Présentation et Mise en Oeuvre de SSF/SSH

Documentation extraite de la formation continue INPG "Internet et Intranet"

Bernard Martinet
UREC - CICG
Bernard.Martinet@grenet.fr

Contributions

Document réalisé d 'après :
la documentation sur SSF de Bernard Perrot (IRMAR), le Tutorial JRES99 sur le Chiffrement de S. Aumont (CRU), R. Dirlewanger et O. Porte (CNRS), les documentations internes de l'Imag (P. Laforgue), de l'UJF (Y. Siret & A. Voutier) et de l'Université d'Evry (P. Petit) sur l'installation et l'utilisation des clients/serveurs SSH

Plan

Partie 1 : Présentation de SSH/SSF

SSH : pourquoi ?

Éviter la circulation en clair sur le réseau des mots de passe donc leur risque de compromission.
Renforcer l'authentification des machines (les authentifications sur les adresses IP sont sensibles à la mascarade).
Sécuriser l'emploi des commandes à distance.
Sécuriser le transfert de données.
Sécuriser les sessions X11.

Autres solutions ?

les r-commandes (rlogin, rcp, rsh)

Elles évitent bien la circulation du mot de passe s'il existe un fichier .rhosts, mais se posent alors d'autres gros problèmes de sécurité échappant totalement à l'administrateur.

Les mots de passe à "usage unique"

Type Carte Securid.
Ils ne protègent que l'authentification, mais pas les transferts de données ou les sessions X11.
Les systèmes sont généralement chers, très contraignants, et difficiles à mettre en place et à maintenir pour un large public.

Telnet avec chiffrement

Il ne couvre que telnet. Pour le transfert de données ou les sessions X11, le problème reste entier.

La solution SSH

Elle est basée sur une authentification forte de type RSA (Rivest, Shamir, Adleman).

Remplace les r-commandes :

rlogin et rsh deviennent ssh (ou slogin), rcp devient scp.
Le remplacement est transparent si on utilise des alias.

Une connexion SSH entre deux machines établit un tunnel chiffré pour la session.

Toute application TCP peut être renvoyée dans le tunnel, toute session X11 l'est automatiquement. De plus, le tunnel peut être compressé sur demande.

SSH : les caractéristiques

Développé par un chercheur finlandais (T. Ylonen).
Les versions 1.x utilisant le protocole v1.5 sont gratuites. La société SSH Inc. développe et commercialise les versions 2.x et 3 basées sur le protocole v2. L'utilisation est gratuite uniquement pour usage privé, et sur OS libres. Actuellement le produit est fourni gratuitement aux universités. (*"SSH grants licenses free-of-charge to universities for the use of SSH Secure Shell among staff, faculty members and students"*).

Il existe également une implémentation sous licence GNU (OpenSSH) basée sur le protocole v2, mais seules les versions 1.x répondent à la législation française encore en vigueur.

Nous traiterons donc ci-après uniquement la version 1.5 du protocole.

L'authentification et les échanges de clés sont faits avec RSA (clés publiques et clés privées). Le chiffrement des communications se fait avec les algorithmes : DES (2^{56}), IDEA (défaut Unix 2^{128}), 3DES (défaut SSH 2^{168}), RC4 (2^{128}), Blowfish (2^{256}).

Une des caractéristiques les plus intéressantes est le relai des échanges X11 à travers le canal chiffré si la variable $DISPLAY existe, et s'il n'y a pas une configuration explicite inverse (en vue de limiter l'action de l'utilisateur sur la machine cible).

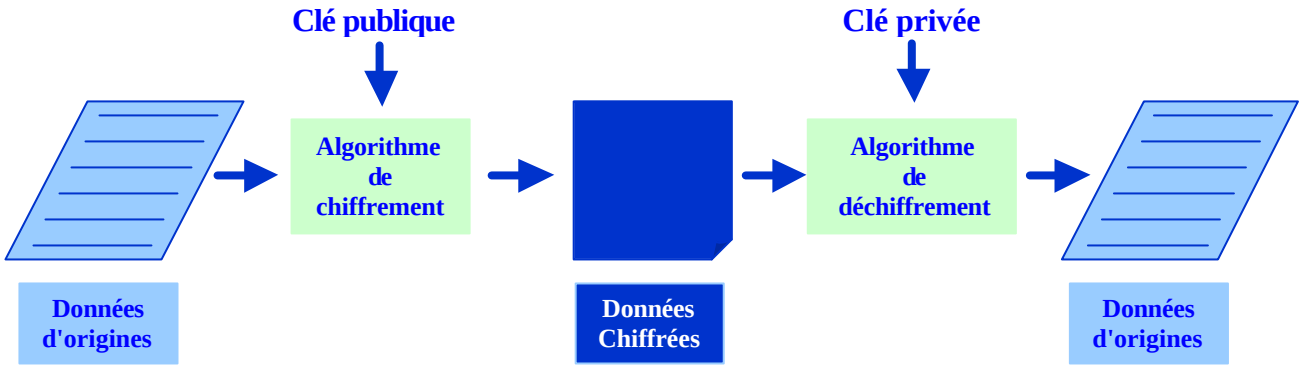
Une tentative de connexion SSH avec une machine cible ne tournant pas le protocole repasse automatiquement à un rsh "classique". Cette possibilité assez confortable crée un danger potentiel s'il existe encore des .rhosts sur la machine cible et que son serveur SSH est arrêté (ou inexistant).

Le principe clé publique/clé privée

Nous allons faire ici une petite présentation de ce qu'est un mécanisme d'authentification à clé asymétrique. Ce paragraphe ne prétend pas expliquer la cryptologie.

Ce mode de chiffrement prend sa source dans les années 1970. Son principe révolutionne le mode de fonctionnement traditionnel à clé secrète dans lequel les deux utilisateurs doivent avoir connaissance de la même clé (d'où une multiplication du nombre de clés : une par couple de correspondants, et quid des nouveaux correspondants potentiels ?)

Le mécanisme asymétrique est illustré par le schéma suivant :



Seul le détenteur de la clé privée peut déchiffrer le message. La même clé publique pouvant être fournie à de nombreux correspondants différents.

Nombreux algorithmes utilisent le principe d'authentification par clés asymétriques dont RSA inventé en 1977 par R. Rivest, A. Shamir et L. Adleman.

Le principe de calcul est basé sur l'exponentiation de grands nombres et un calcul de congruence modulo un nombre premier.
L'exponentiation est relativement facile mais l'inverse est complexe.

L'algorithme RSA

Génération des clés

- choisir 2 grands nombres premiers très grands p et q (100 chiffres minimum),
- calculer n = p * q, n sera appelé le modulo du chiffrement,
- choisir un nombre e plus petit que n et premier avec ((p - 1) * (q - 1)),
- calculer ensuite d tel que d * e = 1 mod ((p - 1) * (q - 1)). d est calculé avec l'algorithme dit "d'Euclide étendu" permettant de trouver l'inverse d'un nombre modulo n.

d et e sont respectivement les composants privés et publics. d est l'inverse de e dans l'arithmétique modulo ((p - 1) * (q - 1)).
La clé publique est donnée par le couple (n,e) et la clé privée par d.

Le chiffrement devient alors :

- découper le message en blocs de même longueur n (en pratique 200 octets ou plus),
- chiffrer en utilisant la formule :

  $c = m^e \bmod n$

  avec c correspondant au bloc chiffré et m correspondant au bloc origine

- pour déchiffrer faire :

  $m = c^d \bmod n$

Avantages et Inconvénients

Avantages et inconvénients du chiffrement asymétrique peuvent se résumer comme suit :

- trouver des grands nombres premiers,
- nécessité de choisir des clés privées et publiques assez longues,
- difficulté à réaliser des opérations modulo n rapidement,
- complexité algorithmique de la méthode,
- la solution reste malgré tout assez générale et sûre si la longueur des clés et les précautions d'emploi sont respectées.

La sûreté dépend alors de :

- la non-divulgation de p et q,
- la difficulté qu'il existe à factoriser de grands nombres,
- l'absence de méthodes mathématiques actuellement connues permettant de calculer la clé privée d à partir du couple (n,e), la clé publique.

Beaucoup de tentatives pour casser ce type de chiffrement (dont les plus prometteuses sont basées sur la factorisation des grands nombres) sont encore mises en échec par l'emploi de clés de longueur suffisantes. En Août 1999, une clé de 512 bits a nécessité 8400 MIPS Années[1] pour être factorisée, une clé de 1024 bits le sera probablement aux alentours des années 2020[2]

[1] 1 MIPS Années = calcul pendant 1 année à la vitesse constante de 1 Million d'Instruction Par Seconde.
[2] cf. ftp://ftp.gage.polytechnique.fr/pub/publications/jma/rsa-155.ps.

Rappel de la législation française

Avant février 1998 : tout emploi de mécanisme de chiffrement nécessitait l'obtention préalable d'une autorisation.

Après le 23 mars 1998 : l'usage de logiciel à clé de chiffrement d'une entropie inférieure à 2^{40} est libre sous condition de déclaration et d'enregistrement auprès du SCSSI.

Depuis le 17 mars 1999 : cette mesure a été étendue aux algorithmes à clé 2^{128}. *" matériels ou logiciels offrant un service de confidentialité mis en œuvre par un algorithme dont la clé est d'une longueur inférieure à 2^{128} "*

En 2001, la libéralisation totale est annoncée, mais nous attendons toujours la publication des décrets (cf. Loi sur la Société de l'Information).

Introduction à SSF

SSF est une adaptation de la suite publique "SSH Unix", destinée à l'usage sur le territoire français en conformité avec la législation française concernant la cryptologie.

SSF-128 est un produit dont la taille de l'espace de clés est limitée à 2^{128}, et dont l'usage est libre (les utilisateurs n'ont aucune démarche à effectuer).
Il a fait l'objet de la déclaration n° 9908271 auprès du SCSSI.
Le client Windows 95/NT (dénommé TTSSF-128) a fait l'objet de la déclaration n° 9908275 auprès du SCSSI.

SSF est entièrement compatible avec le SSH standard (version 1.5 du protocole) avec lequel il interagit en toute transparence.

Il est mis à l'entière disposition de la communauté sous réserve d'acceptation d'une charte et d'inscription en règle auprès de l'IN2P3.
voir : *http://perso.univ-rennes1.fr/bernard.perrot/ssf* rubrique Avertissement

SSF ne veut rien dire de particulier … il pourrait signifier : Shell Sécurisé Français … (rappel: SSH signifie Secure SHell).

SSF : pourquoi ?

Pourquoi adapter SSH ?

Par défaut SSH utilise le chiffrement triple-DES considéré comme un chiffrement sur 168 bits, donc hors de la limite légale des 128 bits.

De plus l'emploi de ce mode de chiffrement est impossible à supprimer : le protocole 1.5 prévoit que le 3-DES est l'algorithme commun. Il n'est pas négocié, et il est le repli obligatoire en cas d'échec d'un autre algorithme.

La distribution Unix standard comporte également Blowfish qui utilise des clés 256 bits.

Comment ?

Par révélation de la partie de clé supérieure à 128 bits.

SSF : les caractéristiques propres

Le chiffrement est limité à 128 bits.
Le fonctionnement est identique à SSH pour les chiffrements inférieurs à 128 bits (DES, IDEA, RC4). Pour le triple-DES ou Blowfish (clé > 2^{128}) une limitation d'entropie de la clé de session à 128 bits entre en jeu en rendant les bits au-delà conventionnels.
On ne note pas d'affaiblissement du chiffrement pour un usage approprié de SSF[3].

Des ajouts ont été apportés à SSF, notamment une fonction pour "unaliaser" les hostnames (fermes de machines), un support de l'identification S/Key et enfin l'ajout d'un message dans le syslog du démon (identifiant du client).

Le Package SSH/SSF

Le Package SSH (SSF) se compose des éléments suivant :

- Un serveur : sshd (ssfd)
- Un client : ssh ou slogin (ssf)
- Un outil de transfert de fichiers : scp

Auxquels s'ajoutent des outils d'administrations des clés :

- ssh-keygen *génération de bi-clé publique/privée*
- ssh-agent *gestion des clés pour une session*
- ssh-add " "
- make-ssh-known-hosts *gestion de la liste des machines connues*

Seuls le démon et le client SSF sont légèrement différents du démon et du client SSH et portent donc un nom différent. Ils doivent être considérés comme une implémentation du protocole SSH version 1.5 réalisant tout ou partie des fonctions du logiciel SSH basé sur le même protocole.

Le protocole SSH v1

SSH définit un protocole entre un client et un serveur pour l'établissement d'une connexion distante chiffrée. Les machines sont connectées par un réseau non sécurisé (qui peut être espionné, dégradé ou abusé par des tiers hostiles).

Une connexion est toujours initialisée par le côté client. Le serveur est en écoute sur le port 22 (par défaut, mais il est possible d'utiliser n'importe quel port). Plusieurs clients peuvent se connecter à la même machine.
Client et serveur communiquent en TCP/IP par une socket bidirectionnelle.

Dans une phase initiale, le client authentifie (par un mécanisme RSA) le serveur au début de chaque connexion pour se prémunir contre les "Chevaux de Troie" (par DNS ou IP spoofing) et les attaques de type "man-in-the-middle". Le serveur authentifie (par le mécanisme RSA) le client avant d'accepter des authentifications de type .rhosts ou /etc/hosts.equiv pour prévenir les attaques de type DNS, Routing, ou IP spoofing.

[3] Dixit l'auteur, Bernard Perrot, qui considère qu'au vu de ses possibilités réelles de canaux subliminaux, SSH ne peut pas être utilisé dans des opérations de type commerce bancaire, mais reste utilisable pour l'échange de données scientifiques.

Puis dans une seconde phase, sur la machine cliente, un agent d'authentification va pouvoir gérer des clés RSA pour l'authentification d'un utilisateur particulier

Phase initiale

- Le client contacte le serveur. Le serveur accepte la connexion et répond en envoyant sa chaîne d'identification de version. Le client analyse l'identification du serveur et renvoie sa propre identification. Ceci a pour but de valider les ports de connexions, et de déclarer la version de logiciel utilisée par les deux parties. Si l'un des deux côtés ne comprend pas ou ne supporte pas l'autre version (accord sur la plus basse), la connexion tombe. Cet échange se fait en clair.
- Les deux parties passent en mode paquet.
- Le serveur transmet sa host-key pour être authentifié (chaque machine possède une clé RSA utilisée pour l'authentification), une server-key (clé RSA régénérée toutes les 2 heures), plus des informations complémentaires comme la liste des codages supportés, la liste des authentifications supportées, et un cookie aléatoire (64 bits) pour rendre plus difficile les attaques en IP spoofing.
- Le client génère une clé de session (256 bits), le crypte en utilisant la paire de clé RSA (host-key et server-key) du serveur, et lui renvoie la clé de session ainsi que l'algorithme de chiffrement retenu, le cookie 64 bits fourni par le serveur et ses flags de protocole (les deux côtés calculent simultanément un session-id = MD5 (host-key →n || server-key→n || cookie). Envoi en clair
- Les deux côtés basculent en mode crypté en utilisant la clé de session et l'algorithme choisi. Le serveur envoie un message de confirmation crypté au client. A partir de cet instant le canal est chiffré, et la phase d'authentification du client peut commencer

Méthodes d'authentification du client

Quatre méthodes possibles seront employées pour l'authentification de l'usager :

- SSH_AUTH_RHOSTS : c'est une authentification à la rlogin (.rhosts), avec la possibilité d'utiliser des fichiers /etc/shosts.equiv et .shosts qui ne seront pas compris par les r-commandes,
- SSH_AUTH_RSA : utilisation d'une séquence challenge/réponse avec RSA,
- SSH_AUTH_PASSWD : identification normale par mot de passe (à la telnet),
- SSH_AUTH_RHOSTS_RSA : mélange entre l'utilisation des .rhosts et de l'authentification RSA.

SSH_AUTH_PASSWORD

Le client envoie le mot de passe sous forme texte (mais il est normalement chiffré par le paquet), le serveur vérifie le mot de passe et accepte si l'authentification réussit.
Le mot de passe est lu par le client directement de l'utilisateur (pas d'interaction entre l'usager et un programme de login)

Il n'y a pas de confiance particulière à avoir envers la machine cliente, le réseau, le DNS ou autre. Seule compte la connaissance du mot de passe de l'usager.

SSH_AUTH_RHOSTS

C'est le mode d'authentification utilisée par rlogin et rsh.
Il nécessite la confiance dans :

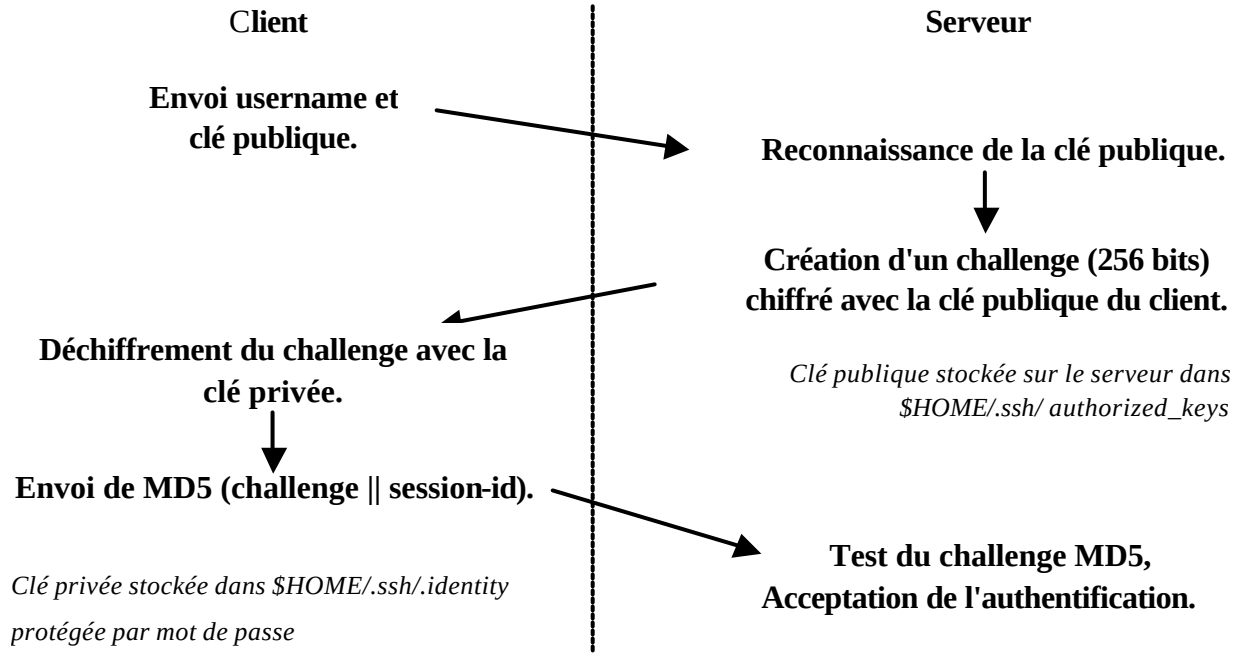
- la machine cliente (root peut usurper toutes les identités d'usagers)
- le DNS (la correspondance nom ⇔ @IP se fait par une résolution en forward puis en reverse)
- le réseau puisque l'écoute du réseau peut permettre de l'IP-spoofing, cependant le protocole interdit l'IP-spoofing aveugle (utilisation du cookie aléatoire)

Il y a possibilité d'utiliser des fichiers .shosts et shosts.equiv à la place de .rhosts et hosts.equiv pour invalider l'usage des r-commandes classiques.

SSH_AUTH_RSA

L'idée derrière l'authentification RSA est que le serveur reconnaisse la clé publique offerte par le client, génère un challenge aléatoire et le chiffre avec la clé publique. Le client doit alors prouver qu'il possède sa clé privée en déchiffrant le challenge.

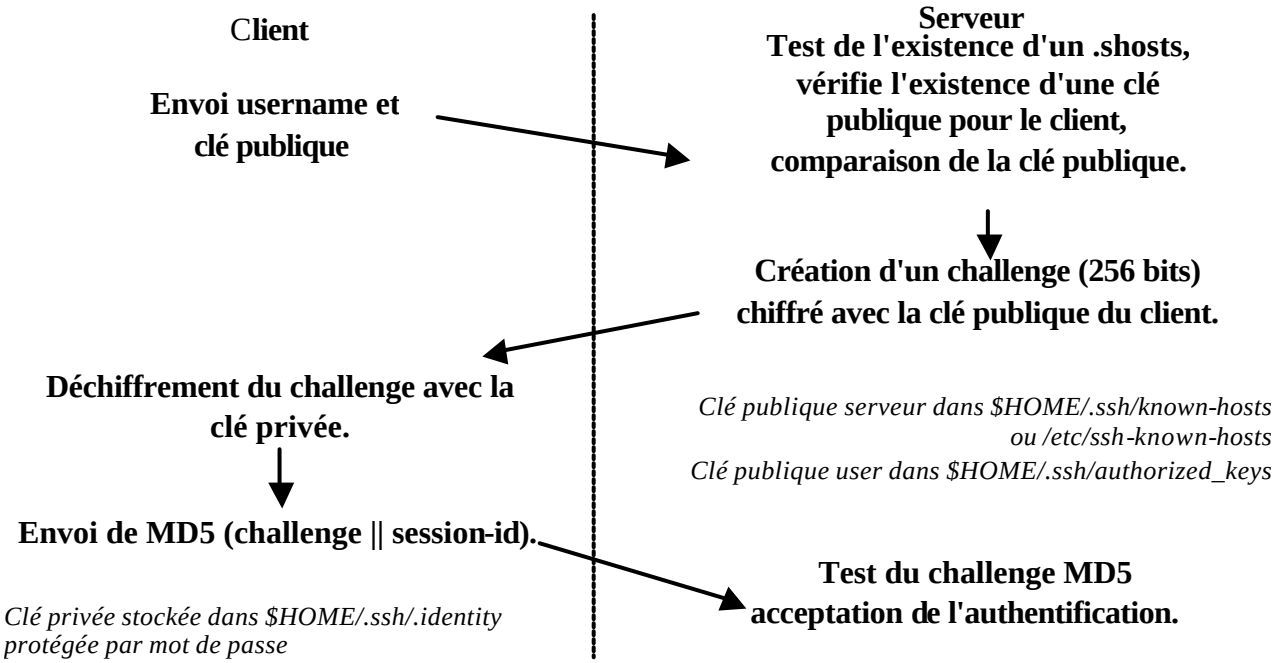
La procédure est la suivante :



L'authentification n'accorde pas de confiance particulière à la machine distante, au réseau ou au DNS, seule la possession de la clé privée compte.

SSH_AUTH_RHOSTS_RSA

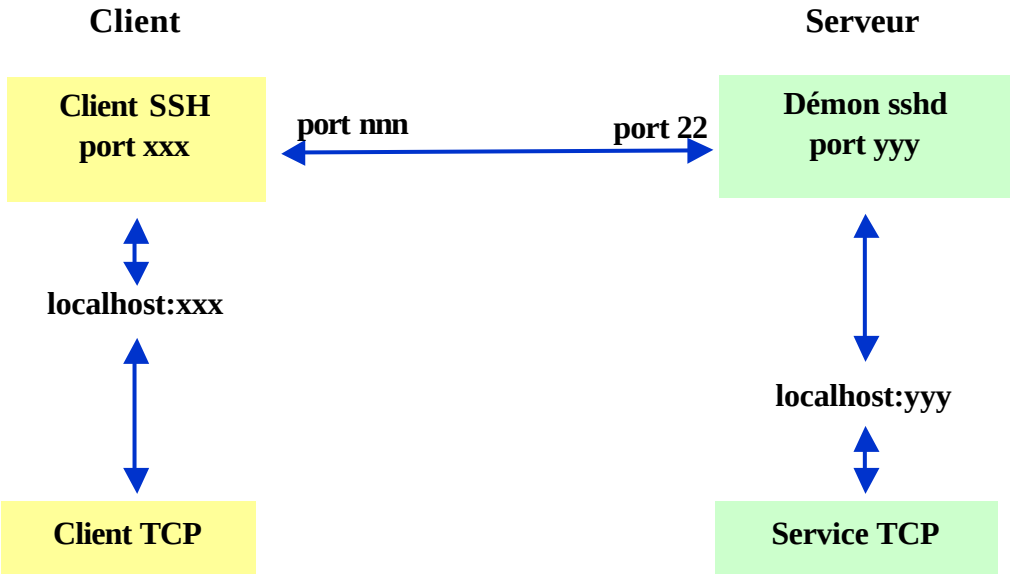
En addition à l'identification traditionnelle par .rhosts ou hosts.equiv, cette méthode requiert du client une authentification par RSA.



Besoin d'une confiance dans la machine côté client (root peut prétendre être n'importe quel usager), pour le reste, seule la possession de la clé privée est importante.

Mode tunnel

Le service TCP est redirigé à travers le tunnel chiffré de la session SSH

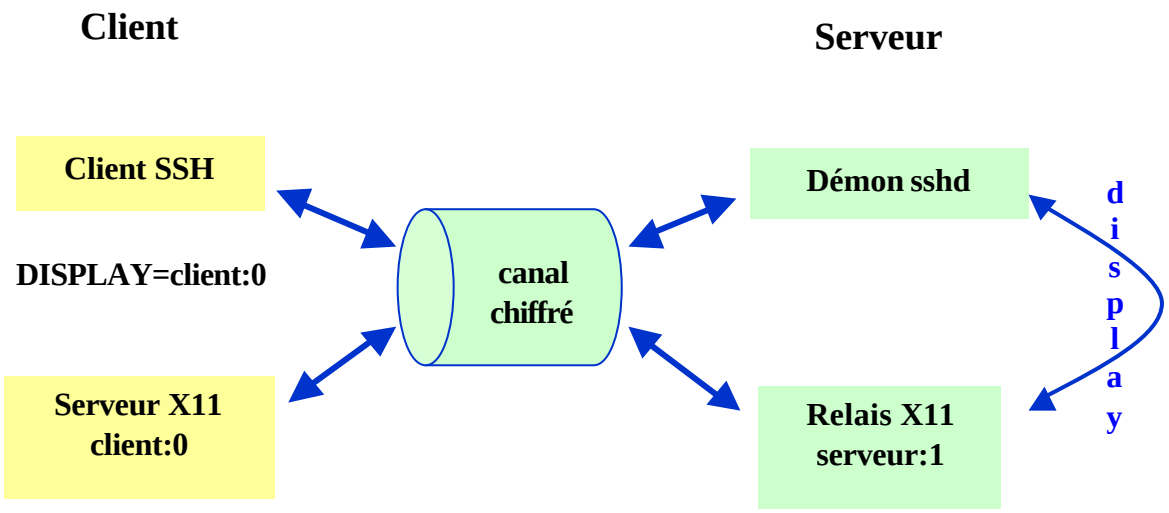


Le client SSH établit un tunnel chiffré avec le port 22 du serveur depuis un port nnn.

Le client TCP établit sa connexion avec localhost (port xxx) sur le client SSH, tandis que sur le serveur SSH le service TCP est en relation avec le port yyy de localhost.
Le service SSH redirige du côté client tout ce qui arrive pour le port xxx sur le port nnn, et du côté serveur tout ce qui arrive sur le port 22, pour le service TCP, sera redirigé sur le port yyy.

Relais X11

Le serveur crée un pseudo-serveur local qui est redirigé à l'intérieur du canal chiffré de la session SSH



Partie 2 : Mise en œuvre de SSH

SSF : installation

Récupérer le programme SSF sur : *http://perso.univ-rennes1.fr/bernard.perrot/ssf*
Cette phase nécessite un enregistrement sur le site. La demande pouvant être valable pour l'ensemble d'une structure (récupération d'un code d'authentification nécessaire au téléchargement des fichiers).

Téléchargements de :

- Archive ssf-1.2.27.6 (kit source client/serveur/outils SSF pour Unix, version 128 bits)
- Archive ssf-1.2.27.6 (kit RPM client/serveur/outils SSF pour Linux RH6.0, version 128 bits)
- Archive ttssf-1.4.4 (extension à TeraTerm Pro, voir ci-dessous, version 128 bits)
- Archive TeraTerm Pro 2.3 (client Telnet Windows 95/NT freeware)

Pour Unix

Décompresser et détarer le fichier ssf-1.2.27.8.tgz
Exécuter :

```
cd ssf-1.2.27.8
./configure --with-rsh='/usr/bin/rsh '--program-transform-name=s/^s/r/'
make
```

puis en root :

```
make install
```

Les options fournies au configure vont permettre aux commandes rsh et rcp d'utiliser SSH lorsque le serveur distant en est capable et de se rabattre sur les anciennes commandes sinon

A la configuration, il est également conseillé d'utiliser l'option **--with-libwrap** afin de contrôler les connexions à la machine par l'emploi de tcp_wrapper.

"make install" effectue :

- l'installation de tous les exécutables (ssf scp ssh-agent ssh-keygen ssh-add ssh-askpass make-ssh-known-hosts),
- l'installation des fichiers d'options /etc/ssh_config et /etc/sshd_config,
- la génération d'un couple de clés identifiant la machine :
  - clé privée : /etc/ssh_host_key
  - clé publique : /etc/ssh_host_key.pub
- l'installation du démon serveur ssfd,
- l'installation des manuels :
  - sous /usr/.../man1 : ssh.1 scp.1 ssh-add.1 ssh-agent.1 ssh-keygen.1
  - sous /usr/.../man8 : sshd.8

Si le répertoire d'installation /usr/local/bin est partagé avec d'autres machines :

- effectuer "./configure ; make ; make install" sur la machine source NFS ;
- effectuer "make hostinstall" sur chacune des autres machines pour générer le couple de clés privées/publiques et installer leurs fichiers de configurations ;

- installer le lancement du serveur sur chacune des autres machines si nécessaire.

Compléments d'installation

Si on veut forcer l'usage de SSH sur une machine dès que le serveur distant en est capable, on peut faire les adaptations suivantes :

- créer un script exécutable /usr/local/bin/rcp contenant :
  #!/bin/sh
  exec /usr/local/bin/scp1 -o 'FallBackToRsh yes' -q $*
- créer un lien :
  ln -s /usr/local/bin/ssf /usr/local/bin/rsh
- s'il existe un script xon (/usr/local/bin/xon) sur la machine :
  enlever le test :
  if [-f /usr/bin/remsh] ... elif [-f /usr/bin/rcmd] ...

  et le remplacer par une assignation :
  rsh=/usr/local/bin/rsh

Dans ce cas l'authentification par SSH se fera (même à l'insu de l'utilisateur), lorsque ce sera possible sur le serveur appelé par une r-commande (authentification par SSH_AUTH_RHOSTS ou SSH_AUTH_PASSWD au minimum). On suppose ici que /usr/local/bin est en tête des règles de recherche de chaque utilisateur.

SSF : configuration

Le choix des options se fait grâce à deux fichiers de configurations :

/etc/ssh_config :

pour modifier les défauts des usagers du client :

Host serv.site.fr	*interdit le retour à l'usage de rsh*
FallBackToRsh no	*sur la machine serv.site.fr*

/etc/sshd_config :

pour modifier les choix du démon serveur

RhostsAuthentication no	*interdit l'authentification par .rhosts ou .shosts*
RhostsRSAAuthentication yes	*autorise l'emploi de .rhosts + RSA*
AllowHosts *.grenet.fr *.inpg.fr *.imag.fr	*Autorise la connexion pour certaines machines ou domaine*

Serveur à accès restreint

Pour réaliser un serveur à accès restreint (exemple : machine obligatoire pour entrer sur un site depuis l'extérieur), il faut :

- commenter les lignes de lancement des démon telnetd, rshd, rlogind et rexecd dans /etc/inetd.conf (kill -HUP inetd) afin d'interdire définitivement telnet et les r-commandes,

- dans /etc/sshd_config :

- définir sa politique d'autorisation de connexion en .rhosts ou .shosts à l'aide de la commande : AllowShosts ….
  Ceci limitera comme client, les machines présentes dans les fichiers ~/.rhosts ou ~/.shosts. Les démons des r-commandes n'étant plus actifs, il n'y aura pas de possibilités de connexion autrement que par SSH, donc en mode SSH_AUTH_RHOSTS_RSA (ou SSH-AUTH-RSA, SSH-AUTH-PASSWD).
  Si on écrit une ligne AllowShosts sans argument les fichiers ~/.rhosts ou ~/.shosts ne pourront pas fonctionner

- ajouter des directives utiles comme :
  PermitRootLogin no
  AllowHosts *.domaine.fr pour limiter le domaine de connexion

- Relancer le démon sshd (kill -HUP `cat /etc/sshd.pid`)

Usage :

connexion

ssf -l <user> <site-distant> ou **ssf <user>@<site-distant>**
ou **slogin -l <user> <site-distant>** (par analogie avec rlogin)
"-l user" est facultatif si on possède le même nom de login sur les 2 sites

si ssh est installé sur le site distant
1 ère connexion :

```
Host key not found from the list of known hosts.
Are you sure you want to continue connecting (yes/no)? yes      (et pas y ou return)
Host '<site-distant>' added to the list of known hosts.
<user>@<site-distant>'s password: XXXXXXXX
```

connexions suivantes :

```
<user>@<site-distant>'s password: XXXXXXXX
```

dans les deux cas, le mot de passe est crypté par la session.
Si ssh n'est pas installé sur le site distant, bascule sur un rlogin normal non crypté

copie de fichier

scp <nomfich.ici> <user>@<site-distant>:<nomfich.labas>
si on possède le même nom de login sur les deux sites, la commande se réduira à :
scp <nomfich.ici> <site-distant>:<nomfich.labas>

```
<user>@<site-distant>'s password: XXXXXXXXX
nomfich.ici          |        19 KB |  19.3 kB/s | ETA: 00:00:00 | 100%
```

la commande fonctionne dans les deux sens
scp <site-distant>:<nomfich.labas> <nomfich.ici>

Comme pour la commande rcp, on a la possibilité d'utiliser les caractères '?' et '*' pour recopier un ensemble de fichiers et l'option -r pour récupérer récursivement tout un répertoire.

Exécution de commandes

ssf <user>@<site-distant> <commande>

si on possède le même nom de login sur les deux sites, la commande se réduira à :
ssf <site-distant> <commande>

exemple :

ssf resto.imag.fr uname -a
<martinet>@resto.imag.fr's password: *XXXXXXXXX*
SunOS resto.imag.fr 5.7 Generic_106541-07 sun4u sparc SUNW,Ultra-30

Pour le lancement d'une session interactive :

ssf -f <site-distant> xterm
<user>@<site-distant>'s password: *XXXXXXXXX*

L'option -f permet de lancer ssf en arrière plan. On peut donc travailler indifféremment sur les deux fenêtres.

Connexion avec clé RSA

Pour plus de sécurité, on peut vouloir réaliser des connexions sans utiliser son mot de passe Unix (fortement conseillé).

Dans ce cas la connexion se fera avec l'algorithme RSA (clé privée - clé publique)
Cela reste du ressort de l'utilisateur qui doit se créer son propre jeu de clés (on peut néanmoins l'imposer dans sshd_config).

L'utilisateur s'identifiera à l'aide d'une passphrase, sorte de super mot de passe (maximum de 127 caractères avec espaces et ponctuation autorisés), tenue secrète servant à authentifier le propriétaire de la clé privée et éviter son emprunt par un tiers.
Il n'y a plus de transmission du password Unix (même crypté) sur le réseau. Celui-ci ne sert plus qu'à la connexion directe sur la machine.

La clé privée et la passphrase sont codées, il est donc nécessaire d'utiliser une phrase facile à retenir pour l'usager :

ex : "je n'aime pas choisir des mot2passes compliqués !"

Création des clés

La création des clés se fait avec la commande **ssh-keygen**
Dans l'exemple qui suit les clés ont été créées sur la machine bistrot par l'usager martinet :

```
bistrot (143) > ssh-keygen
Initializing random number generator...
Generating p:  .....................++ (distance 370)
Generating q:  .....++ (distance 46)
Computing the keys...
Testing the keys...
Key generation complete.
Enter file in which to save the key (/home/martinet/.ssh/identity): <return>
Enter passphrase: maître corbeau sur ...
Enter the same passphrase again: maître corbeau sur ...
Your identification has been saved in /home/martinet/.ssh/identity.
Your public key is:
1024 35 1518765082145…..540991711537 martinet@bistrot
        (on trouve ici une série de 1024 chiffres)
Your public key has been saved in /home/martinet/.ssh/identity.pub
```

La clé publique (en clair) est stockée dans le fichier /home/martinet/.ssh/identity.pub
La clé privée (cryptée) est stockée dans le fichier /home/martinet/.ssh/identity

Il faut ensuite recopier sur chacune des machines distantes où on voudra se connecter le fichier de clé publique identity.pub dans ~/.ssh/authorized_keys

exemple :

scp ~/.ssh/identity.pub resto.imag.fr:.ssh/authorized_keys
martinet@resto.imag.fr's password : *XXXXXXXX*

(ici on a le même nom d'usager sur la machine locale et sur la machine resto)

Pour les commandes slogin ou scp ultérieures :
exemple :

slogin resto.imag.fr
Enter passphrase for RSA key 'martinet@bistrot': maître corbeau sur …
(passphrase en aveugle)
Last login: Fri Mar 17 14:09:10 2000 from bistrot.grenet.fr
Sun Microsystems Inc. SunOS 5.7 Generic October 1998

Attention ce n'est pas symétrique : les clés pour se connecter de A vers B ne servent pas pour se connecter de B vers A ; il faut répéter ce qu'on a fait sur A et B dans l'autre sens avec la même passphrase ou une autre (conseillé).

Réglages utilisateur

Le fichier $HOME/.ssh/config permet à l'utilisateur de redéfinir certains paramètres pour une cible (cf. /etc/ssh-config)

exemple (*entre parenthèse la valeur par défaut*) :

- User <nomuser> (*nom local*) : nom d'utilisateur sur la machine cible
- IdentityFile <nomfich> (*$HOME/.ssh/identity*) : fichier contenant la clé privée pour la machine
- Compression <yes/no> (*no*) : contrôle de la compression de données pour la connexion. La compression augmente le rendement réseau au dépend d'une perte de temps de calcul
- CompressionLevel <1-9> (*6*) : niveau de compression 1 (faible) à 9 (fort)
- FallBackToRsh <yes/no> (*yes*) : retour ou non au rsh normal en cas d'échec de connexion ssh

Chaque ensemble de paramètres à redéfinir est précédé d'une ligne comportant le nom d'une machine ou d'un groupe de machines (utilisation de * et ?)

Exemple :

```
Host resto.imag.fr
        User martinet
        Compression no
        IdentityFile ~/.ssh/resto-identity
Host *.grenet.fr
        FallBackToRsh no
        IdentityFile ~/.ssh/grenet-identity
```

Host *
Compression yes
Compression level 9
FallBackToRsh yes

Le particulier l'emporte sur le général (ex : compression pour tous sauf pour resto…)

Connexion à partir de plusieurs machines sur un site cible

Un utilisateur veut pouvoir accéder à un même site serveur distant (ex: bistrot) depuis plusieurs machines clientes (ex: chablis, sancerre).

- Sur chablis et sancerre, créer des clés propres (l'utilisation de passphrases différentes est recommandée),
- transférer la clé publique de chaque machine dans ~/.ssh/authorized_keys de bistrot (une ligne par machine).

exemple:

scp ~/.ssh/identity.pub bistrot.site.fr:.ssh/temp.pub
<martinet>@bistrot.site.fr's password: XXXXXXXXX
ssf bistrot.site.fr 'cat ~/.ssh/temp.pub >> ~/.ssh/authorized_keys'
<martinet>@bistrot.site.fr's password: XXXXXXXXX

Le fichier authorized_keys de la machine serveur cible contiendra autant de lignes que de machines clientes susceptibles de s'y connecter.

Connexion à partir d'une machine sur plusieurs sites cibles

Ici l'utilisateur veut avoir accès depuis la même machine cliente (ex: bistrot) sur plusieurs machines serveurs cibles (ex: chablis, sancerre...).

- Sur le client (bistrot), créer des clés avec des passphrases différentes pour chaque serveur cible. Les clés seront sauvées dans des fichiers distincts des fichiers identity et identity.pub utilisés par défaut,

bistrot (167) ssh-keygen
Initializing random number generator...
Generating p: .....++ (distance 596)
Generating q: .....++ (distance 1130)
Computing the keys...
Testing the keys...
Key generation complete.
Enter file in which to save the key (/home/martinet/.ssh/identity): ***/home/martinet/.ssh/identity-chablis***
Enter passphrase: *xxxx xxxx xx xxxxxx*
Enter the same passphrase again: *xxxx xxxx xx xxxxxx*
Your identification has been saved in /home/martinet/.ssh/identity-chablis.
Your public key is:
1024 35 127929861606…….310213978397 martinet@bistrot
Your public key has been saved in /home/martinet/.ssh/identity-chablis.pub

- Recopier la clé publique créée dans chaque fichier ~/.ssh/authorized_keys des machines serveurs cibles (chablis, sancerre, …).
- Mettre à jour le fichier ~/.ssh/config sur bistrot.

Host chablis

IdentityFile ~/.ssh/identity-chablis
Host sancerre
IdentityFile ~/.ssh/indentity-sancerre

Une autre méthode est la gestion des clés en mémoire grâce à la commande ssh-agent qui est définie ci-dessous.

Gestion des clés par ssh-agent

ssh-agent gère les clés en mémoire pour la durée d'une session. Pour chacun des sites serveurs distants connectés par SSH, la passphrase du site n'est tapée qu'une fois (et non à chaque appel de commande).

ssh-agent se lance en associant par exemple le nom d'un shell (csh, tcsh…), ce qui créera un processus sur la machine source.

exemple : **ssh-agent tcsh**

Ensuite on indique au système à l'aide de la commande ssh-add le nom des clés privées que l'on veut utiliser et on entre la passphrase adéquate pour chacune d'elles.

Exemple (si pas de paramètre, utilisation de ~/.ssh/identity) :

bistrot (11) **ssh-add /home/martinet/ssh/identity-resto**
Need passphrase for /home/martinet/.ssh/identity-resto (martinet@bistrot).
Enter passphrase: xxxx xxxxx xx xxxx
Identity added: /home/martinet/.ssh/identity-resto (martinet@bistrot)
bistrot(12) **ssf resto.imag.fr uname -a**
SunOS resto.imag.fr 5.7 Generic_106541-07 sun4u sparc SUNW,Ultra-30

Exemple de session complexe

bistrot 10 >**eval `ssh-agent`** #permet l'usage du process courant
Agent pid 9191
bistrot 11 > **ssh-add .ssh/identity-chablis**
Need passphrase for /home/martinet/.ssh/identity-chablis (martinet@bistrot)
Enter passphrase: xxxxxx xxxx xx xxxx
Identity added: /home/martinet/.ssh/identity-chablis (martinet@bistrot)
bistrot 12 > **ssh-add .ssh/identity-sancerre**
Need passphrase for /home/martinet/.ssh/identity-sancerre (martinet@bistrot)
Enter passphrase: yyyy y yyyyy yyyyyyyy
Identity added: /home/martinet/.ssh/identity-sancerre (martinet@bistrot)
bistrot 13 > **scp chablis:.netscape/bookmarks.html book.tmp**
bistrot 14 > **scp book.tmp sancerre:.netscape/bookmarks.html**
bistrot 15 > **ssf -f sancerre netscape**
bistrot 16 > **ssf -f chablis xterm**
bistrot 17 > etc ...

Les clients PC/Mac

Client PC : TTSSF

C'est le seul client utilisable d'après la législation française. Développé à l'IN2P3, il a fait l'objet d'une déclaration auprès du SCSSI.
Il fonctionne sur les plateformes Windows 9x et NT. C'est en fait une extension sécurisée de ttermp (TeraTerm Pro 2.3) qui est un client telnet gratuit.

La version 128 bits (ttssf-1.4.4) est disponible à :
http://perso.univ-rennes1.fr/bernard.perrot/ssf

Il permet :

- la connexion de type telnet
- la redirection X-Window si un émulateur X est actif sur le PC
- la redirection de n'importe quel service TCP (par exemple pop ou imap)

Il ne permet pas le transfert de fichiers. Pour cela, il faut installer un produit supplémentaire FTP acceptant le mode PASV comme LeechFTP par exemple (*http://stud.fh-heilbronn.de/~jdebis/leechftp*) que l'on pourra rediriger dans le canal crypté.

Client Mac :

Il n'existe pas à l'heure actuelle de client Mac répondant à la législation (clé 2^{128} max.).
Par contre, si on sort du territoire français, il sera possible d'utiliser soit une extension sécurisée de NiftyTelnet, soit d'utiliser MindTerm qui est un client universel puisque développé en Java.

NiftyTelnet (étendu)

Réservé à la plateforme MacOS, c'est une adaptation sécurisée du produit (telnet gratuit) de NiftyTelnet, qui fonctionne avec un serveur en protocole v1.

NiftyTelnet1.1sshr3b est disponible à :
http://www.lysator.liu.se/~jonasw/freeware/niftyssh/

Il permet :

- la connexion telnet,
- le transfert de fichiers (scp).

Il existe une très bonne documentation sur le produit à l'URL :
http://www.ufrp7.math.jussieu.fr/doc/ssh/ssh.html

MindTerm

Développé en Java, il peut tourner sur toutes les plateformes supportant un Kit Java plus ou moins récent suivant les types d'OS.
Il fonctionne sous : Linux, Unix (Solaris, Aix, HP/UX, IRIX…), FreeBSD, NetBSD, MacOS(>8.1) Win95/98/NT/2000…

Actuellement disponible à :
http://www.appgate.org/products/mindterm

Il permet :

- la connexion telnet,
- le transfert de fichiers (scp et vrai tunneling ftp),
- la redirection X-Window,
- la redirection de n'importe quel service TCP (par exemple pop ou imap)

Utilisation de clés RSA sur les clients

Il n'existe pas de ssh-keygen sur les clients PC ou Mac (sauf sur MacSSH, mais le produit est en SSH v2 et son usage n'est pas possible avec un serveur SSF). On ne pourra donc pas générer des clés sur les clients. Si on veut pouvoir bénéficier du service d'authentification par clés RSA, il faut :

- générer les clés RSA sur une machine Unix (les clés étant indépendantes du host où elles ont été générées).
- ramener les clés sur le PC, ou le Mac, par un scp et les ranger dans le répertoire adéquat

TTSSF :

- Fichier dans le répertoire principal Ttermpro
- Dans le menu Setup/SSF Authentification, indiquer le nom du fichier contenant la clé privée

NiftyTelnet :

- Indiquer le nom du fichier contenant la clé privée.

Récupération de courrier

Une des applications les plus intéressantes de SSH est la possibilité de récupérer le courrier depuis un PC sur un serveur distant et ce, de façon sécurisée.

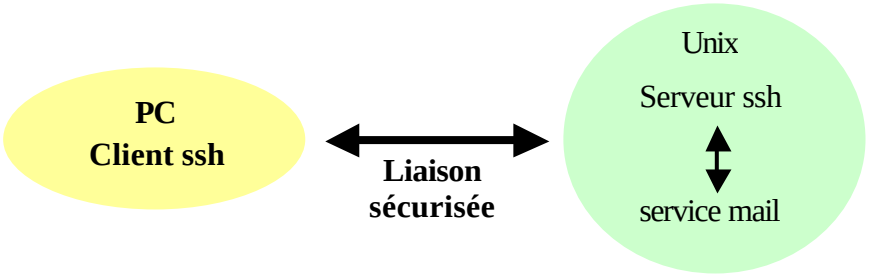
Principe :

On établit une liaison sécurisée entre le PC et une machine Unix et on redirige la connexion vers le serveur de mail (par POP ou IMAP) à travers le canal sécurisé.

Le serveur de Mail croit que le courrier est rapatrié depuis la machine SSH Unix, le logiciel de mail du PC croit que le PC est son propre serveur de mail (port local redirigé sur le serveur à travers le tunnel SSH).

Cette architecture nécessite que l'on ait une confiance totale dans la partie réseau entre les serveurs Unix (serveur SSH et serveur Mail). Ceci peut être réalisé si on considère que le PC est n'importe où sur le réseau (même en région hostile) et que les serveurs Unix sont eux sur un réseau local protégé.

Le serveur de mail peut être le serveur ssh, dans ce cas, il n'y aura pas de mot de passe en clair sur le réseau, la connexion finale se faisant en interne.



Configuration TTSSF

- configurer "Setup/ssf forwarding"
- ajouter : redirection port local 110 vers serveur-pop.site: port 110
- ajouter : redirection port local 25 vers serveur-smtp.site: port 25
- configurer "Setup/Tcp/IP"
- entrer les noms de machines auxquelles on se connecte avec port 22 (ssh) et non 23 (telnet)
- configurer localhost comme serveur de courrier entrant et sortant sur le logiciel de courrier du PC

Utilisation

- lancer ttssf et se connecter (port 22)
- lancer la récupération de courrier

Redirection X-Window

Elle ne sera possible que si le client PC possède un serveur X local.

Configuration TTSSF

- configurer "Setup/ssf forwarding"
- activer "afficher les clients X distants sur le serveur X local"
- configurer "Setup/Tcp/IP"
- entrer les noms de machines auxquelles on se connecte avec port 22 (ssh) et non 23 (telnet)

Utilisation

- lancer le serveur X local,
- lancer ttssf et se connecter (port 22),
- dans la fenêtre obtenue, sans changer le DISPLAY, lancer les applications X. Le plus simple étant de lancer un xterm, puis les autres applications dans ce xterm.

Liens intéressants

SSH/SSF

http://perso.univ-rennes1.fr/bernard.perrot/ssf

SSH Usage

http://www.medicis.polytechnique.fr/doc/medicis/siret
http://polywww.in2p3/informatique/doc/ssf

Forwarding X11 et mail

ftp://ftp.lami.univ-evry.fr/pub/specif/petit/Documentation/doc-ssf-X11.txt
ftp://ftp.lami.univ-evry.fr/pub/specif/petit/Documentation/doc-ssf-mail.txt

Documentation Jussieu
NiftyTelnet

http://www.ufrp7.math.jussieu.fr/doc/ssh/ssh.html

SSH-SSL-S/Key…

http://www.math.jussieu.fr/informatique/acces.html

POP, IMAP, SMTP, FTP sur SSH

http://www.math.jussieu.fr/informatique/tunnel_ssh.html

Tutorial Chiffrement de JRES99

Merci à Joël Marchand et Bernard Perrot pour leurs remarques et leurs corrections !