Ssh - généralités

Secure Shell (SSH) est un protocole de connexion à distance sécurisé (authentification, chiffrement).

Le **client `ssh`** est disponible de façon standard sur les systèmes *Linux* ou *MacOS X*. Pour *Windows* le logiciel Putty offre une interface graphique pour SSH, mais Microsoft fournit aussi le client `ssh` en ligne de commande via WSL (Windows Subsystem for Linux).

Putty existe aussi pour Linux et MacOS X, mais il y est nettement moins souvent utilisé.

Des *services supplémentaires* existent au-dessus de SSH, entre autre `scp` et `sftp` pour le transfert de fichiers. SSH permet de mettre en place des *tunnels* afin de faire transiter d'autres flux de communication par le même canal (x2go, Proxy web...).

Connexion login/mot de passe

Le protocole SSH peut permettre de s'authentifier sur une machine distante simplement à l'aide de son **login** et **mot de passe** (c'est le cas de la plupart des machines internes au laboratoire lorsqu'on y accède d'une autre machine interne). On utilise simplement :

```
ssh ma_machine
```

(éventuellement `ssh autrelogin@ma_machine` si on dispose sur cette machine d'un login différent)

Le mot de passe est demandé au moment de la connexion.

Exemple

```
bidule@plop:~$ ssh ma_machine
bidule@ma_machine's password: XXXXXXXXX
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 5.4.0-52-generic x86_64)
…
bidule@ma_machine:~$
```

Première connexion / vérification machine cible

S'il s'agit de la **première connexion** à cette machine, `ssh` affiche le **“fingerprint”** correspondant à la machine et demande de le valider. Il va ensuite le mémoriser dans `~/.ssh/known_hosts` comme identifiant d'une machine connue.

```
bidule@plop:~$ ssh  ma_machine
The authenticity of host 'ma_machine (129.175.17.124)' can't be established.
ECDSA key fingerprint is SHA256:ffp+FmYkXoGXXB0pbZXjUKTYjEdMIBsrXhgoibx375Y.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ma_machine,192.168.122.243' (ECDSA) to the list
of known hosts.
```

```
bidule@ma_machine's password: XXXXXXXXX
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 5.4.0-52-generic x86_64)
…
bidule@ma_machine:~$
```

Au cas où un autre ordinateur essaierait d'usurper l'adresse de ma_machine (typiquement pour récupérer votre mot de passe), sa signature serait invalide, ssh afficherait un message d'erreur et refuserait la connexion:

```
bidule@plop:~$ ssh ma_machine
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    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 a host key has just been changed.
The fingerprint for the ECDSA key sent by the remote host is
SHA256:niH36DM6CtVqxZpe3iEWtbQbWpJoh3MZYrnsWsIo9pM.
Please contact your system administrator.
Add correct host key in /home/bidule/.ssh/known_hosts to get rid of this
message.
Offending ECDSA key in /home/bidule/.ssh/known_hosts:3
  remove with:
  ssh-keygen -f "/home/bidule/.ssh/known_hosts" -R "ma_machine"
ECDSA host key for ma_machine has changed and you have requested strict
checking.
Host key verification failed.
```

En cas d'alerte « **WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!** », si le service informatique ne vous a pas prévenu d'un tel changement, alors prenez contact afin de vérifier qu'il n'y a pas effectivement une machine usurpatrice.

Si le changement de fingerprint est normal, alors vous pouvez nettoyer l'ancien à l'aide de la commende indiquée dans le message d'alerte.

Il se peut aussi que la même machine soit connue sous un autre nom, ou ait changé d'adresse IP... vous pouvez dans ce cas avoir des messages comme:

```
Warning: the ECDSA host key for 'ma_machine' differs from the key for the IP
address '129.175.17.124'
Offending key for IP in /home/bidule/.ssh/known_hosts:2
```

Si tout est bon, il suffit de supprimer la ligne correspondante indiquée du fichier `~/.ssh/known_hosts`.

Connexion avec des clés SSH

Les clés SSH permettent une sécurité plus forte que le login/mot de passe, grace à la passphrase plus solide. Elles permettent d'automatiser certaines connexions de façon transparente.

Création d'un jeu de clés ssh

Lors de la création d'un jeu de clés SSH, vous fabriquez en fait une **clé_publique** (la **serrure**) et une **clé_privée** correspondante. Ce jeu de clés permet de vous authentifier sur des machines (accès ssh) ou des services (par exemple github ou sourcesup).

- La *clé_publique* (serrure) sera positionnée sur les machines (ou services) auxquelles vous désirez vous connecter.
- La *clé_privée* doit être protégée, avec une passphrase forte — sinon c'est comme si vous donniez vos clés au monde entier pour entrer chez vous. Elle peut être recopiée sur vos différents postes de travail habituels, ou encore transférée lorsque vous changez de machine.

Pensez à **sauvegarder vos fichiers de clés ssh**[1)] au cas où vous les perdriez sur l'ordinateur sur lequel elles ont été créées, afin de ne pas avoir non seulement à les re-créer mais aussi à les remettre en place partout où elles sont utilisées

Lorsque vous désirez vous connecter à une machine sur laquelle vous avez positionné la *clé_publique*, ssh vous demande de déverrouiller la *clé_privée* et l'utilise dans le cadre d'un échange avec le serveur qui contrôle si elles correspondent.

Pour éviter d'avoir à saisir trop souvent votre passphrase, vous pouvez utiliser un "agent ssh", programme qui fonctionne en tache de fond et qui se charge de conserver la *clé_privée* dans un trousseau déverrouillé en mémoire et de la rendre disponible à chaque fois que besoin.

Linux / MacOS / Windows WSL

Sur son poste de travail, on génère un jeu de clés SSH avec la commande:

```
ssh-keygen -t rsa -b 4096
```

Il faut répondre aux questions:

- **Enter file in which to save the key** — pour choisir le nom des fichiers dans lesquels seront stockées la *clé_privée* et la *clé_publique*. Le choix proposé par défaut est `~/.ssh/id_rsa`, le plus simple est de le valider. Mais il est possible de spécifier un autre nom (avec le chemin) si on désire faire des tests ou avoir différents jeux de clés.
- **Enter passphrase (empty for no passphrase)** — là il faut saisir une phrase “normale”, assez longue, et dont on se souvient facilement… et qu'un tiers ne pourra pas deviner facilement en se renseignant sur vous.

Il est obligatoire de mettre une passphrase, solide, afin de sécuriser cette clé privée, sinon c'est comme si vous mettiez votre mot de passe en clair dans un fichier !

S'il n'existe pas encore, le répertoire `~/.ssh` est créé, avec les droits `rwx` pour l'utilisateur seulement.

La *clé_privée* est stockée de façon chiffrée dans `~/.ss/id_rsa` avec les droits `rw` pour l'utilisateur seulement. Même chiffré, ce fichier ne doit ni être transféré par email, ni être déposé dans des espaces de partage publics.

La *clé_publique* est stockée dans un fichier correspondant **.pub** `~/.ss/id_rsa.pub` avec les droits `rw` pour l'utilisateur et `r` pour tout le monde.

Exemple

```
bidule@plop:~$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/bidule/.ssh/id_rsa):
Created directory '/home/bidule/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/bidule/.ssh/id_rsa
Your public key has been saved in /home/bidule/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:2VdyBiOfF2y7AxdVyEmSBnGtAVfJOdKPrjl0aurgYPE bidule@plop
The key's randomart image is:
+---[RSA 4096]----+
|          =+*X+*+|
|           ===#. |
|           .+==* |
|         o  o*+ .|
|      . S . .+ . |
|       o   .. =  |
|      o E  . = . |
|     . o .  *    |
|        ..oo .   |
+----[SHA256]-----+
bidule@plop:~$ ls -l .ssh
total 8
-rw------- 1 bidule bidule 3434 févr.  7 16:12 id_rsa
-rw-r--r-- 1 bidule bidule  737 févr.  7 16:12 id_rsa.pub
```

La *clé_publique* (serrure) est un fichier texte d'une ligne, contenant une indication sur le type de clé, la valeur de la clé, et un texte libre pour identifier celle-ci (texte que l'on peut modifier sans incidence):

~/.ssh/id_rsa.pub

```
ssh-rsa AAAAB3NzaC1yc… …Ufq3PeJXSIN4eDkP7hQ== bidule@plop
```

La *clé_privée* est aussi un fichier texte, content une représentation chiffrée de la clé.

~/.ssh/id_rsa

```
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAACmFlczI1Ni1jdHIAAAAGYmNyeXB0AAAAGAAAABCF6qDIcB
udEpeWoB79ob2CAAAAEAAAAAEAAAIXAAAAB3NzaC1yc2EAAAADAQABAAACAQDjn4GXfstx
… …
uSRJHWq206wx/Bdl+mBaSc/1/pMm3G7JKj2zmKKI1r9jLvAh+U62wInoR6BsiefUwMNGY4
m6day8zfc+ZZW5p+hZ56yGVFs=
-----END OPENSSH PRIVATE KEY-----
```

À la place de l'algorithme RSA, il est possible d'utiliser l'algorithme ed25519 avec la commande `ssh-keygen -t ed25519 -a 100` – dans ce cas les fichiers de clés créés sont `id_ed25519` et `id_ed25519.pub`.
On peut spécifier le commentaire informatif placé à la fin de la clé publique (par défaut `loginlocal@nommachinelocale`) avec l'option `-C "le texte que je veux"` .

Mise en place de la clé publique (serrure)

L'installation de la *clé_publique* sur une machine distante nécessite de s'authentifier au moins une fois sur celle-ci à l'aide de votre login et mot de passe. Et d'ajouter le contenu du fichier *clé_publique* au fichier `~/.ssh/authorized_keys` de la machine distante.

Le plus simple est d'utiliser la commande **ssh-copy-id** :

```
ssh-copy-id -i ~/.ssh/id_rsa.pub ma_machine.lisn.upsaclay.fr
```

La commande `ssh-copy-id` établit une connexion SSH (par mot de passe, avec gestion du fingerprint de la machine comme indiqué ci-dessus), se charge de la création de `~/.ssh/` et de `authorized_keys` sur la machine distante si besoin (avec les bons droits d'accès), et enfin installe votre *clé_publique* dans les clés autorisées sur la machine distante.

Exemple:

```
bidule@plop:~$ ssh-copy-id -i ~/.ssh/id_rsa.pub ma_machine.lisn.upsaclay.fr
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed:
"/home/bidule/.ssh/id_rsa.pub"
The authenticity of host 'ma_machine.lisn.upsaclay.fr (129.175.17.124)'
can't be established.
ECDSA key fingerprint is SHA256:ffp+FmYkXoGXXB0pbZXjUKTYjEdMIBsrXhgoibx375Y.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to
filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are
prompted now it is to install the new keys
bidule@ma_machine.lisn.upsaclay.fr's password:
```

```
Number of key(s) added: 1

Now try logging into the machine, with:   "ssh
'ma_machine.lisn.upsaclay.fr'"
and check to make sure that only the key(s) you wanted were added.
```

Toutefois, si cette commande n'est pas disponible, il est possible de se connecter sur la machine cible avec son login/mot de passe et de réaliser l'opération à la main via la console et la ligne de commande.

Attention : pour que la clé soit prise en compte, vous seul (ni le groupe, ni other) devez avoir les accès en écriture sur les répertoires `$HOME` et `$HOME/.ssh` ainsi que sur le fichier `$HOME/.ssh/authorized_keys`.

Utilisation de la clé privée

On utilise ssh normalement :

```
ssh ma_machine.lisn.upsaclay.fr
```

Comme il existe des fichiers de clés, au lieu de demander le mot de passe, c'est la passphrase de la *clé_privée* qui est demandée:

```
bidule@plop:~$ ssh ma_machine.lisn.upsaclay.fr
Enter passphrase for key '/home/bidule/.ssh/id_rsa':
…
bidule@ma_machine:~$
```

Il est possible de forcer l'utilisation du login/mot de passe malgré la présence de clés, en utilisant l'option `-o PubkeyAuthentication=no` *de ssh.*

Utilisation d'un agent ssh

Pour éviter d'avoir à saisir sa passphrase à chaque connexion, il est possible de conserver sa *clé_privée* déverrouillée en mémoire le temps de la session courante. Si vous utilisez un environnement graphique, celui-ci fournit généralement un outil (“*wallet*”) pour gérer les trousseaux de clés ou mots de passe, dont les clés ssh, qui peuvent être déchiffrées à l'ouverture de votre session.

On peut aussi gérer l'agent en ligne de commande avec **ssh-add**:

```
ssh-add
```

Utilise directement la *clé_privée* `~/.ssh/id_rsa`, mais on peut lui spécifier un fichier de clé privée en paramètre.

Si l'agent n'est pas démarré sur une machine, il est possible de l'activer dans une session console avec:

```
eval "$(ssh-agent)" ; ssh-add ~/.ssh/id_rsa
```

Pour connaître les clés privées actuellement déchiffrées en mémoire: `ssh-add -L`. Pour supprimer les clés déchiffrées de la mémoire `ssh-add -D`.

Transmission d'un agent ssh

L'option `-A` de ssh (ou encore la présence de l'option `ForwardAgent yes` pour la connexion cible dans votre fichier de configuration), permet de transmettre l'accès à votre agent ssh local pour la session ssh distante que vous ouvrez. Ceci autorise l'utilisation de votre *clé_privée* dans cette session distante sans avoir à installer le fichier ni à la dé-verrouiller.

Quand **ne pas transmettre son agent ssh**... si cette option est pratique, elle ouvre un accès à l'ensemble des clés déverrouillées au niveau de votre agent ssh à toute personne qui a un accès à votre compte sur la machine cible (root, cracker) – cette personne peut alors se faire passer pour vous sur toutes les machines où vos *clés_publiques* ont été installées.

Cette option est donc à utiliser avec parcimonie, uniquement lorsque besoin et pour des connexions à des machines de confiance.

Passerelles

Il est courant qu'un site protège les accès SSH en obligeant le passage par une machine relai qui assure que le service de connexion est bien à jour au niveau sécurité, et que seules les personnes autorisées peuvent se connecter aux machines du site.

SSH permet ainsi de rebondir (jump) de machine en machine pour rejoindre une machine cible.

Lors de l'utilisation des machines pour "rebondir" vers d'autres machines, il est possible de transférer un accès au trousseau de clés de votre agent ssh.

Tunnels

Sécurité

- Il est conseillé de conserver une sauvegarde des fichiers correspondant à vos jeux de clés — une *clé_publique* ou une *clé_privée* perdue l'est définitivement.
- La *clé_privée*, protégée par une passphrase, ne devrait jamais être transmise par des moyens

de communication non sûrs (par exemple on ne l'envoie pas en utilisant un système email hors institutions).

- Sur les machines dont vous avez la gestion (par exemple une machine personnelle), il est primordial d'effectuer très régulièrement les mises à jour afin de corriger les failles de sécurité lorsqu'elles apparaissent (ce n'est d'ailleurs pas particulier à SSH).
- Par défaut la conservation d'une *clé_privée* en mémoire par l'agent SSH dure tant que l'agent fonctionne, mais on peut spécifier une limite de temps.
- Le transfert de l'accès à votre trousseau de clés SSH (Agent Forwarding) ne doit se faire que sur des machines auxquelles vous pouvez faire entièrement confiance. Une personne malintentionnée avec un accès d'administration sur ces machines pourrait en effet utiliser le transfert de votre trousseau pour se faire passer pour vous.
- Lorsqu'on dépose un *clé_publique* pour autoriser un accès SSH, il est possible de spécifier des règles de sécurité à appliquer (restrictions sur la commande possible à lancer, sur les adresses des machines d'où on peut se connecter…).
- On ne place jamais la *clé_publique* d'un utilisateur tiers sur son compte, car celui-ci pourra alors se connecter en usurpant votre identité. S'il y a des informations à partager ce n'est pas le bon moyen.
- Le logiciel d'accès distant serveur SSH est sensible aux droits d'accès donnés sur les répertoire ~/.ssh et les fichiers qu'il contient. Typiquement vous seul devez y avoir accès (chmod 700 ~/.ssh et chmod 600 ~/.ssh/*).
- Il est possible d'avoir plusieurs jeux de clés, par exemple utilisés dans des contextes de sécurité différents. Attention toutefois à ne pas les multiplier, et à la complexification des procédures de connexion.

Configuration

Un fichier de configuration `~/.ssh/config`[2] permet de définir des options à appliquer suivant les machines sur lesquelles on doit se connecter.

Ce fichier enchaîne, du début à la fin, l'identification de modèles et l'application des options qui suivent. Pour chaque connexion, l'ensemble du fichier est parcouru (plusieurs modèles peuvent être reconnus et leurs options appliquées). S'il y a une connexion relai par une passerelle, il est aussi parcouru *séparément* pour celle-ci. Les options positionnées avant le premier modèle sont systématiquement appliquées.

Si une option a déjà été fixée, sa valeur ne peut pas être redéfinie. **Les modèles les plus génériques doivent donc être positionnés à la fin du fichier.**

L'identification de base des modèles, avec le mot clé **Host**, repose sur une reconnaissance de la machine cible de la connexion ssh (généralement par son nom DNS). Il est possible d'utiliser * pour spécifier différentes machines, et d'indiquer plusieurs chaînes de reconnaissance séparées par des espaces. Il est possible d'utiliser des noms à notre choix et de spécifier la machine cible avec l'option

HostName ; les noms seront utilisés par la complétion ssh sous Unix.

```
…
Host pl-ssh.lisn.upsaclay.fr
…
Host m123.lisn.upsaclay.fr m167.lisn.upsaclay.fr m54.lisn.upsaclay.fr
berlioz.lisn.upsaclay.fr
…
Host *.lisn.upsaclay.fr
…
Host via
…
Host via-inside
…
Host via-outside
…
```

L'identification avancée, avec le mot clé **Match**, permet de combiner logiquement différentes expressions pouvant être de la reconnaissance de machine cible (encore avec Host) ou le résultat d'exécution de programmes/scripts externes (avec **Exec** ou **!Exec**).

```
…
Match Host *.lisn.upsaclay.fr Exec ~/bin/inside_lisn.sh
…
Match Host *.lisn.upsaclay.fr !Exec ~/bin/inside_lisn.sh
…
Match Host subversion.renater.fr,git.renater.fr !Exec ~/bin/inside_lisn.sh
…
Match Exec "nslookup %h | grep 'pl-ssh.lisn.upsaclay.fr'"
…
```

Avec `Match Host`, il faut séparer les différents noms de machines cibles par des virgules.

Avec `Match Exec`, l'expression est considérée valide si le programme/script sort avec un code `0` et invalide s'il sort avec un code ≠`0` (≈erreur).

Options ssh

Liste non exhaustive, se référer à la documentation pour la liste complète. Ces options peuvent être surchargées lorsd de l'appel à `ssh` avec des arguments `-o Option=Valeur`.

Accès à la machine distante

- **HostName** indique le nom DNS ou bien l'adresse IP du serveur distant à contacter (défaut fourni par la ligne de commande). La possibilité de spécifier ici la machine cible permet de se créer de multiples entrées dans la configuration du client ssh et d'utiliser la complétion basée sur les modèles *Host* avec des noms de machine arbitraires de son choix.
  `HostName 129.175.17.124`
- **ServerAliveInterval** délai pour le renvoi d'un message en cas d'inactivité afin de maintenir la connexion ouverte (défaut 0, pas de message renvoyé).
  `ServerAliveInterval 60`
- **ServerAliveCountMax** nombre maximum de messages envoyés au cas où le serveur ne répond pas (défaut 3).
  `ServerAliveCountMax 2`
- **ProxyJump** indique une machine par laquelle rebondir. À noter que le fichier de configuration ssh est à nouveau utilisé pour cette connexion de rebond (ie. la configuration pour accéder à la machine de rebond est utilisée).
  `ProxyJump pl-ssh.lisn.upsaclay.fr`
- **Port** permet de faire une connexion ssh vers un port non standard (défaut 22).
  `Port 443`

Avant l'existence de l'option ProxyJump, on utilisait l'option **ProxyCommand** `ProxyCommand ssh -W %h:%p pl-ssh.lisn.upsaclay.fr`.

Authentification

- **User** permet de spécifier le login par défaut à utiliser (défaut login local actuel).
  `User monlogin`
- **IdentityFile** indique le fichier de clé à utiliser pour l'authentification.
  `IdentityFile ~/.ssh/cle_specifique`
- **IdentitiesOnly** indique de n'utiliser que les clés fournies par IdentityFile[3] (défaut no).
  `IdentitiesOnly yes`
- **ForwardAgent** indique de transmettre une connexion vers l'agent ssh actuel (celui utilisé pour cette connexion) vers la machine distante afin de pouvoir utiliser ces mêmes clés (défaut no). Comme déjà indiqué, pour des raisons de sécurité cette option ne doit être activée qu'avec des machines de confiance et ne surtout pas être mise par défaut pour toutes les machines.
  `ForwardAgent yes`
- **PubkeyAuthentication** permet de désactiver/activer l'utilisation des clés pour la connexion (défaut yes).
  `PubkeyAuthentication no`
- **KbdInteractiveAuthentication** évite une possible demande de passphrase ou de mot de passe (défaut yes).
  `KbdInteractiveAuthentication no`

Tunnels

- **ForwardX11** ouvre un tunnel spécifique pour l'acheminement des requêtes d'affichage X11

vers la machine locale (défaut no). Cette option positionne la variable DISPLAY sur la session distante.
`ForwardX11 yes`

Inclusion

- **Include** (client OpenSSH≥7.3) permet de prendre en compte des éléments de configuration définis dans un autre fichier. Cela permet par exemple d'avoir votre configuration ssh sur un espace synchronisé entre différents ordinateurs.
`Include ~/nextCloudCirrus/configs/ssh_config`

Environnement

- **SendEnv** spécifie les variables d'environnement (ou ensemble de variables avec un pattern) qui doivent être transmises de la session sur la machine locale vers la session sur la machine distante.
`SendEnv GIT_*`

Il faut aussi que le serveur ssh de la machine distante ait été configuré pour accepter les variables d'environnement, par exemple dans `/etc/ssh/sshd_config` :

```
AcceptEnv LANG LC_* GIT_*
```

Exemples configuration ssh

À adapter avec **VOS paramètres de configuration** (login, passerelle, machines cibles, éventuellement fichiers de clés...).

Rappel, la configuration va du particulier au général, une option fixée avec un modèle ne peut pas être modifiée avec un modèle postérieur.

Exemple simple

```
# File: ~/.ssh/config
Host passerelle-labo
    HostName 129.175.8.242
    User bidule
    IdentityFile ~/.ssh/id_rsa
    IdentitiesOnly yes
```

```
Host ma_machine ma_machine.lisn.upsaclay.fr
    ForwardAgent yes
    ForwardX11 yes
    IdentitiesOnly yes

Host ma_machine *.lisn.upsaclay.fr
    User bidule
    IdentityFile ~/.ssh/id_rsa

Host *.lisn.upsaclay.fr
    ProxyJump passerelle-labo
```

Exemple intermédiaire

Exemple avancé

Plus complet, avec un test si vous êtes dans ou hors du laboratoire (besoin de passer par la passerelle ou non) en utilisant un script externe. Et avec un exemple d'accès à Lab-IA avec un ou deux rebonds (rebond par une machine autorisée vers Lab-IA, avec rebond par la passerelle si nécessaire).

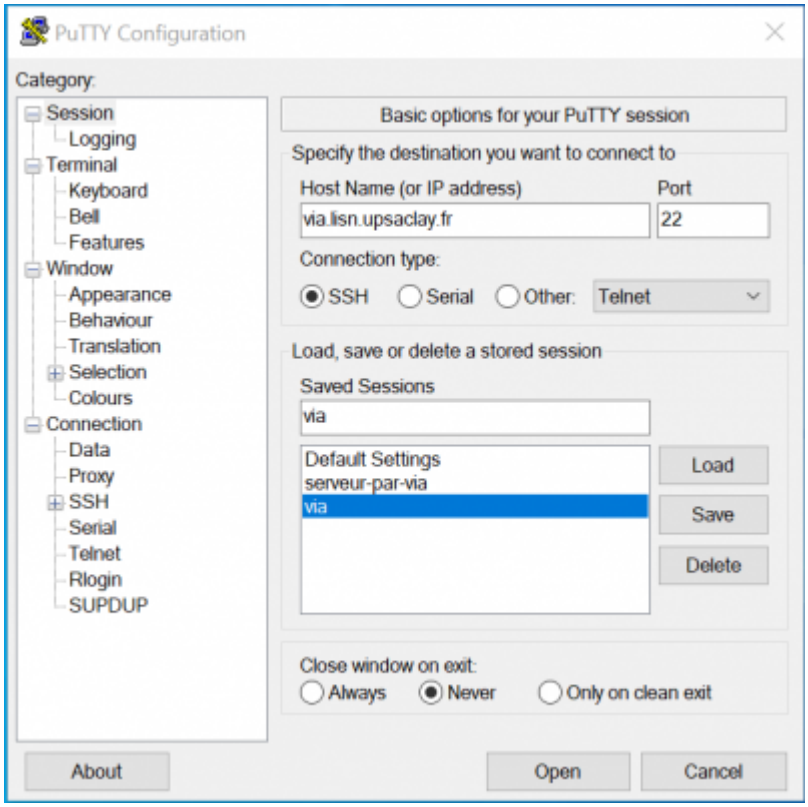
Attention: l'utilisation d'un script externe avec Exec peut compliquer la recherche lors de problèmes de connexion ssh. C'est une fonctionnalité pour utilisateur averti (et capable de se débrouiller...).
En alternative il vaut mieux utiliser une configuration un peu plus longue basée sur la reconnaissance de la machine saisie sur la ligne de commande (par exemple avec/sans le nom de domaine) pour sélectionner les options à utiliser.

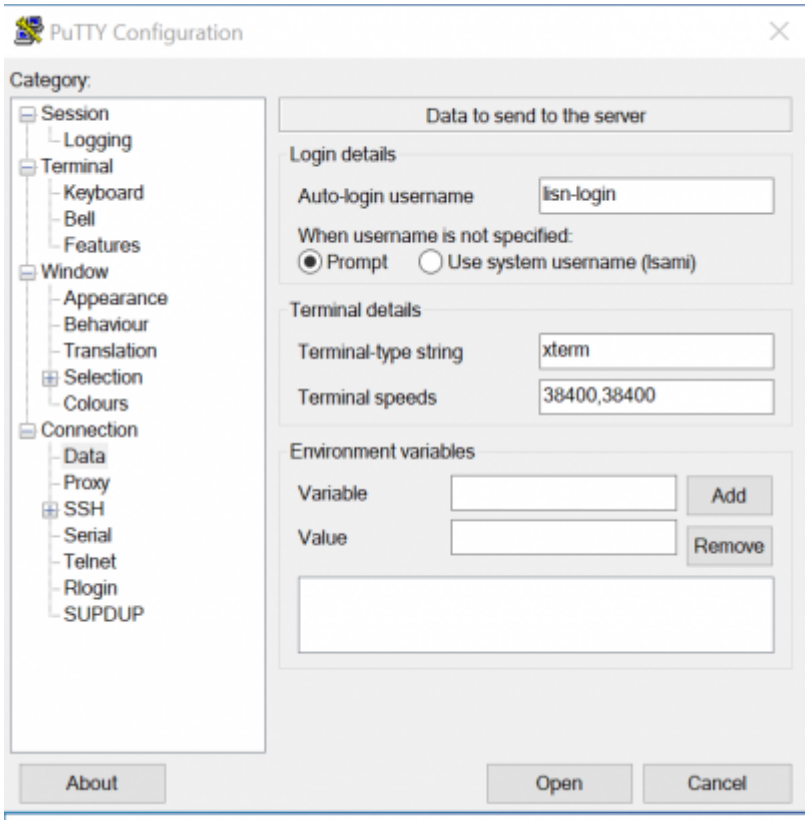
```
# File: ~/.ssh/config
Host passerelle-labo
    HostName 129.175.8.242
    User bidule
    IdentityFile ~/.ssh/id_rsa
    IdentitiesOnly yes

Match Host ma_machine.lisn.upsaclay.fr
    ForwardAgent yes
    ForwardX11 yes
    IdentitiesOnly yes

Match Host ma_machine.lisn.upsaclay.fr !Exec
~/nextCloudCirrus/bin/inside_lisn.sh
    ProxyJump passerelle-labo

Host *.lisn.upsaclay.fr
    User bidule
    IdentityFile ~/.ssh/id_rsa
```

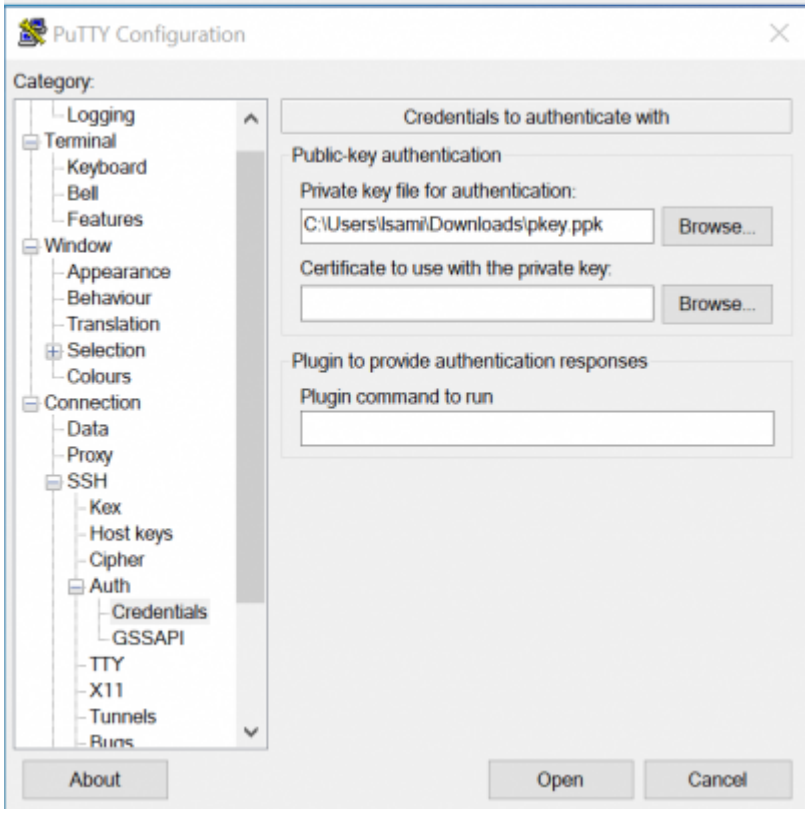
```
Match Host *.lisn.upsaclay.fr !Exec ~/nextCloudCirrus/bin/inside_lisn.sh
    ProxyJump passerelle-labo

Host slurm slurm.lab-ia.fr
    User bidulealt
    IdentityFile ~/.ssh/id_rsa
    IdentitiesOnly yes
    # Là la passerelle dépend si vous êtes côté belvédère ou plaine.
    ProxyJump verslabia.lisn.upsaclay.fr
```

Configuration vec putty

Création d'un profil de connexion à via.lisn.upsaclay.fr

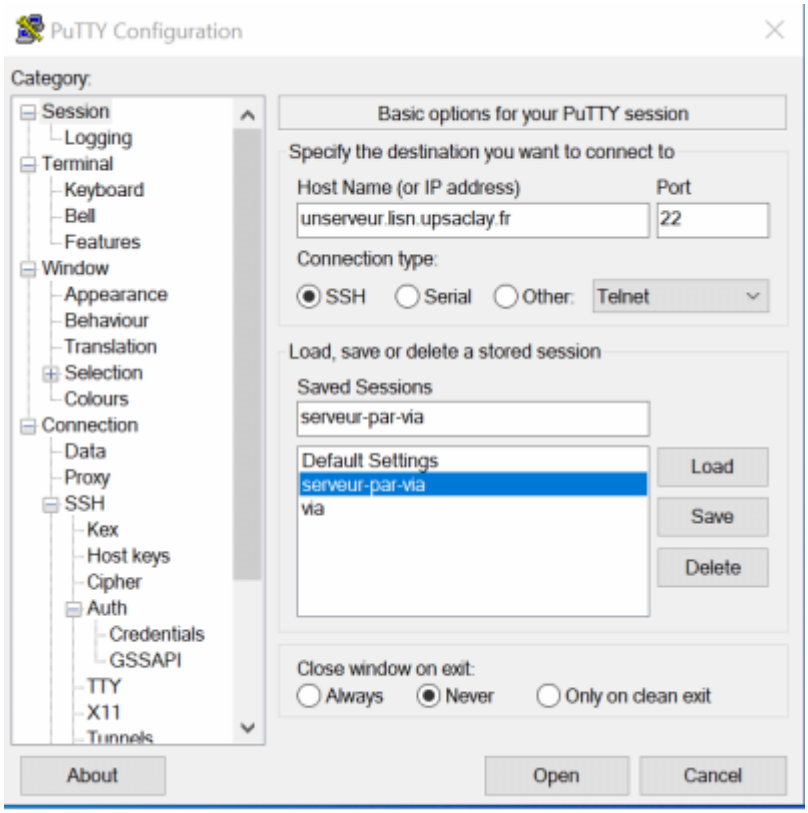


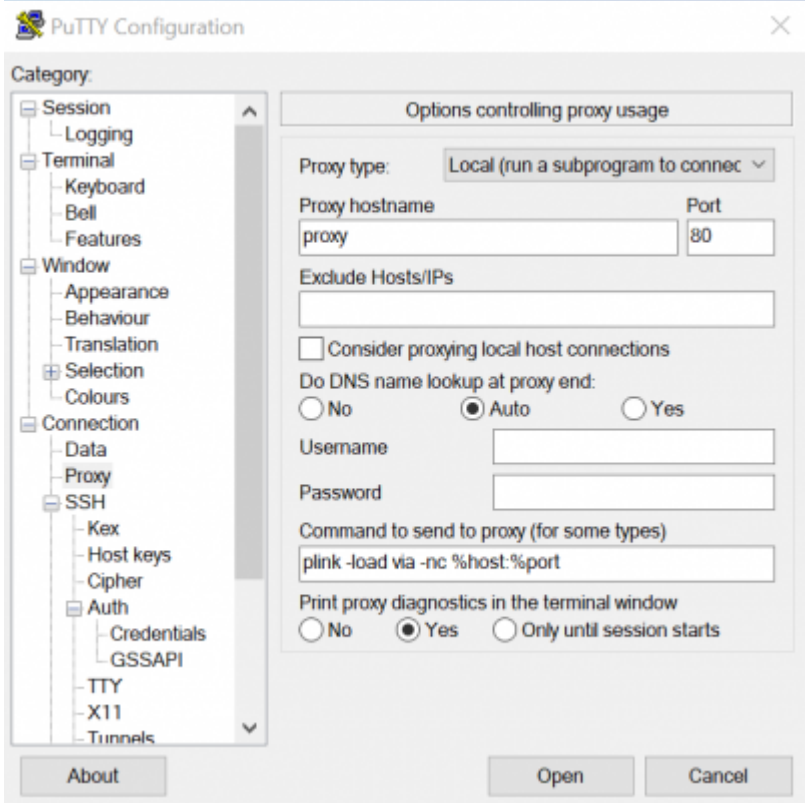




- Cette connexion sera utilisée pour le rebond à l'intérieur du labo.
- remplacer lisn-login par votre login
- attention au format de la clé, propre à putty

Création d'une connexion vers une machine interne en utilisant via





- utilisez ce profil pour rebondir sur une machine interne
- remplacer 'unserveur.lisn.upsaclay.fr' par le serveur sur lequel vous voulez aboutir

Trucs & astuces

Raccourcis avec complétion

Il est possible de définir, dans le fichier de config `.ssh/config`, des **Host avec les noms que l'on veut**. En ligne de commande il y a une complétion sur `ssh` qui utilise ces définitions de Hosts, on peut donc avoir des *raccourcis* pour les machines couramment utilisées, pour lesquelles on fournit tous les paramètres de connexion, et qui sont facilement activables (raccourcis utilisables aussi avec `scp`...).

```
Host lisn-perso
    User bidule
    IdentitiesOnly yes
    IdentityFile ~/.ssh/id_rsa
    Hostname ma_machine.lisn.upsaclay.fr
    ForwardX11 yes
    ProxyJump passerelle-labo
```

En saisissant `ssh lisn<tab>` tous les hosts définis en *lisn* dans le fichier de config vont s'afficher et s'il y en a un seul qui est sélectionnable il sera directement complété, par exemple `ssh lisn-perso`.

acces distant
[1)]

la publique ET la privée ; en fichiers dans votre KeePass, ou bien sur votre MyCore du CNRS, ou encore sur votre Cirrus de l'Université
[2)]

L'option `-F` de ssh permet de spécifier le fichier de configuration à utiliser.
[3)]

Peut être utile si on a de nombreuses clés ssh déverrouillées dans l'agent ssh, pour ne pas dépasser le nombre d'essais avant que la bonne clé ne soit testée.

From:
https://docsami.lisn.upsaclay.fr/ - **Service d'Administration des Moyens Informatiques**

Permanent link:
https://docsami.lisn.upsaclay.fr/ssh?rev=1681997580

Last update: **2023/04/20 15:33**