



Projet de communication réseau

Alice & Bob

SOMMAIRE

I - Introduction.....	3
II - SSH.....	5
A - Histoire.....	5
B - Objectifs.....	5
C - Paramétrage.....	6
III - OpenSSL.....	14
A - Histoire.....	14
B - Objectif.....	14
C - Paramétrage.....	14
IV - SCP.....	16
A - Histoire.....	16
B - Objectif.....	16
C - Paramétrage.....	16
V - Conclusion.....	20

I - Introduction

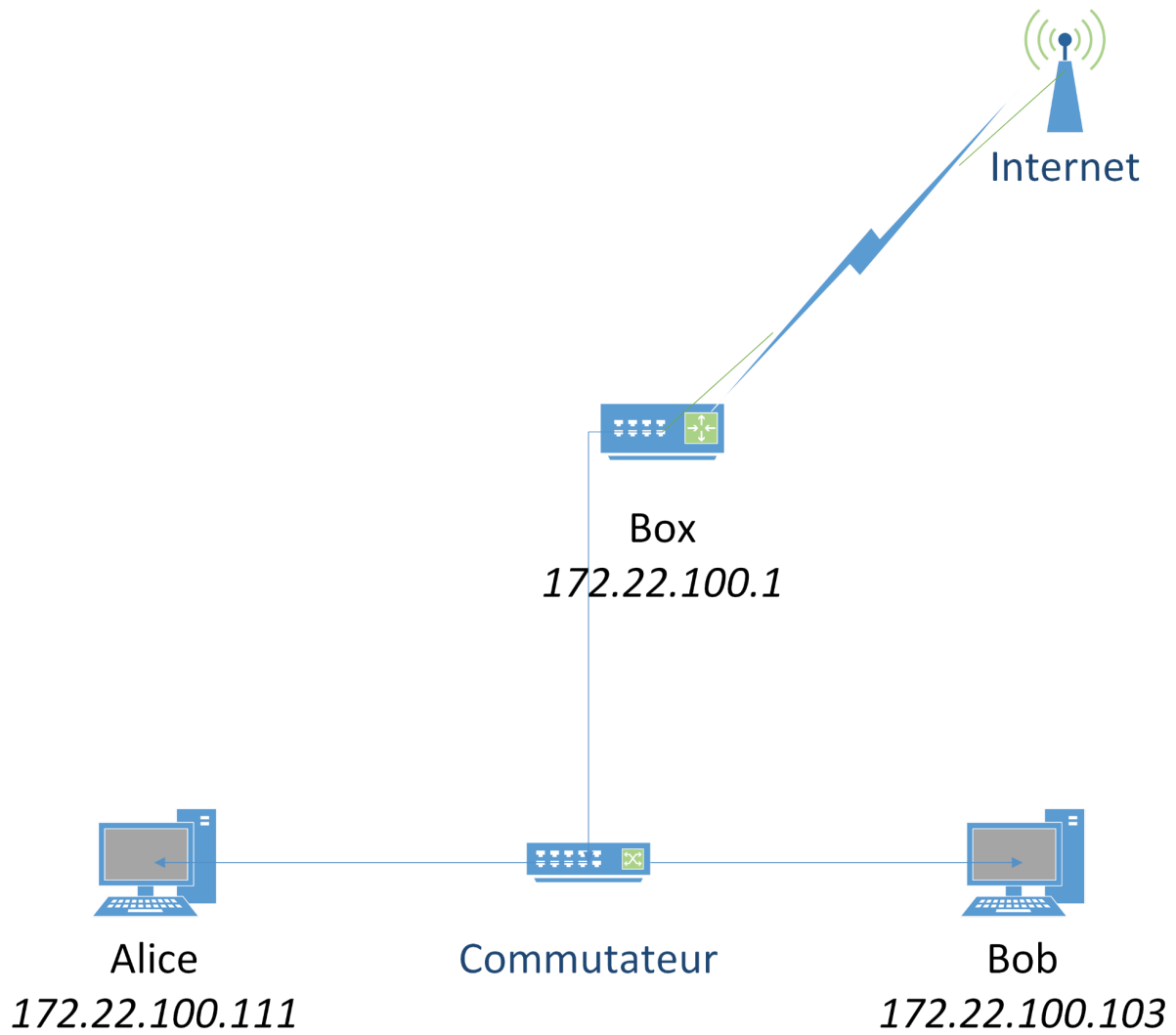
De nos jours, les logiques et comportements des utilisateurs sur Internet doivent être de plus en plus sécurisés, notamment en matière de communication. Ainsi, nous allons explorer trois technologies permettant de créer un espace de communication sécurisé dans lequel les informations vont pouvoir transiter sans risque de fuite ou de vol.

Tout d'abord, afin de pouvoir connecter les deux machines entre elles, nous allons utiliser le protocole **Secure Shell (SSH)**.

De plus, pour permettre de sécuriser les messages et les informations, nous allons créer un jeu de clé asymétrique à l'aide du protocole **Secure Socket Layer (SSL)** et du logiciel **OpenSSL**. Enfin, pour transmettre nos clés, sans risque de fuite de données, nous allons communiquer grâce au protocole **Secure Copy Protocol (SCP)**.

Grâce à cet environnement, nos deux machines pourront communiquer en minimisant les risques et en augmentant leur confidentialité.

L'idée, à terme, est que nos deux utilisateurs, **Alice** et **Bob** puissent se connecter mutuellement sur l'ordinateur de l'un et de l'autre, de façon sécurisée.



Topologie du réseau local

II - SSH

A - Histoire

SSH (Secure Shell) est un protocole de communication introduit en 1995. Il permet d'accéder de manière sécurisée à une machine depuis son ordinateur.

Il est l'héritier de Telnet, un protocole qui agit de la même manière mais qui ne chiffrait pas les données, pouvant provoquer d'importantes failles de sécurité en cas d'interception de données sensibles. C'est pour corriger ses défauts que SSH a été introduit.

La première version de SSH, conçue par Tatu Ylönen en Finlande, est apparue en 1995. SSH utilise alors le chiffrement des données asymétriques pour sécuriser la connexion et les échanges entre les machines. La deuxième version, arrivée en 2006, réécrit le protocole avec des mécanismes encore plus sécurisés pour éviter les vulnérabilités. C'est cette version de SSH qui est aujourd'hui la plus répandue.

B - Objectifs

Comme précisé plus tôt, le protocole **SSH** a permis de remplacer le protocole **Telnet** qui n'était pas sécurisé. Maintenant, quels sont les objectifs de ce protocole ?

SSH, comme son nom l'indique, a pour objectif de créer un accès à un shell depuis un hôte distant. Le shell, sur une machine Linux, est un espace permettant d'exécuter des commandes. Ainsi, un hôte quelconque peut, à travers ce protocole, se connecter à une machine et exécuter des commandes.

La pertinence du **SSH**, comme reflet du shell, est sa liaison aux utilisateurs de l'hôte Linux. Par conséquent, le **SSH** intègre une hiérarchie de permissions, tirée des utilisateurs de la machine, qui peut sécuriser les accès lors des communications. Dans la logique, chaque utilisateur se connectant à une machine possède un utilisateur avec lequel il a des accès précis sur les fichiers et les commandes.

De plus, **SSH** intègre une double logique d'authentification, de façon native : le mot de passe et les clés asymétriques. Dans le contexte commun, le mot de passe est une solution facile et prise en main mais elle implique un risque, celui du mot de passe volé. Ainsi, la possibilité de se connecter à l'aide d'un jeu de clé publique et clé privée a été intégrée afin de mieux sécuriser les connexions.

C - Paramétrage

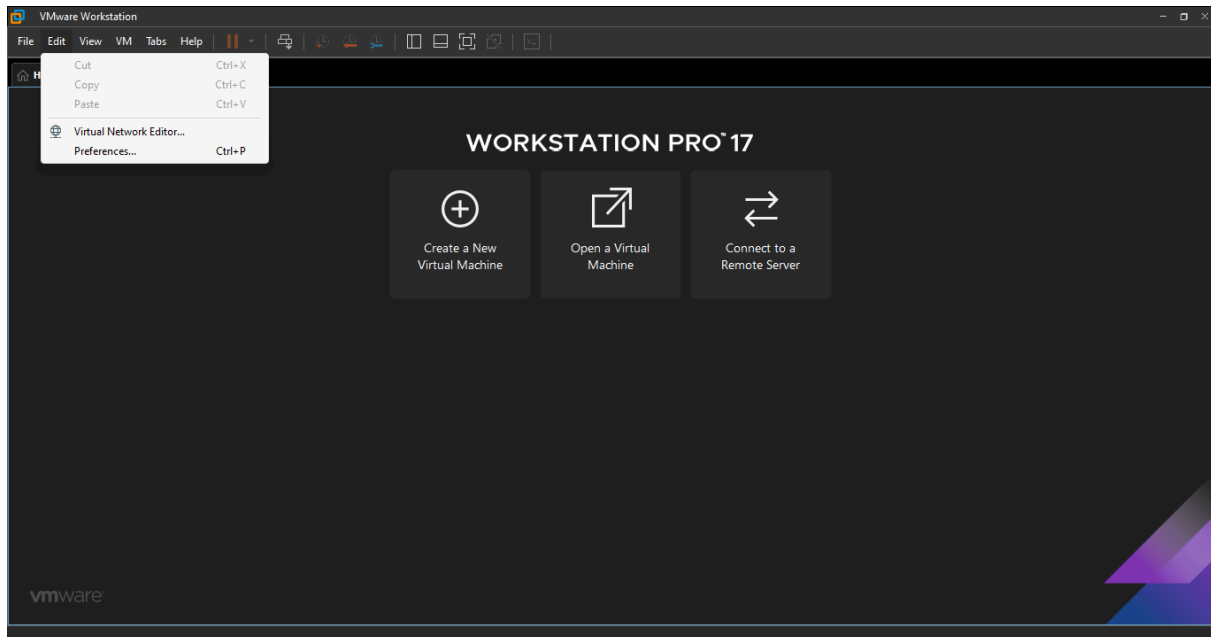
Nous allons donc commencer notre processus de fonctionnement par la mise en communication des deux machines à l'aide du protocole **SSH**.

Tout d'abord, les deux machines doivent identifier leur IP et se les partager.

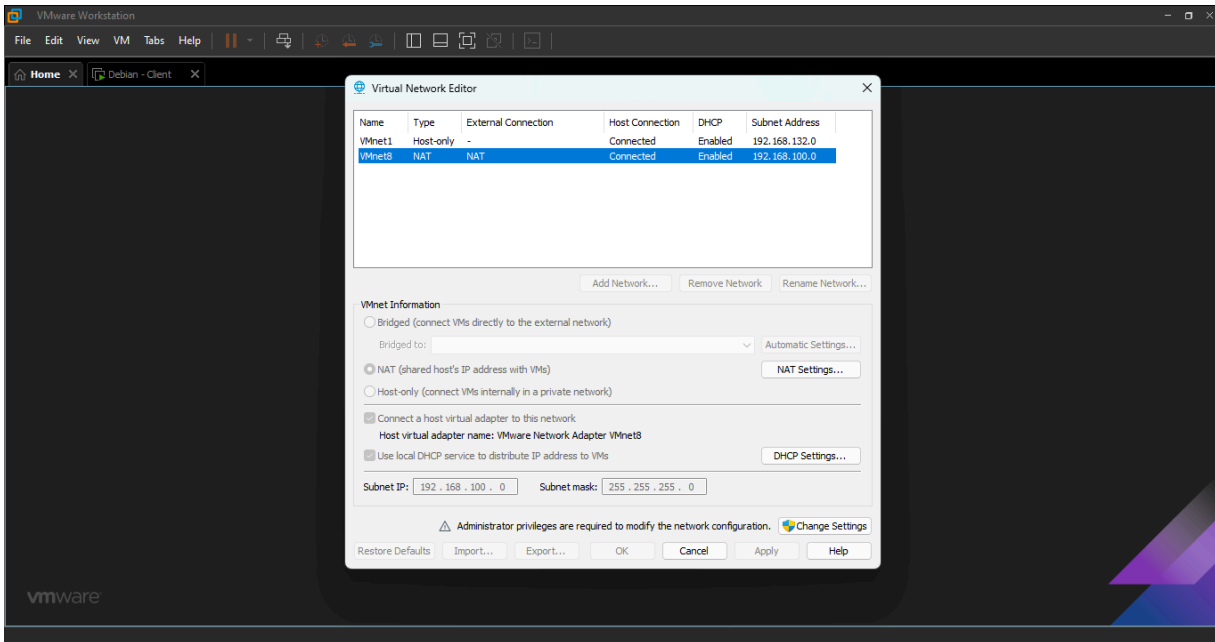
Bob utilise une machine virtuelle Debian sur VMWare et Alice utilise une machine virtuelle Debian sur UTM. Afin de pouvoir communiquer entre eux, sans exposer leur machine virtuelle sur Internet, les deux doivent configurer une redirection de ports vers leur machine et laisser la carte réseau en NAT.

L'objectif, ici, est de permettre aux requêtes sur le port **22**, utilisé par SSH, d'être redirigées vers les machines virtuelles.

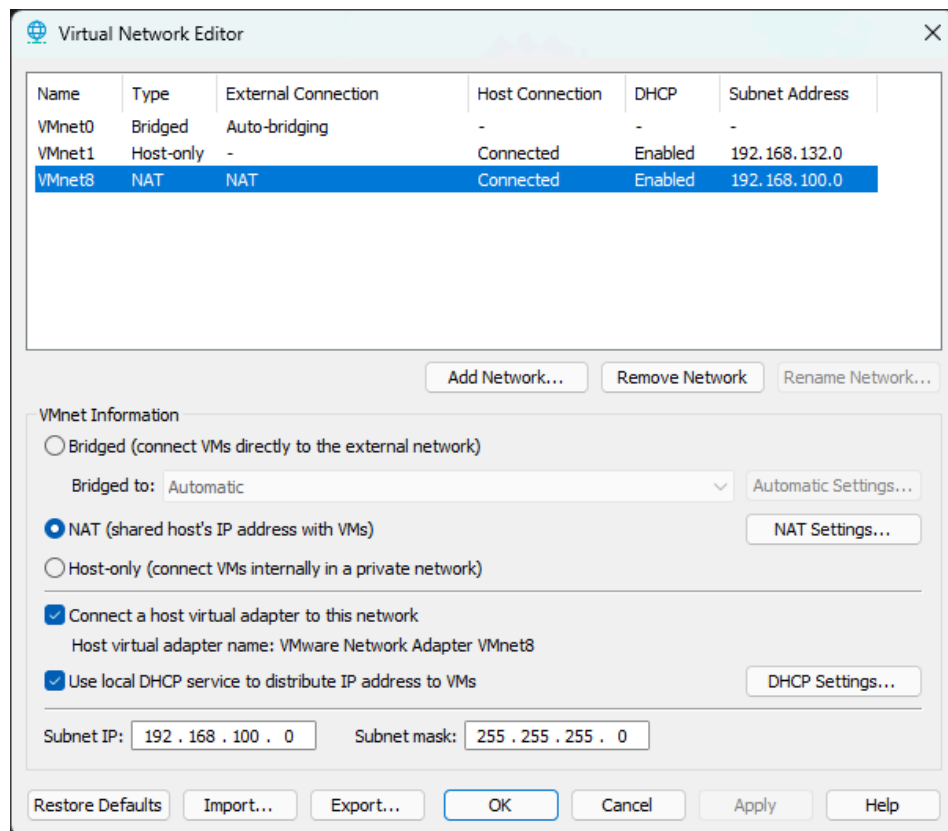
Ainsi, nous allons configurer la redirection de ports sur VMWare. Il faut d'abord se rendre dans **Edit** :



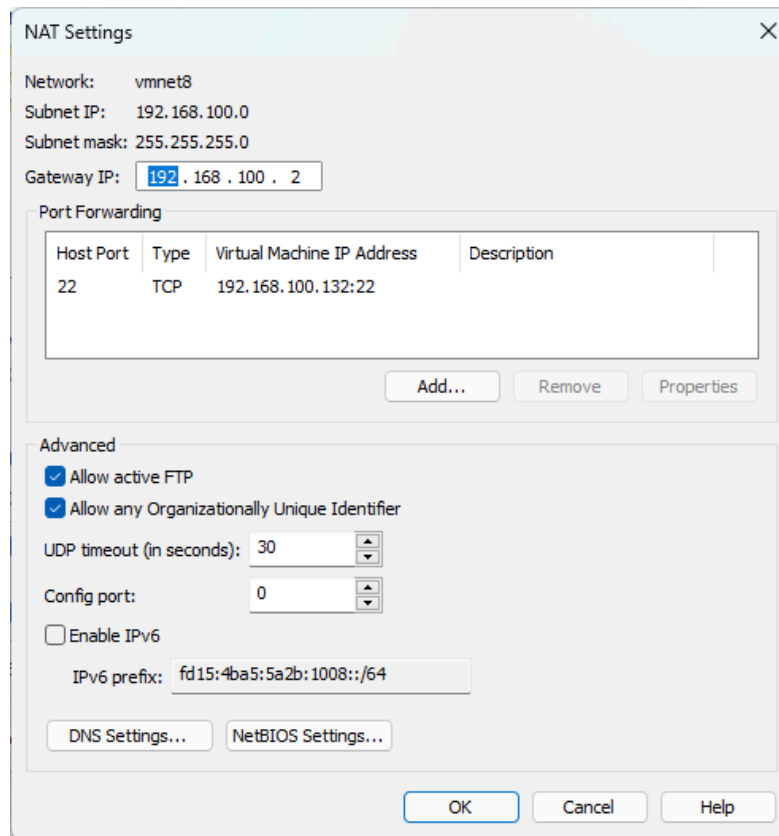
Puis, dans **Virtual Network Editor** et sélectionner la carte de type **NAT** :



Ensuite, il faut cliquer sur **Change Settings** :



Enfin, nous pouvons cliquer sur **NAT Settings** qui ouvrira le menu permettant de gérer les redirections de ports :



Une fois dans cet espace, nous allons pouvoir créer une redirection mais il faut d'abord connaître l'IP locale de la machine virtuelle. Bob va donc taper dans sa machine virtuelle :

ip a

L'IP local qui lui est donnée est celle à entrer lors de la création :

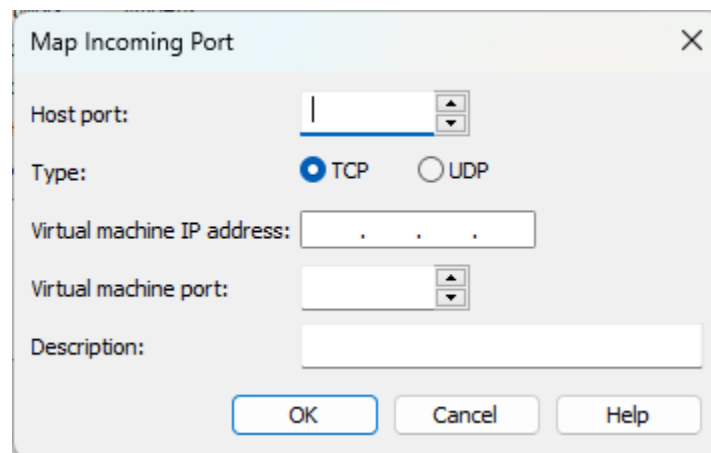
```
Debian GNU/Linux 12 debian tty1

debian login: root
Password:
Linux debian 6.1.0-32-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.129-1 (2025-03-06) x86_64

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Mar 17 15:08:48 CET 2025 on tty1
root@debian:~# ip a
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 noprefixroute
        valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:5c:3f:03 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    inet 192.168.100.132/24 brd 192.168.100.255 scope global dynamic ens33
        valid_lft 1575sec preferred_lft 1575sec
    inet6 fe80::20c:29ff:fe5c:3f03/64 scope link
        valid_lft forever preferred_lft forever
root@debian:~#
```

Ici, Bob a pour IP locale "192.168.100.132", nous pouvons donc créer notre redirection avec cette IP. Pour se faire il faut appuyer sur le bouton **Add** :



The screenshot shows a 'Map Incoming Port' dialog box. It contains the following fields and controls:

- Host port:** A text input field with a spinner control.
- Type:** Two radio buttons, 'TCP' (selected) and 'UDP'.
- Virtual machine IP address:** A text input field with three dots as a placeholder.
- Virtual machine port:** A text input field with a spinner control.
- Description:** A text input field.
- Buttons:** 'OK', 'Cancel', and 'Help' buttons at the bottom.

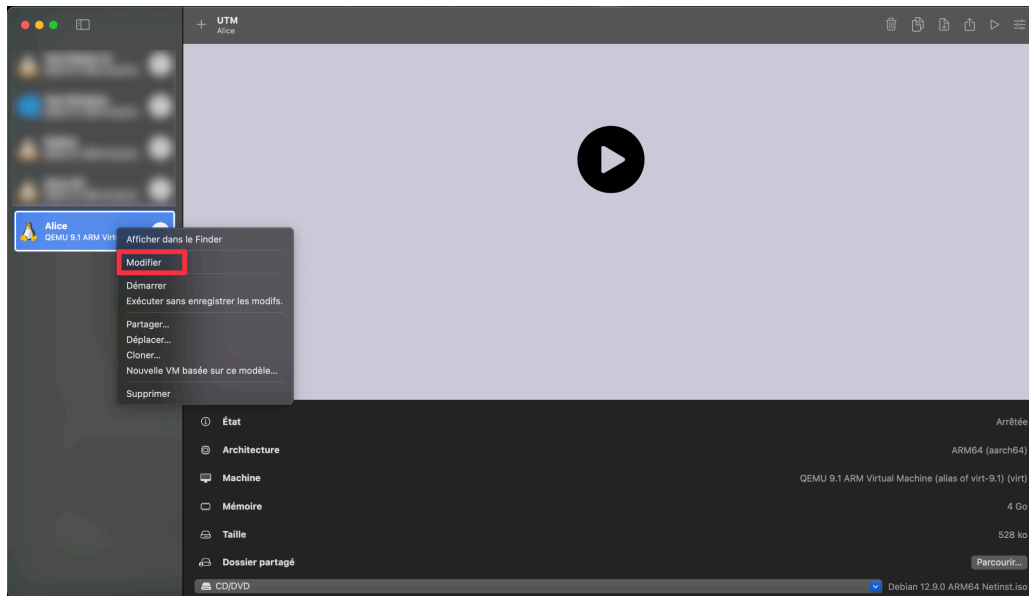
Nous pouvons remplir les différents champs avec ces informations :

- Host Port : 22
- Type : TCP
- Virtual machine IP address : 192.168.100.132 (L'IP de la machine virtuelle de Bob)
- Virtual machine port : 22
- Description : SSH

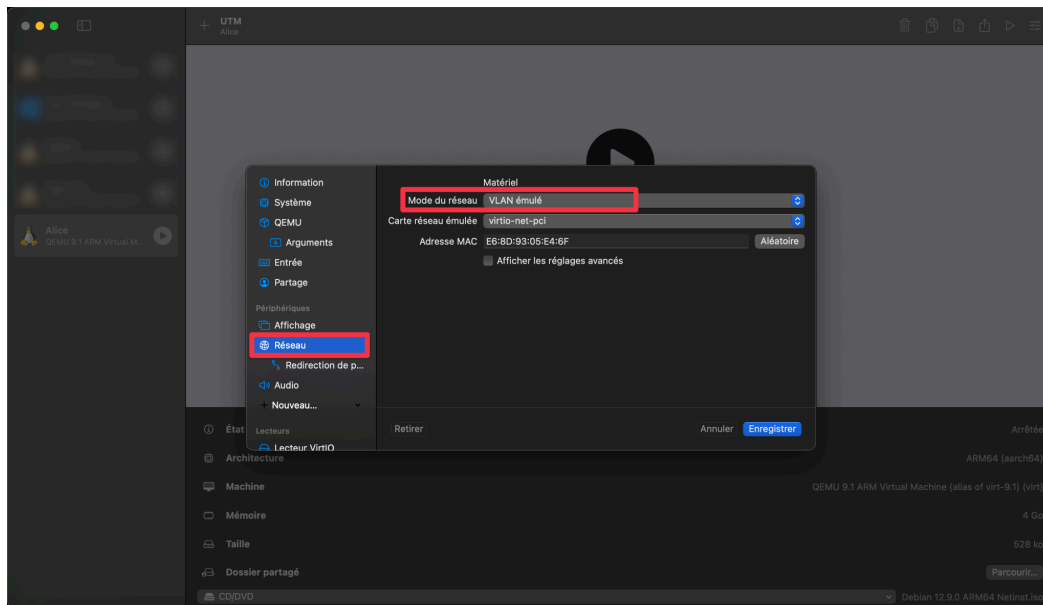
Enfin, nous pouvons appuyer sur **OK** pour valider notre redirection.

En cas d'utilisation d'un mac (apple silicon), voici comment faire pour effectuer la redirection de ports avec le logiciel UTM :

Éteindre sa VM si ce n'est pas déjà fait. Ensuite, faire un clique droit sur sa machine virtuelle, et sélectionner "Modifier".

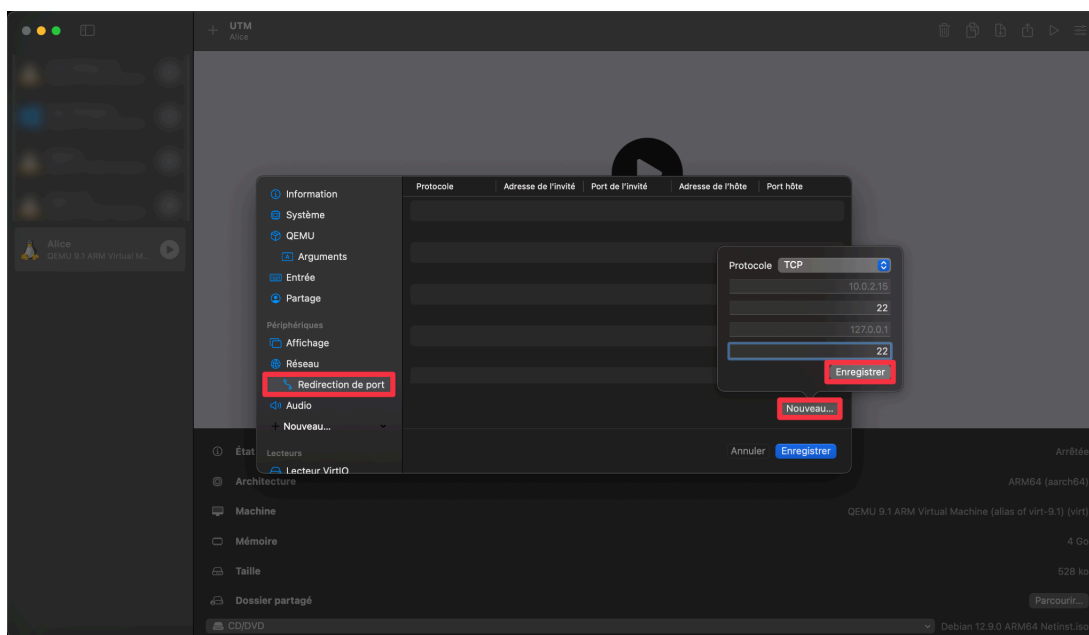


Ensuite, dans l'interface de configuration qui s'est ouverte, aller dans la partie "réseau", puis changer le mode du réseau en "VLAN émulé".

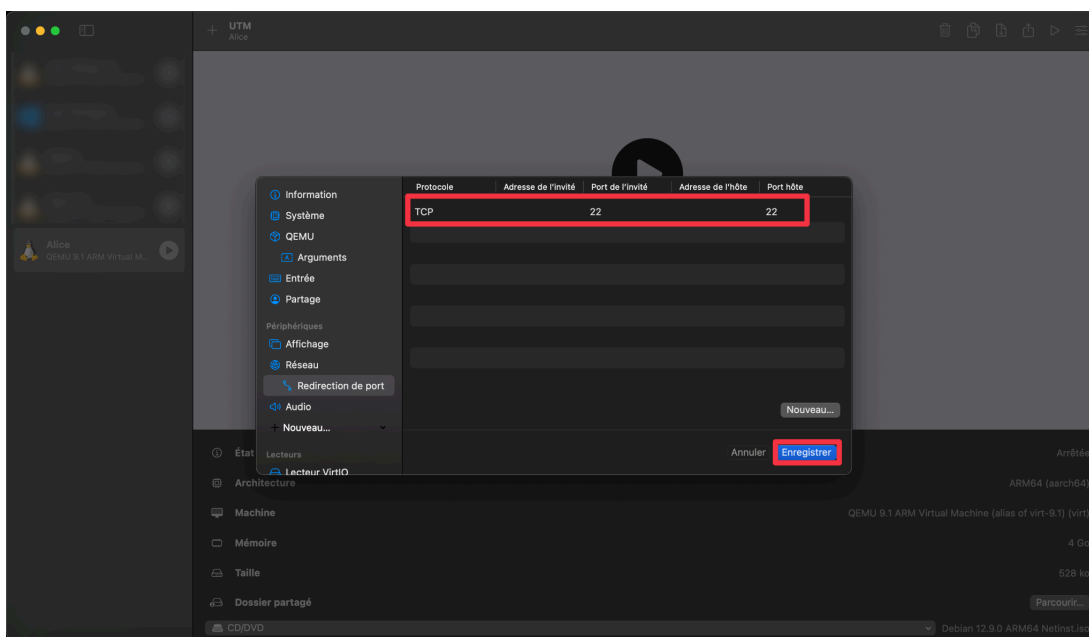


Une nouvelle option est apparue en dessous de réseau : "Redirection des ports". Dans cette option, créer une redirection des ports en sélectionnant "Nouveau". Ensuite, laisser TCP, et

remplir le deuxième et le quatrième champ par le port 22, et laisser les autres vides. Puis enregistrer.



Si tout s'est bien passé, une ligne est apparue avec la redirection saisie. Sélectionner "enregistrer". La redirection sur le port 22 est maintenant active.



Maintenant que notre redirection est prête, nous allons créer un utilisateur sur chacune des machines pour permettre à Alice et Bob de pouvoir se connecter mutuellement. Ainsi, dans notre shell linux nous allons pouvoir faire, sur la machine d'Alice la commande :

adduser bob

Et remplir toutes les informations et le mot de passe :

```
root@alice:~# adduser bob
Ajout de l'utilisateur « bob » ...
Ajout du nouveau groupe « bob » (1001) ...
Ajout du nouvel utilisateur « bob » (1001) avec le groupe « bob » (1001) ...
Création du répertoire personnel « /home/bob » ...
Copie des fichiers depuis « /etc/skel » ...
Nouveau mot de passe :
Retapez le nouveau mot de passe :
passwd : mot de passe mis à jour avec succès
Modifier les informations associées à un utilisateur pour bob
Entrer la nouvelle valeur, ou appuyer sur ENTER pour la valeur par défaut
  NOM []:
  Numéro de chambre []:
  Téléphone professionnel []:
  Téléphone personnel []:
  Autre []:
Cette information est-elle correcte ? [0/n]Y
Ajout du nouvel utilisateur « bob » aux groupes supplémentaires « users » ...
Ajout de l'utilisateur « bob » au groupe « users » ...
root@alice:~# _
```

De la même façon, sur la machine de Bob nous allons créer un utilisateur :

adduser alice

Et remplir toutes les informations et le mot de passe :

```
root@debian:~# adduser alice
Ajout de l'utilisateur « alice » ...
Ajout du nouveau groupe « alice » (1002) ...
Ajout du nouvel utilisateur « alice » (1002) avec le groupe « alice » (1002) ...
Création du répertoire personnel « /home/alice » ...
Copie des fichiers depuis « /etc/skel » ...
Nouveau mot de passe :
Retapez le nouveau mot de passe :
passwd : mot de passe mis à jour avec succès
Modifier les informations associées à un utilisateur pour alice
Entrer la nouvelle valeur, ou appuyer sur ENTER pour la valeur par défaut
  NOM []:
  Numéro de chambre []:
  Téléphone professionnel []:
  Téléphone personnel []:
  Autre []:
Cette information est-elle correcte ? [0/n]Y
Ajout du nouvel utilisateur « alice » aux groupes supplémentaires « users » ...
Ajout de l'utilisateur « alice » au groupe « users » ...
```

Nos utilisateurs étant créés, nous pouvons maintenant utiliser le protocole **SSH**.

Pour se faire, il faut installer sur chaque machine un serveur SSH grâce à la commande :

sudo apt install openssh-server

Un serveur SSH est maintenant présent sur chacune des machines. Alice et Bob peuvent donc se connecter mutuellement.

Prenons l'exemple de Bob qui voudrait se connecter sur la machine d'Alice.

Ce dernier va se rendre sur sa machine virtuelle, ouvrir un terminal et taper la commande suivante :

ssh bob@IP_ALICE

Lors de la toute première tentative de connexion, un avertissement va apparaître afin de savoir si la clé publique fournie par la machine d'Alice est de confiance et si vous acceptez de le signer. Vous pouvez simplement valider et ensuite entrer le mot de passe du compte **bob** sur la machine d'Alice :

```
root@debian:~# ssh bob@172.22.100.111
The authenticity of host '172.22.100.111 (172.22.100.111)' can't be established.
ED25519 key fingerprint is SHA256:nGJxGN5F1+uueeDKo1V2M61huupDkNa8hADJq0Jzk4k.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '172.22.100.111' (ED25519) to the list of known hosts.
bob@172.22.100.111's password:
Linux alice 6.1.0-32-arm64 #1 SMP Debian 6.1.129-1 (2025-03-06) aarch64

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
bob@alice:~$
```

Bob est maintenant connecté sur l'utilisateur **bob** sur la machine d'Alice !

Maintenant, Alice va se connecter au compte **alice** sur la machine de Bob, elle va taper la commande :

ssh alice@IP_BOB

De la même façon, Alice reçoit un avertissement sur la clé publique de la machine de Bob et elle se connecte :

```
root@alice:~# ssh alice@172.22.100.103
The authenticity of host '172.22.100.103 (172.22.100.103)' can't be established.
ED25519 key fingerprint is SHA256:aLsDpo8s/0hq8YQPnqp8jxzz9Csy7t0ZVXDI2uMJXzw.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '172.22.100.103' (ED25519) to the list of known hosts.
alice@172.22.100.103's password:
Linux debian 6.1.0-32-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.129-1 (2025-03-06) x86_64

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
alice@debian:~$ _
```

III - OpenSSL

A - Histoire

OpenSSL est un logiciel embarquant des outils de chiffrement et de déchiffrement qui a été créé en 1998. Le logiciel a été créé sur la base d'une bibliothèque nommée **SSLeasy** développée par Eric A. Young et Tim J. Hudson. Il est développé afin de pouvoir mettre à disposition une suite d'outils de chiffrement libre, donc à source ouverte, sous une interface de ligne de commande.

Aujourd'hui, c'est l'un des logiciels les plus utilisés dans le milieu du chiffrement et notamment dans la génération de certificats **SSL/TLS** (Secure Socket Layer / Transport Layer Security) permettant de sécuriser les communications sur le Web.

B - Objectif

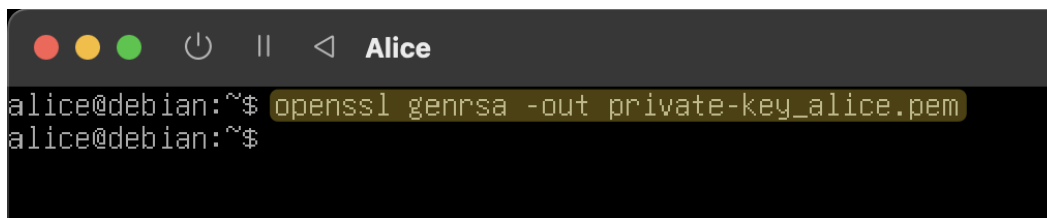
OpenSSL inclut plusieurs catégories d'outils. Parmi eux, nous retrouvons la génération de certificats SSL et clés asymétriques mais aussi le chiffrement et déchiffrement de données. De plus, le logiciel **OpenSSL** est réputé pour embarquer une grande quantité d'algorithmes de chiffrement / déchiffrements différents et permet donc de répondre à une demande large de la part des utilisateurs.

Pour rappel, le principe d'un jeu de clé publique / privée est de pouvoir chiffrer un message grâce à une clé publique, accessible pour tous, et qui ne peut être déchiffré que par la clé privée, qui n'est accessible qu'à son propriétaire.

C - Paramétrage

Afin qu'Alice et Bob puissent communiquer des informations sans risque de fuite / vol de données, ils vont chacun créer un jeu de clé publique / privée, sur leur machine respective. Tout d'abord, Alice va créer une clé privée avec la commande :

openssl genrsa -out private-key_alice.pem



```
Alice
alice@debian:~$ openssl genrsa -out private-key_alice.pem
alice@debian:~$
```

Cette commande va donc, si nous regardons argument par argument, créer une clé, à l'aide de l'algorithme **RSA**, dans un fichier, qui va se nommer `private-key_alice.pem`, dans le dossier ouvert.

Ensuite, Alice va créer sa clé publique, elle a donc besoin de faire la commande :

openssl rsa -pubout -in private-key_alice.pem -out public-key_alice.pem

```
alice@debian:~$ openssl rsa -pubout -in private-key_alice.pem -out public-key_alice.pem
writing RSA key
alice@debian:~$ _
```

Ici, la commande va créer une clé publique, basée sur notre précédente clé privée et la placer dans le fichier public-key_alice.pem.

Pour vérifier la création de nos clés privées et publiques, on peut lister le contenu de notre dossier personnelle (/home) avec la commande "ls" :

```
root@alice:~# ls
private-key_alice.pem  public-key_alice.pem
root@alice:~# _
```

Maintenant, nous allons effectuer les mêmes commandes du côté de la machine de Bob pour que lui aussi puisse avoir un jeu de clés :

```
openssl genrsa -out private-key_bob.pem
openssl rsa -pubout -in private-key_bob.pem -out public-key_bob.pem
ls
```

Nous avons donc ce résultat :

```
root@debian:~# openssl genrsa -out private-key_bob.pem
root@debian:~# openssl rsa -pubout -in private-key_bob.pem -out public-key_bob.pem
writing RSA key
root@debian:~# ls
private-key_bob.pem  public-key_bob.pem
root@debian:~#
```

Nos deux utilisateurs ont maintenant un jeu de clé publique / privée et ils peuvent ainsi communiquer de manière sécurisée.

IV - SCP

A - Histoire

Le protocole **SCP** (Secure Copy Protocol) a été conçu en 1983 dans l'objectif de créer un protocole de transmission de fichiers entre des ordinateurs, basé sur le protocole **SSH**. En effet, **SCP** utilise le serveur SSH tournant sur les machines pour communiquer. Ce protocole a été déprécié au profit du **SFTP** qui permet une meilleure gestion de ses espaces de fichiers et des transferts de fichiers plus robustes.

B - Objectif

L'objectif de ce protocole est donc de transmettre des fichiers de pair à pair de façon sécurisée, en se basant sur le **SSH**. Aujourd'hui, il est utilisé pour faire du transfert direct de fichier, notamment des fichiers simples et légers.

C - Paramétrage

En sachant que le **SCP** se base sur le protocole **SSH** et que Alice et Bob possèdent déjà un serveur **SSH** chacun fonctionnel, avec des utilisateurs qui leurs sont assignés. Il n'y a pas de configuration supplémentaire à apporter dans notre contexte.

Afin de tirer profit du **SCP**, Alice et Bob vont se transmettre leur clé publique pour, à terme, pouvoir chiffrer des messages et se les envoyer sans risques.

Du côté de Bob, il va pouvoir envoyer sa clé publique dans le dossier de son utilisateur du côté de la machine d'Alice avec la commande :

scp public-key_bob.pem bob@IP_ALICE:public-key_bob.pem

```
root@debian:~# scp public-key_bob.pem bob@172.22.103.132:public-key_bob.pem
bob@172.22.103.132's password:
public-key_bob.pem
root@debian:~# _
```

De la même façon, Alice va envoyer sa clé publique dans le dossier de son utilisateur sur la machine de Bob avec :

scp public-key_alice.pem alice@IP_BOB:public-key_alice.pem


```
root@alice:~# ls
private-key_alice.pem  public-key_alice.pem
root@alice:~# scp public-key_alice.pem alice@172.22.103.126:public-key_alice.pem
The authenticity of host '172.22.103.126 (172.22.103.126)' can't be established.
ED25519 key fingerprint is SHA256:aLsDpo8s/0hq8YQFnqp8jxzz9Csy7tCZVXDIZuMJXzw.
This host key is known by the following other names/addresses:
  ~/.ssh/known_hosts:1: [hashed name]
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '172.22.103.126' (ED25519) to the list of known hosts.
alice@172.22.103.126's password:
Permission denied, please try again.
alice@172.22.103.126's password:
Permission denied, please try again.
alice@172.22.103.126's password:
public-key_alice.pem
root@alice:~#
```

Maintenant que Alice et Bob possèdent les clés publiques de l'un et l'autre, ils vont pouvoir se transmettre des messages chiffrés sans risque qu'on puisse comprendre leurs transmissions. Tout d'abord, Alice va envoyer un message à Bob dans un fichier appelé **msg_alice.txt**, pour se faire elle va créer et éditer le contenu du fichier avec :

nano msg_alice.txt

```
root@alice:~# echo Hello World > msg_alice.txt
root@alice:~# ls
msg_alice.txt  private-key_alice.pem  public-key_alice.pem
root@alice:~# cat msg_alice.txt
Hello World
root@alice:~# _
```

Et elle va écrire un "Hello World" dans son fichier.

Ensuite, pour sécuriser le message, Alice va chiffrer le fichier grâce à la clé publique de Bob avec la commande :

openssl pkeyutl -encrypt -in msg_alice.txt -pubin -inkey /home/bob/public-key_bob.pem -out msg_alice.enc

```
root@alice:~# openssl pkeyutl -encrypt -in msg_alice.txt -pubin -inkey /home/bob/public-key_bob.pem -out msg_alice.enc
root@alice:~# ls
msg_alice.enc  msg_alice.txt  private-key_alice.pem  public-key_alice.pem
root@alice:~# _
```

Si on regarde la commande, argument par argument, Alice demande à l'utilitaire **OpenSSL** d'utiliser son outil de clé privé, en chiffrant un fichier appelé **msg_alice.txt**, à partir d'une clé publique (celle de Bob), et de sortir le résultat dans un fichier appelé **msg_alice.enc**. Ensuite, Alice va envoyer son message chiffré à Bob à travers le protocole **SCP** avec la commande :

scp msg_alice.enc alice@IP_BOB:msg_alice.enc

```
root@alice:~# scp msg_alice.enc alice@172.22.103.126:msg_alice.enc
alice@172.22.103.126's password:
msg_alice.enc
root@alice:~# _
```

Bob va pouvoir ainsi le déchiffrer et voir son contenu avec la commande :

```
openssl pkeyutl -decrypt -in /home/alice/msg_alice.enc -inkey  
private-key_bob.pem -out msg_alice.txt
```

```
root@debian:~# openssl pkeyutl -decrypt -in /home/alice/msg_alice.enc -inkey private-key_bob.pem -out msg_alice.txt
root@debian:~# ls
msg_alice.txt  msg_bob.enc  msg_bob.txt  private-key_bob.pem  public-key_bob.pem  snap
root@debian:~#
```

Cette commande, argument par argument, va permettre à Bob de déchiffrer avec l'utilitaire de clé de **OpenSSL** le message d'Alice chiffré, en utilisant sa clé privée, et sortir le contenu dans un fichier appelé **msg_alice.txt**.

Ainsi, il peut consulter le contenu du message déchiffré avec :

```
cat msg_alice.txt
```

```
root@debian:~# openssl pkeyutl -decrypt -in /home/alice/msg_alice.enc -inkey private-key_bob.pem -out msg_alice.txt
root@debian:~# ls
msg_alice.txt  msg_bob.enc  msg_bob.txt  private-key_bob.pem  public-key_bob.pem  snap
root@debian:~# cat msg_alice.txt
Hello World
root@debian:~# _
```

Bob peut également envoyer un message à Alice, sous la même forme il va créer un fichier appelé **msg_bob.txt** avec comme contenu "Au revoir." et le chiffrer avec la clé publique d'Alice :

```
nano msg_bob.txt
```

```
GNU nano 7.2
Au revoir.
```

Puis il va chiffrer le message avec la clé publique d'Alice :

```
openssl pkeyutl -encrypt -in msg_bob.txt -pubin -inkey  
/home/alice/public-key_alice.pem -out msg_bob.enc
```

```
root@debian:~# openssl pkeyutl -encrypt -in msg_bob.txt -pubin -inkey /home/alice/public-key_alice.pem -out msg_bob.enc
root@debian:~# ls
msg_bob.enc  msg_bob.txt  private-key_bob.pem  public-key_bob.pem  snap
root@debian:~# _
```

Il va ainsi l'envoyer à Alice avec le protocole **SCP** :

```
scp msg_bob.enc bob@IP_ALICE:msg_bob.enc
```

```
root@debian:~# scp msg_bob.enc bob@172.22.103.132:msg_bob.enc
bob@172.22.103.132's password:
msg_bob.enc
root@debian:~# _
```

Une fois reçu, Alice va pouvoir le déchiffrer avec la commande :

**openssl pkeyutl -decrypt -in /home/bob/msg_bob.enc -inkey
private-key_alice.pem -out msg_bob.txt**

```
Alice
root@alice:~# openssl pkeyutl -decrypt -in /home/bob/msg_bob.enc -inkey private-key_alice.pem -out msg_bob.txt
root@alice:~# ls
msg_alice.enc  msg_alice.txt  msg_bob.txt  private-key_alice.pem  public-key_alice.pem
root@alice:~#
```

Et consulter le contenu du message déchiffré avec :

cat msg_bob.txt

```
Alice
root@alice:~# cat msg_bob.txt
Au revoir.
root@alice:~# _
```

V - Conclusion

Pour conclure, Alice et Bob peuvent maintenant communiquer grâce aux protocoles **SSH / SCP** et l'utilitaire **OpenSSL**.

SSH permettant à l'un et l'autre d'avoir un accès à un **Shell** virtuel sur la machine de l'autre, **SCP** qui permet sur la base du **SSH** de transférer des fichiers de manière sécurisée et **OpenSSL** avec l'algorithme **RSA** pour chiffrer et déchiffrer des fichiers ou contenu afin que seul le destinataire puisse les déchiffrer.