

Procédure d'Implémentation et de Sécurisation du Protocole SSH

Objectif

L'objectif principal de cette procédure technique est de détailler l'établissement, la configuration et la sécurisation des communications à distance via le protocole **SSH (Secure Shell)**. Cette procédure est essentielle pour garantir une administration et une maintenance sécurisées des systèmes critiques, en conformité avec les standards de sécurité en entreprise.

La procédure vise spécifiquement à :

- Installer les composants client et serveur OpenSSH sur les systèmes d'exploitation Linux (Debian 11 *Bullseye*).
- Assurer la connectivité et l'interopérabilité entre les systèmes d'administration (postes clients) et les serveurs critiques.
- Déployer l'**authentification par clé publique/privée (identité numérique)** pour renforcer la sécurité de l'accès et éliminer la dépendance aux mots de passe seuls.
- Sécuriser l'accès aux clés privées en utilisant la **Passphrase et l'agent SSH** pour une gestion des identités en mémoire.
- Optimiser l'expérience utilisateur et l'administration des connexions via la configuration client (`.ssh/config`).
- Établir les bases pour une sécurité périphérique avancée (notamment par l'exploration de la double authentification et des ports de communication).

Contexte de la Réalisation



Le protocole **Secure Shell (SSH)** est un standard de chiffrement de bout en bout indispensable pour l'administration des serveurs dans tout environnement de production moderne. Son implémentation remplace les protocoles non sécurisés comme Telnet et assure la confidentialité et l'intégrité des commandes exécutées à distance.



Cette procédure d'implémentation est réalisée dans un contexte d'infrastructure réseau comprenant plusieurs machines, simulées pour représenter un environnement de serveurs et de postes d'administration hétérogènes. Cette infrastructure de démonstration inclut :

- **Systèmes Serveurs (Debian 11)** : Désignés comme machines cibles (`sshserver`), elles hébergent l'instance OpenSSH Server et nécessitent des accès distants sécurisés.
- **Postes d'Administration (Clients Linux et Windows)** : Utilisés pour initier les sessions sécurisées (`sshclient`, postes Windows Server et postes physiques de l'administrateur).

L'accent est mis sur le déploiement de l'**authentification sans mot de passe** (basée sur la clé) pour les administrateurs (`userclient` et `userserver`), une pratique hautement recommandée pour l'automatisation et pour l'élevation du niveau de sécurité de l'ensemble de l'infrastructure d'accès distant.

De plus, cette réalisation intègre la vérification des **ports réseau** et la gestion des **services bien connus** pour garantir le bon fonctionnement des communications et l'identification des processus d'écoute (Port 22 pour SSH).



A- Mise en place de l'infrastructure

- 1- Mettre en place deux machine Debian 11 nom de code bullseye et une machine windows2019
- 2- Mettre à jour les deux machines debian11
- 3- Changer le nom des deux machines pour sshclient et sshserver
- 4- Installer le paquet openssh-client et openssh-server
- 5- Créez un utilisateur **userserver** sur le serveur **sshserver** Et **userclient** sur le serveur **sshclient**
- 6- Exploration des fichiers de configurations sur le serveur ssh et le client ssh
- 7- Connexion au :
 - a- Serveur ssh
 - b- Serveur client
 - c- Serveur Windows 2019
 - d- Votre machine physique Windows 10

B- Crédation d'une identité NUMERIQUE pour les comptes utilisateurs ssh en créant une paire de clés asymétrique

- 1- Crédation d'une identité NUMERIQUE pour userclient en créant une paire de clés asymétrique sur sshclient
- 2- J'envoie la clé publique à userserver sur sshserver
- 3- Connexion au serveur ssh avec la Passphrase
- 4- Passphrase et agent ssh

C- Passphrase et agent SSH

- a- Crédation d'une nouvelle instance bash

- b- Charger la clé dans la mémoire de l'agent
- c- Connexion ssh en utilisant l'agent ssh avec la Passphrase en mémoire
- d- Autres options

D- Créations du fichier config

- a- Créer le fichier config dans .ssh
- b- Modifier le fichier config
- c- Modifier le fichier hosts

E- La double Authentification google

F- Création d'une bannière

G- Connection ssh avec une solution mobile

Introduction

(SSH) Secure Shell est un protocole de communication sécurisé. Ce protocole de connexion exige un échange de clés de chiffrement pour toutes connexions, tous les segments TCP sont **authentifiés et chiffrés**. Toutes attaque de types **man-in-the-middle (MITM)**

Le protocole SSH a été conçu pour remplacer les différents protocoles non chiffrés comme rlogin, telnet, rcp et rsh.

OpenSSH utilise la **cryptographie*** asymétrique comme mécanisme d'authentification. OpenSSH gère les clés RSA, les clés DSA et les clés DSA basées sur les courbes elliptiques. OpenSSH reconnaît aussi les certificats X509 et les fichiers au format PKCS#12.

*La cryptographie sert principalement 3 objectifs :

- a- La confidentialité
 - b- L'authentification (identification des partenaires, contrôle d'accès aux ressources et données, non répudiation de l'origine de l'information)
 - c- L'intégrité des données (données non altérées accidentellement ou frauduleusement lors du transfert ou sur place)
- ^{2&}

OpenSSh est, on l'a dit, une alternative à SSH. Il est également tout indiqué pour remplacer La principale différence entre OpenSSH et SSH est bien entendu le fait qu'OpenSSH soit libre. Il y a aussi quelques petites différences de configuration.

Modèle client/serveur

- a- Comme tout protocole qui se respecte, OpenSSH est composé d'un serveur et d'un client :
- b- Le serveur (sshd) écoute sur un port (22) par default. Pour une connection entrante, il vérifie que c'est bien un client ssh qui est à l'autre bout, demande à l'utilisateur de s'identifier via paire de clés asymétriques ou un mot de passe. Dans le premier cas, le serveur va lire un fichier de clés publiques (~/.ssh/authorized_keys) dans le répertoire home de l'user annoncé. S'il y trouve une clé ayant le même nom que celle proposée par le client, il va les vérifier par l'envoi d'un 'challenge' crypté avec la clé publique.

Seule la clé privée pourra le décrypter, et elle renverra le challenge décrypté, prouvant que le client connaît la clé privée, mais sans jamais la dévoiler.

Dans le deuxième cas, le serveur vérifie que le mot de passe et le nom correspondent à un utilisateur de la machine. Cette méthode n'est utilisée que si la méthode par paire de clés a échoué. Le mot de passe étant lui-même crypté durant le transfert, il n'est pas possible de l'intercepter en clair. Si le client est autorisé, la machine locale est disponible en interactif, avec les droits de la personne entrée.

- c- Le client sert à se connecter au serveur. C'est lui qui choisit les algorithmes (et par là, la version du protocole). Si le serveur ne reconnaît pas l'algorithme demandé, c'est au client d'en proposer un autre.
- d- Certaines options peuvent être configurées des deux côtés, serveur et client. La règle qui prévaut alors est celle du 'Tout ce qui n'est pas interdit est autorisé'. Par exemple, le client autorisera les applications X-window à passer par le tunnel ssh, que si la configuration du serveur ne l'interdit pas expressément.

Mise en place

- 1- Mettre en place deux machines Debian 11 nom de code bullseye
Cloner deux machines à partir de la machine d'origine

- 2- Mettre à jour les deux machines

```
root💀debian:~# apt update
root💀debian:~# apt upgrade
```

- 3- Changer le nom des deux machines pour sshclient et sshserver

```
root💀debian:~# hostnamectl set-hostname sshclient
root💀debian:~# hostnamectl set-hostname sshserver
```

On vérifiera que les noms ont changé dans le fichier hostname

```
root💀debian:~# vim /etc/hostname
```

sshserver

Ou avec la commande hostnamectl

```
root💀sshserver:~# hostnamectl
      Static hostname: sshserver
      Icon name: computer-vm
```

- 4- Installer le paquet :

- OpenSSH-client sur le serveur sshclient
- OpenSSH-server sur le serveur sshserver

- a- Sur la machine sshserver on installe le paquet openssh-server

```
root💀sshserver:~# apt install openssh-server -y
```

Vérification : on constate que le paquet openssh est installé

```

root@sshclient:~# dpkg -l openssh*
Souhait=inconnu/Installé/suppRimé/Purgé/H=à garder
| État=Non/Installé/fichier-Config/dépaqueté/échec-conFig/H=semi-installé/W=attend-traitement-déclenchements
|/ Err?=(aucune)/besoin Réinstallation (État,Err: majuscule=mauvais)
||| Nom Version Architecture Description
+++-+-----+-----+-----+-----+
ii  openssh-client  1:8.4p1-5  amd64   secure shell (SSH) client, for secure access to remote machines
un  openssh-server  <aucune>    <aucune> (aucune description n'est disponible)
ii  openssh-sftp-server 1:8.4p1-5  amd64   secure shell (SSH) sftp server module, for SFTP access from remote ma
un  openssh-sk-helper <aucune>    <aucune> (aucune description n'est disponible)
lines 1-9/9 (END)

```

- b- Sur la machine sshclient on installe le paquet openssh-client

```
root@sshclient:~# apt install openssh-client -y
```

Vérification : on constate que le paquet openssh est installé

```

root@sshclient:~# dpkg -l openssh*
Souhait=inconnu/Installé/suppRimé/Purgé/H=à garder
| État=Non/Installé/fichier-Config/dépaqueté/échec-conFig/H=semi-installé/W=attend-traitement-déclenchements
|/ Err?=(aucune)/besoin Réinstallation (État,Err: majuscule=mauvais)
||| Nom Version Architecture Description
+++-+-----+-----+-----+
ii  openssh-client  1:8.4p1-5  amd64   secure shell (SSH) client, for secure access to remote machines
un  openssh-server  <aucune>    <aucune> (aucune description n'est disponible)
un  openssh-sk-helper <aucune>    <aucune> (aucune description n'est disponible)
root@sshclient:~#

```

Vérification

```

root@sshclient:~# ssh -V
OpenSSH_8.4p1 Debian-5, OpenSSL 1.1.1k 25 Mar 2021
root@sshclient:~# dpkg -l libssl*
Souhait=inconnu/Installé/suppRimé/Purgé/H=à garder
| État=Non/Installé/fichier-Config/dépaqueté/échec-conFig/H=semi-installé/W=atten>
|/ Err?=(aucune)/besoin Réinstallation (État,Err: majuscule=mauvais)
||| Nom Version Architecture Description
+++-+-----+-----+-----+
ii  libssl1.1:amd64 1.1.1k-1+deb11u1 amd64   Secure Sockets Layer toolkit ->
lines 1-6/6 (END)

```

Les différentes commandes liées à ssh

```

root@sshserver2:/usr/bin# ls *ssh*
ssh  ssh-add  ssh-agent  ssh-argv0  ssh-copy-id  ssh-keygen  ssh-keyscan

```

```
root@sshserver:~# dpkg -s openssh-server |
```

```
root@sshserver:~# apt show openssh-server|
```

On vérifie si le service ssh est démarré

```
root@sshserver2:~# /etc/init.d/ssh status|
```

Ou

```
root@sshserver2:~# systemctl status ssh|
```

ou

```

root@sshserver:~# service ssh status
● ssh.service - OpenBSD Secure Shell server
  Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
  Active: active (running) since Sat 2021-09-25 21:15:40 CEST; 18s ago
    Docs: man:sshd(8)
          man:sshd_config(5)
  Process: 2603 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
 Main PID: 2604 (sshd)
   Tasks: 1 (limit: 2303)
  Memory: 1.1M
     CPU: 24ms
    CGroup: /system.slice/ssh.service
            └─2604 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups

sept. 25 21:15:40 sshserver systemd[1]: Starting OpenBSD Secure Shell server...
sept. 25 21:15:40 sshserver sshd[2604]: Server listening on 0.0.0.0 port 22.
sept. 25 21:15:40 sshserver sshd[2604]: Server listening on :: port 22.
sept. 25 21:15:40 sshserver systemd[1]: Started OpenBSD Secure Shell server.

```

5- Créez un utilisateur **userserver** sur le serveur **sshserver** Et **userclient** sur le serveur **sshclient**

6- Les fichiers de configuration

a- Sur le serveur ssh

```

root@sshserver:~# ls -l /etc/ssh
total 608
-rw-r--r-- 1 root root 577771 13 mars 2021 moduli
-rw-r--r-- 1 root root 1650 13 mars 2021 ssh_config
drwxr-xr-x 2 root root 4096 13 mars 2021 ssh_config.d
-rw-r--r-- 1 root root 3289 13 mars 2021 sshd_config
drwxr-xr-x 2 root root 4096 13 mars 2021 sshd_config.d
-rw----- 1 root root 505 25 sept. 21:15 ssh_host_ecdsa_key
-rw-r--r-- 1 root root 176 25 sept. 21:15 ssh_host_ecdsa_key.pub
-rw----- 1 root root 411 25 sept. 21:15 ssh_host_ed25519_key
-rw-r--r-- 1 root root 96 25 sept. 21:15 ssh_host_ed25519_key.pub
-rw----- 1 root root 2602 25 sept. 21:15 ssh_host_rsa_key
-rw-r--r-- 1 root root 568 25 sept. 21:15 ssh_host_rsa_key.pub

```

Moduli : Il n'est pas fait pour être utilisé directement par l'utilisateur c'est un fichier qui est géré par ssh on peut le générer avec la commande ssh-keygen c'est un fichier qui va contenir des données utiles quand on va faire un échange avec la clé diffie-hellman

La clé diffi-hellman permet à deux correspondant de concevoir une clé symétrique en échangeant une quantité de donnée qui ne permet pas de calculer cette clé

Ssh_config : la configuration du client permet d'identifier les options configurées pour ssh

Sshd_config :

```

#AddressFamily any
#ListenAddress 0.0.0.0
#ListenAddress ::

#HostKey /etc/ssh/ssh_host_rsa_key
#HostKey /etc/ssh/ssh_host_ecdsa_key
#HostKey /etc/ssh/ssh_host_ed25519_key

```

```
#LoginGraceTime 2m
#PermitRootLogin prohibit-password
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10

#PubkeyAuthentication yes
```

Paire de clé qui identifiée

Rsa basé sur la factorisation des nombres premiers
Ecdsa et ed25519 basé sur les courbes elliptiques

b- Sur le serveur client

On remarque qu'il existe juste un fichier et un répertoire

```
root@sshclient:~# ls /etc/ssh
ssh_config  ssh_config.d
root@sshclient:~#
```

7- Connexion au :

a- Serveur ssh

On se connecte toujours avec les comptes hébergés sur le serveur ssh et non les comptes locaux.

Depuis le client sshclient on va établir une connexion ssh au serveur ssh ; il existe deux façons,

```
#ssh -l <nom d'utilisateur sur le serveur ssh> <adresse ip du serveur ssh>
```

```
root@sshclient:/etc/ssh# ssh -l userserver 192.168.44.152
The authenticity of host '192.168.44.152 (192.168.44.152)' can't be established.
ECDSA key fingerprint is SHA256:izZkTe4Q0Low872yX0Zf8e0S0W05behllq9fIj1S11g.
Are you sure you want to continue connecting (yes/no/[fingerprint])? y|
```

Ou

```
#ssh nom_utilisateur@adresse ip
```

exemple

```
root@sshclient:/etc/ssh# ssh userserver@192.168.44.152
The authenticity of host '192.168.44.152 (192.168.44.152)' can't be established.
ECDSA key fingerprint is SHA256:izZkTe4Q0Low872yX0Zf8e0S0W05behllq9fIj1S11g.
Are you sure you want to continue connecting (yes/no/[fingerprint])? |
```

Le serveur ssh envoi sa clé publique ainsi que l'empreinte de cette clé

On peut vérifier cette empreinte sur le serveur ssh avant d'accepter cette clé publique, après comparaison on remarque que les deux empreintes correspondent

```
# ssh-keygen -l -E sha256 -f ssh_host_ecdsa_key.pub
```

ou

```

root@sshserver:/etc/ssh# ssh-keygen -lv -f ssh_host_ecdsa_key.pub
256 SHA256:izZkTe4Q0Low872yX0Zf8e0S0W05behllq9fIj1S11g root@sshserver (ECDSA)
+---[ECDSA 256]---+
|   .. |
|   .. |
|   ... . +E|
| + . =   o.**|
| = o+ S   . *oX|
| oo.= o . + O.|
|   +. = . o * +|
|   ...+   o =.|
|   .+   ...|
+---[SHA256]---+

```

on trouve cette clé dans le fichier `/root/.ssh/known_hosts` on peut afficher cette clé

```

root@sshclient:~/ssh# cat known_hosts
|1|/dkwRRR3fLLuNpbFHd3Agq0kiPc=|h0ifiUKMM0sHr5/fskD66sVJtAY= ecdsa-sha2-nistp256 AA
AAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmjlzdHAyNTYAAABBBmDcokbIMrWStT4pMdx fzjJet+xkindg
MUbfqLVE078ZdAZGDNF7d+YqzLoEGWqTE231QRmTRC/4VJUCuQNYzI=
root@sshclient:~/ssh#

```

Pour effacer une clé

```

root@debian:~# ssh-keygen -f "/root/.ssh/known_hosts" -R "192.168.44.135"

```

On remarque que le prompt a changé on est bien sur le serveur ssh (**sshserver**)

il faut noter qu'on ne peut pas se connecter par défaut en ssh avec le compte root ; pour des raisons de sécurité le compte est désactivé dans le fichier de configuration `sshd_config`

```

#LoginGraceTime 2m
#PermitRootLogin prohibit-password

root@sshclient:/etc/ssh# ssh root@192.168.44.152
The authenticity of host '192.168.44.152 (192.168.44.152)' can't be established.
ECDSA key fingerprint is SHA256:izZkTe4Q0Low872yX0Zf8e0S0W05behllq9fIj1S11g.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.44.152' (ECDSA) to the list of known hosts.
root@192.168.44.152's password:
Permission denied, please try again.

```

b- Serveur client

Depuis le serveur ssh on va essayer de se connecter sur le client ssh, la connexion est refusé sur un client ssh on ne peut pas se connecter si le paquet openssh-server n'est pas installé.

```

root@sshserver:~# ssh userclient@192.168.44.158
ssh: connect to host 192.168.44.158 port 22: Connection refused

```

c- Serveur Windows 2019, 2022 et Windows 10

Installer OpenSSH avec les Paramètres Windows

Les deux composants OpenSSH client et serveur doivent être installés doivent être installés. Pour installer les composants OpenSSH :

1. Ouvrez **Paramètres**, sélectionnez **Applications > Applications et fonctionnalités**, puis **Fonctionnalités facultatives**.

- Parcourez la liste pour voir si OpenSSH est déjà installé. Si ce n'est pas le cas, sélectionnez **Ajouter une fonctionnalité** en haut de la page, puis :
 - Recherchez **OpenSSH Client** et cliquez sur **Installer**.
 - Recherchez **OpenSSH Server** et cliquez sur **Installer**.

Une fois l'installation terminée, revenez à **Applications > Applications et fonctionnalités et Fonctionnalités facultatives**. Vous devriez voir OpenSSH dans la liste.

	Client OpenSSH	5,05 Mo
	Serveur OpenSSH	4,71 Mo

On peut installer ssh en ligne de commande sur un power shell
Installation d'OpenSSH en ligne de commande

Ouvrez le PowerShell en tant qu'administrateur. Pour s'assurer qu'OpenSSH est disponible, exécutez l'applet de commande suivante :[Get-WindowsCapability -Online | ? Name -like 'OpenSSH*'!](#)

La sortie suivante doit être retournée si aucun n'est déjà installé :

```
Name : OpenSSH.Client~~~~0.0.1.0
State : NotPresent
```

```
Name : OpenSSH.Server~~~~0.0.1.0
State : NotPresent
```

Installation des composants serveur ou client :

```
# Install the OpenSSH Client
Add-WindowsCapability -Online -Name OpenSSH.Client~~~~0.0.1.0

# Install the OpenSSH Server
Add-WindowsCapability -Online -Name OpenSSH.Server~~~~0.0.1.0
```

Quel que soit le composant, la sortie suivante devrait être retournée :

Démarrer et configurer OpenSSH Server

Pour la première utilisation de ssh, ouvrez PowerShell en tant qu'administrateur, il faut exécuter les commandes suivantes pour démarre

```
# démarrer le service sshd
Start-Service sshd

# configurer un démarrage automatique du service sshd:
Set-Service -Name sshd -StartupType 'Automatic'
```

Il faut désactiver le firewall ou il faut le configurer pour le configurer il faut suivre les etapes suivantes

```
# autorisez ssh sur le firewall.  
Get-NetFirewallRule -Name *ssh*
```

```
# la règle suivante doit etre activer "OpenSSH-Server-In-TCP", si cette regle n'existe pas il faut la créer
```

```
New-NetFirewallRule -Name sshd -DisplayName 'OpenSSH Server (sshd)' -Enabled True -  
Direction Inbound -Protocol TCP -Action Allow -LocalPort 22
```

On vérifie l'installation o remarque les même fichiers que sur linux

```
PS C:\> cd .\ProgramData\ssh\  
PS C:\ProgramData\ssh> dir  
  
 Répertoire : C:\ProgramData\ssh  
  
Mode          LastWriteTime      Length Name  
----          -----          ----  --  
d---- 26/09/2021    11:21          logs  
-a---- 26/09/2021    11:21          6 sshd.pid  
-a---- 05/09/2018   15:07          2253 sshd_config  
-a---- 26/09/2021    11:21          668 ssh_host_dsa_key  
-a---- 26/09/2021    11:21          622 ssh_host_dsa_key.pub  
-a---- 26/09/2021    11:21          227 ssh_host_ecdsa_key  
-a---- 26/09/2021    11:21          194 ssh_host_ecdsa_key.pub  
-a---- 26/09/2021    11:21          432 ssh_host_ed25519_key  
-a---- 26/09/2021    11:21          114 ssh_host_ed25519_key.pub  
-a---- 26/09/2021    11:21          1675 ssh_host_rsa_key  
-a---- 26/09/2021    11:21          414 ssh_host_rsa_key.pub
```

Dans system32 le dossier openssh regroupe toutes les commandes ssh

```
PS C:\Windows\System32\OpenSSH> ls  
  
 Répertoire : C:\Windows\System32\OpenSSH  
  
Mode          LastWriteTime      Length Name  
----          -----          ----  --  
-a---- 05/09/2018   17:07          322560 scp.exe  
-a---- 05/09/2018   15:07          322048 sftp-server.exe  
-a---- 05/09/2018   17:07          390144 sftp.exe  
-a---- 05/09/2018   17:07          491520 ssh-add.exe  
-a---- 05/09/2018   17:07          384512 ssh-agent.exe  
-a---- 05/09/2018   17:07          637952 ssh-keygen.exe  
-a---- 05/09/2018   17:07          530432 ssh-keyscan.exe  
-a---- 05/09/2018   15:07          149504 ssh-shellhost.exe  
-a---- 05/09/2018   17:07          882688 ssh.exe  
-a---- 05/09/2018   15:07          974336 sshd.exe  
-a---- 05/09/2018   15:07          2253 sshd_config_default
```

Maintenant on va tenter une connexion sur le serveur ssh Windows 2019 depuis le serveur sshclient

Avant il faut créer un utilisateur **userw19** avec **Azerty123** comme mot de passe

Nom	Nom complet	Description
Administrator		Compte d'utilisateur d'administra...
bendi		
DefaultAcco...		Compte utilisateur géré par le syst...
Invité		Compte d'utilisateur invité
sshd	sshd	
userw19	userw19	

```
root@sshclient:~# ssh userw19@192.168.44.145  
The authenticity of host '192.168.44.145 (192.168.44.145)' can't be established.  
ECDSA key fingerprint is SHA256:D4axPeJ5a0v10T0+hIQnJUVZotXaJxYFsdWbvcA0hXQ.  
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
```

On tombe sur le prompt de notre machine windows sur une console dos.

On peut lancer une console powershell à partir de cet endroit :

```
PS C:\Windows\System32\WindowsPowerShell\v1.0> powershell
```

```
userw19@SSH-W-2019 C:\Users\userw19>
```

On peut vérifier comme auparavant l'envoie de la clé publique par le serveur ssh windows au serveur linux sshclient

Sur sshclient

```
root@sshclient:~/ssh# cat known_hosts
|1|/dkwRRR3fLLuNpbFhd3Agq0kiPc=|h0ifiUKMM0sHr5/fskD66sVJtAY= ecdsa-sha2-nistp256 AA
AAE2VjZHNhLXNoYTItbmlzdHAYNTYAAAAIBmlzdHAYNTYAAABBBmDcokbIMrWStT4pMdx fzjJet+xkindg
MUbfqlVE078ZdAZGDNF7d+YqzLoEGWqTE2310RmTRC/4VJCUcONYzT=
|1|iAnhwgWTAzd4hKv06uLMst0u+kY=|Vz8CyJFgZF4n8RvT6UjNPuweDXk= ecdsa-sha2-nistp256 AA
AAE2VjZHNhLXNoYTItbmlzdHAYNTYAAAAIBmlzdHAYNTYAAABBBANG/HxgTba9d/+4POe+vMDGuxnQWcxrX
G5w0HF3gK1AEuGwWksXddyKVT62bWaGs6VPzoeY5LbRqgduos/FYhY=
root@sshclient:~/ssh#
```

Sur w2019

```
PS C:\ProgramData\ssh> cat .\ssh_host_ecdsa_key.pub
ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAYNTYAAAAIBmlzdHAYNTYAAABBBANG/HxgTba9d/+4POe+
vMDGuxnQWcxrXG5w0HF3gK1AEuGwWksXddyKVT62bWaGs6VPzoeY5LbRqgduos/FYhY= autorite nt\système@ss
h-w-2019
PS C:\ProgramData\ssh>
```

On remarque que c'est la même clé

On peut tenter d'établir une connexion de Windows 2019 vers un serveur linux

On ouvre une connexion ssh sur un terminal PowerShell ou dos

```
Windows PowerShell
PS C:\> ssh userserver@192.168.44.152
The authenticity of host '192.168.44.152 (192.168.44.152)' can't be established.
ECDSA key fingerprint is SHA256:PuHV+MhVrRqd1E2AzJ2jDBOThrkUXpQwbEpPKrWniAU.
Are you sure you want to continue connecting (yes/no)?
```

Yes pour confirmer puis on tombe sur le terminal du serveur ssh sur linux on peut vérifier aisément

Que le serveur ssh a envoyé sa clé publique

```
userserver@sshserver :/$
```

Ce PC > Disque local (C:) > Utilisateurs > user > .ssh				Rechercher dans : .ssh
Nom	Modifié le	Type	Taille	
known_hosts	26/09/2021 13:41	Fichier	1 Ko	

- c- Votre machine physique Windows 10
- Idem que Windows 2019

B- Création d'une identité NUMERIQUE pour les comptes utilisateurs ssh en créant une paire de clés asymétrique

- 1- Création d'une identité NUMERIQUE pour userclient en créant une paire de clés asymétrique sur sshclient

Création d'une paire de clés asymétrique la clé **privée** est protégé par un **mot de passe**
La syntaxe pour créer cette paire de clés asymétriques

Accès à distance

Création de clés

- Clés asymétriques
 - RSA ou DSA
- Création des fichiers
 - id_rsa (privé)
 - id_rsa.pub (publique)

```
#ssh-keygen -t algorth -b la taille de laclé -f le chemin ou la clé sera stokée
```

```
#ssh-keygen -t rsa -b 1024 -f /home/userclient/uc_rsa
```

```
root@sshclient:~# ssh-keygen -t rsa -b 1024 -f /home/userclient/uc_rsa
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/userclient/uc_rsa
Your public key has been saved in /home/userclient/uc_rsa.pub
The key fingerprint is:
SHA256:EZIqNtPh2rpv4Wg5ytNSaQb0zQFa04FFaLGFNhk0qAI root@sshclient
The key's randomart image is:
+---[RSA 1024]---+
|+XB=. . .
|EBO o...
|*+oooo. .
|o.=.+o...
|...*. S
| .=o
| == .
| .o*.o
| .++=.
+---[SHA256]---+
```

Je vérifie la création de la paire de clé

```
root@sshclient:~# ls /home/userclient/
uc_rsa  uc_rsa.pub
```

Je peux changer le mot de passe d'une clé privée

```
root@sshserver1:~# ssh-keygen -p -f /home/user/rsa_user
```

- J'affiche la clé privée

```

userclient@sshclient:~$ cat uc_rsa
-----BEGIN OPENSSH PRIVATE KEY-----
b3B1bnNzaC1rZktjdEAAAAACmFlczI1Ni1jdHIAAAAGYmNyeXB0AAAAGAAAABDZe1Rmg2
KxpLKVuOER+WsAAAAEAAAAEAAACXAAAAB3NzaC1yc2EAAAADAQABAAAQgQDdKVWTmNOC
WpBq4oMLbEqsfAES+ZDvDhdBHzYXW1vUT+U/Xg5RgrjSJKFH3GZ3Ie9MHznwZv4uXzRGg9
hcrvzMtIdJbDpzjk8Z6pPhW71W9u8NJqeRfidoh//7M6+v/f0XDxkrIJtT3yXnuGHulf8
OSYJRrXmNa3YP+J2SgFwQwAAAhDg3VDRhbrbrup/AA/LnrljuhqiinpgOJxRPK4ZLo8u9k
NakQbtanU62cd7QicFxNnYPjofSyKCTZRCC/6vrxaQLOyuWw+YU0jd9qpZSF5Q+4D6dANj
or1VOQ0ptwxwEWA98YpfJ0xWpgQsNPWx+tXyK1zgX40ZxRxrUuQLHX6oKSDXAvcjKi+YHf
HPuPVarH59QEYqbzep06HQxdhjtezNn1hFGsaJobT9BF2Lbxr+mIBY/Gdgb+TOyLPtzzcK
YASh7imzyc7xl9ufnVZ4h5FK9chxbrz2Q9mZb25CyF5ryJpqtb7cRDdnEA1ID8MLgqyAZV
0mr0hLD3LS9Dx7FJqiTY1ewqTpp3JvDGGVlwDzOPVPwVSG45hPNk01Fqp6rmmZeeRgg3ny
pNTk+fTXAaZNOZ8DcjIGKmZLz7zpCooPuZw8b390ZI44YuhH/1EehXuZGq80DzoEjY01ch
27IHRFsfpBzeJIKs1+Y4zfVgAkKlnP/w9cmjicodXS60sPx1277UTWT9tOCqNDo2G7wpv2
hjtZ7u1lEJ8N7Uw9ZtmBHM/v7NX4pDXauxLzMMUlyW8+84suINWpAKaNIjcX5kpz1t28t
S5yMZVlsaKrkhka+2FG52T0d0/9GwmKtsjY3Q4HOT8/UCyeYdvkgqmpayy6+p3s3hrdbe5
ck9xBYCfpBCIXJkDhfnrFbCHCcPrhpq=
-----END OPENSSH PRIVATE KEY-----

```

- J'affiche la clé publique

```

userclient@sshclient:~$ cat uc_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAAQgQDdKVWTmNOCWpBq4oMLbEqsfAES+ZDvDhdBHzYXW1vUT+U/Xg5RgrjSJKFH3GZ3Ie9
MHznwZv4uXzRGg9hcrvzMtIdJbDpzjk8Z6pPhW71W9u8NJqeRfidoh//7M6+v/f0XDxkrIJtT3yXnuGHulf8OSYJRrXmNa3YP+J2Sg
FwQw== userclient@sshclient
userclient@sshclient:~$ 

```

Je vérifie l'empreinte numérique de ma clé publique

```

root@sshclient:~# ssh-keygen -l -E sha256 -f /home/userclient/uc_rsa.pub
1024 SHA256:EZIqNtPh2rpv4Wg5ytNSaQb0zQFao4FFaLGFnhk0qAI root@sshclient (RSA)

```

- 2- J'envoie la clé publique à userserver sur sshserver

```

userclient@sshclient:~$ ssh-copy-id -i uc_rsa.pub userserver@192.168.44.152

```

```

userclient@sshclient:~$ ssh-copy-id -i uc_rsa.pub userserver@192.168.44.152
/usr/bin/ssh-copy-id: INFO: source of key(s) to be installed: "uc_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
userserver@192.168.44.152's password:

Number of key(s) added: 1

Now try logging into the machine, with:  "ssh 'userserver@192.168.44.152'"
and check to make sure that only the key(s) you wanted were added.

```

On vérifie sur le serveur sshserver si la clé publique est envoyée elle doit être stocker dans le répertoire personnel de l'utilisateur userserver

```

userserver@sshserver:~$ pwd
/home/userserver
userserver@sshserver:~$ ls -a
. . . .bash_aliases .bash_history .bash_logout .bashrc .config .profile .ssh .viminfo
userserver@sshserver:~$ ls .ssh
authorized_keys
userserver@sshserver:~$ cat .ssh/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQDdKVWTmNOCWpBq4oMLbEqsfAES+ZDvDhdBHzyXW1vUT+U/Xg5RgrjSJKFH3GZ3Ie9
MHZnwZv4uXzRGg9hcrvzMtIdJbDpzjk8Z6pPhW71W9u8NJqeRfidoh//7M6+v/f0XDxkrIJtT3yXnuGHulf80SYJRrXmNa3YP+J2Sg
FwQw== userclient@sshclient
userserver@sshserver:~$ |

```

On remarque qu'elle correspond à celle affiché précédemment sur le serveur sshclient

- 3- Je me connecte en ssh sur sshserver avec le compte sshserver ; la machine distante va remarquer qu'elle possède la clé publique attaché à mon compte elle va m'envoyer un message chiffré (challenge) je le déchiffre le message et je l'envoie je prouve que j'ai en ma possession la clé privée donc je certifie mon identité dans le cas contraire je ne pourrais pas me connecter

```

userclient@sshclient:~$ ssh -i uc_rsa userserver@192.168.44.152
Enter passphrase for key 'uc_rsa': useruc

```

Remarque

Si on ne change pas le nom par défaut de la clé privée on n'est pas obligé de mettre le **-i nom de la clé privée** c'est-à-dire si dans l'exemple on change le nom de la clé privée de uc_rsa à id_rsa on peut juste taper :

```
#ssh user@192.168.44.152
```

Accès à distance

Authentification par clé publique

- Recopie de la clé publique du client vers le serveur grâce à ssh-copy-id ou simple concaténation dans **\$HOME/.ssh/authorized_keys**
- Clé privée disponible dans **\$HOME/.ssh/id_rsa** côté client
- La directive **PubkeyAuthentication** doit être positionnée à yes

C- Passphrase et agent SSH

Ssh-agent est un outil côté client ; Ssh-agent permet de rajouter dans la mémoire de mon agent ma clé privée tout en-déverrouillant-la à clé privée pendant un certain temps
La commande ssh-add permet de charger la clé privée en mémoire.

```
root@server1:~# man ssh-add
```

```
SSH-ADD(1)      BSD General Commands Manual     SSH-ADD(1)
```

NAME

ssh-add – adds private key identities to the OpenSSH authentication agent

SYNOPSIS

```
ssh-add [-cDdKkLlqvXx] [-E fingerprint_hash]
        [-S provider] [-t life] [file ...]
ssh-add -s pkcs11
ssh-add -e pkcs11
ssh-add -T pubkey ...
```

l' **agent** est un programme qui garde les clés en mémoire afin de les déverrouiller *qu'une seule fois*.

Il y'a deux méthodes pour démarrer *ssh-agent* :

- a- eval `ssh-agent` - cela exécute l'agent en arrière-plan et définit les variables d'environnement appropriées pour l' instance de shell *actuelle* .
- b- exec ssh-agent bash- démarre une *nouvelle* instance du bashshell, en remplaçant l'actuelle.

La deuxième méthode est parfois préférée, car elle arrête automatiquement ssh-agent lorsque on arrête la session.

Après il faut utiliser **ssh-add** , pour déverrouiller les clés et les associés à l'agent, les connexions *ssh, scp, sftp* ... pourront tirer profit de cette méthodes afin de simplifier les connexions.

- a- Création d'une nouvelle instance bash

exec ssh-agent bash- démarre une *nouvelle* instance du bashshell, en remplaçant l'actuelle.

```
user@sshclient:~$ exec ssh-agent bash
```

- b- Charger la clé dans la mémoire de l'agent

Utiliser **ssh-add** , déverrouille les clés (généralement *~/.ssh/id_**) et les charge dans l'agent, les rendant accessibles aux connexions *ssh ou sftp* .

```
userclient@sshclient:~$ ssh-add uc_rsa
Enter passphrase for uc_rsa: useruc
Identity added: uc_rsa (userclient@sshclient)
userclient@sshclient:~$ |
```

Pour vérifier on peut lister les clés en mémoire

```
userclient@sshclient:~$ ssh-add -l
1024 SHA256:sJNv70kVxqI2+m1pZagLgQSJaeOtE2ZoAtujOMQ0mFg userclient@sshclient (RSA)
```

- c- Connexion ssh en utilisant l'agent ssh avec la Passphrase en mémoire

Je me connecte en ssh on me demande plus la clé passphrase

```
userclient@sshclient:~$ ssh -i uc_rsa userserver@192.168.44.152
```

- d- Autres options

Si je veux effacer des clés en mémoire de l'agent ssh

```
userclient@sshclient:~$ ssh-add -d uc_rsa
Identity removed: uc_rsa RSA (userclient@sshclient)
userclient@sshclient:~$ ssh-add -l
The agent has no identities.
userclient@sshclient:~$ |
```

Mémoriser la clé pour une durée limité

```
userclient$ sshclient:~$ ssh-add -t 60 uc_rsa
Enter passphrase for uc_rsa:
Identity added: uc_rsa (userclient@sshclient)
Lifetime set to 60 seconds
userclient$
```

Après 60 secondes la clé est effacée de la mémoire de l'agent

```
userclient$ sshclient:~$ ssh-add -l
The agent has no identities.
```

D- Créations du fichier config

Dans le cas où connaît une multitude de serveurs le fichier config qu'on peut créer dans le dossier ./ssh nous facilitera la tâche car au lieu de taper la commande

#ssh user@adresse ip du serveur

On peut juste entrer le nom de la machine

#neptune par exemple c'est plus facile car mémoriser toutes les adresses ip est fastidieux

a- Créer le fichier config dans .ssh

```
userclient$ sshclient:~$ cd .ssh
userclient$ sshclient:~/ssh$ touch config
userclient$ sshclient:~/ssh$ ls
config known_hosts
```

b- Modifier le fichier config

```
userclient$ sshclient:~/ssh$ vim config |
host sshserver
    user userserver
    port 22
    identityfile /home/userserver/uc_rsa
~
```

c- modifier le fichier hosts

```
root$ sshclient:~# vim /etc/hosts
127.0.0.1      localhost
127.0.1.1      sshclient
192.168.44.152  sshserver

# The following lines are desirable for IPv6 capable hosts
::1      localhost ip6-localhost ip6-loopback
ff02::1  ip6-allnodes
ff02::2  ip6-allrouters
```

Maintenant on peut se connecter sur notre serveur

```
userclient$ sshclient:/root$ ssh sshserver
```

E- La double Authentification google

On commence par installer le paquet libpam-google-authenticator sur le serveur ssh

```
root$ sshserver:~# apt install libpam-google-authenticator
```

On modifie le fichier sshd dans le dossier /etc/pam.d/

```
root@sshserver:~# vim /etc/pam.d/sshd
```

On rajoute à la fin dans ce fichier le bloc suivant

```
#Authentication Google
auth required pam_google_authenticator.so
```

On modifie aussi le fichier sshd_config dans /etc/ssh/

```
root@sshserver:/etc/ssh# vim sshd_config
# Change to yes to enable challenge-response passwords (beware issues with
# some PAM modules and threads)
ChallengeResponseAuthentication yes
```

On redémarre le service sshd puisque on modifier le fichier sshd_config

En même temps on vérifie si le service a bien redémarrer

```
root@sshserver:/etc/ssh# service sshd restart
root@sshserver:/etc/ssh# service sshd status
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2021-09-26 20:24:22 CEST; 13s ago
     Docs: man:sshd(8)
           man:sshd_config(5)
     Process: 6911 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
    Main PID: 6912 (sshd)
      Tasks: 1 (limit: 2303)
     Memory: 1.1M
        CPU: 29ms
       CGroup: /system.slice/ssh.service
               └─6912 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups

sept. 26 20:24:22 sshserver systemd[1]: Starting OpenBSD Secure Shell server...
sept. 26 20:24:22 sshserver sshd[6912]: Server listening on 0.0.0.0 port 22.
sept. 26 20:24:22 sshserver sshd[6912]: Server listening on :: port 22.
sept. 26 20:24:22 sshserver systemd[1]: Started OpenBSD Secure Shell server.
```

On lance google-authenticator

```
root@sshserver:/etc/ssh# google-authenticator
Do you want authentication tokens to be time-based (y/n) y|
```

Attention il faut lancer google-authenticator pour chaque utilisateur si on veut appliquer la double authentification à tous les utilisateurs



Sur le smartphone il faut télécharger sur le store l'application



Authenticator

Synchronisation automatique

Une horloge possède une dérive inhérente dû au quartz de la carte mère, l'horloge a tendance à se décaler de l'heure officielle (plusieurs secondes par jour pour certaines cartes mère). Pour éviter ce décalage, on utilise le protocole NTP pour synchroniser notre serveur avec un serveur de temps.

Il faut d'abord installer les paquets ntp et ntpdate avec la commande suivante

```
root@server1:~# apt install ntp ntpdate
```

Il faut configurer NTP on peut utiliser :

Serveurs NTP du projet **pool.ntp.org**

Pour la France, les serveurs NTP a utiliser sont : **server 0.fr.pool.ntp.org server 1.fr.pool.ntp.org server 2.fr.pool.ntp.org**

Donc on va modifier le fichier /etc/ntp.conf et de remplacer les lignes correspondantes aux adresses des serveurs NTP par celles ci-dessus.

On peut régler l'heure manuellement avec la commande suivante :

```
root@server1:~# ntpdate pool.ntp.org
8 Oct 18:45:44 ntpdate[1920]: adjust time server 45.33.84.208 offset -0.021789 sec
```

On vérifie si le service ntp est démarrer

```
root@server1:~# service ntp status
● ntp.service - Network Time Service
  Loaded: loaded (/lib/systemd/system/ntp.service; enabled; vendor preset: enabled)
  Active: active (running) since Fri 2021-10-08 18:49:03 CEST; 27s ago
    Docs: man:ntpd(8)
 Process: 1930 ExecStart=/usr/lib/ntp/ntp-systemd-wrapper (code=exited, status=0/SUCCESS)
 Main PID: 1936 (ntpd)
   Tasks: 2 (limit: 2303)
  Memory: 948.0K
     CPU: 41ms
    CGroup: /system.slice/ntp.service
            └─1936 /usr/sbin/ntpd -p /var/run/ntpd.pid -g -u 106:112

oct. 08 18:49:05 server1 ntpd[1936]: Soliciting pool server 95.81.173.155
oct. 08 18:49:05 server1 ntpd[1936]: Soliciting pool server 37.187.105.73
oct. 08 18:49:06 server1 ntpd[1936]: Soliciting pool server 129.250.35.251
oct. 08 18:49:06 server1 ntpd[1936]: Soliciting pool server 162.159.200.1
oct. 08 18:49:07 server1 ntpd[1936]: Soliciting pool server 212.83.179.156
oct. 08 18:49:07 server1 ntpd[1936]: Soliciting pool server 162.159.200.123
oct. 08 18:49:07 server1 ntpd[1936]: Soliciting pool server 80.74.64.2
oct. 08 18:49:08 server1 ntpd[1936]: Soliciting pool server 51.255.197.148
oct. 08 18:49:09 server1 ntpd[1936]: Soliciting pool server 78.196.167.192
oct. 08 18:49:10 server1 ntpd[1936]: Soliciting pool server 92.243.6.5
root@server1:~#
```

F- Cration d'une banniere

Le but est d'avoir une bannière d'accueil au cours de notre connexion Ssh

Pour réaliser cette bannière il faut :

- a- Créer un fichier qui va présenter cette bannière on peut lui donner le nom qu'on veut dans notre cas je lui ai donné le nom de banner

- b- Il faut renseigner le chemin du fichier banner dans sshd config

```
root@server1:/etc/ssh# vim sshd_config
```

```
# no default banner path  
Banner /root/banner
```

G- Connection ssh avec une solution mobile

- a- Connexion avec une redirection de port

Maintenant on va essayer d'accéder à notre serveur ssh à partir de l'extérieur en utilisant notre adresse publique.

Tout d'abord :

Attention mettez votre machine en bridge au lieu du Nat

- Il faut accéder à la boîte internet et ouvrir le port 22 en créant une redirection de port

Service	Adresse IP du serveur	Protocole	Ports externes	Ports internes	Activer la règle
▲ Utilisateur					
SSH	192.168.1.250	TCP/UDP	22 • 22	22 • 22	<input checked="" type="button"/> on
Activer UPnP off					
Règles actives UPnP v4 2					
Service	Adresse IP du serveur	Protocole	Ports externes	Ports internes	Activer la règle
NAT for STB 6035C0E8E710 port STBCONTROL	192.168.1.150	TCP	1290 • 1290	8000 • 8000	<input checked="" type="button"/> on
NAT for STB 6035C0E8E6D4 port STBCONTROL	192.168.1.137	TCP	1291 • 1291	8000 • 8000	<input checked="" type="button"/> on

- Ensuite il faut déterminer notre adresse publique soit à partir de la boîte ou un site internet <http://www.whatismyip.com>

- Après sur notre smartphone on télécharge un client ssh sur  google store **Juicessh**

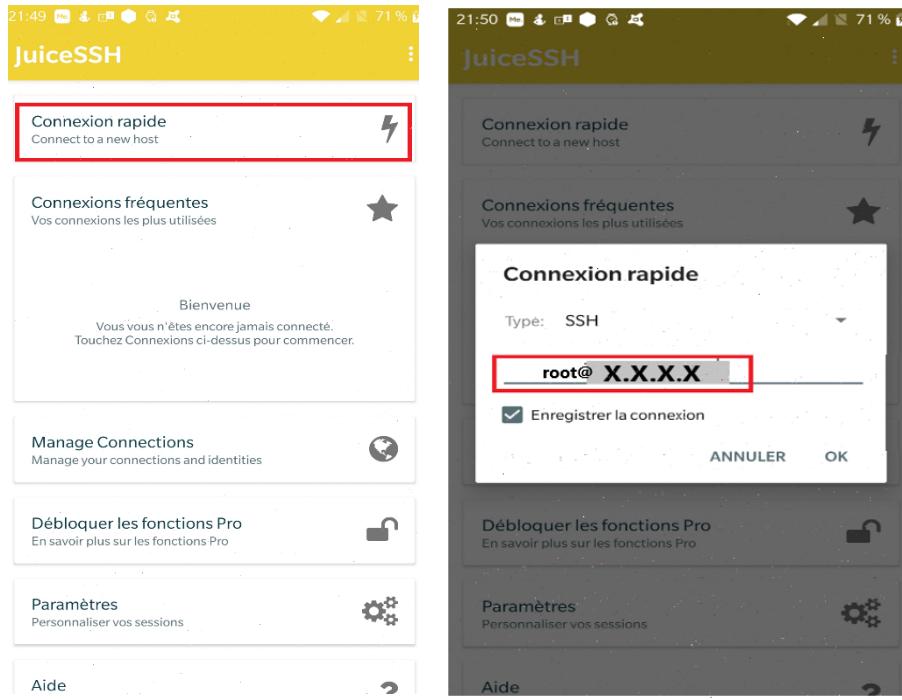


JuiceSSH - SSH Client

- Sur notre smartphone il faut qu'on se mette en 4G et non en wifi.
On peut effectuer cette procédure en wifi sur notre smartphone et non en 4G.
- On ouvre l'application et on commence à établir notre connexion ssh

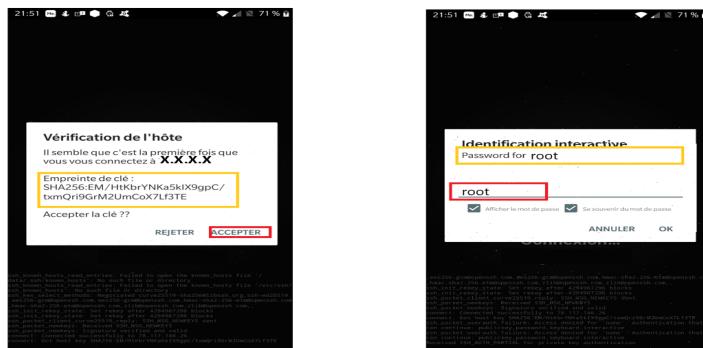
On sélectionne connexion rapide

ssh root@ adresse publique



Le serveur nous envoie l'empreinte de sa clé publique
 On constate que c'est la même que celle qu'on a calculé
 Sur le serveur

EM/HtKbrYNKa5kIX9gpC/txmQri9GrM2UmCoX7Lf3TE rentre le mot de passe root

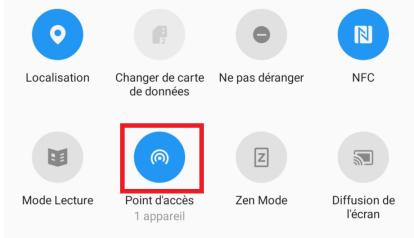


Après on tombe sur notre console VMWare

```
16:23 ⚡ 🔍 🌐 🌐 🌐 42% ⓘ
[...]
66 RENDINE 66 >
X
root@server:~# ip ad
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 brd 127.255.255.255 scope host lo
            valid_lft forever preferred_lft forever
inet6 ::/128 brd :: scope host
        valid_lft forever preferred_lft forever
2: ens3: <NO-CARRIER,BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state DOWN group default qlen 1000
    link/ether 00:0c:29:f7:2e:20 brd ff:ff:ff:ff:ff:ff
        altname enp2s1
        inet 192.168.1.16/24 brd 192.168.1.255 scope global dynamic ens3
            valid_lft 157352sec preferred_lft 65575sec
            inet6 fe80::20c:29ff:feff:2e20/64 scope link
                valid_lft forever preferred_lft forever
root@server:~#
```

b- Connexion avec un partage de connexion 4G

- Faites un partage de connexion à partir de votre smartphone



- Connectez-vous en wifi sur votre machine physique en utilisant le wifi correspondant au partage de connexion de votre smartphone
 - Mettez votre serveur SSH en Bridge pour avoir une adresse IP du même réseau que la machine physique et le smartphone
 - Installez un terminal sur le smartphone :
Exemple de terminal :



Termux

Fredrik Fornwall

- Relevez l'adresse IP smartphone, machine physique et la VM serveur SSH

IP du smartphone : 192.168.43.234 /24

```
12:25 <_> 40 %  
$ ip ad  
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1  
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00  
    inet 127.0.0.1/8 scope host lo  
        valid_lft forever preferred_lft forever  
        inet6 ::1/128 scope host  
            valid_lft forever preferred_lft forever  
26: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 3000  
    link/ether da:7e:ff:f6:a7:79 brd ff:ff:ff:ff:ff:ff  
    inet 192.168.43.234/24 brd 192.168.43.255 scope global wlan0  
        valid_lft forever preferred_lft forever  
        inet6 fe80::d87e:ffff:fe6:a779/64 scope link  
            valid_lft forever preferred_lft forever
```

IP de la machine physique : 192.168.43.197 /24

```
[root💀ETANIUM] ~ # ip ad
```

```

22: wifi0: <BROADCAST,MULTICAST,UP> mtu 1500 group default qlen 1
    link/ieee802.11 f8:16:54:2b:40:f8
      inet 192.168.43.197/24 brd 192.168.43.255 scope global dynamic
        valid_lft 2341sec preferred_lft 2341sec
      inet6 fe80::3410:fa32:28b6:999a/64 scope link dynamic
        valid_lft forever preferred_lft forever
22: wifi0: <BROADCAST,MULTICAST,UP> mtu 1500 group default qlen 1
    link/ieee802.11 f8:16:54:2b:40:f8
      inet 192.168.43.197/24 brd 192.168.43.255 scope global dynamic
        valid_lft 2341sec preferred_lft 2341sec
      inet6 fe80::3410:fa32:28b6:999a/64 scope link dynamic
        valid_lft forever preferred_lft forever

```

IP de la machine VM serveur SSH : 192.168.43.122 /24

```

root@server1:~# ip ad
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
      inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
      inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:f7:2e:20 brd ff:ff:ff:ff:ff:ff
      altname enp2s1
      inet 192.168.43.122/24 brd 192.168.43.255 scope global dynamic ens3
        valid_lft 3207sec preferred_lft 3207sec
      inet6 fe80::20c:29ff:fe7:2e20/64 scope link
        valid_lft forever preferred_lft forever

```

- Faites un test de ping entre smartphone, machine physique et la VM serveur SSH
Smartphone à Machine physique



```

$ ping 192.168.43.197
PING 192.168.43.197 (192.168.43.197) 56(84) bytes of data.
64 bytes from 192.168.43.197: icmp_seq=1 ttl=128 time=15.8 ms
64 bytes from 192.168.43.197: icmp_seq=2 ttl=128 time=5.17 ms

```

Smartphone à Serveur SSH



```

$ ping 192.168.43.122
PING 192.168.43.122 (192.168.43.122) 56(84) bytes of data.
64 bytes from 192.168.43.122: icmp_seq=1 ttl=64 time=5.11 ms
64 bytes from 192.168.43.122: icmp_seq=2 ttl=64 time=14.5 ms

```

- Utilisez maintenant votre application client SSH sur votre smartphone pour se connecter la VM serveur SSH, avec **ip ad** on vérifie bien qu'on est sur le serveur ssh

```

15:20 ① 🌐 ⚡ 🌐 🌐 🌐 >_ ① N 🌐 🌐 🌐 35 %
root@server:~# ip ad
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue
    state UNKNOWN group default qlen 1000
        link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
    qdisc pfifo_fast state UP group default qlen 1000
        link/ether 00:0c:29:f7:2e:20 brd ff:ff:ff:ff:ff:ff
        altname enp2s1
        inet 192.168.43.122/24 brd 192.168.43.255 scope global dynamic ens33
            valid_lft 3180sec preferred_lft 3180sec
        inet6 fe80::20c:29ff:fef7:2e20/64 scope link
            valid_lft forever preferred_lft forever
root@server:~#

```

H- Les Ports de communication:

Un numéro de port est codé sur 16 bits, ce qui fait qu'il existe un maximum de 2^{16} , soit **65 536** ports distincts par machine. Ces ports sont classés en 3 catégories en fonction de leur numéro:

- les numéros de port de 0 à 1 023 correspondent aux ports "bien-connus" (**well-known ports**), utilisés pour les services réseaux les plus courants.
- sous Windows, ce fichier est dans **C:\Windows\System32\drivers\etc**

Nom	Modifié le	Type	Taille
hosts	09/10/2021 22:20	Fichier	2 Ko
hosts	16/10/2021 15:13	Fichier iCalendar	1 Ko
lmhosts.sam	07/12/2019 10:12	Fichier SAM	4 Ko
networks	29/09/2017 15:44	Fichier	1 Ko
protocol	29/09/2017 15:44	Fichier	2 Ko
services	29/09/2017 15:44	Fichier	18 Ko

- Sous linux Le fichier services indiquant la liste des services dits well-known, est dans /etc ;

```

root@server1:/etc# ls services
services

```

- les numéros de ports de 1 024 à 49 151 correspondent aux ports enregistrés (**registered ports**), assignés par l'IANA
- les numéros de ports de 49 152 à 65 535 correspondent aux ports dynamiques, utilisables pour tout type de requêtes TCP ou UDP autres que celle citées précédemment.

Lorsqu'un logiciel client veut dialoguer avec un logiciel serveur, aussi appelé service, il a besoin de connaître le port écouté par ce dernier. Les ports utilisés par les services doivent être connus par les clients, les principaux types de services utilisent des ports qui sont dits réservés. Par convention, ce sont tous ceux compris entre 0 et 1 023 inclus et leur utilisation

par un logiciel serveur nécessite souvent que celui-ci s'exécute avec des droits d'accès particuliers. Les services utilisant ces ports sont appelés les services bien connus ("Well-Known Services").

Le fichier services indique la liste de ces services dits well-known. Sous UNIX, ce fichier est directement dans /etc ; sous Windows, ce fichier est par défaut dans C:\Windows\System32\drivers\etc. Les services les plus utilisés sont :

Toutefois, ces conventions ci-dessus peuvent ne pas être respectées pourvu que le client et le serveur soient cohérents entre eux et que le nouveau numéro choisi ne soit pas déjà utilisé
Pour afficher les port on utilise netstat

```
#netstat -paunt
```

- -a : Tous les ports
- -t : Tous les ports TCP
- -u : Tous les ports UDP
- -l : Tous les ports en écoute
- -n : Affiche directement les IP. Pas de résolution de nom.
- -p : Affiche le nom du programme et le PID associé.