TP1 - Mots de passes, clés, certificats

1 Stockage d'un mot de passe (1/2h)

Lorsque l'on stocke un mot de passe sur un serveur, on évite de le stocker "en clair" : on applique d'abord une fonction de hachage, et l'on ne stocke que la version hachée. Ainsi, le mot de passe n'existe en mémoire que temporairement : une fois lorsque l'on définit le mot de passe, et chaque fois que la machine vérifie le mot de passe que l'utilisateur a entré. Par exemple, voici un extrait de `/etc/shadow`, qui contient les mots de passe des comptes unix locaux (séparé par des ':')

`root:uapFHwPEN5AaA:12745:0:99999:7:::`

Lisez `man crypt`. Jouez avec la fonction `crypt()` : essayez de crypter un même mot de passe avec différents sels, DES ou MD5 (n'oubliez pas de donner `-lcrypt` à gcc, et d'inclure l'en-tête `crypt.h`).

Pourquoi utilise-t-on un sel au fait ?

Est-ce que le mot de passe root ci-dessus est difficile à deviner ?

Essayez également la commande unix `mkpasswd`, elle est plus pratique à utiliser au quotidien.

2 Introduction à SSH (40m)

2.1 Quelques rappels sur les clés

Le principe d'une *paire de clés publique/privée* est que toute donnée cryptée avec la clé publique ne peut être décryptée qu'avec la clé privé, et toute signature d'une donnée avec la clé privée peut être vérifiée avec la clé publique : la signature assure que c'est bien celui qui possède la clé privé qui a fourni la donnée et sa signature.

Le principe d'une *clé symétrique* est que l'on partage en commun une clé qui permet à la fois de chiffrer et de déchiffrer. Les algorithmes utilisés sont en général bien plus léger que ceux utilisant une paire de clés publique/privée.

2.2 Le protocole SSH

Brièvement, le protocole SSH (Secure SHell) permet à des utilisateurs d'accéder à une machine distante à travers une communication chiffrée (appelée tunnel). L'établissement d'une liaison passe par deux étapes. Le client établit d'abord un tunnel sécurisé avec le serveur en utilisant un couple de clés privée/publique : le serveur possède une clé privée (qui ne change que lorsqu'on réinstalle entièrement la machine) et les clients ont une copie de la clé publique. Le client utilise la clé publique du serveur pour crypter une clé symétrique dite *de session* qui est utilisée pour chiffrer tout le reste des communications. L'authentification de l'utilisateur peut alors être faite sans que le mot de passe soit transmis en clair sur le réseau.

- Expliquez brièvement la nécessité de l'utilisation de SSH et comment ce protocole garantit à la fois la confidentialité, l'authentification (dans les deux sens !) et l'intégrité des données échangées. N'hésitez pas à vous faire un petit schéma pour expliquer l'établissement d'une connexion.
- Connectez-vous à la machine `ssh.enseirb-matmeca.fr`. Expliquez pourquoi lorsque vous vous connectez pour la première fois à une machine donnée votre client vous demande si la clé publique récupérée est bien celle du serveur. Pourquoi la clé privé du serveur doit-elle rester secrète ?

2.3 Assez des mots de passe ?

Lors de la connexion vers une machine distante, ssh demande à l'utilisateur son mot de passe. Imaginez un administrateur système qui doit exécuter cette opération plusieurs fois par jour... Heureusement, de façon analogue à l'authentification des machines, un utilisateur peut utiliser un couple de clés privée/publique pour s'authentifier auprès d'un serveur.

2.3.1 On oublie les mots de passe

Voici la marche à suivre :

1. Création d'un couple de clés : `ssh-keygen -t rsa`, utilisez pour l'instant le choix par défaut pour les 3 questions posées. La clé privée se trouve dans `~/.ssh/id_rsa` et votre clé publique est dans `~/.ssh/id_rsa.pub`.
2. Transfert de votre clé publique sur le serveur (utilisez par exemple le serveur `ssh.enseirb-matmeca.fr`) :
   `ssh username@machine "cat >> ~/.ssh/authorized_keys" < ~/.ssh/id_rsa.pub`
   (`authorized_keys` s'appelle selon les versions `authorized_keys2`). Voyez aussi la commande `ssh-copy-id` qui permet de le faire bien plus simplement.
3. Vous pouvez désormais vous connecter sur le serveur distant sans mot de passe !

Si (n'essayez pas) l'on laissait le droit en lecture sur `~/.ssh/id_rsa` , ssh râlerait que ce n'est pas sécurisé, pourquoi ?

Si (n'essayez pas) l'on laissait le droit en écriture sur `~/.ssh/authorized_keys` , ssh râlerait de nouveau, pourquoi ?

Je veux fournir à un collègue un accès `ssh` à ma machine. Quelle est la manière la plus sûre de procéder ?

2.3.2 Mode paranoïaque

Pour protéger votre clé privée et s'assurer que c'est bien la bonne personne qui utilise la clé, il est possible lors de la création des clés de saisir une passphrase servant à crypter la clé privée, et qui sera donc demandée à chaque connexion pour pouvoir l'utiliser. Recommencez la procédure de création des clés en utilisant cette fois-ci une passphrase. On avait évité d'avoir à taper un mot de passe à l'aide d'une paire de clés, mais maintenant on doit taper une passphrase ! Mais on a quand même amélioré la sécurité. Pourquoi ? (Quelles données passent par le réseau ? Pourquoi est-ce notamment intéressant pour des ordinateurs portables ?)

Pour éviter d'avoir à retaper la passphrase à chaque connexion, on peut utiliser un agent ssh qui va conserver dans un cache votre clé privée décryptée durant toute la durée de votre session sur la machine cliente.

1. Tuez d'abord tout agent que gnome aurait lancé pour vous : `killall ssh-agent`
2. Initialisez l'agent ssh : `ssh-agent`
3. Copiez-collez les définitions de variable d'environnement dans le shell. En pratique on utilisera plutôt `` eval `ssh-agent` `` pour le faire directement (attention, ce sont des apostrophes inverses, obtenues avec altgr-7).
4. Ajoutez votre clé au cache de l'agent : `ssh-add`
5. Essayez de vous connecter sur une autre machine. Alors ?
6. Ouvrez une autre fenêtre de terminal. Connectez-vous sur une autre machine. Expliquez.
7. Revenez sur le terminal précédent, lancez `ssh-add -d` , et réessayez.
8. Essayez de vous connecter en cascade (*i.e.* un `ssh` à l'intérieur d'un autre `ssh`) sur différentes machines. Est-ce que cela fonctionne toujours ? Remarquez l'option `-A`.

2.4 Transferts de fichiers

Les commandes `scp` et `sftp` permettent de transférer des fichiers par l'intermédiaire de `ssh`. Testez ! Essayez aussi de transférer un fichier entre deux ordinateurs distants tout en étant sur un troisième.

En pratique, `scp` n'est pas très efficace pour transmettre de nombreux fichiers (option `-r`). Expliquez comment la commande suivante fonctionne et l'importance de l'antislash devant le `;` :

```
tar c myproject | ssh monserveur cd foobar \; tar x
```

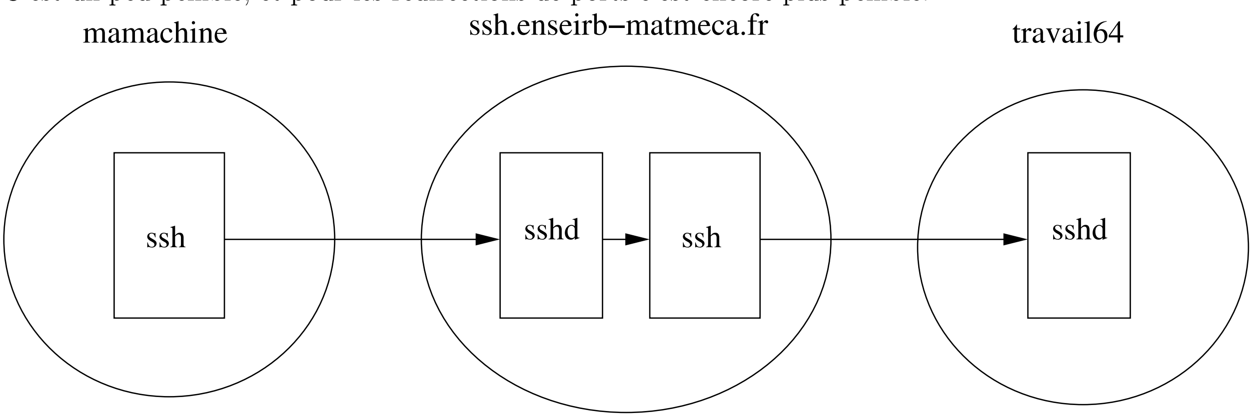
2.5 Affichage X distant

Le protocole `ssh` permet aussi d'exporter un affichage X11 vers votre machine cliente à l'aide de l'option -X.

3 Comment cascader du `ssh` (20m)

Supposons que vous être à l'extérieur de l'école et vous souhaitez vous connecter à `travail64`, mais comme seule `ssh.enseirb-matmeca.fr` est visible depuis l'extérieur, il est nécessaire de passer par elle. Il y a trois solutions.

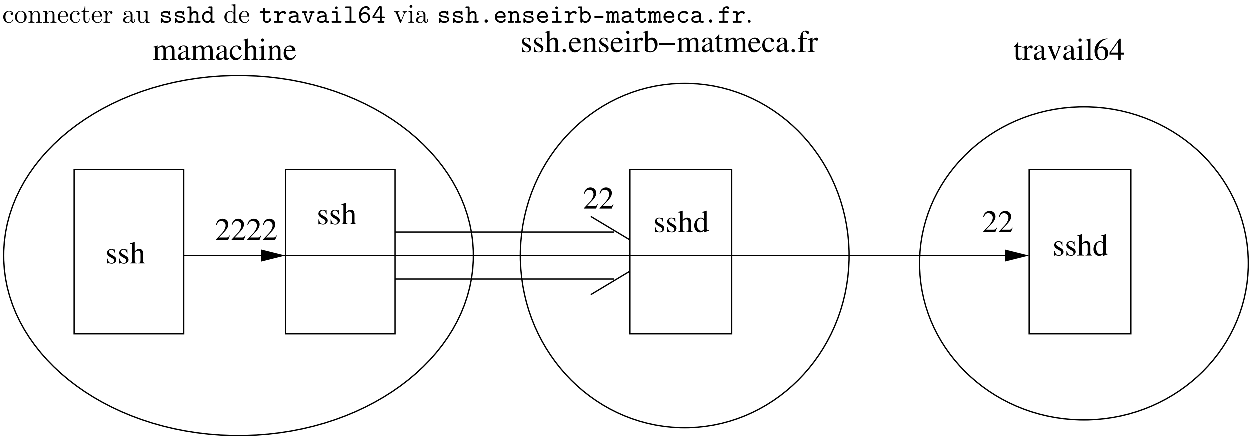
La plus simple est de lancer `ssh ssh.enseirb-matmeca.fr` et à l'intérieur, de lancer `ssh travail64`. C'est un peu pénible, et pour les redirections de ports c'est encore plus pénible.



Une deuxième solution est d'effectuer une redirection de port : lancer un premier

`ssh ssh.enseirb-matmeca.fr -L 2222:travail64:22`

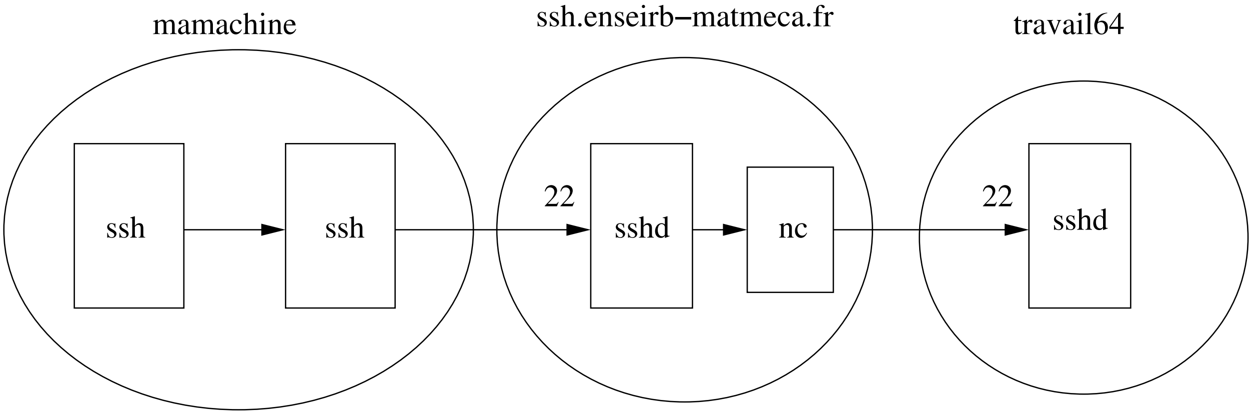
qui ouvre sur votre machine une redirection (sur le port 2222) vers le port ssh de `travail64`. Laissez-le tourner, et dans un autre terminal sur votre machine, un simple `ssh localhost -p 2222` suffit alors à se connecter au `sshd` de `travail64` via `ssh.enseirb-matmeca.fr`.



Une troisième solution est d'utiliser `ProxyCommand` : dans votre `~/.ssh/config`, ajoutez les lignes suivantes :

```
Host montravail64
   User monlogin
   ProxyCommand ssh ssh.enseirb-matmeca.fr /bin/nc -w 600 travail64 22
```

Ces deux lignes indiquent que lorsque l'on lance `ssh montravail64`, plutôt qu'ouvrir une connection TCP vers un port 22, `ssh` va lancer un autre `ssh` vers `ssh.enseirb-matmeca.fr`, et faire exécuter `nc` dessus pour établir la connection vers le port `ssh` de `travail64`.



4 Gestion de certificats (30m)

Le système de clé publique/privée, c'est pratique dans une certaine mesure, mais ça ne passe pas à l'échelle si l'on veut qu'un grand nombre de clients ou serveurs vérifient l'identité d'un grand nombre d'autres clients ou serveurs. Typiquement, que le navigateur de n'importe qui puisse vérifier l'identité d'un serveur web lorsqu'il utilise le protocole `https`. On utilise pour cela une tierce personne, une Autorité de Certification, qui délivre des certificats, que tout un chacun peut alors vérifier à l'aide du certificat racine de cette autorité. Mettons cela en œuvre à l'aide d'`openssl`, en jouant les trois rôles du mécanisme : autorité, serveur et client. Fabriquons d'abord les trois rôles :

```
mkdir autorite
mkdir serveur
mkdir client
```

4.1 Autorité de certification

A priori il est rare d'avoir à jouer ce rôle de tierce personne, on préfère utiliser les services de Verisign, CAcert, ... Il reste intéressant de comprendre ce qui se passe. Prenons donc d'abord le rôle de l'autorité

```
cd autorite/
```

Il s'agit essentiellement de créer un certificat racine :

```
openssl req -x509 -newkey rsa:4096 -keyout ca_priv.key -out ca.crt
```

Des questions sont posées, pour enregistrer dans le certificat des informations utiles pour se souvenir de qui gère cette autorité. Vous pouvez garder les réponses par défaut. On peut revoir le contenu du certificat à l'aide de

```
openssl x509 -in ca.crt -text -noout
```

Le fichier `ca_priv.key` est la partie privée du certificat, à ne pas divulguer à qui que ce soit d'autre. C'est justement là que s'appuie l'autorité : seul celui qui connait le contenu de `ca_priv.key` pourra créer des certificats. `ca.crt` est par contre le certificat racine, à diffuser aux clients, on en reparle plus loin. Il nous reste à créer un répertoire où seront gérés les certificats (ces répertoires et fichiers correspondent à la configuration par défaut d'`openssl`) :

```
mkdir demoCA
mkdir demoCA/newcerts
touch demoCA/index.txt
echo 00 > demoCA/serial
```

4.2 Requête du serveur

Jouons maintenant le rôle d'un administrateur de serveur qui veut s'authentifier auprès d'un client.

```
cd ../serveur/
```

Pour cela il a besoin que l'autorité de certification lui délivre un certificat. Il doit donc d'abord faire la requête (attention à ne pas remettre `-x509`, car il s'agit maintenant d'une requête de certification) :

```
openssl req -newkey rsa:4096 -keyout priv.key -out cert.csr
```

De même, on peut enregistrer des informations utiles, les valeurs par défaut iront bien, sauf le champ Common Name, qui ne doit pas être vide. Avec la configuration par défaut, les champs `countryName`, `stateOrProvinceName`, et `organizationName` doivent être les mêmes que l'autorité.

Le fichier `priv.key` est la partie privé du certificat, à ne pas divulguer à qui que ce soit. C'est justement là que s'appuie la certification : seul celui qui connait le contenu de `priv.key` pourra être certifié. Il faut par contre transmettre `cert.csr` à l'autorité de certification.

```
cp cert.csr ../autorite/
```

Changeons de casquette, nous sommes de nouveau l'autorité de certification.

```
cd ../autorite/
```

Nous vérifions l'identité de l'administrateur du serveur par un moyen « classique » (registre d'industrie, carte d'identité, connaissance personnelle, ...), et que cela correspond bien à ce qui est écrit dans le certificat demandé :

```
openssl req -in cert.csr -text -verify -noout
```

et nous validons alors la requête :

```
openssl ca -cert ca.crt -keyfile ca_priv.key -in cert.csr -out cert.crt
```

Nous pouvons alors transmettre `cert.crt`, le certificat ainsi fraîchement créé, à l'administrateur du serveur.

```
cp cert.crt ../serveur/
```

Reprenons donc la casquette d'administrateur de serveur. Nous pourrions par exemple configurer Apache pour utiliser ce certificat, il lui faut à la fois la partie publique, et la clé privée, nécessaire pour s'authentifier vraiment, typiquement ceci dans /etc/apache2/sites-available/monsite-ssl :

```
SSLCertificateFile cert.crt
SSLCertificateKeyFile priv.key
```

Faisons-le avec un serveur web plus rapide à installer, bozohttpd :

```
apt-get source bozohttpd
cd bozohttpd-*
make -f Makefile.boot
./bozohttpd -b -f -I 1234 -Z ~/serveur/cert.crt ~/serveur/priv.key $PWD
```

Il faut éventuellement taper la passphrase, et le serveur web est prêt.

4.3 Client

Souvent, le client possède déjà le certificat racine de l'autorité de certification, installé typiquement par la distribution système dans `/etc/ssl/certs` . Le navigateur vérifie alors que le certificat fourni par le serveur (et authentifié par challenge sur la clé privée) a bien été délivré par l'autorité, on peut le faire à la main ainsi :

```
cd ../client/
cp ../autorite/ca.crt
cp ../serveur/cert.crt
openssl verify -CAfile ca.crt cert.crt
```

Si le client ne possède pas déjà le certificat racine de l'autorité de certification, le navigateur web du client affiche un avertissement de sécurité et propose d'utiliser le certificat tout de même : essayez de lancer un navigateur sur https ://localhost :1234/ , et demandez à voir le certificat, constatez que c'est bien celui que vous avez émis. Quel risque peut-il y avoir à accepter le certificat sans vérifier l'empreinte ? Si le client voit en personne l'administrateur du site web et confirme avec lui l'identité de la signature du certificat fourni par le serveur web, est-ce qu'il peut alors faire confiance au site ? Pourquoi c'est dommage de faire ainsi ?

Ajoutez l'exception au navigateur, constatez que l'on obtient bien la page web (même si c'est bien sûr une page 404), fermez complètement le navigateur, relancez-le. Que constate-t-on ?

Refabriquez un autre certificat (ne changez pas d'autorité de certification, juste de certificat, et mettez un Common Name différent sinon il y aura un conflit dans la base avec le certificat précédent), relancez le serveur web avec ce certificat, et rechargez la page dans le client, que constate-t-on ?

En effet, le client n'a récupéré que le certificat du serveur, et pas le certificat racine. Plutôt qu'ajouter ce certificat seulement, ajoutez le certificat racine, c'est-à-dire celui de l'autorité de certification, `autorite/ca.crt` (c'est en général dans le panel de préférences, dans les options avancées de sécurité). Rechargez la page, constatez qu'il n'a pas demandé de confirmation : il suffit de confirmer seulement le certificat de l'autorité, pour que tous les certificats produits par cette autorité soient acceptés. Regénérez encore un autre certificat (en changeant de nouveau le Common Name) et relancez le serveur web pour confirmer.

4.4 Bilan

Au final, il suffit que

- Les clients récupèrent auprès de l'autorité de certification son certificat racine.
- Les administrateurs de serveurs obtiennent auprès de l'autorité des certificats

et les clients peuvent alors être sûrs de l'identité des administrateurs des serveurs, sans avoir à interagir directement avec eux. Où sont les maillons faibles ?