Table des matières

Public : utilisateurs de SSH

Utiliser SSH

← Accès distants

Le protocole SSH est très utilisé pour se connecter à distance à des systèmes de type Unix, mais aussi pour tunneliser des communications à la manière d'un VPN ou pour transférer des fichiers. Dans tous ces cas, les communications sont chiffrées.

L'authentification par clef publique est considérée comme plus robuste que l'authentification par mot de passe. Protégée par une passphrase, la clef privée offre deux facteurs d'authentification puisqu'il faut être en possession de la clef et de la passphrase.

Elle repose sur les mêmes mécanismes que ceux protégeant les communications Web.

Le principe est simple : un challenge est généré par le serveur et chiffré avec la clef publique. Seule la clef privée permet de le déchiffrer, authentifiant ainsi le client.

Dans toute la suite, sont brièvement présentés les outils OpenSSH sous Unix et PuTTY sous Windows (depuis la version 0.77), leur installation n'est pas détaillée ici. NB : le client OpenSSH natif de Windows 10/11 fonctionne comme sous Unix.

Clefs SSH

Supports

Traditionnellement, les clefs SSH sont stockées dans des fichiers. Néanmoins, leur stockage dans des jetons matériels (PKCS#11, U2F) se démocratise vite. Évidemment, subtiliser une clef SSH stockée dans un tel jeton oblige à voler le jeton lui-même, puis parvenir à en exploiter les secrets, ces jetons étant justement conçus pour que les secrets ne puissent pas en être extraits.

La DSI peut guider le choix de jetons matériels, par exemple pour faciliter l'authentification sur la passerelle SSH, cf. Authentification U2F et passerelle SSH.

Information : les puces TPM les plus récentes peuvent faire office de jeton pour stocker de tels secrets.

Générer une clef avec les outils OpenSSH

Règle : seules les clefs Ed25519 et P-256, P-384, P-521 sont acceptées par la DSI (RSA avec des clefs de plus de 3072 bits est toléré).

clef Ed25519

Pour générer un couple de clefs, utiliser la commande `ssh-keygen(1)`. Pour une clef Ed25519, ajouter l'option `-t ed25519` :

```
$ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/u/user/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /u/user/.ssh/id_ed25519.
Your public key has been saved in /u/user/.ssh/id_ed25519.pub.
The key fingerprint is:
SHA256:jbJGYcLohytAENcDPAWd4OK0k2cOe0IqQWGN4q1955U user@host
The key's randomart image is:
+--[ED25519 256]--+
|o=BB..           |
|++++=            |
|+++.o.o          |
|=+oo o . o       |
|oB=o. o S o      |
|+oB+ o + E       |
|+o.o. = .        |
|..o  . .         |
|                 |
+----[SHA256]-----+
```

clef RSA

Pour une clef RSA, ajouter les options `-t rsa` pour le type et `-b 3072` pour la taille des clefs.

```
$ ssh-keygen -t rsa -b 3072
Generating public/private rsa key pair.
Enter file in which to save the key (/u/user/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /u/user/.ssh/id_rsa.
Your public key has been saved in /u/user/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:nE6rzJEryMwTTxlVVWyyo7Ku54dvb2hROmItmagFVG0 user@host
```

```
The key's randomart image is:
+---[RSA 3072]----+
|    ... ....o.   |
|  .    E   . o   |
| .    o     +    |
|  . .  . .+      |
|   . + +S+ .     |
|  . = B+*.       |
| + B .o*o+       |
|  B oo++= .      |
|   ..*B=.o.      |
+----[SHA256]-----+
```

Dans tous les cas, **la clef doit être protégée par une *passphrase***. Une passphrase est plus longue qu'un mot de passe et peut ne pas être aléatoire comme doit l'être un mot de passe, pourvu qu'elle ait une plus grande entropie qu'un bon mot de passe (en 2022 : 12 caractères aléatoires).

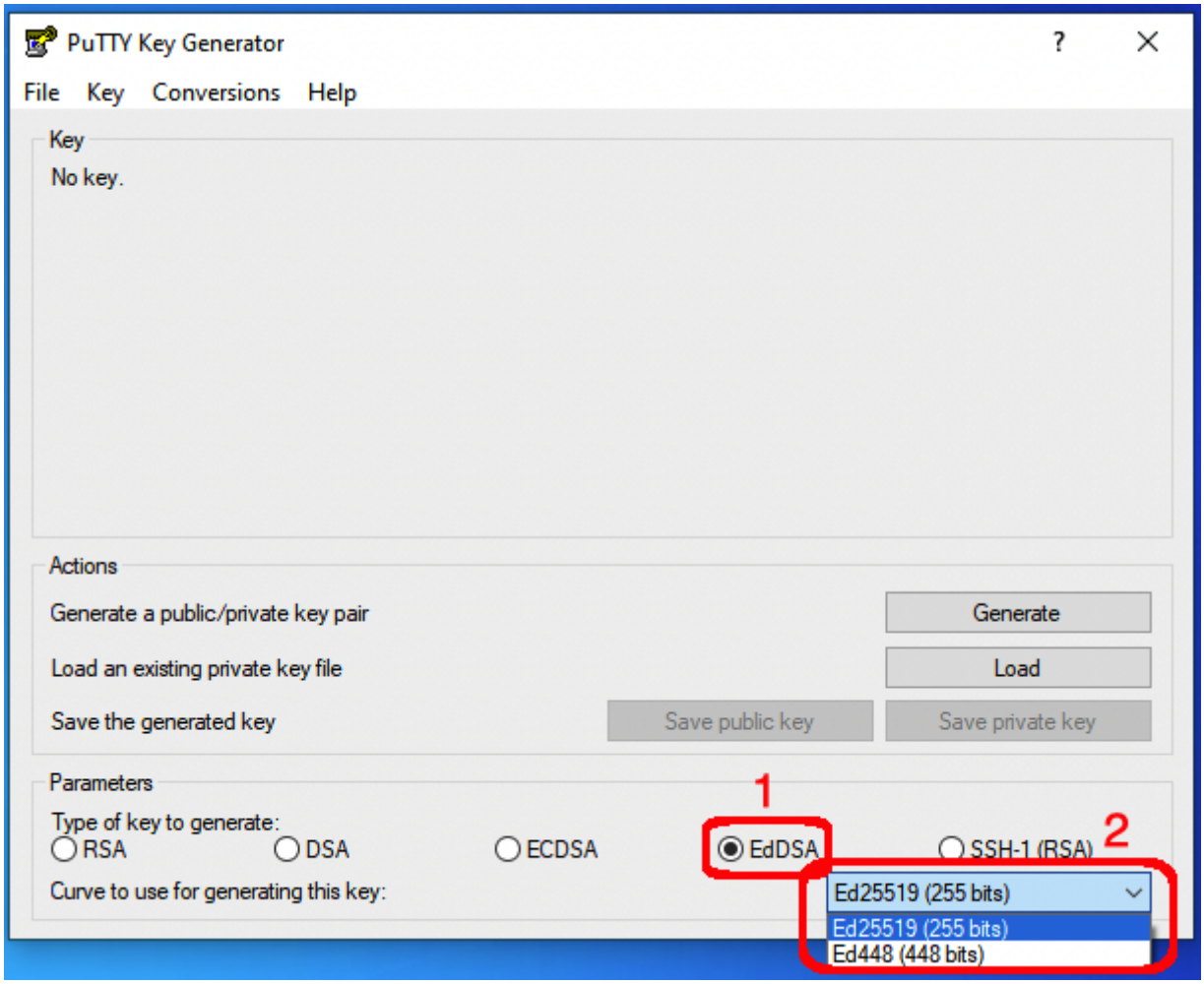
La clef publique est alors dans le fichier `~/.ssh/id_rsa.pub` ou `~/.ssh/id_ed25519.pub` comme cela est clairement indiqué dans la sortie de la commande. C'est **uniquement** ce fichier qui doit être distribué ; il peut bien évidemment être envoyé en clair. Par contre, la clef privée, elle, doit rester privée et ne pas quitter la station de travail de l'utilisateur, réputée « maîtrisée et sûre ».

Attention à protéger ses clefs : avec l'authentification par clefs publiques, une partie de la sécurité repose sur le soin que les utilisateurs déploient pour protéger leurs clefs.

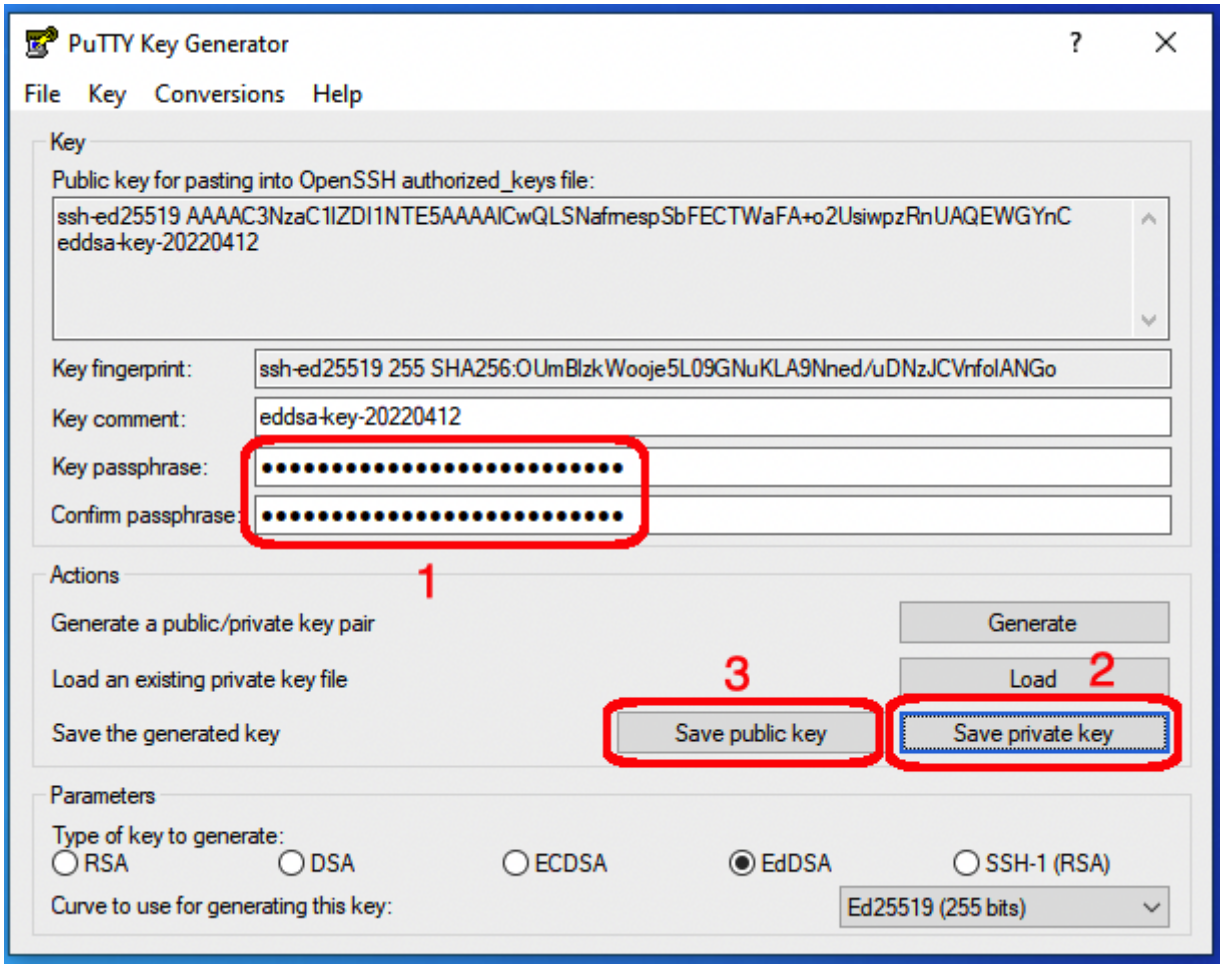
Se souvenir que la sécurité n'est pas un produit qui s'achète, mais des procédures que l'on applique, pour se protéger soi, mais aussi et surtout, pour protéger les autres.

Générer une clef avec PuTTYgen sous Windows

Lancer PuTTYgen et paramétrer le type de clef : par défaut RSA ; ici l'utilisateur a choisi une clef Ed25519. Le paramétrage fait, cliquer sur *Generate*.



La clef est maintenant générée. Il faut l'enregistrer et pour cela la protéger par une *passphrase* : **la clef doit être protégée par une *passphrase***. Une passphrase est plus longue qu'un mot de passe et peut ne pas être aléatoire comme doit l'être un mot de passe, pourvu qu'elle ait une plus grande entropie qu'un bon mot de passe (en 2022 : douze caractères aléatoires).



Enregistrer les clefs sur le disque. Les clefs privées PuTTY ont une extension `.ppk` ; on pourra utiliser `.pub` pour les clefs publiques. Il est aussi pratique de garder par devers soi la clef publique au format OpenSSH *i.e.* copier-coller le texte dans le cartouche intitulé *Public key for pasting into OpenSSH authorized_keys file* et le garder dans un fichier à conserver avec les deux précédents et nommé comme on voudra. C'est ce dernier fichier qu'il faudra déployer.

Attention à protéger ses clefs : avec l'authentification par clefs publiques, une partie de la sécurité repose sur le soin que les utilisateurs déploient pour protéger leurs clefs.

Se souvenir que la sécurité n'est pas un produit qui s'achète, mais des procédures que l'on applique, pour se protéger soi, mais aussi et surtout, pour protéger les autres.

Authentification avec une clef

Utiliser une clef avec les outils OpenSSH

Par défaut, les clients `ssh(1)`, `slogin(1)`, `scp(1)` et `sftp(1)` cherchent à utiliser les clefs sur la machine locale *i.e.* cherchent à charger `~/.ssh/id_rsa` ou `~/.ssh/id_ed25519` si le serveur

distant propose l'authentification par clef :

```
$ ssh remote-host
Enter passphrase for key '/home/pnom/.ssh/id_ed25519':
```

Il faut alors saisir la passphrase pour que le client déchiffre la clef privée et s'authentifie avec auprès du serveur.

Plutôt que de saisir la passphrase à chaque connexion, on utilise `ssh-agent(1)`, un *daemon* qui tourne sous l'identité de l'utilisateur (sur la machine locale). Ce programme échange avec les clients *via* une socket Unix connue par la variable d'environnement `$SSH_AUTH_SOCK` :

```
$ echo $SSH_AUTH_SOCK
SSH_AUTH_SOCK=/tmp/ssh-wAwzfWyDazvK/agent.1014
```

Le plus sûr moyen de passer une variable d'environnement est d'utiliser l'héritage de `fork(2)`. `ssh-agent(1)` affiche les variables au lancement. Avec l'option `-s`, ces variables seront présentées comme instructions Bourne :

```
$ ssh-agent -s
SSH_AUTH_SOCK=/tmp/ssh-wAwzfWyDazvK/agent.1014; export SSH_AUTH_SOCK;
SSH_AGENT_PID=1014; export SSH_AGENT_PID;
echo Agent pid 1014;
```

Si les clients `ssh(1)`, `slogin(1)`, `scp(1)` et `sftp(1)` connaissent cette variable, ils utiliseront les clefs chargées par l'agent.

Il est d'usage dans la configuration du shell de login d'ajouter :

```
eval $( ssh-agent -s )
```

pour charger l'agent et ensuite hériter des variables d'environnement.

Méthode recommandée : il est possible de simplifier encore en utilisant Keychain.

Les interactions avec l'agent se font avec `ssh-add(1)` : pour charger la clef pour 10 minutes :

```
$ ssh-add -c -t 600 ~/.ssh/id_ed25519
Enter passphrase for ~/.ssh/id_ed25519 (will confirm each use):
Identity added: ~/.ssh/id_ed25519 (nomp@machine-locale)
Lifetime set to 600 seconds
The user must confirm each use of the key
```

Astuce : pour se faciliter la vie, sur le poste local avec

interface graphique *i.e.* sur la machine où est stockée la clef, installer `ssh-askpass` (sur Linux) ou `ssh-askpass-mac` (sur macOS).

Pour vérifier les clefs chargées dans l'agent (la première commande montre le haché, la seconde la clef publique) :

```
$  ssh-add -l
256 SHA256:a4gp...r620 nomp@machine-locale (ED25519)
$ ssh-add -L
ssh-ed25519 AAA...FdA9 nomp@machine-locale
```

Pour vider les clefs de l'agent :

```
$ ssh-add -D
```

Pour terminer le processus `ssh-agent(1)` :

```
$ ssh-agent -k
unset SSH_AUTH_SOCK;
unset SSH_AGENT_PID;
echo Agent pid 41740 killed;
```

utile, par exemple dans un fichier `$HOME/.bash_logout`.

Il est évidemment possible d'avoir des clefs dans des fichiers autres qu'`id_rsa` ou `id_ed25519`. Il suffit alors d'indiquer le chemin vers le fichier considéré :

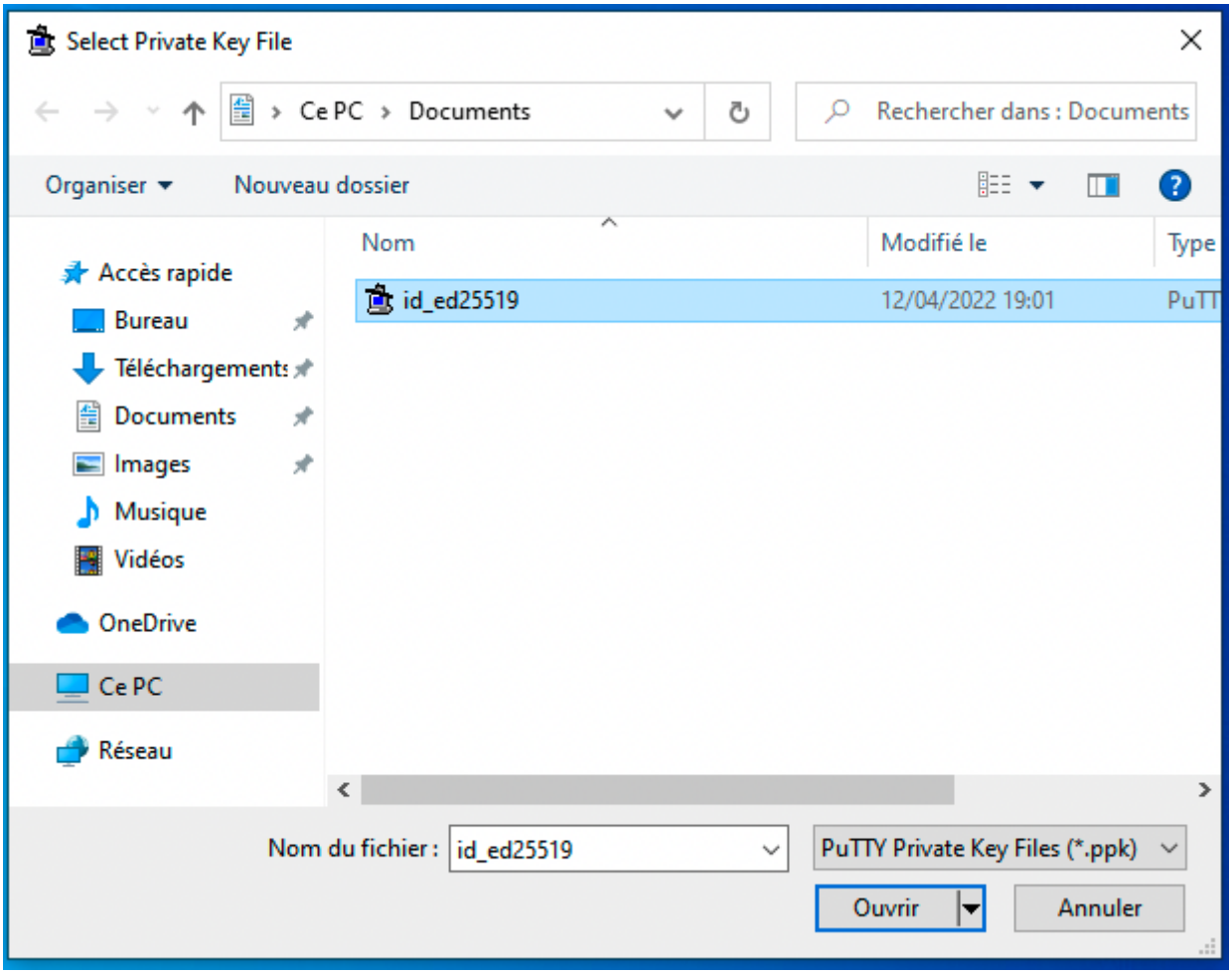
```
$ ssh -i ~/.ssh/cnam_rsa remote-host
```

Pour ne pas saisir l'option à chaque connexion, on peut aussi enregistrer le réglage dans `$HOME/.ssh/config` sur la machine locale avec :

```
Host remote-host
    IdentiyFile /home/pnom/.ssh/cnam_rsa
```

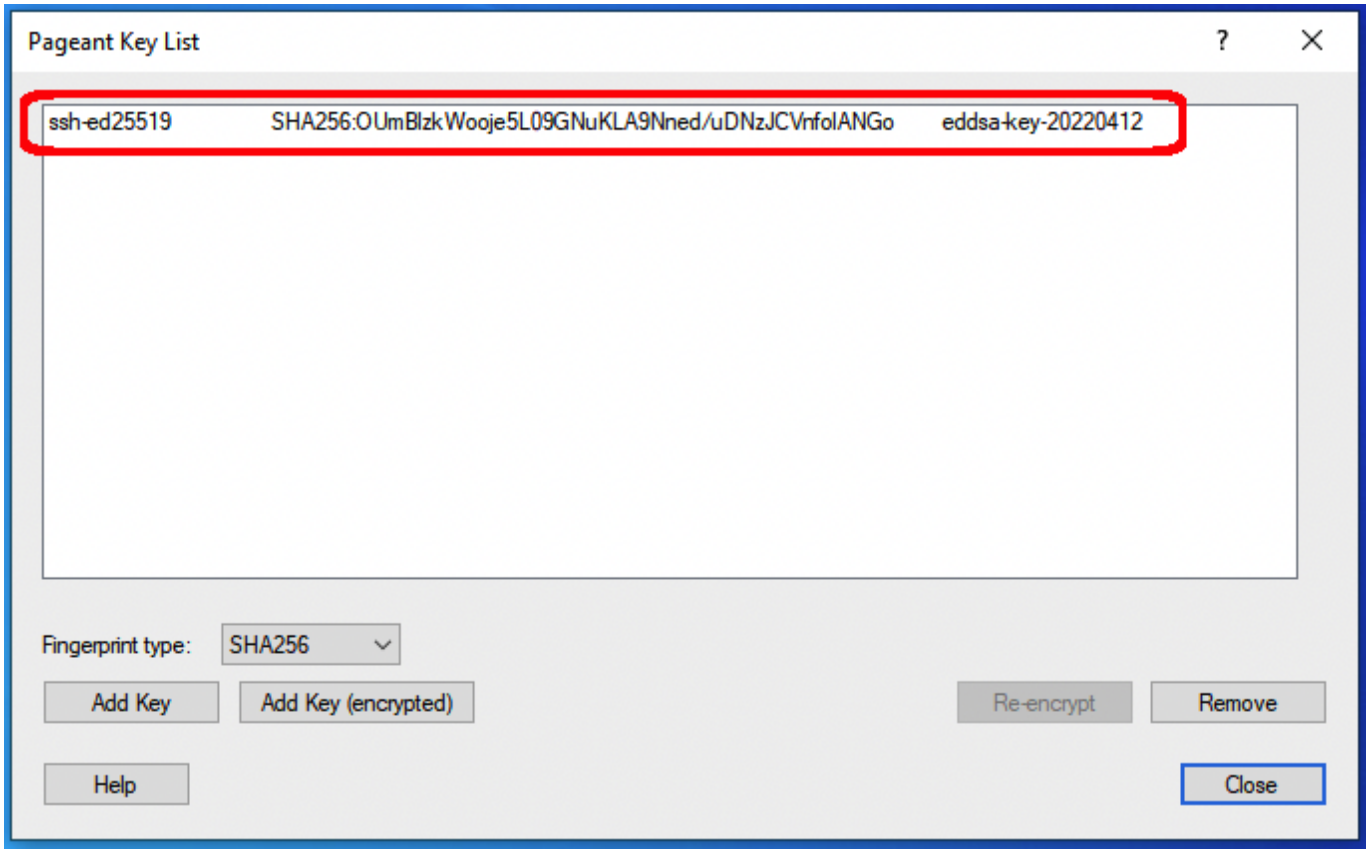
Utiliser une clef avec pageant

Sous Windows, lancer `pageant.exe`. Il apparaît dans le *systray* à côté de l'horloge. Cliquer dessus puis sur *Select Private Key File* et ouvrir la clef privée.



La passphrase est demandée pour accéder à la clef privée :

La clef est enfin chargée :



La clef n'est chargée que pour une durée limitée. De plus, cette opération est à faire avant d'utiliser la clef.

Déployer une clef publique sur un serveur Unix

La clef publique (*i.e.* le contenu de ce fichier `.pub`) doit être enregistrée sur le serveur dans `$HOME/.ssh/authorized_keys` (fichier par défaut utilisé par le serveur OpenSSH). On peut transmettre le contenu de ce fichier par tout canal de communication, chiffré ou non : comme son nom l'indique, c'est une donnée publique. Il contient une ligne au format :

```
<type> <clef publique en base64> <identifiant de clef>
```

où le `type` indique `ssh-rsa` ou `ssh-ed25519` par exemple suivant le type de clef. L'identifiant est une chaîne de caractères à la discrétion de l'utilisateur, elle n'est qu'indicative.

Attention à ne pas transmettre la clef privée ! Avec PuTTY, la clef privée est dans fichier ayant pour extension `.ppk`. Avec OpenSSH, cette dernière est dans un fichier au format PEM qui commence par une ligne `-----BEGIN OPENSSH PRIVATE KEY-----` et termine par une ligne `-----END OPENSSH PRIVATE KEY-----`.

Règle : si jamais une clef privée est transmise par inadvertance, elle doit être considérée comme compromise et détruite sans délai.

Sur le serveur, les droits sur `$HOME/.ssh/authorized_keys` doivent être soigneusement réglés :

- le répertoire de l'utilisateur `$HOME` ne doit pas être ouvert en écriture à quiconque hormis l'utilisateur lui-même (`0750` en octal par exemple),
- le répertoire `$HOME/.ssh` ne doit permettre lecture et écriture qu'à l'utilisateur (`0700` en octal),
- le fichier `$HOME/.ssh/authorized_keys` ne doit permettre lecture et écriture qu'à l'utilisateur (`0600` en octal).

Ainsi, pour l'utilisateur `pnom` membre de `group_u`, on doit avoir :

```
$ ls -ld ~
drwx------. 16 pnom group_u 4096 Apr  7 14:29 /home/pnom
$ ls -ld ~/.ssh
drwx------. 2 pnom group_u    82 Feb 18 00:04 /home/pnom/.ssh
$ ls -l ~/.ssh/authorized_keys
-rw-------. 1 pnom group_u   745 Oct 29  2017
/home/pnom/.ssh/authorized_keys
```

Le format du fichier `$HOME/.ssh/authorized_keys` est simple : une ligne par clef publique. C'est documenté dans le manuel de `sshd(8)`.

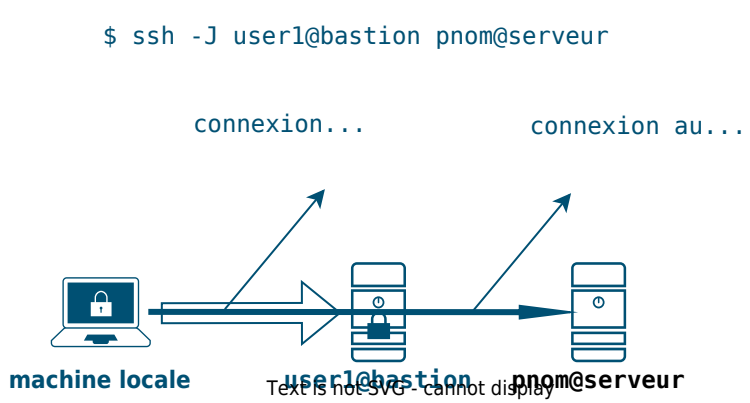
Si l'ordinateur (ou tout du moins le support) sur laquelle est stockée la clef privée est volé ou compromis, il faut supprimer la clef publique des serveurs sur lesquels elle était déployée *i.e.* faire le tour des `$HOME/.ssh/authorized_keys` pour supprimer la clef. De même, si la clef privée est envoyée par mégarde, elle ne doit plus jamais être utilisée.

Utilisation d'une machine de rebond avec ProxyJump

Pour ne pas exposer sur Internet de nombreuses machines, on peut utiliser une machine de rebond : il faut alors se connecter à une première machine et, de là, aux autres. Seule la première machine est atteignable depuis l'extérieur, les autres ne sont accessibles que sur le réseau interne.

Plutôt que de se connecter une première fois puis une seconde, on peut faire les deux connexions en une. Pour cela, on peut utiliser l'option `-J` suivie du nom de la machine de rebond, appelons-là `bastion` :

```
$ ssh -J user1@bastion pnom@serveur
```



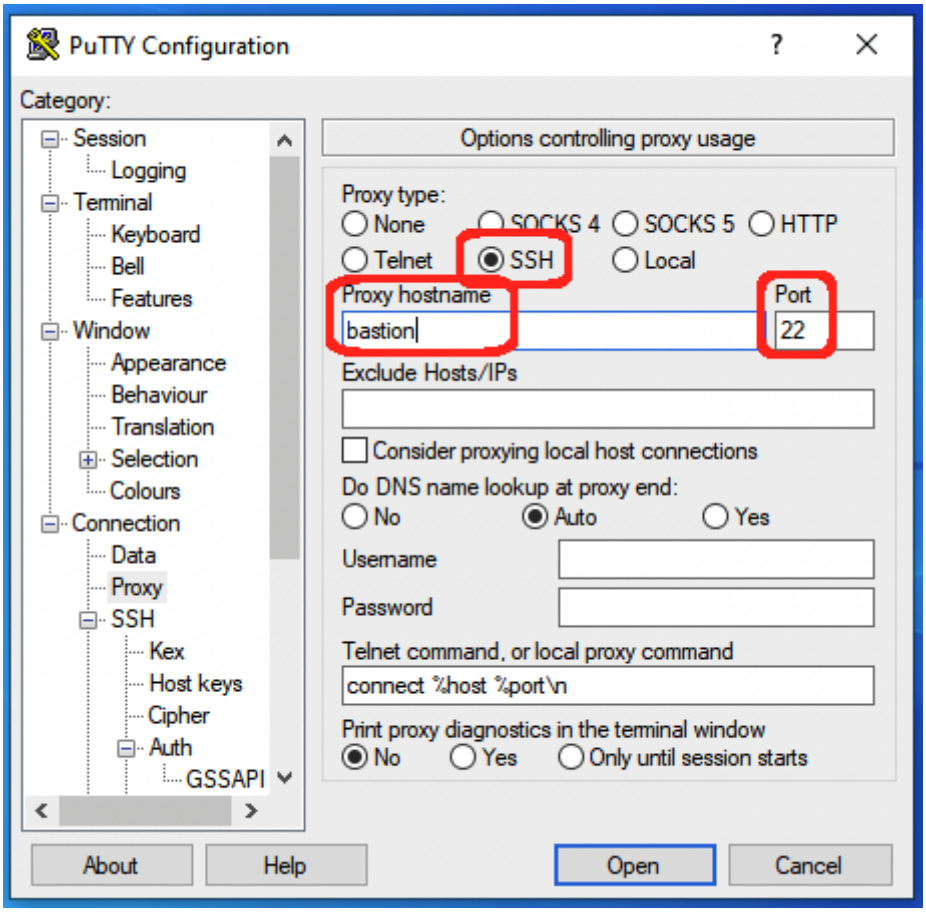
On peut enregistrer le réglage sur la machine locale dans `$HOME/.ssh/config` :

```
Host bastion
    User user1

Host srv
    ProxyJump bastion
    Hostname serveur
    User pnom
```

Pour se connecter, on utilisera alors la commande :

```
$ ssh srv
```

Avec PuTTY 0.77, dans *Connexion* » *Proxy*, cocher *SSH* et remplir adresse et port.



Agent forwarding

Comme il n'est pas question pour l'utilisateur de stocker sa clef privée ailleurs que sur son propre poste, mais qu'il est souvent utile d'en disposer sur une machine distante, on utilise alors le transfert d'agent ou *agent forwarding*.

Comme les accès à l'agent se font *via* des sockets Unix, il ne faut activer le transfert d'agent que vers des machines dont on est sûr : `root`, s'il est malicieux ou indélicat, peut accéder aux sockets et donc utiliser les clefs. L'option `-h` de `ssh-add(1)` permet de spécifier lors du chargement d'une clef, les machines sur lesquelles la clef est utilisable.

Lors de connexions à travers un bastion, configurer le client sur la machine locale. Cela se fait dans `$HOME/.ssh/config` (permissions `0600` en octal), par exemple avec :

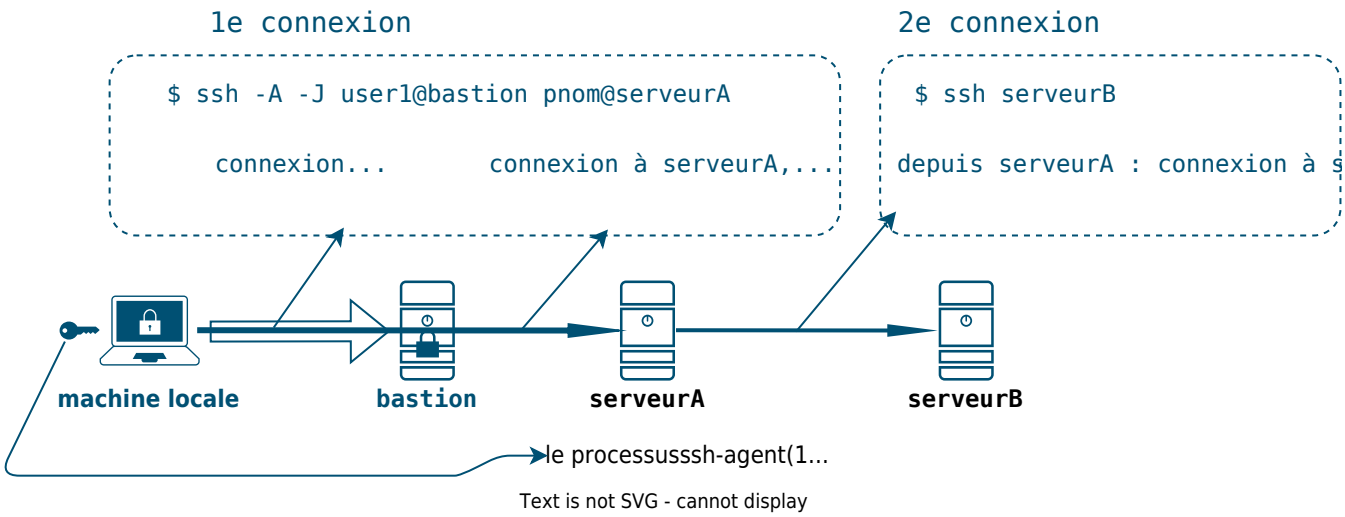
```
ForwardAgent no
IdentityFile ~/.ssh/id_ed25519

Host bastion
    User user1

Host srv
    ForwardAgent yes
    ProxyJump bastion
    Hostname serveur
    User pnom
```

Pour se connecter :

```
$ ssh srv
```

Ça peut être fait directement sans configuration :

```
$ ssh -i ~/.ssh/id_ed25519 -J user1@bastion -A pnom@serveur
```



← Accès distants