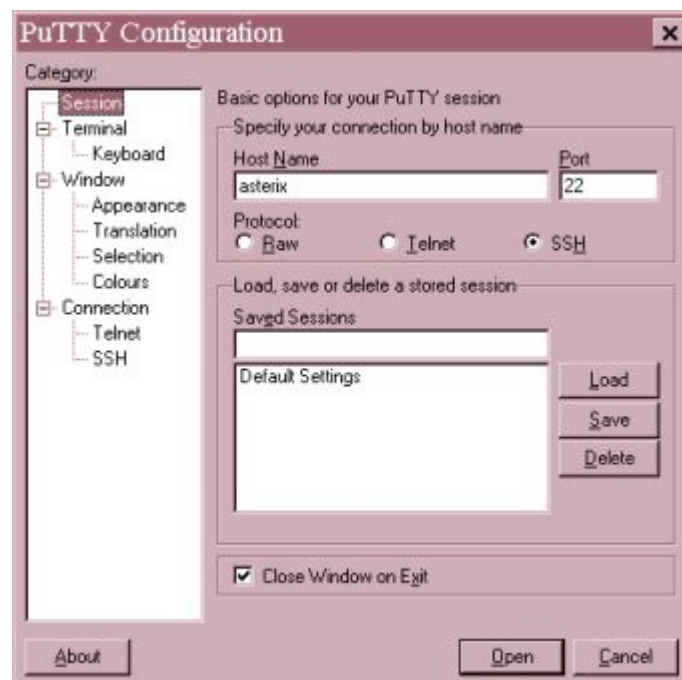
Je vous conseille bien évidemment de ne pas mettre la clé privée à un endroit visible par tout le monde, l'idéal est de le mettre dans un répertoire UNIX monté avec les droits qui vont bien, dans un répertoire windows crypté ou bien encore sur une disquette ou un CD-ROM.

Maintenant il faut sauvegarder la clé publique, sélectionner la clé publique se trouvant dans le champ **Public key for pasting into authorized_keys file**, puis cliquez sur le bouton droit de la souris, **Copier**, coller dans le bloc note, sauvegarder le fichier, c'est la clé publique que vous pouvez échanger par la suite.

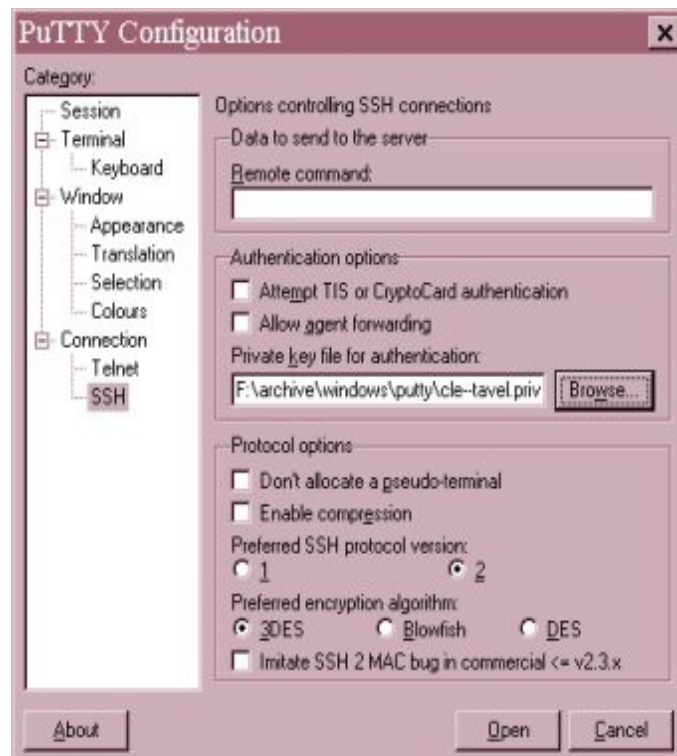
Fermer la fenêtre en cliquant sur la croix en haut à droite.

putty.exe

putty.exe est le client **telnet** et **SSH**



On peut sauvegarder les sessions (connexions réussies), plusieurs paramètres de configuration sont disponibles, pour la configuration de la connexion SSH, il faut cliquer sur **Connection>SSH**



Vous pouvez choisir de préférer le protocole SSH2, d'envoyer des données sous forme compressée.

Une fois tous ses paramètres saisis, on clique sur **Open** :



Comme c'est la première fois qu'on se connecte, il nous avertit que la machine sur laquelle on croit se connecter n'est peut être pas la bonne, la question est, est-on vraiment sûr de se connecter sur la bonne machine, une fois répondue, la fenêtre suivante apparaît, on doit y saisir le login et le mot de passe avant d'être connecté :

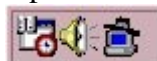


Pour information les informations sur les hôtes visités sont inscrites dans la base de registre de windows :

HKEY_CURRENT_USER/SOFTWARE/Simon Tatham\PuTTY\SshHostKeys

pageant.exe

Voilà un utilitaire dont je n'ai pas bien compris l'intérêt.



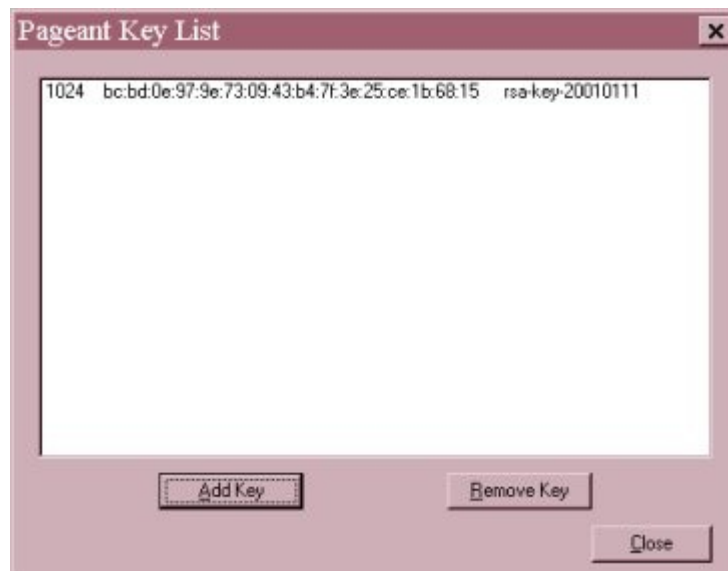
Quand on lance **pageant.exe** on retrouve une petite icône représentant un ordinateur avec un chapeau par dessus quand on clique dessus avec le bouton droit en sélectionnant **View Keys** la fenêtre suivante apparaît :



On clique sur **Add Key**, et on nous demande alors de sélectionner une clé privée **Select Private Key File**, comme on est sensé pouvoir sélectionner que sa propre clé, j'ai sélectionné ma clé, on nous demande alors de saisir le passphrase défini avec **puttygen.exe**.



La clé privée apparaît alors ou du moins sa longueur en octets, son empreinte et le **Key Comment**.



Les autres commandes disponibles de **pageant.exe** accessibles par le menu sont :

- **View Keys**
- **Add Key**
- **About**
- **Exit**

pscp.exe

pscp.exe est un utilitaire de copie sécurisée équivalent à **ftp** (en sécurisé), il marche dans une fenêtre DOS, la syntaxe est la suivante :

pscp [options] source destination

avec source et destination = **user@host:fichier**

Les options disponibles sont :

- p** préserver les attributs du fichier
- q** pour ne pas montrer les stats (vitesse de transfert...)
- r** pour copier des répertoires
- v** mode bavard
- P** pour spécifier un numéro de port particulier
- pw passwd** mot de passe de l'utilisateur spécifié dans la source ou la destination
- gui hWnd** mode graphique

Un petit exemple

```
pscp toto.txt olivier@obelix:/tmp  
olivier@obelix's password :
```

Mais ça marche pas chez moi, il y a un problème de compatibilité **SSH1**, il cherche un programme **scp1** qui n'est pas dans le **PATH** et qui n'est pas sur mon disque non plus.

plink.exe

plink.exe permet de se connecter ou de lancer des commandes d'une fenêtre DOS, la syntaxe est la suivante :

```
plink [options] [user@]host [command]
```

Un petit exemple :

```
plink -v -ssh olivier@obelix
```

```
Server version: SSH-2.0-2.4.0 SSH Secure Shell (non-commercial)
```

```
We claim version: SSH-2.0-PuTTY
```

```
Using SSH protocol version 2
```

```
Host Key fingerprint is:
```

```
1024: 11:25:31:f0:26:1e:c5:59:b3:bb:f4:a1:58:9c:87:37
```

```
Using username "olivier".
```

```
olivier@obelix's password:
```

```
Opened channel for session
```

```
Allocated pty
```

Started a shell/command
Last Login: Thu Jan 11 2001 18:27:19 +0100
You have mail.
[olivier@obelix olivier]\$

Pour lancer qu'une commande sur le serveur distant :

plink -v -ssh olivier@obelix df