

[Accueil](#) > [Cours](#) > [Reprenez le contrôle à l'aide de Linux !](#) > La connexion sécurisée à distance avec SSH

Reprenez le contrôle à l'aide de Linux !

 30 heures  Facile

Mis à jour le 29/06/2021



La connexion sécurisée à distance avec SSH

Voici probablement l'un des chapitres les plus intéressants de ce livre. Nous allons découvrir comment se connecter à distance à une machine équipée de Linux.

Je vous en ai déjà un peu parlé au début de cet ouvrage : toutes les machines sous Linux peuvent être configurées pour que l'on s'y connecte à distance, pour peu qu'elles restent allumées.

Ici, nous n'allons pas seulement découvrir la connexion à distance. Nous allons aussi essayer de comprendre comment cela fonctionne et comment les données sont sécurisées grâce au **protocole SSH**. Ce chapitre sera donc l'occasion de découvrir de nouvelles notions sur le monde passionnant des réseaux et de la sécurité (chiffrement).

Prenons un cas concret : votre ordinateur chez vous est sous Linux, vous le laissez allumé. Pendant la journée au boulot, vous avez besoin de lancer un téléchargement ou de récupérer un document. Vous vous connectez à distance sur votre machine et vous ouvrez une console comme si vous étiez en face de votre PC ! Tout ce que vous avez appris à faire dans une console, vous pouvez le faire à distance depuis n'importe quelle machine dans le monde.



Ce chapitre intéressera en particulier ceux qui ont besoin d'apprendre à gérer un serveur dédié. Ils sont de plus en plus nombreux. Avoir son serveur permet d'héberger soi-même son site web ou fournir des services, comme un serveur de jeux. L'administration de serveur sous Linux se fait presque exclusivement en ligne de commande à distance, à l'aide de SSH.

Se connecter à une console à distance



Jusqu'ici, vous avez utilisé Linux de la même façon que Windows : vous étiez **en face** de votre ordinateur. Vous étiez physiquement à côté de votre machine, vous avez par exemple appuyé sur le bouton « Power » pour l'allumer. Jusque-là, rien de nouveau.

Pourtant, une des grandes forces de Linux est que l'on peut s'en servir même si l'on est à des centaines de kilomètres de la machine. Ce fonctionnement date de l'époque d'Unix où il était nécessaire d'administrer des machines à distance.

Aujourd'hui, si j'habite à Paris, je peux très bien contrôler un ordinateur sous Linux situé à Tokyo, au Japon, en même temps qu'un autre ordinateur situé au fin fond du Nevada, aux États-Unis. Je peux même ordonner à l'ordinateur de Tokyo d'envoyer un fichier à celui du Nevada.

Ce genre de manipulation est désormais possible grâce à l'internet, et cela se fait tous les jours. Les personnes qui s'emploient à gérer des machines Linux, souvent à distance, sont appelées **administrateurs système** (c'est un métier recherché !).

Heureusement qu'il n'est pas nécessaire d'être présent physiquement à côté de la machine pour travailler dessus ! Vous imaginez, devoir se payer un billet aller-retour pour Tokyo juste parce que l'on a besoin d'installer un programme sur un serveur...



Un **serveur** est un ordinateur qui reste allumé 24 h / 24, 7 j / 7. Cet ordinateur est semblable au vôtre (quoique souvent plus puissant et plus bruyant) : il possède un processeur, un ou plusieurs disques durs, etc.

Le principe d'un serveur est de rester allumé et connecté à l'internet tout le temps. Il offre des services. Par exemple, le Site d'OpenClassrooms possède plusieurs serveurs chargés de vous envoyer les pages web du site à toute heure du jour et de la nuit. :-)

Le PC qui se connecte au serveur est appelé le **client**. Nous allons les représenter comme sur la figure suivante dans les prochains schémas.



Actuellement, votre petit PC chez vous n'est pas considéré comme un serveur... mais vous pouvez très facilement le transformer en serveur si vous le désirez, à condition d'installer les bons programmes et de les configurer correctement.

Et de le laisser allumé aussi, parce qu'un serveur éteint, c'est un serveur qui ne sert à rien. 🙄

Nous allons suivre ce plan pour découvrir SSH :

1. Pourquoi faut-il sécuriser les échanges ?
2. Comment fait SSH pour sécuriser les échanges ?
3. Comment utiliser SSH concrètement ?

De Telnet à SSH



Les protocoles

Pour communiquer entre eux en réseau, deux ordinateurs doivent utiliser le même **protocole**. C'est un peu

comme une langue : pour parler à quelqu'un, vous devez parler la même langue que lui, sinon vous ne vous comprendrez pas.

Il existe de très nombreux protocoles pour que les ordinateurs puissent communiquer entre eux. Il y en a un que vous avez forcément vu, c'est le HTTP (*HyperText Transfer Protocol*). Si, si, regardez par exemple [l'adresse d'OpenClassrooms](#). Le préfixe `http` signifie que vous communiquez avec les serveurs d'OpenClassrooms à l'aide du protocole HTTP. C'est le protocole utilisé sur le web pour s'échanger des pages web.

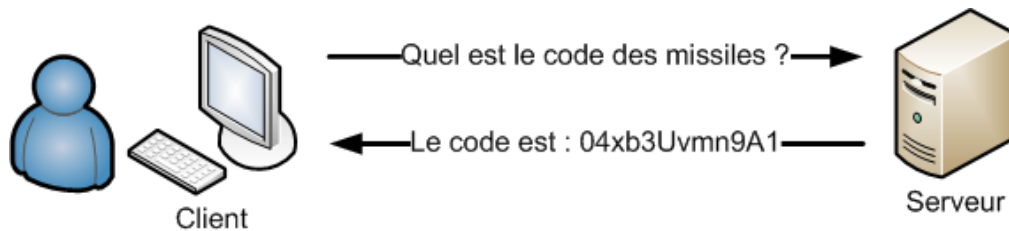
Mais il existe bien d'autres protocoles ! Par exemple le FTP (*File Transfer Protocol*, protocole de transfert de fichiers), l'IMAP (*Internet Message Access Protocol*, utilisé pour s'échanger des e-mails), etc.

Le protocole Telnet : simple mais dangereux

Un protocole très simple, très basique, a été créé dans les années 80 : c'est **Telnet**. Il sert juste à échanger des messages simples d'une machine à une autre.

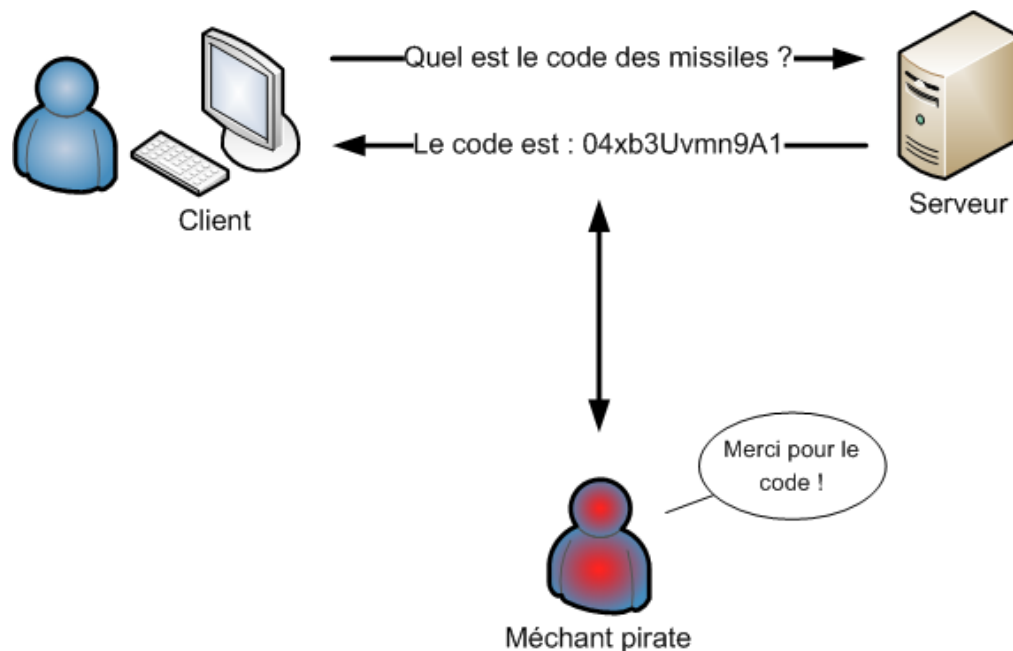
En théorie donc, on peut communiquer avec un serveur à l'aide du protocole Telnet. Le problème de ce protocole... c'est justement qu'il est trop simple : les données sont transférées en clair sur le réseau. Il n'y a aucun chiffrement.

Voici ce qui pourrait se passer. Je force le trait, mais c'est pour vous donner une idée. Imaginez qu'un PC militaire demande à un serveur de l'armée le code de lancement de missiles (nucléaires, soyons fous), comme sur la figure suivante.

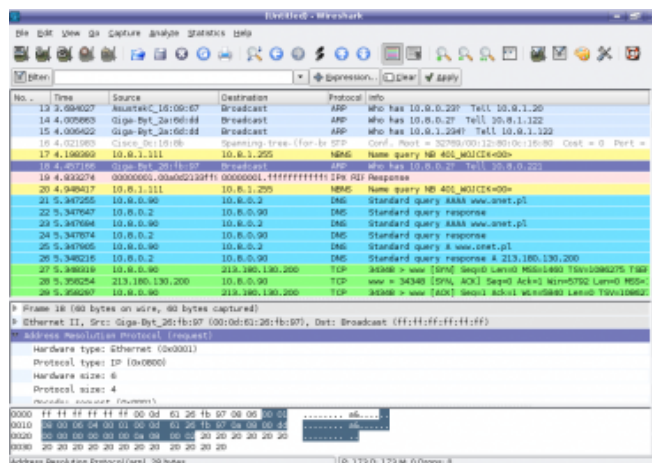


Après tout, il n'y a rien de choquant. Le message n'est envoyé qu'au client qui l'a demandé.

Mais en fait, un pirate aurait la possibilité d'« **écouter** » ce qui se passe sur le réseau, et donc d'intercepter les données en chemin (figure suivante).



Vous pouvez difficilement empêcher que quelqu'un intercepte les données. Intercepter les données peut être compliqué à réaliser, mais possible. Sachez qu'il existe par exemple des programmes comme Wireshark capables d'écouter ce qui se passe — notamment sur un réseau local — et donc d'intercepter les données (figure suivante).

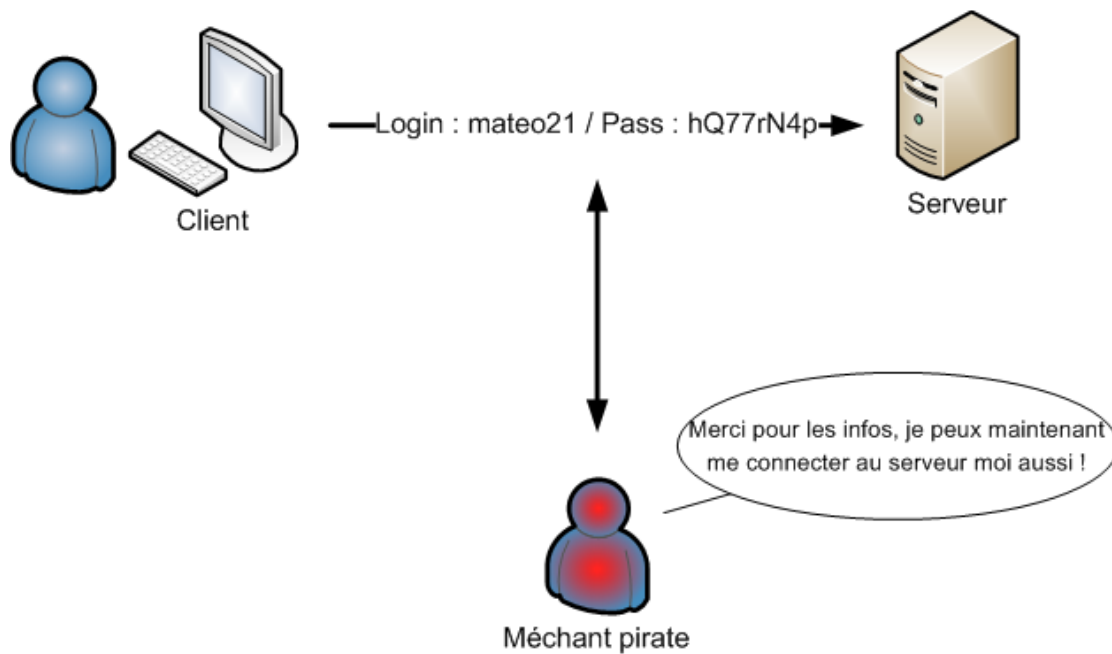


?

Euh... attends, là : moi, je veux juste me connecter à distance à ma machine ou à un serveur pour avoir l'accès à la console. Je ne vais pas échanger de code de lancement de missiles nucléaires ! Je vois pas en quoi c'est un problème si quelqu'un sait que je suis en train de faire un **grep** sur ma machine, par exemple...

Ça ne vous dérange pas que l'on vous espionne ? Soit.

Mais quand vous allez vous connecter au serveur, vous allez donner votre login et votre mot de passe. Rien que ça, c'est dangereux (figure suivante). Il ne faut pas que le login et le *pass* apparaissent en clair sur le réseau !



Rien que pour cela, il faut que les données soient chiffrées. Vous ne voulez pas que quelqu'un récupère votre mot de passe tout de même !

Le protocole SSH : la solution pour sécuriser les données

Comme on ne peut pas complètement empêcher quelqu'un d'intercepter les données qui transitent sur l'internet, il faut trouver un moyen pour que le client et le serveur communiquent de manière sécurisée. Le chiffrement sert précisément à ça : si le pirate récupère le mot de passe chiffré, il ne peut rien en faire.

Mais tout cela est plus compliqué que ça en a l'air. Comment chiffrer les données ?

Comment sont chiffrés les échanges avec SSH ?



SSH est un protocole assez complexe, mais il est vraiment intéressant de savoir comment il fonctionne. Plutôt que de l'utiliser bêtement, je vous propose de vous expliquer dans les grandes lignes son mode de fonctionnement.

Nous allons ici nous intéresser aux deux questions suivantes.

1. Quelles sont les différentes méthodes de chiffrement qui existent ?
2. Comment SSH utilise-t-il ces méthodes de chiffrement pour garantir la sécurité ?

Quelles sont les différentes méthodes de chiffrement ?

Il existe des tonnes d'algorithmes de chiffrement. Je ne vais pas tous vous les présenter : cela demanderait trop de notions mathématiques, on pourrait y consacrer 30 chapitres et on n'aurait pas tout vu.

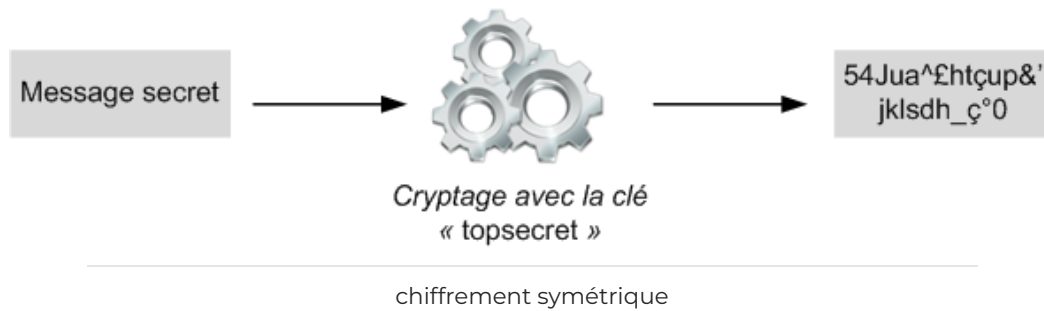
Si l'on ne peut pas connaître tous les algorithmes de chiffrement, il faut par contre savoir que l'on peut les classer en deux catégories : les chiffrements *symétriques* et les chiffrements *asymétriques*.

Le chiffrement symétrique

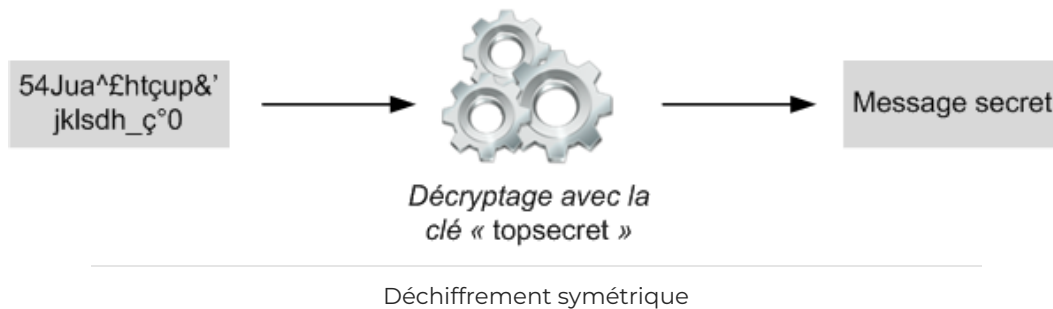
C'est la méthode de chiffrement la plus simple. Cela ne veut pas dire qu'elle n'est pas robuste (il existe des

chiffrements symétriques très sûrs). Cela veut plutôt dire que le fonctionnement est simple à comprendre. :-)

Avec cette méthode, on utilise une clé (un mot de passe secret) pour chiffrer un message. Par exemple, imaginons que cette clé soit **topsecret** (figure suivante).

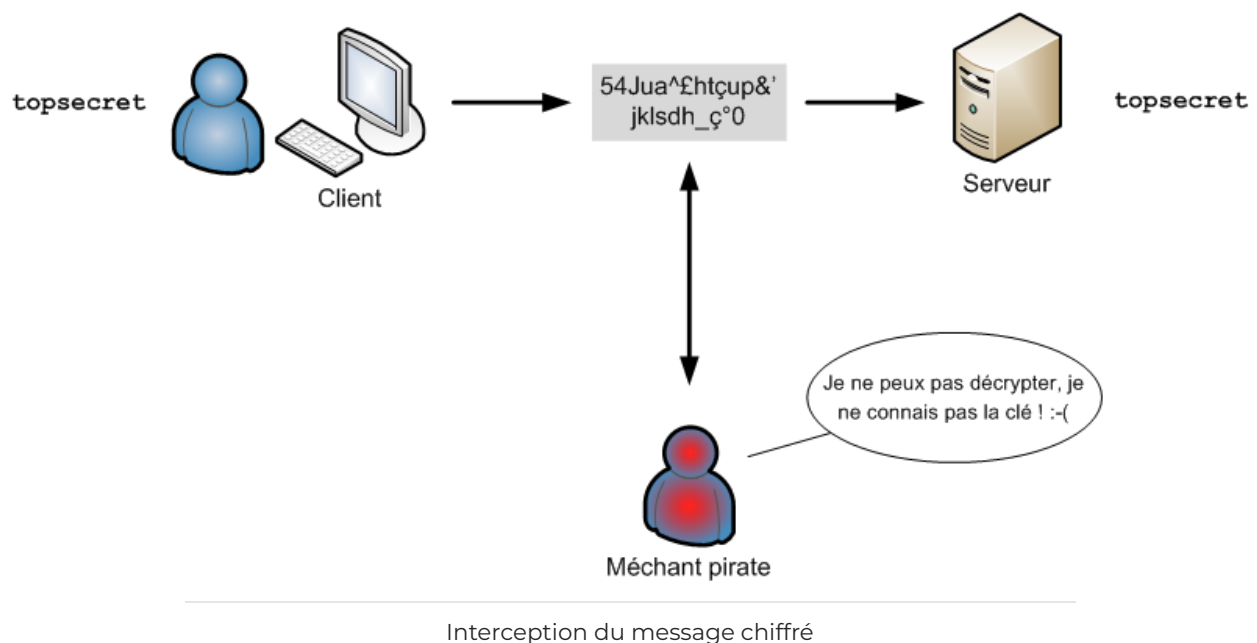


Pour déchiffrer ensuite le message, on utilise cette même clé (figure suivante)...



Il faut donc que la personne qui chiffre et celle qui déchiffre connaissent toutes deux cette clé qui sert à chiffrer et déchiffrer.

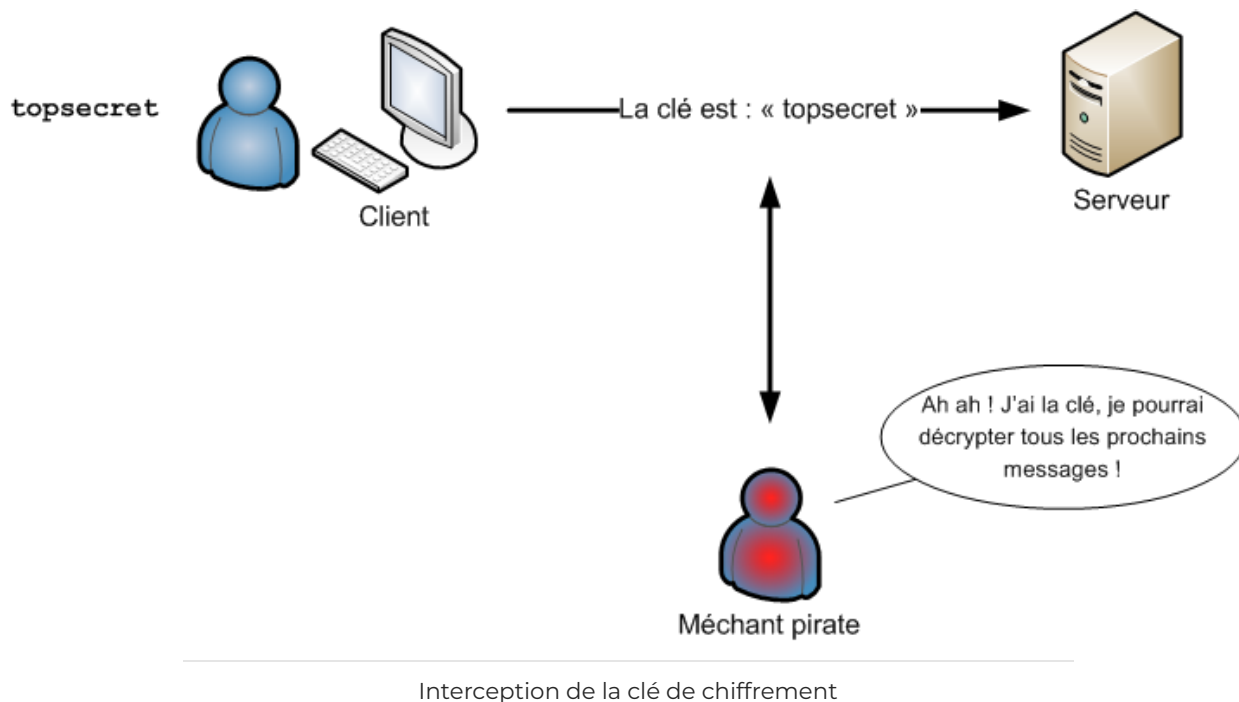
Si le pirate intercepte un message chiffré, **il ne peut rien en faire s'il ne connaît pas la clé secrète** (figure suivante) !



Ah... c'est bien, ça ! Mais il faut que le client et le serveur connaissent tous les deux la clé de chiffrement. Il faut donc que le client envoie d'abord au serveur la clé pour que celui-ci puisse déchiffrer ses futurs messages...

Très bonne remarque : je vois que vous suivez, c'est bien. ;-)

En effet, pour que le schéma que l'on vient de voir puisse fonctionner, il faut que le client et le serveur se soient transmis auparavant la clé magique qui sert à chiffrer et déchiffrer. Mais comment font-ils pour se l'échanger ? S'ils l'envoient en clair, le pirate va pouvoir l'intercepter et sera ensuite capable de déchiffrer tous les messages chiffrés qui passeront sur le réseau (voyez la figure suivante) !



Le chiffrement symétrique est donc puissant, mais il a un gros défaut : il faut communiquer « discrètement » la clé de chiffrement... mais c'est impossible : il faut bien envoyer la clé en clair au début !

...

À moins de... non...

Et pourquoi pas ? Si l'on chiffrait la clé de chiffrement lors de son envoi ? :-p

Pour chiffrer la clé de chiffrement symétrique, on va utiliser une autre méthode : le chiffrement asymétrique. Avec cette autre méthode, on ne risque pas de connaître à nouveau le problème que l'on vient de rencontrer.

Le chiffrement asymétrique

Le chiffrement symétrique utilise une seule clé pour chiffrer et déchiffrer.

Le chiffrement asymétrique, lui, utilise une clé pour chiffrer, et une autre pour déchiffrer.

Il y a donc deux clés :

- une clé dite « **publique** » qui sert à **chiffrer** ;

- une clé dite « **privée** » qui sert à **déchiffrer**.

La clé publique ne sert qu'à chiffrer. Avec ce type d'algorithme, on ne peut déchiffrer un message que si l'on connaît la clé privée.

On demande à l'ordinateur de générer une paire de clés : une privée et une publique. Elles vont ensemble. Ne me demandez pas comment il les génère ni pourquoi elles vont ensemble, c'est trop compliqué à expliquer ici. Admettez simplement que l'ordinateur est capable de générer aléatoirement un couple de clés qui vont ensemble.

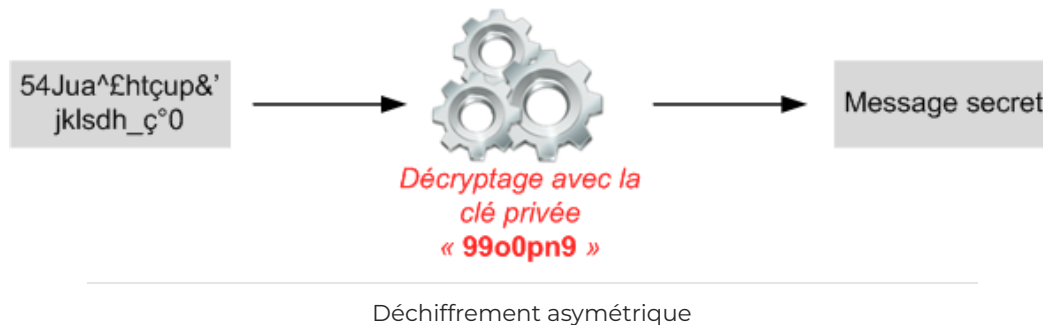
Prenons un exemple et imaginons que :

- la clé publique soit **74A48vXX** ;
- la clé privée soit **99o0pn9**.

Pour chiffrer, on utilise la clé publique, comme sur la figure suivante.



Pour déchiffrer, la clé publique ne fonctionne pas. Il faut obligatoirement utiliser la clé privée (figure suivante).



Voilà pourquoi on dit que c'est un chiffrement **asymétrique** : il faut deux clés différentes. L'une d'elles permet de chiffrer le message, l'autre de le déchiffrer. Il n'y a pas d'autre moyen.

La clé publique peut être transmise en clair sur le réseau (elle est « publique »). Ce n'est pas grave si un pirate l'intercepte. En revanche, la clé privée — qui permet donc de déchiffrer — doit rester secrète.



L'algorithme de chiffrement asymétrique le plus connu s'appelle RSA. Si vous voulez savoir comment RSA fonctionne et pourquoi il faut une clé différente pour chiffrer et pour déchiffrer, il existe sur le Site d'OpenClassrooms [un tutoriel sur RSA](#). Je vous préviens : il faut aimer les maths.

La création d'un tunnel sécurisé avec SSH

SSH combine chiffrement asymétrique et chiffrement symétrique

SSH utilise les deux chiffrements : asymétrique et symétrique. Cela fonctionne dans cet ordre.

1. On utilise d'abord le chiffrement asymétrique pour s'échanger discrètement une clé secrète de chiffrement symétrique.
2. Ensuite, on utilise tout le temps la clé de chiffrement symétrique pour chiffrer les échanges.



Pourquoi ne pas utiliser uniquement du chiffrement asymétrique tout le temps ?

Ce serait possible mais il y a un défaut : le chiffrement asymétrique demande beaucoup trop de ressources au processeur. Le chiffrement asymétrique est 100 à 1 000 fois plus lent que le chiffrement symétrique !

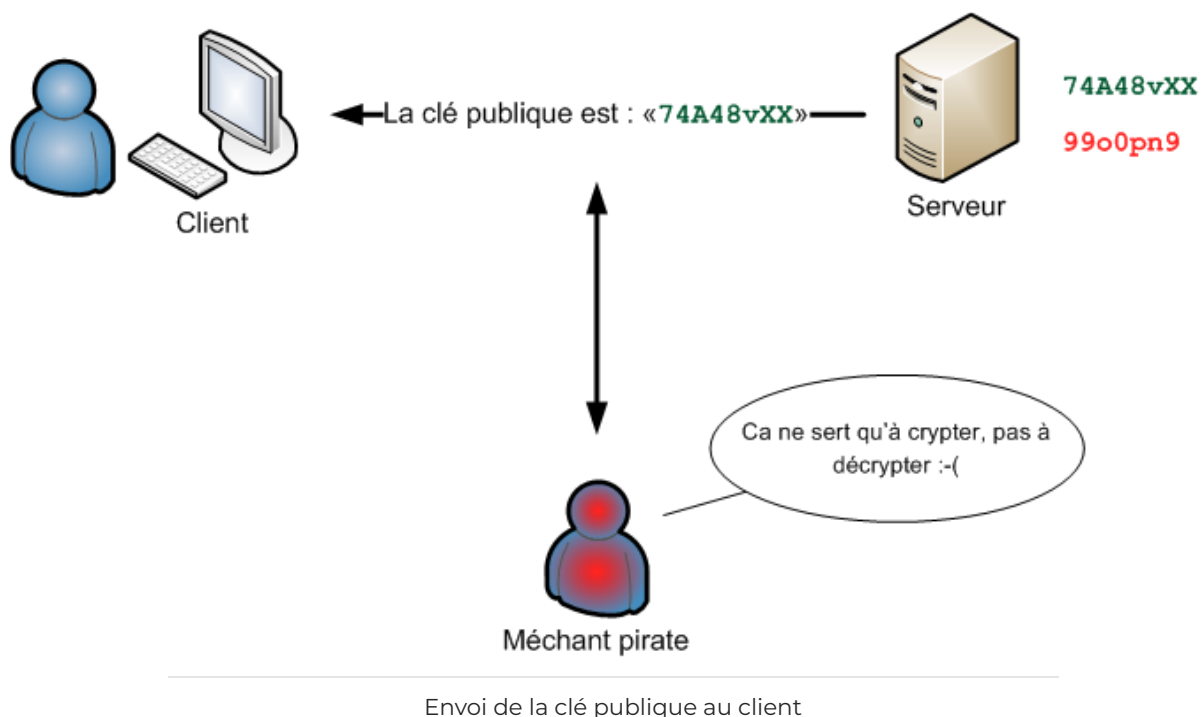
Les ordinateurs s'échangent donc la clé de chiffrement symétrique de manière sécurisée (grâce au chiffrement asymétrique) et peuvent ensuite communiquer plus rapidement en utilisant en permanence le chiffrement symétrique.

Le chiffrement asymétrique est donc utilisé seulement au début de la communication, afin que les ordinateurs s'échangent la clé de chiffrement symétrique de manière sécurisée. Ensuite, ils ne communiquent que par chiffrement symétrique.

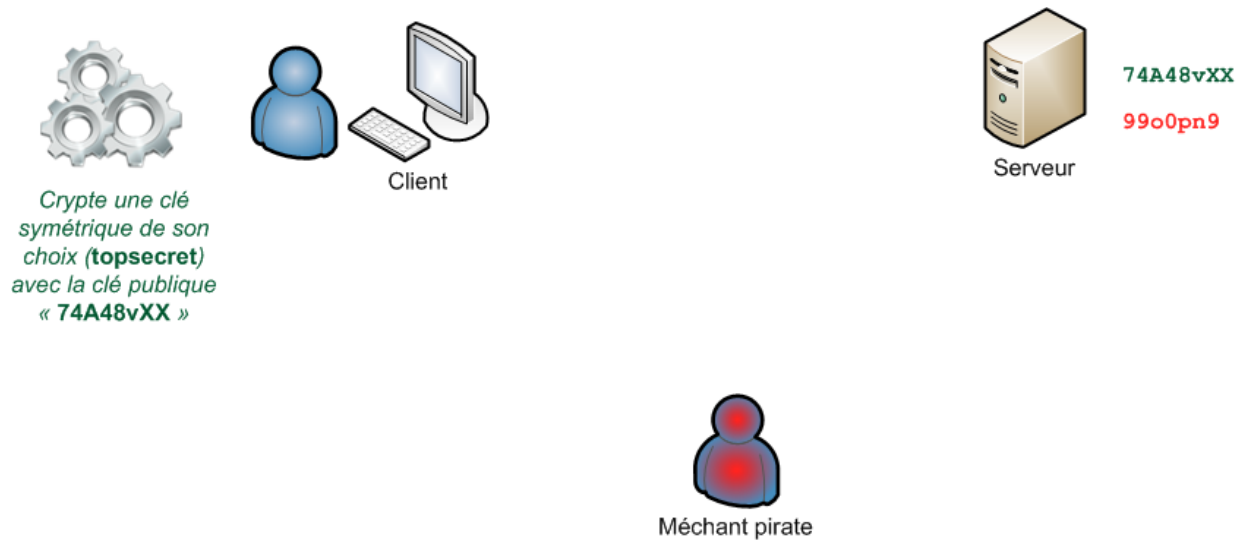
Les étapes de la création d'un canal sécurisé avec SSH en images

Je résume en images. On veut s'échanger une clé de chiffrement symétrique, mais on ne peut pas le faire en clair sinon le pirate peut l'intercepter. On va donc chiffrer la clé grâce au chiffrement asymétrique.

Le serveur envoie la clé publique en clair au client pour qu'il puisse chiffrer (figure suivante).

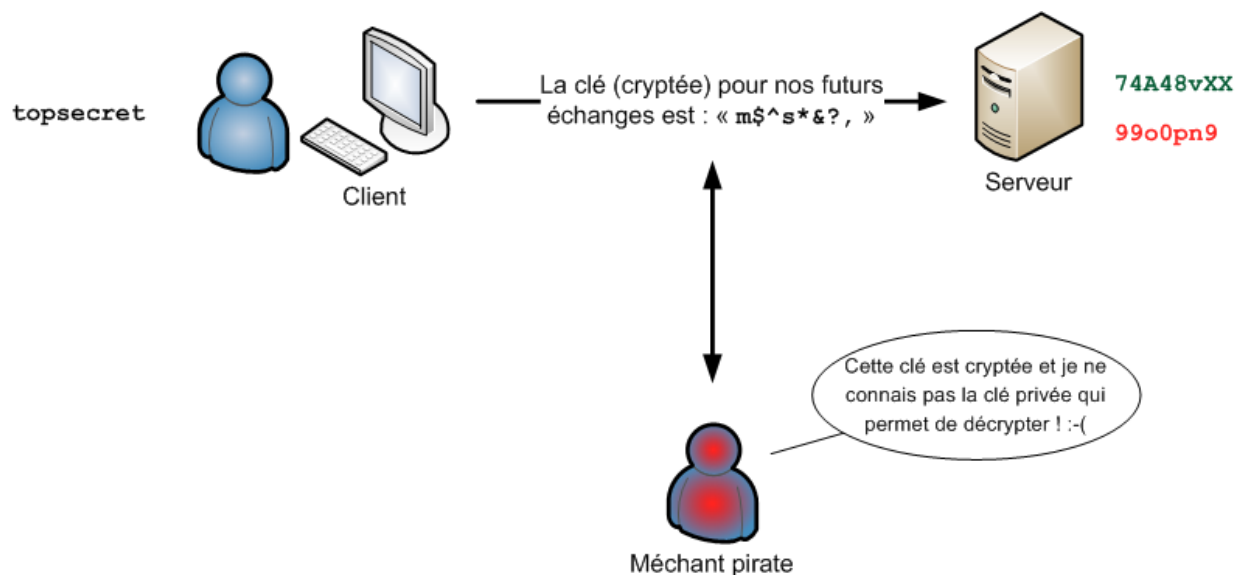


Le client génère une clé de chiffrement symétrique (par exemple `topsecret`) qu'il chiffre grâce à la clé publique qu'il a reçue (figure suivante).



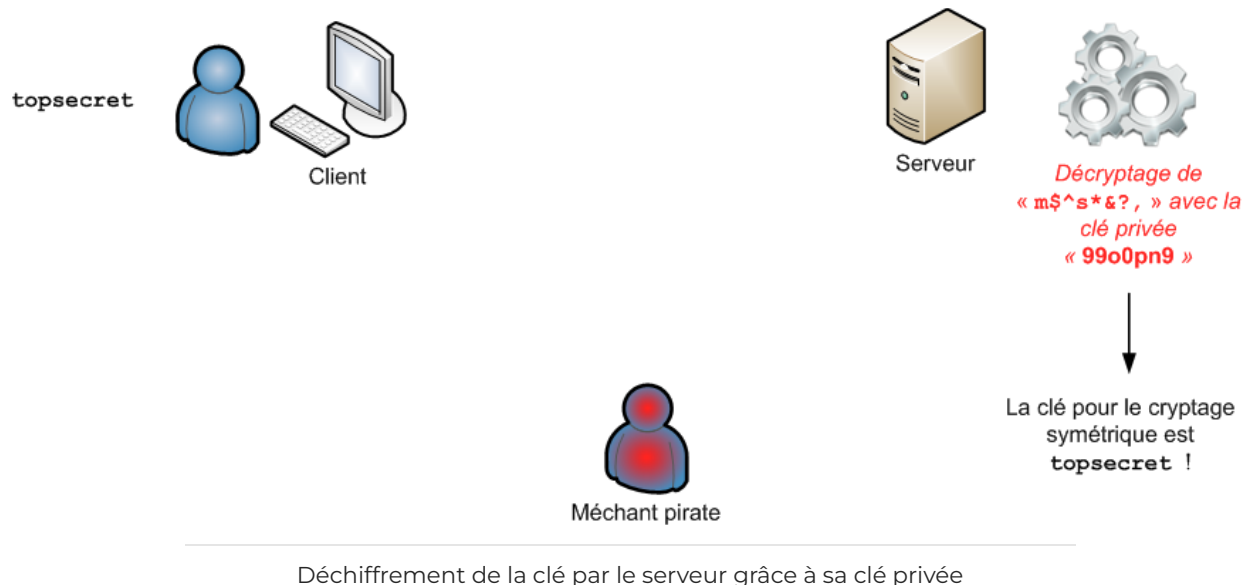
Création de clé symétrique et chiffrement par le client

Le client envoie la clé symétrique chiffrée au serveur. Le pirate peut l'intercepter, mais ne peut pas la déchiffrer car il faut pour cela la clé privée, connue seulement du serveur (figure suivante).



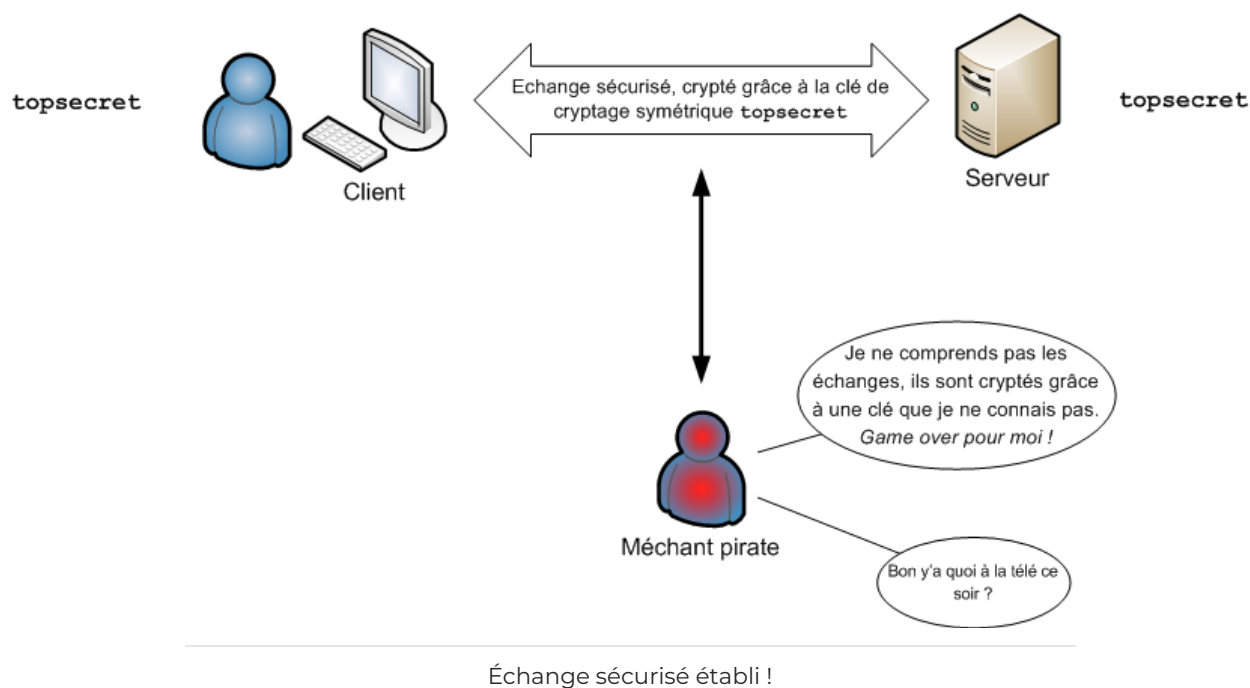
Envoi de la clé chiffrée au serveur

Le serveur déchiffre la clé reçue grâce à sa clé privée qu'il a gardée bien au chaud chez lui (figure suivante).



Le client et le serveur connaissent maintenant tous les deux la clé symétrique **topsecret**, et à aucun moment ils ne l'ont échangée en clair sur le réseau !

Ils peuvent donc s'envoyer des messages chiffrés de manière symétrique en toute tranquillité. Ce chiffrement est plus rapide et tout aussi sûr que le chiffrement asymétrique car le pirate ne connaît pas la clé (figure suivante) !

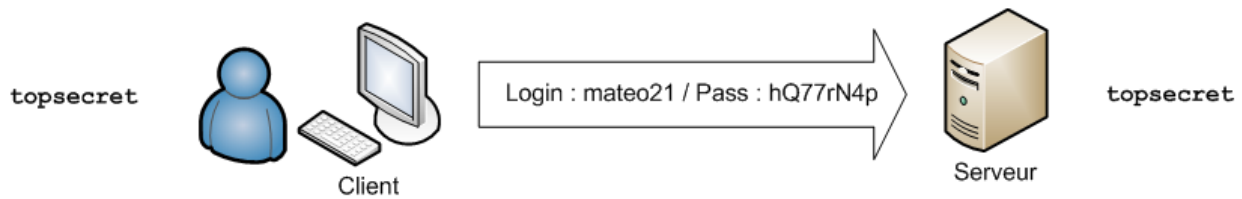


Voilà comment SSH fonctionne pour créer un canal d'échange sécurisé. Tout est chiffré grâce à la clé symétrique que le client et le serveur se sont **astucieusement** communiquée.



Maintenant qu'ils discutent de manière sécurisée, que font le client et le serveur ?

Eh bien **seulement** maintenant, le client peut se connecter au serveur : il peut donner son login et son mot de passe sans craindre de se les faire voler par le pirate (figure suivante) ! ;-)



Le client peut envoyer son login et son mot de passe de manière sécurisée



Faut-il savoir tout cela pour utiliser SSH ?

Non. En fait, tout se fait automatiquement. Vous allez juste avoir à entrer un login et un mot de passe pour vous connecter à votre machine à distance.

Mais j'estime que c'était l'occasion idéale de vous expliquer comment fonctionne le protocole SSH. Ce système est utilisé partout dans le monde ! Plus personne n'envisage de se connecter en Telnet aujourd'hui.

Se connecter avec SSH et PuTTY



Assez de théorie, passons à la pratique ! Vous allez voir, ça sera beaucoup plus simple car les ordinateurs effectuent les chiffrements entre eux sans nous demander d'intervenir... et c'est tant mieux. ;-)

À partir de maintenant, de deux choses l'une.

- Soit **vous louez déjà un serveur dédié** (ce qui devrait être le cas d'une minorité d'entre vous). Celui-ci est déjà configuré comme serveur SSH, vous n'avez donc rien à faire pour le « transformer » en serveur.



Si vous voulez louer un serveur dédié, sachez qu'il existe de très nombreux hébergeurs qui en proposent, comme par exemple *OVH*. Comme vous pourrez le constater, ça coûte cher (en même temps, c'est un ordinateur à part entière que vous louez !). Sachez qu'il existe aussi des serveurs *low cost*, moins chers (moins puissants mais ils peuvent suffire), comme *Kimsufi* et *Dedibox*. Je vous recommande d'attendre un peu avant de louer un serveur dédié : cela représente un gros investissement et il vaut mieux être sûr d'en avoir vraiment besoin.

- Soit **vous n'avez pas de serveur dédié**, ce qui, je suppose, est le cas de la plupart d'entre vous. Dans ce cas, nous allons voir tout de suite comment transformer votre PC en serveur.

Transformer sa machine en serveur

Cette étape vous concerne si vous voulez transformer votre PC en serveur. Par exemple, si vous voulez accéder à votre PC depuis un autre lieu (et donc suivre le reste de ce chapitre), vous devez le transformer en serveur au préalable.

Il faut tout simplement installer le paquet `openssh-server` :

```
sudo apt-get install openssh-server
```

Lors de l'installation, vous devriez voir certaines étapes intéressantes s'effectuer automatiquement :

```
Creating SSH2 RSA key; this may take some time ...  
Creating SSH2 DSA key; this may take some time ...  
* Restarting OpenBSD Secure Shell server sshd [ OK ]
```

RSA et DSA sont deux algorithmes de chiffrement asymétrique. Comme je vous l'ai dit plus tôt, SSH peut travailler avec plusieurs algorithmes de chiffrement différents.

Ce que vous voyez là est l'étape de création d'une clé publique et d'une clé privée pour chacun des deux algorithmes (RSA et DSA).

Ensuite, le programme de serveur SSH (appelé `sshd`) est lancé.

Normalement, le serveur SSH sera lancé à chaque démarrage. Si ce n'est pas le cas, vous pouvez le lancer à tout moment avec la commande suivante :

```
sudo /etc/init.d/ssh start
```

Et vous pouvez l'arrêter avec cette commande :

```
sudo /etc/init.d/ssh stop
```

Logiquement, vous ne devriez pas avoir besoin de configurer quoi que ce soit, mais sachez au besoin que le fichier de configuration se trouve dans `/etc/ssh/ssh_config` . Il faudra recharger SSH avec la commande

```
sudo /etc/init.d/ssh reload
```

 pour que les changements soient pris en compte.

Voilà : votre machine est désormais un serveur SSH ! Vous pouvez vous y connecter depuis n'importe quel ordinateur sous Linux ou sous Windows dans le monde (pour peu que vous ne soyez pas derrière un pare-feu).

Nous commencerons dans un premier temps par voir comment accéder à votre PC à distance depuis une machine Linux.

Se connecter via SSH à partir d'une machine Linux

Toutes les machines équipées de Linux proposent la commande `ssh` qui permet de se connecter à distance à une autre machine.



À partir d'ici, je suppose que vous avez installé `openssh-server` et que votre machine est allumée. L'idéal serait d'aller chez un ami qui a Linux (ou d'utiliser de chez vous un autre PC équipé de Linux).

Ouvrez une console sur le second PC et utilisez la commande `ssh` comme ceci :

```
ssh login@ip
```

Il faut remplacer `login` par votre login (mateo21, dans mon cas) et `ip` par l'adresse IP de votre ordinateur.



Si vous vous connectez depuis chez un ami, il vous faut entrer l'IP internet de votre PC que vous pouvez

obtenir en allant sur www.whatismyip.com par exemple.

Si vous vous connectez depuis un autre PC chez vous (sur le même réseau local), il vous faut entrer l'IP locale que vous devriez voir en tapant la commande `ifconfig` (par exemple 192.168.0.3).

Si **vraiment** vous n'avez ni ami sous Linux ni second PC dans la maison, vous pouvez simuler une connexion réseau en vous connectant depuis votre PC à votre PC. Utilisez pour cela l'IP 127.0.0.1 (ou le mot `localhost`), ça marche toujours.

Si je suis chez un ami et que l'IP internet de mon ordinateur est 87.112.13.165, je vais taper :

```
ssh mateo21@87.112.13.165
```

Si, faute de mieux, vous voulez tester en vous connectant chez vous depuis chez vous, vous pouvez taper :

```
ssh mateo21@localhost
```

Cette seconde méthode marche toujours, mais c'est moins impressionnant parce que vous ne faites que simuler une connexion réseau.

Normalement, le serveur devrait répondre au bout d'un moment et vous devriez voir quelque chose comme ce qui suit :

```
The authenticity of host 'localhost (127.0.0.1)' can't be established.  
RSA key fingerprint is 49:d9:2d:2a:df:fd:80:ab:e9:eb:59:37:58:34:de:f7.  
Are you sure you want to continue connecting (yes/no)?
```



Si vous n'avez pas de réponse du serveur, vérifiez que vous ne vous êtes pas trompés d'IP. Vérifiez aussi que le port 22 n'est pas bloqué par un pare-feu, car c'est celui utilisé par SSH par défaut.

Si le serveur tourne sur un autre port, il faudra préciser le numéro de ce port comme ceci :

```
ssh mateo21@87.112.13.165 -p 12451
```

(si le serveur fonctionne sur le port 12451 au lieu du port 22).

Que se passe-t-il ? On vous dit que le *fingerprint* (empreinte) du serveur est

`49:d9:2d:2a:df:fd:80:ab:e9:eb:59:37:58:34:de:f7`. C'est un numéro unique qui vous permet d'identifier le serveur. Si demain quelqu'un essaie de se faire passer pour le serveur, le *fingerprint* changera forcément et vous saurez qu'il se passe alors quelque chose d'anormal. Ne vous inquiétez pas, SSH vous avertira de manière très claire si cela arrive.

En attendant, tapez « yes » pour confirmer que c'est bien le serveur auquel vous voulez vous connecter. Le serveur et le client vont alors s'échanger une clé de chiffrement, comme je vous l'ai expliqué un peu plus tôt. Normalement, le serveur devrait vous demander au bout de quelques secondes votre mot de passe :

```
mateo21@localhost's password:
```

Vous pouvez l'entrer en toute sécurité, la communication est chiffrée. ;-)

Si vous entrez le bon mot de passe, la console du PC de votre ami (ou votre propre console) devrait vous afficher un message de bienvenue puis un **prompt** qui correspond à la console de votre PC. Bravo, vous êtes connectés !

```
mateo21@mateo21-desktop:~$
```

Si aucune erreur ne s'affiche, c'est que vous êtes bien connectés et que vous travaillez désormais à distance sur votre machine ! Vous pouvez effectuer toutes les opérations que vous voulez comme si vous étiez chez vous. Essayez de parcourir les dossiers pour voir que ce sont bien les vôtres, et amusez-vous (pourquoi pas) à créer un fichier (avec Nano). Lorsque vous reviendrez sur votre PC, vous l'y retrouverez.

Vous pouvez aussi commander l'exécution d'un programme, d'une recherche, etc. Vous savez déjà comment lancer un programme en tâche de fond pour qu'il continue de s'exécuter même quand vous n'êtes pas connectés à la machine.

Vous vous souvenez de `nohup` et de `screen` ? 🤔

Pour vous déconnecter, tapez `logout` ou son équivalent : la combinaison de touches `Ctrl + D` .

Se connecter via SSH à partir d'une machine Windows

Si vous voulez avoir accès à la console de votre machine Linux mais que vous n'avez pas d'autre machine Linux sous la main, pas de panique ! Il existe des programmes pour Windows faits pour cela. Le plus connu d'entre eux – celui que j'utilise personnellement – s'appelle **PuTTY**.

Vous pouvez [télécharger PuTTY depuis son site officiel](#).



Il est possible de se connecter en SSH via Windows 10 sans l'installation de PuTTY.

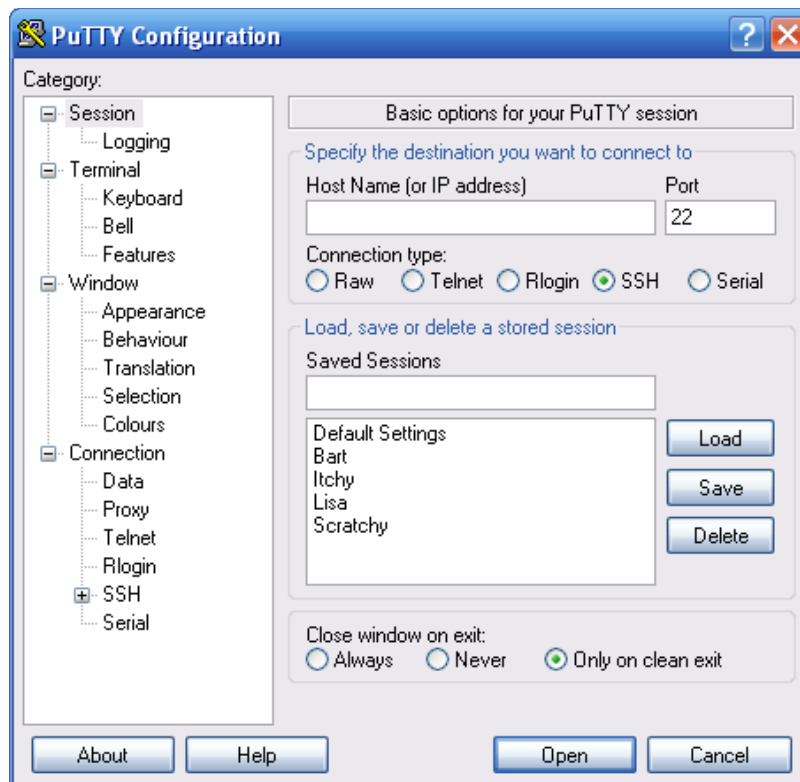
Je sais : vous devez vous dire que ce n'est pas très clair et que vous ne voulez pas chercher sur quel lien cliquer sur cette page.

Repérez la section « Binaries ». C'est un tableau. Vous avez le choix entre :

- cliquer sur `putty.exe` pour télécharger le programme principal. Il ne nécessite pas d'installation ;
- cliquer sur le programme d'installation (par exemple `putty-0.60-installer.exe`). Celui-ci installera PuTTY et d'autres utilitaires dont vous aurez besoin dans quelques minutes.

`putty.exe` suffit, mais je vous recommande de prendre le package complet en récupérant le programme d'installation.

Une fois que c'est fait et installé, lancez PuTTY. Une fenêtre comme celle de la figure suivante devrait s'afficher.

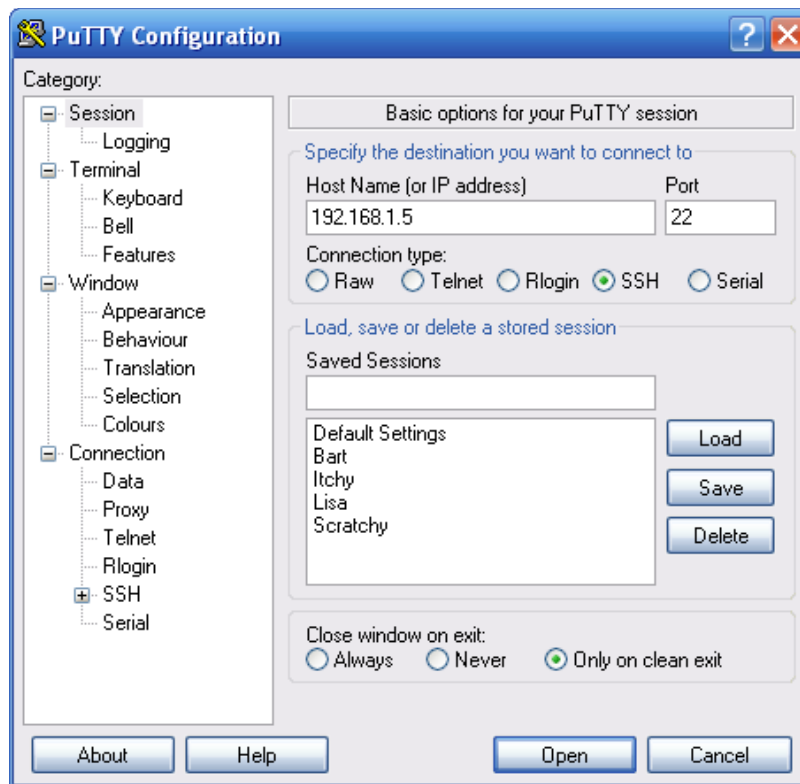


Il y a beaucoup de pages d'options, comme vous pouvez le voir au niveau de la section « Category » sur le côté. Pour le moment, pas de panique : vous avez juste besoin de remplir le champ en haut « Host Name (or IP address) ». Entrez-y l'adresse IP de votre ordinateur sous Linux.



J'ai donné quelques explications à propos de l'adresse IP un peu plus haut, lorsque j'ai parlé de la connexion SSH depuis Linux. Lisez donc les paragraphes précédents si vous voulez plus d'informations à ce sujet.

Dans mon cas, je vais entrer l'adresse IP de mon PC sous Linux situé sur mon réseau local (192.168.1.5 – figure suivante).



Vous pouvez changer le numéro du port si ce n'est pas 22, mais normalement c'est 22 par défaut.

Ensuite, vous n'avez plus qu'à cliquer sur le bouton « Open » tout en bas pour lancer la connexion. Rien d'autre !



Si vous voulez sauvegarder l'IP et les paramètres pour ne pas devoir les retaper à chaque fois, donnez un nom à cette connexion (par exemple, le nom de votre ordinateur) dans le champ sous « Saved Sessions », puis appuyez sur le bouton « Save ». La prochaine fois, vous n'aurez qu'à double-cliquer sur le nom de votre PC dans la liste pour vous y connecter directement.

La première fois que vous vous connectez à votre serveur, PuTTY devrait vous demander une confirmation comme sur la figure suivante.



C'est la même chose que sous Linux : on vous donne l'empreinte (*fingerprint*) de votre serveur. Vous devez confirmer que c'est bien chez lui que vous voulez vous connecter. Cliquez sur « Oui » pour confirmer. À l'avenir, on ne vous reposera plus la question. Par contre, si le *fingerprint* change, un gros message

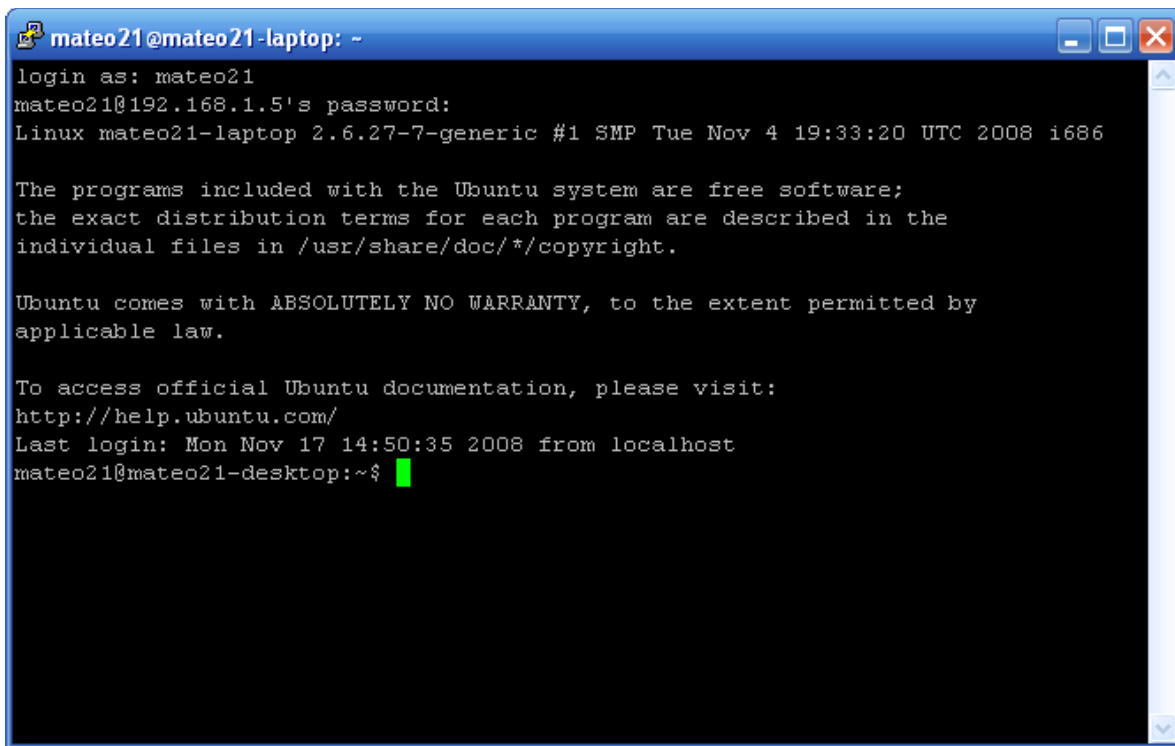
d'avertissement s'affichera. Cela signifiera soit que le serveur a été réinstallé, soit que quelqu'un est en train de se faire passer pour le serveur (c'est ce que l'on appelle une attaque *man-in-the-middle*). Cela ne devrait fort heureusement pas vous arriver, du moins je l'espère. :-)

Le serveur vous demande alors le login et le mot de passe (figure suivante).



Rappelez-vous qu'il est normal que les caractères ne s'affichent pas quand vous tapez votre mot de passe. Il n'y a même pas d'étoiles pour des raisons de sécurité, afin que quelqu'un ne soit pas tenté de compter le nombre de caractères en regardant derrière votre épaule !

Si tout est bon, vous devriez être connectés à votre machine (figure suivante) !

A terminal window titled 'mateo21@mateo21-laptop: ~' with standard window controls. The terminal shows the login process for user 'mateo21' on an Ubuntu system. It displays the system version 'Linux mateo21-laptop 2.6.27-7-generic #1 SMP Tue Nov 4 19:33:20 UTC 2008 i686', a copyright notice for Ubuntu, a warranty disclaimer, and the official Ubuntu documentation URL. The prompt changes from 'login as:' to 'mateo21@mateo21-laptop:' and then to 'mateo21@mateo21-desktop:~\$' after a successful login. A green cursor is visible at the end of the last prompt.

```
mateo21@mateo21-laptop: ~
login as: mateo21
mateo21@192.168.1.5's password:
Linux mateo21-laptop 2.6.27-7-generic #1 SMP Tue Nov 4 19:33:20 UTC 2008 i686

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/
Last login: Mon Nov 17 14:50:35 2008 from localhost
mateo21@mateo21-desktop:~$
```

Et voilà, vous êtes chez vous ! Vous pouvez faire ce qui vous chante : lire vos fichiers, écrire des fichiers, lancer une recherche, exécuter un programme... bref, vous êtes chez vous.

Pour vous déconnecter, tapez `logout` ou son équivalent, la combinaison de touches `Ctrl + D` .

L'identification automatique par clé



Il y a plusieurs façons de s'authentifier sur le serveur, pour qu'il sache que c'est bien vous. Les deux plus utilisées sont :

- l'authentification par mot de passe ;
- l'authentification par clés publique et privée du **client**.

Pour le moment, nous avons vu uniquement l'authentification par mot de passe (le serveur vous demandait votre mot de passe).

Il est possible d'éviter que l'on vous le demande à chaque fois grâce à une authentification spéciale par clé. Cette méthode d'authentification est plus complexe à mettre en place, mais elle est ensuite plus pratique.



Avec cette nouvelle méthode d'authentification, c'est le client qui va générer une clé publique et une clé privée. Les rôles sont un peu inversés.

L'avantage, c'est que l'on ne vous demandera pas votre mot de passe à chaque fois pour vous connecter. Si vous vous connectez très régulièrement à un serveur, c'est vraiment utile. Si vous faites bien les choses, cette méthode est tout aussi sûre que l'authentification par mot de passe.

Je vais, là encore, distinguer les deux cas :

- vous essayez de vous connecter depuis une machine Linux ;
- vous essayez de vous connecter depuis une machine Windows (avec PuTTY).

Tout le monde ne met pas une phrase de passe. En fait, ça dépend du risque que quelqu'un d'autre utilise la machine du client et puisse lire le fichier contenant la très secrète clé privée. Si le PC du client est votre PC chez vous et que personne d'autre ne l'utilise, il y a assez peu de risques (à moins d'avoir un virus, un spyware...). Si c'est en revanche un PC public, je vous recommande vivement de mettre une **passphrase** pour chiffrer la clé qui sera enregistrée.

Si vous hésitez entre les deux méthodes, je vous recommande de rentrer une **passphrase** : c'est quand même la méthode la plus sûre.

Envoyer la clé publique au serveur

Il faut maintenant envoyer au serveur votre clé publique pour qu'il puisse vous chiffrer des messages.

Votre clé **publique** devrait se trouver dans `~/.ssh/id_rsa.pub` (*pub* comme *public*). correspond à votre home (`/home/mateo21/` dans mon cas). Notez que `.ssh` est un dossier caché.

Votre clé **privée**, elle, se trouve dans `~/.ssh/id_rsa`. Ne la communiquez à personne ! Elle est normalement chiffrée si vous avez entré une **passphrase**, ce qui constitue une sécurité de plus.

Vous pouvez déjà vous rendre dans le dossier `.ssh`, pour commencer :

```
cd ~/.ssh
```

Si vous faites un `ls`, vous devriez voir ceci :

```
$ ls
id_rsa id_rsa.pub known_hosts
```

Les trois fichiers sont :

- `id_rsa` : votre clé privée, qui doit rester secrète. Elle est chiffrée si vous avez rentré une **passphrase** ;
- `id_rsa.pub` : la clé publique que vous pouvez communiquer à qui vous voulez, et que vous devez envoyer au serveur ;
- `known_hosts` : c'est la liste des **fingerprint** que votre PC de client tient à jour. Ça lui permet de se souvenir de l'identité des serveurs et de vous avertir si, un jour, votre serveur est remplacé par un autre (qui pourrait être celui d'un pirate !). Je vous en ai déjà parlé un peu plus tôt.

L'opération consiste à envoyer la clé publique (`id_rsa.pub`) au serveur et à l'ajouter à son fichier **authorized_keys** (clés autorisées). Le serveur y garde une liste des clés qu'il autorise à se connecter.

Le plus simple pour cela est d'utiliser la commande spéciale **ssh-copy-id**. Utilisez-la comme ceci :

```
ssh-copy-id -i id_rsa.pub login@ip
```

Remplacez-y votre **login** et l' **ip** de votre serveur.

```
$ ssh-copy-id -i id_rsa.pub mateo21@88.92.107.7
mateo21@88.92.107.7's password:
```

```
Now try logging into the machine, with "ssh 'mateo21@localhost'", and check in:
```

```
.ssh/authorized_keys
```

```
to make sure we haven't added extra keys that you weren't expecting.
```

Si vous devez vous connecter au serveur par un autre port que celui par défaut, basez-vous sur la commande suivante : `ssh-copy-id -i id_rsa.pub "-p 14521 mateo21@88.92.107.7"` .

On vous demande votre mot de passe (celui de votre compte, pas la `passphrase`). En fait, vous vous connectez par mot de passe encore une fois, pour pouvoir ajouter votre clé publique sur le serveur.

La clé est ensuite automatiquement ajoutée à `~/.ssh/authorized_keys` sur le serveur. On vous invite à vérifier si l'opération s'est bien déroulée en ouvrant le fichier `authorized_keys` , ce que vous pourrez faire plus tard si vous le voulez.

Se connecter !

Maintenant, connectez-vous au serveur comme vous le faisiez auparavant :

```
ssh login@ip
```

Par exemple :

```
$ ssh mateo21@88.92.107.7
Enter passphrase for key '/home/mateo21/.ssh/id_rsa':
```

On vous demande la phrase de passe pour déchiffrer votre clé privée. Entrez-la.

Normalement, si tout va bien, vous devriez être alors connectés au serveur.



Où je suis le dernier des nuls, ou alors c'est ce système qui est nul. Auparavant, on me demandait mon mot de passe. Maintenant, on me demande une phrase de passe pour déchiffrer la clé privée. Où est le progrès ???

Je comprends votre frustration. ;-)

En fait, si vous n'aviez pas mis de phrase de passe, on ne vous aurait rien demandé et vous auriez été directement connectés. Heureusement, il y a une solution pour ceux qui ont choisi la sécurité en utilisant une phrase de passe, mais qui ne veulent quand même pas avoir à l'entrer à chaque fois : l'agent SSH.

L'agent SSH

L'agent SSH est un programme qui tourne en arrière-plan en mémoire. Il retient les clés privées pendant toute la durée de votre session.

Tout ce que vous avez à faire est de lancer le programme `ssh-add` sur le PC du client :

```
$ ssh-add
Enter passphrase for /home/mateo21/.ssh/id_rsa:
```

```
Identity added: /home/mateo21/.ssh/id_rsa (/home/mateo21/.ssh/id_rsa)
```

Il va automatiquement chercher votre clé privée. Pour la déchiffrer, il vous demande la `passphrase` . Entrez-la.

Maintenant que c'est fait, chaque fois que vous vous connecterez à un serveur, vous n'aurez plus besoin d'entrer la `passphrase` . Essayez de vous connecter à votre serveur pour voir !



L'intérêt de l'agent SSH est qu'il ne vous demande la `passphrase` qu'une seule fois au début. Ensuite, vous pouvez vous connecter plusieurs fois sur le même serveur, ou même sur plusieurs serveurs différents, le tout sans avoir besoin de retaper votre `passphrase` !

Authentification par clé depuis Windows (PuTTY)

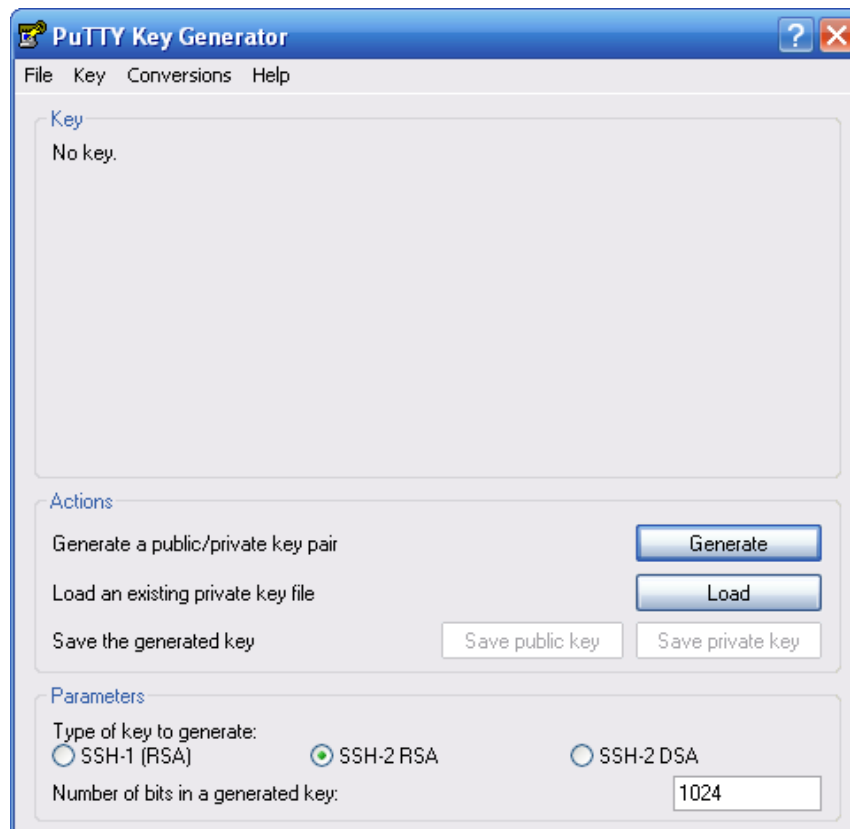
Il est tout à fait possible d'utiliser l'authentification par clé avec PuTTY. C'est là justement qu'il est recommandé d'avoir téléchargé le programme d'installation, et non pas juste le programme principal `putty.exe` .

Le principe est le même que sous Linux : il faut d'abord que l'on génère une paire de clés sur le PC du client, puis qu'on les envoie au serveur. Nous retrouverons aussi un équivalent de l'agent SSH pour éviter d'avoir à entrer une `passphrase` à chaque fois.

Commençons par la génération des clés.

Générer une paire de clés (publique et privée) avec Puttygen

Normalement, vous devriez avoir installé un programme appelé Puttygen (figure suivante – il se trouvait dans le gestionnaire d'installation de PuTTY). Lancez-le.



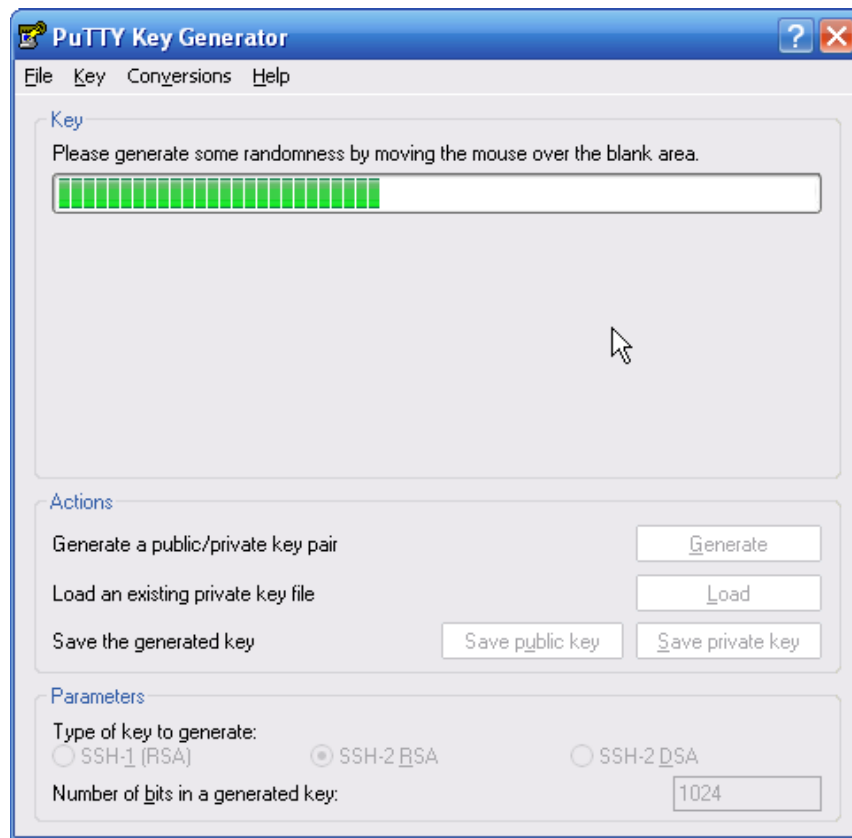
Puttygen permet de générer une paire de clés

En bas de la fenêtre, vous pouvez choisir vos paramètres : algorithme de chiffrement et puissance du chiffrement. Les valeurs par défaut (ici RSA 1024 bits) sont tout à fait convenables. Vous pouvez les changer, mais sachez qu'elles sont sûres et que vous pouvez vous en contenter.

Cliquez sur le bouton « Generate ». Le programme va générer une paire de clés (publique et privée).



Pour l'aider à générer cette paire, le programme vous propose quelque chose d'assez amusant : vous devez bouger la souris dans la fenêtre (figure suivante)



Génération des clés grâce aux mouvements de la souris

Une fois que c'est fait, on vous affiche la clé publique (figure suivante) :



Enregistrement des clés

Comme vous le voyez, cela ne me dérange pas que tout le monde voie ma clé publique. Le principe, c'est justement que tout le monde peut voir cette clé, mais ne peut rien en faire. Par contre, la clé privée doit rester secrète.

Vous pouvez choisir d'entrer une **passphrase** ou non. Comme je vous l'ai expliqué plus tôt, cela renforce la sécurité en chiffrant la clé privée.

Saisissez la **passphrase** dans les champs « Key passphrase » et « Confirm passphrase ».

Ensuite, enregistrez la clé publique dans un fichier en cliquant sur « Save public key ». Vous pouvez nommer ce fichier comme vous voulez, par exemple **cle.pub** . Enregistrez-le où vous voulez.

Puis enregistrez la clé privée en cliquant sur « Save private key ». Donnez-lui l'extension **.ppk** : **cle.ppk** par exemple.

Ne fermez pas encore Puttygen.

Envoyer la clé publique au serveur

Comme sous Linux tout à l'heure, il faut envoyer la clé publique au serveur pour qu'il nous autorise à nous connecter par clé. Le problème, c'est qu'il n'y a pas de commande pour le faire automatiquement depuis Windows. Il va falloir ajouter la clé à la main dans le fichier **authorized_keys** . Heureusement, ce n'est pas très compliqué.

Ouvrez PuTTY et connectez-vous au serveur comme auparavant (en entrant votre mot de passe habituel).

Rendez-vous dans **~/ .ssh** :

```
cd ~/.ssh
```

Si le dossier `.ssh` n'existe pas, pas de panique, créez-le :

```
mkdir .ssh
```

Rajoutez votre clé publique à la fin du fichier `authorized_keys` (s'il n'existe pas, il sera créé). Vous pouvez utiliser la commande suivante :

```
echo "votre_cle" >> authorized_keys
```



Rappel : votre clé publique est affichée dans Puttygen, que vous ne devriez pas avoir fermé. Pour coller la clé dans la console, utilisez la combinaison de touches `Shift + Insert` plutôt que `Ctrl + V` .

Par exemple :

```
echo "ssh-rsa AAAAB3NzaC1yc2E [...] AAAABJQAP++UWB0kLp0= rsa-key-20081117" >> authorized_keys
```

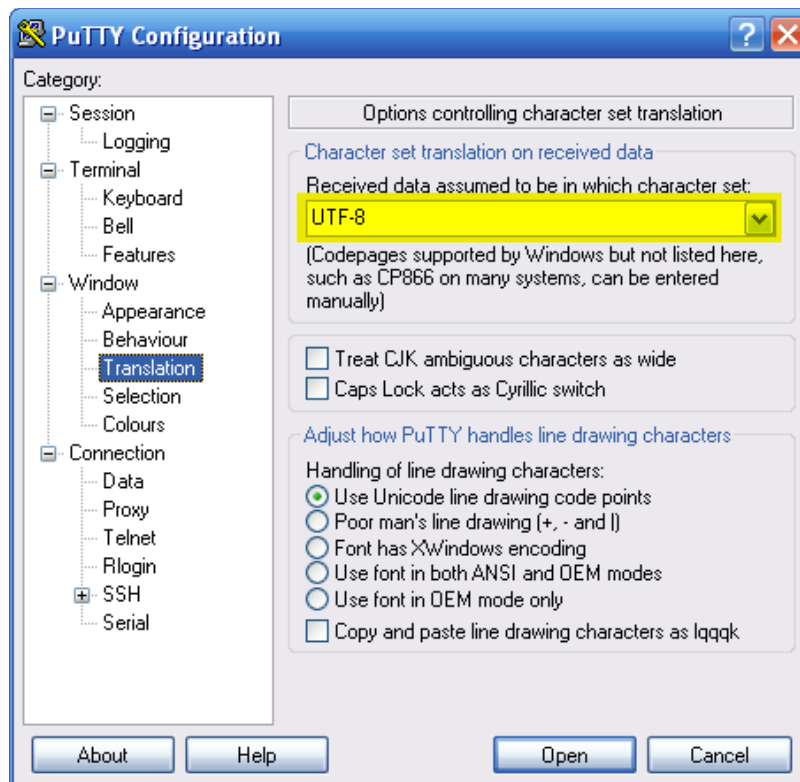
Voilà, c'est fait. ;-)

Déconnectez-vous, et relancez PuTTY. On va maintenant le configurer pour qu'il se connecte à l'aide de la clé.

Configurer PuTTY pour qu'il se connecte avec la clé

Une fois PuTTY ouvert, rendez-vous dans la section `Window → Translation` pour commencer. Ça n'a pas de rapport direct avec les clés, mais cela vous permettra de régler le problème des accents qui s'affichent mal dans la console si vous l'avez rencontré.

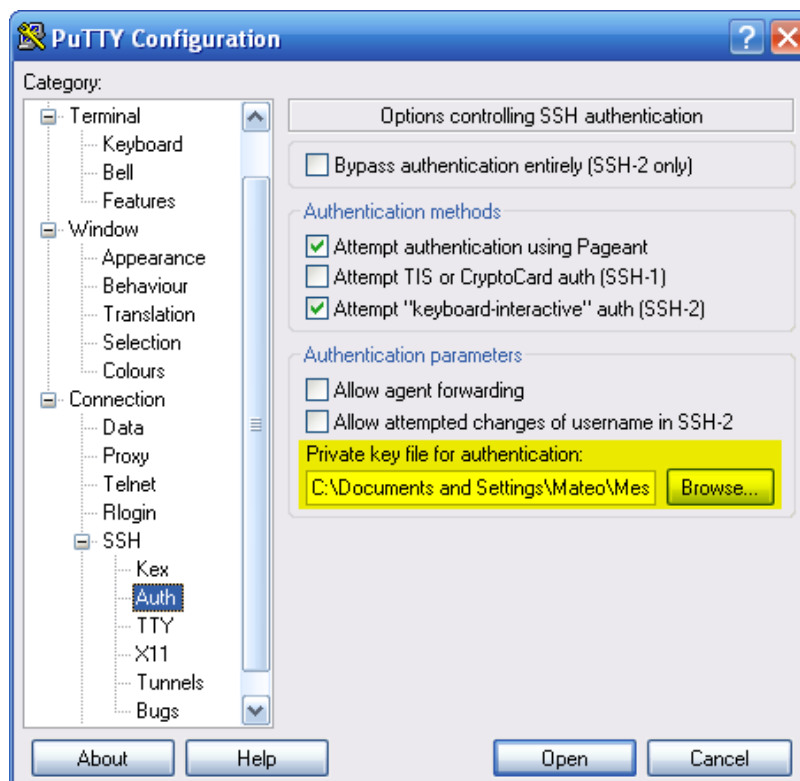
Réglez la valeur de la liste déroulante à UTF-8, comme sur figure suivante.



Encodage de l'invite de commandes

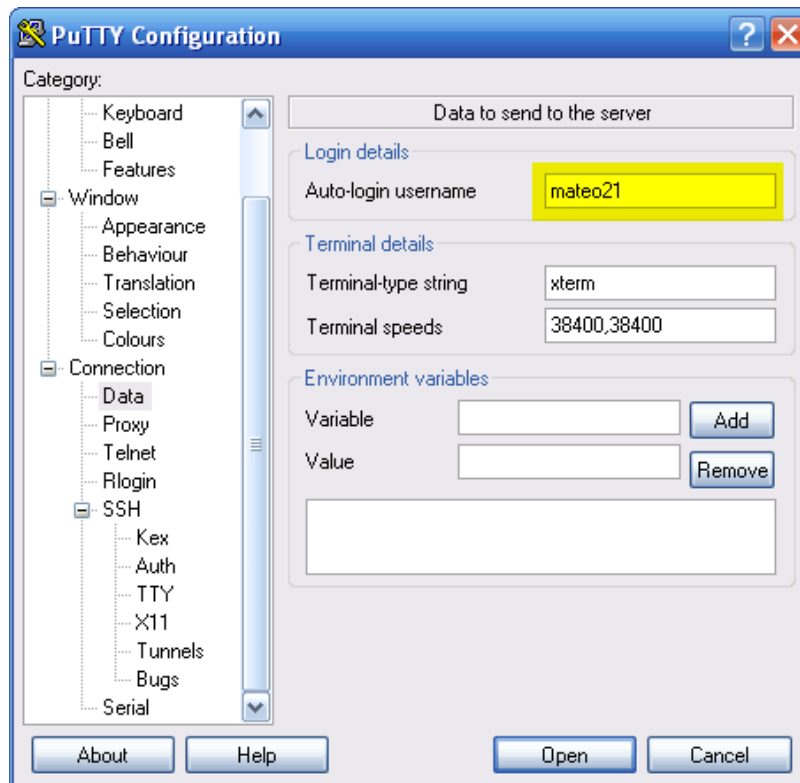
La plupart des serveurs encodent désormais les caractères en UTF-8, cela devrait donc vous éviter des soucis d'affichage.

Maintenant, rendez-vous dans **Connection → SSH → Auth**. Cliquez sur le petit bouton « Browse » pour sélectionner votre clé privée (figure suivante).



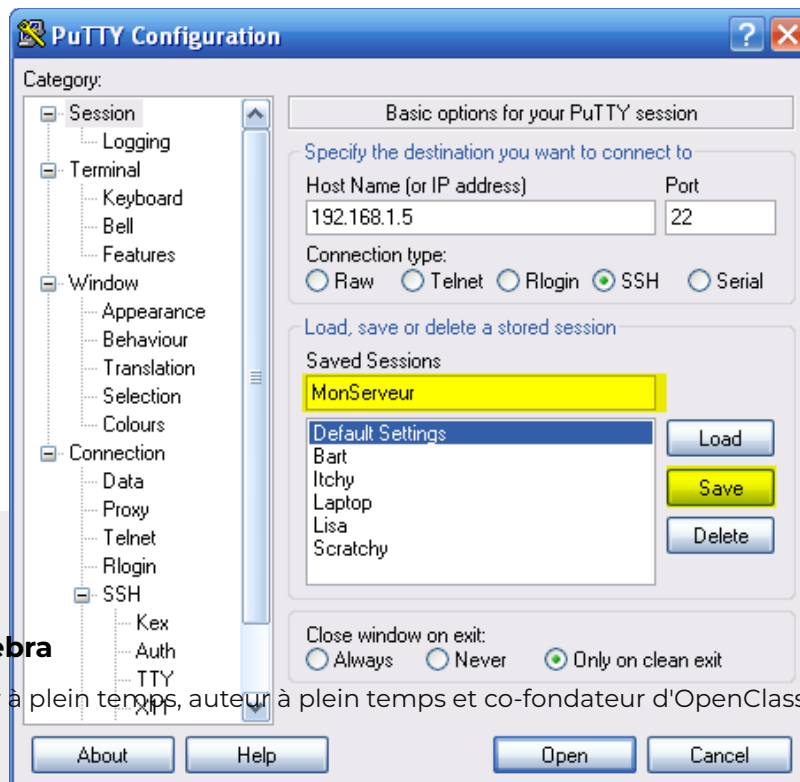
Sélection de la clé privée

Je vous recommande aussi d'aller dans **Connection → Data** et d'entrer votre login dans « Auto-login username », comme la figure suivante vous le montre.



Choix d'un login par défaut

Retournez à l'accueil en cliquant sur la section « Session » tout en haut (figure suivante). Entrez l'IP du serveur. Ensuite, je vous recommande fortement d'enregistrer ces paramètres.



Le professeur



Mathieu Nebra

Entrepreneur à plein temps, auteur à plein temps et co-fondateur d'OpenClassrooms :o)

Découvrez aussi ce cours en...

L'enregistrement des paramètres dans PuTTY

Double-cliquez sur votre serveur (par exemple **MonServeur**) sous « Saved Sessions ». Cliquez ensuite sur « Save ».

À l'avenir, vous n'aurez qu'à double-cliquer sur le nom de votre serveur dans la liste pour vous y connecter directement avec les bons paramètres.

Cliquez sur « Open » pour vous connecter au serveur.

Vous devriez voir PuTTY utiliser automatiquement votre pseudo, puis vous demander votre **passphrase**.

Entrez-la pour vérifier que cela fonctionne, comme sur la figure suivante.

Qui sommes-nous ?

Financements

Expérience de formation

Forum

Blog [↗](#)

Presse [↗](#)

OPPORTUNITÉS

Nous rejoindre [↗](#)

Devenir mentor [↗](#)



Conditions générales d'utilisation

PuTTY demande la phrase de passe

Technique de protection des données personnelles

Euh... et si je ne veux pas avoir à entrer la **passphrase** à chaque fois ? Non, parce que c'est pareil que d'entrer un mot de passe, là...

Cookies

Accessibilité

En effet, et ma réponse sera la même que pour ceux qui se connectent depuis Linux : il faut utiliser un agent SSH. Ce programme va rester en mémoire et retenir votre clé privée. Il ne vous demandera la **passphrase** qu'une fois au début, ensuite, vous pourrez vous connecter autant de fois que vous le souhaitez à autant de serveurs que vous voulez sans avoir à entrer quoi que ce soit.



L'agent SSH installé avec PuTTY s'appelle « Pageant ». Je vous recommande de le lancer au démarrage de l'ordinateur automatiquement (il ne prend que 4 Mo en mémoire), en le plaçant dans le dossier **Démarrage** du



Lorsque vous lancez « Pageant », la petite icône d'un ordinateur avec un chapeau s'ajoute dans la barre des tâches à côté de l'horloge, comme sur la figure suivante.

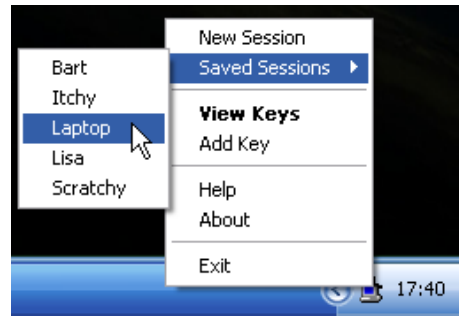


Pageant

Faites un clic droit dessus, puis cliquez sur « Add key ». On vous demande où se trouve la clé privée (**cle.ppk**). Entrez ensuite la **passphrase** .

C'est bon. Vous avez juste besoin de le faire une fois. Maintenant, vous pouvez vous connecter au serveur que

vous voulez en faisant un clic droit sur l'icône, puis en sélectionnant « Saved Sessions » (figure suivante).



Menu de Pageant

On ne vous demandera plus votre clé. :-)



Notez que si l'agent SSH « Pageant » est pratique, il vaut mieux l'arrêter si vous devez vous absenter de votre ordinateur un long moment et que quelqu'un risque de l'utiliser. Sinon, n'importe qui peut se connecter à vos serveurs sans avoir à entrer de mot de passe.

Retenez bien : l'agent SSH est un compromis entre la sécurité et le côté pratique. Il retient les clés pour vous (du moins tant que le programme tourne). Si vous êtes des utilisateurs intensifs de SSH, cela vous fera gagner beaucoup de temps.

Vous pouvez modifier le raccourci qui lance « Pageant » pour que celui-ci charge votre clé privée automatiquement dès son lancement. Faites un clic droit sur l'icône de « Pageant », allez dans « Propriétés ». Dans le champ « Cible », rajoutez à la fin en paramètre le chemin de la clé à charger. Par exemple :

```
"C:\Program Files\PuTTY\pageant.exe" c:\cle.ppk
```

La clé sera alors chargée dès que vous lancerez « Pageant ».

En résumé

- On peut se connecter à distance à un ordinateur équipé de Linux et accéder à sa console. C'est comme cela que l'on administre les serveurs sous Linux.
- Le PC qui se connecte au serveur équipé de Linux est appelé le client. On peut se connecter à une console Linux à distance depuis n'importe quel autre système d'exploitation (Windows, Mac OS ou Linux).
- Sous Windows, il faut installer le programme PuTTY pour se connecter à distance à un PC équipé de Linux.
- Sous Linux et Mac OS, on utilise la commande `ssh` à laquelle on indique son login et l'adresse IP de la machine. Par exemple : `ssh mateo21@74.141.18.33` .
- Les données qui sont échangées entre le client et le serveur sont chiffrées grâce au protocole SSH afin de garantir la confidentialité des échanges.
- Pour éviter de devoir entrer son mot de passe à chaque fois que l'on se connecte au serveur, on peut se créer une paire de clés d'identification. La clé publique ainsi générée doit être envoyée sur le serveur, la clé privée restant sur le PC du client. La connexion se fait alors sans mot de passe et reste sécurisée.

J'AI TERMINÉ CE CHAPITRE ET JE PASSE AU SUIVANT



ARCHIVER ET COMPRESSER

TRANSFÉRER DES FICHIERS

