

Fiche de lecture

Initiation au Protocole de communication SSH


PROPENTA TECH SAVING SOLUTIONS

20 Octobre 2024



Sommaire

1. Introduction au Protocole SSH
2. Fonctionnement du Protocole SSH
3. Installation et Utilisation de SSH sur Ubuntu et avec GitHub
 - Installation de SSH sur Ubuntu
 - Comment utiliser SSH pour se connecter à un serveur distant
 - Utilisation de SSH avec GitHub



Le **protocole SSH (Secure Shell)**, c'est un peu comme une porte magique qui te permet d'accéder à un autre ordinateur à distance, en toute sécurité.

Imagine ça :

Disons que tu veux entrer dans la maison d'un ami. Si la porte est ouverte à tout le monde, n'importe qui pourrait entrer, y compris des voleurs. C'est un risque ! Maintenant, ton ami te donne une **clé spéciale**, que toi seul peux utiliser pour entrer chez lui. C'est un moyen sûr de vérifier que c'est bien toi, et non quelqu'un d'autre.

En informatique, SSH fonctionne un peu de la même façon :

- Tu veux **te connecter à un autre ordinateur ou un serveur** (peut-être dans un autre pays !).
- Tu vas utiliser un **identifiant** et une **clé** (ou un mot de passe) pour prouver que tu es autorisé à entrer.
- SSH s'assure que **personne ne peut écouter ou voir ce que tu fais**, même si tu es sur un réseau public comme le Wi-Fi.

Maintenant que tu vois SSH comme une sorte de "porte magique" sécurisée, voyons un peu plus en détail **comment il fonctionne réellement**. Si tu souhaites travailler en administration système ou dans le développement, il est important de bien comprendre ce qui se passe en arrière-plan.

Introduction au Protocole SSH

Avant d'explorer en détail le protocole **SSH (Secure Shell)**, il est essentiel de comprendre ce qu'est un **protocole** en général.

Qu'est-ce qu'un protocole ?

Un **protocole** est un ensemble de règles et de conventions qui régissent la manière dont des appareils ou des systèmes échangent des informations. Ces règles définissent les éléments suivants :

- **Le format des données** transmises.
- **L'ordre des échanges** entre les parties (client et serveur, par exemple).
- **Les méthodes d'authentification** pour vérifier l'identité des interlocuteurs.
- **Les mécanismes de gestion des erreurs** pour assurer la fiabilité des transmissions.


Ces protocoles garantissent l'interopérabilité entre des systèmes divers, permettant à des technologies variées de fonctionner ensemble sans conflit. Parmi les protocoles les plus courants figurent **HTTP**, **FTP**, et **SSH**.

Présentation du Protocole SSH

SSH (Secure Shell) est un protocole de communication sécurisé conçu pour offrir un accès à distance et un contrôle sécurisé des systèmes sur un réseau. Il a été introduit pour remplacer des méthodes moins sûres telles que **Telnet** et **FTP**, qui transmettent les données en clair, exposant ainsi les informations sensibles à des attaques.

SSH assure à la fois :

1. **L'authentification** sécurisée entre un client et un serveur.

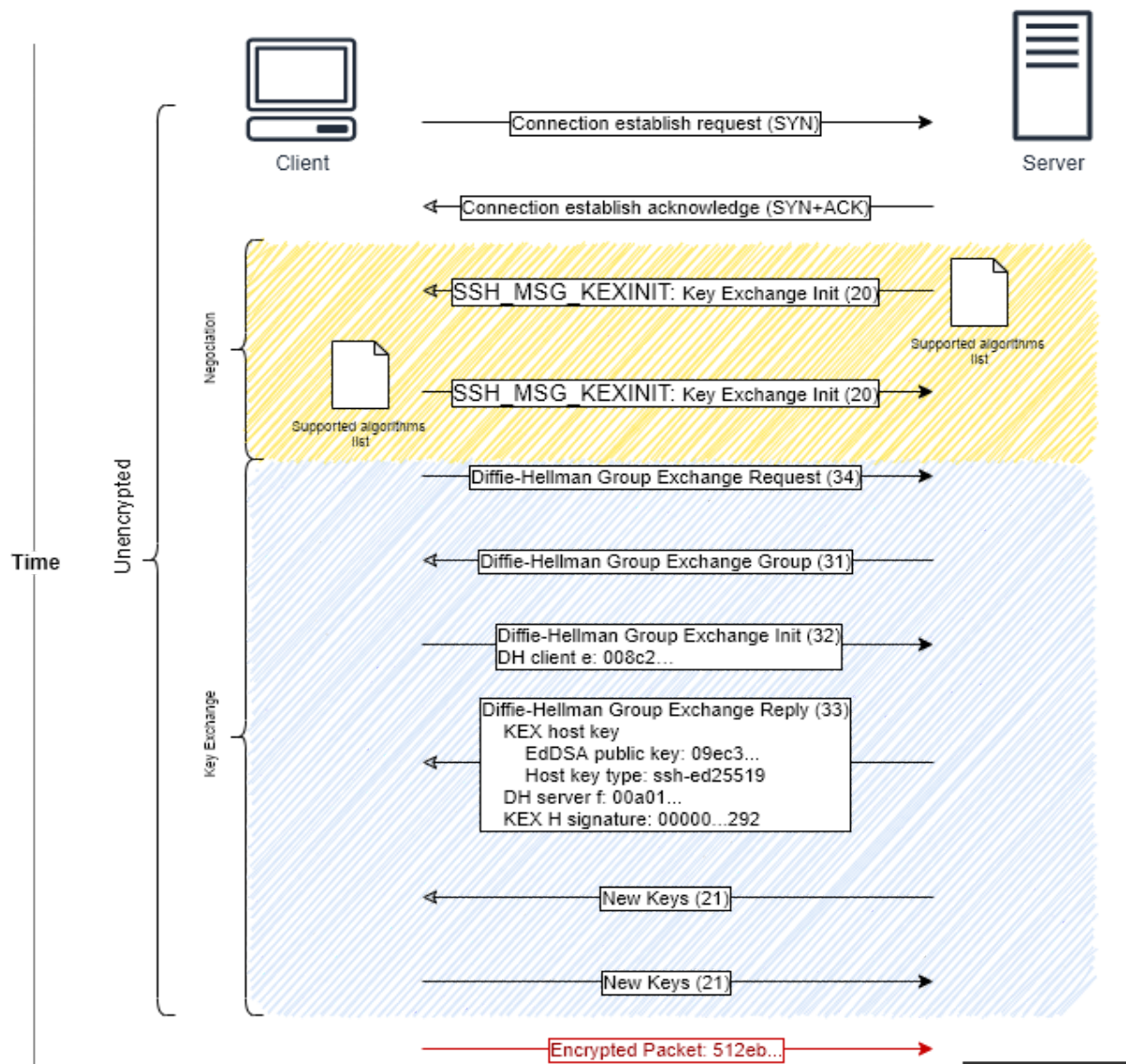
- 
2. **Le chiffrement** des données échangées, garantissant que les communications ne puissent être interceptées ou lues par des tiers.
 3. **L'intégrité** des messages échangés, pour vérifier que les données ne sont pas altérées pendant le transfert.

SSH n'est pas seulement un protocole mais aussi une **suite d'outils** permettant :

- L'exécution de commandes à distance.
 - Le transfert sécurisé de fichiers.
 - L'établissement de tunnels sécurisés pour protéger d'autres communications.
-

Fonctionnement du Protocole SSH

Le processus de connexion SSH repose sur un **échange sécurisé entre un client et un serveur**. Voici les étapes détaillées d'une session SSH :



1. Initiation de la connexion

- Le **client SSH** (par exemple, un terminal utilisateur) envoie une demande de connexion au **serveur SSH**.
- Le serveur écoute généralement sur le **port 22**, qui est le port par défaut du protocole SSH.

2. Négociation des algorithmes

- Lors de la réception de la demande, le serveur et le client **négocient plusieurs paramètres** :
 - Les **algorithmes de chiffrement** (ex. AES, ChaCha20).
 - Les **algorithmes d'authentification** (ex. RSA, Ed25519).
 - Les méthodes de vérification de l'intégrité des messages (ex. SHA-256).

Cette négociation permet aux deux parties de s'accorder sur des algorithmes compatibles pour la session.

3. Authentification du serveur

- Une fois les algorithmes choisis, le serveur envoie sa **clé publique** au client.
- Le client compare cette clé à une copie déjà enregistrée (généralement dans le fichier **known_hosts**).
 - **Si la clé est valide**, la connexion se poursuit.
 - **Si la clé est inconnue ou modifiée**, une alerte est déclenchée, indiquant un possible piratage (attaque de type **man-in-the-middle**).

4. Authentification du client

- Ensuite, le client doit prouver son identité. Il peut s'authentifier via :
 - **Un mot de passe.**
 - **Une clé privée** (si une clé publique a été préalablement installée sur le serveur).
- Si la clé privée est utilisée, elle est comparée à la **clé publique** correspondante enregistrée sur le serveur. Si elles correspondent, l'accès est accordé sans nécessité d'un mot de passe.

5. Établissement du canal sécurisé

- Une fois l'authentification réussie, le serveur et le client **génèrent une clé de session temporaire**.

- Cette clé sert à **chiffrer toutes les communications** entre le client et le serveur jusqu'à la fin de la session.


6. Échange de données sécurisées

- Après l'établissement du canal sécurisé, le client peut :
 - Exécuter des **commandes à distance** sur le serveur.
 - **Transférer des fichiers** en toute sécurité à l'aide de protocoles comme **SCP** ou **SFTP**.
 - **Créer des tunnels SSH** pour sécuriser d'autres communications.

7. Clôture de la session

- À la fin de la session, le client ou le serveur peut **fermer la connexion**.
- Les clés de session temporaires sont supprimées, garantissant que chaque connexion ultérieure nécessite un nouvel échange sécurisé.

Installation et Utilisation de SSH sur Ubuntu et avec GitHub



Après avoir compris le fonctionnement de **SSH**, voyons comment l'installer sur un système **Ubuntu** et l'utiliser pour établir une connexion sécurisée sur une machine distante et avec **GitHub**.

Installation de SSH sur Ubuntu

Voyons maintenant comment installer et configurer **SSH** sur Ubuntu.

Étape 1 : Mise à jour des paquets

Avant toute installation, mettez à jour les paquets du système :

```
sudo apt update && sudo apt upgrade -y
```

Étape 2 : Installation du serveur OpenSSH

Installez le paquet OpenSSH, qui contient le serveur et le client SSH :

```
sudo apt install openssh-server -y
```

Étape 3 : Vérification du service SSH

Assurez-vous que le service SSH est actif :

```
sudo systemctl status ssh
```

- Si le service est inactif :

```
sudo systemctl start ssh
```

- Pour activer SSH au démarrage :

```
sudo systemctl enable ssh
```

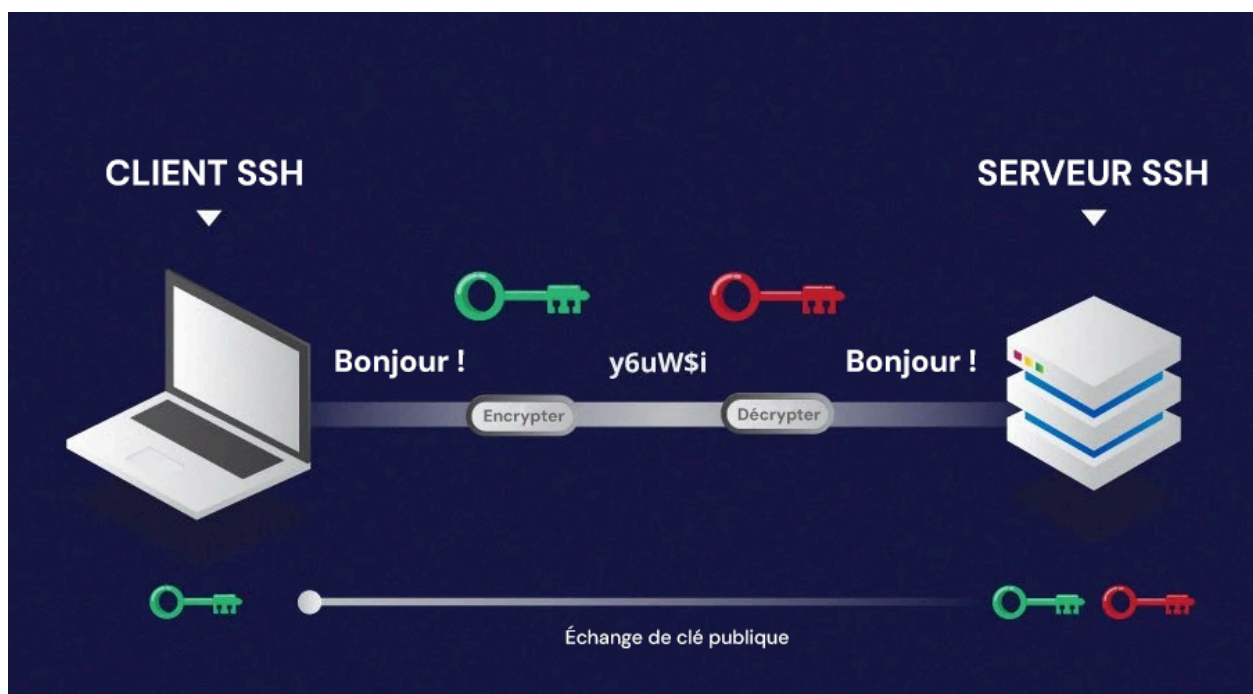
Étape 4 : Configurer le pare-feu (ufw)

Autorisez le port 22 utilisé par SSH dans le pare-feu :

```
sudo ufw allow ssh
```

```
sudo ufw reload
```

Comment utiliser SSH pour se connecter à un serveur distant



Une clé privée , ca ressemble à ceci :

-----BEGIN OPENSSH PRIVATE KEY-----

b3BlbnNzaC1rZXktdjEAAAABG5vbmUAAAABbm9uZQAAAAAAAAABAAAAMwAAAAAtzc2gtZW

QyNTUxOQAAACATbd0/42RFdVJRftYdqw4igZeD1Bp/MZUZBXjCJYgjlQAAAJhFXOsVRVzr

FQAAAAAtzc2gtZWQyNTUxOQAAACATbd0/42RFdVJRftYdqw4igZeD1Bp/MZUZBXjCJYgjlQ

AAAEAcHpM/LBPhy5M82GUSIUB0XQ5xjNJQKg5dmcWp5K7PZxNt3T/jZEV1UIF+1h2rDiKB

I4PUGn8xlRkFeMlliCOVAAAADmd1aWxoZW1Ac3NsdnBuAQIDBAUGBw==

-----END OPENSSH PRIVATE KEY-----

La clé publique est stockée dans un fichier .pub créé lors de la génération de clé et ressemble à :

ssh-ed25519

AAAAC3NzaC1lZDI1NTE5AAAAIBNt3T/jZEV1UIF+1h2rDiKB I4PUGn8xlRkFeMlliCOV root@wikipedia

Il est important que vous le sachiez car nous ne cesseront d'évoquer ces termes au cours de notre apprentissage.

Dans ce guide, nous verrons comment les utiliser pour se connecter à un système distant.

Pré-requis :

1. Un accès à la machine distante (adresse IP ou domaine).
2. Un utilisateur avec un compte sur cette machine.
3. **SSH** doit être installé et activé sur les deux machines (client et serveur).

Étape 1 : Connexion par mot de passe (option rapide)

Si vous ne souhaitez pas utiliser de clé SSH, essayez d'abord une **connexion basique** par mot de passe.

ssh user@ip_serveur

Remplacez **user** par l'utilisateur de la machine distante et **ip_serveur** par son adresse IP.

Exemple :

ssh alice@192.168.1.10


- **Si demandé**, entrez le mot de passe de l'utilisateur :

alice@192.168.1.100's **password:** *****

Étape 2 : Générer une paire de clés SSH (méthode recommandée)

Sur la **machine client**, générez une paire de clés.

ssh-keygen -t rsa -b 4096



Appuyez sur Entrée pour accepter les chemins par défaut (`~/ .ssh/id_rsa`).

Si vous souhaitez, définissez une **passphrase** ou laissez vide.

Étape 3 : Copier la clé publique sur la machine distante

Pour se connecter sans mot de passe, transférez la **clé publique** vers le serveur distant.

ssh-copy-id user@ip_serveur

Exemple :

ssh-copy-id alice@192.168.1.100

Cela ajoutera votre **clé publique** au fichier `~/ .ssh/authorized_keys` sur la machine distante.

Étape 4: Connexion sans mot de passe

Maintenant que la clé publique est en place, essayez de vous connecter sans mot de passe.

ssh user@ip_serveur

Étape 7 : Déconnexion

Lorsque vous avez fini, tapez :

exit

Vous voilà prêt ! 🎉 Vous avez appris à vous connecter à une machine distante via SSH en utilisant une **connexion sécurisée avec clés SSH**.

Utilisation de SSH avec GitHub

GitHub permet d'utiliser une **authentification par clé SSH** au lieu de mots de passe pour sécuriser les interactions avec les dépôts.

Étape 1 : Génération d'une paire de clés SSH

Pour générer une nouvelle paire de clés :

ssh-keygen -t rsa -C "votre_email@example.com"

- **Emplacement des fichiers** : Par défaut, les clés seront générées dans `~/.ssh/`.
 - **Fichiers générés** :
 - `id_ed25519` (ou `id_rsa`) : Clé privée à garder secrète.
 - `id_ed25519.pub` (ou `id_rsa.pub`) : Clé publique à partager.
-

Étape 2 : Ajouter la clé publique à GitHub

1. **Afficher et copier la clé publique** :

cat ~/.ssh/id_rsa.pub

2. **Ajouter la clé sur GitHub** :

- Connectez-vous à **GitHub**.
- En haut à droite, cliquez sur votre **photo de profil**, puis allez dans **Settings**.
- Dans le menu de gauche, cliquez sur **SSH and GPG keys**.

- Cliquez sur **New SSH key**.
- Donnez un nom à la clé (par exemple, "Clé SSH Ubuntu").
- **Collez la clé publique** dans le champ prévu.
- Cliquez sur **Add SSH key**.
- Si vous avez activé **l'authentification à deux facteurs**, entrez votre code de sécurité.

après quoi tous vos transferts de fichiers sur github seront sécurisés par le protocole **ssh**

Étape 3 : Tester la connexion SSH avec GitHub

Dans le terminal, testez la connexion avec la commande suivante :

ssh -T git@github.com

- **Message attendu :**

Hi username! You've **successfully authenticated, but GitHub does not provide shell access.**

Ce message indique que la connexion SSH avec GitHub est correcte.



Conclusion

Le protocole **SSH** joue un rôle essentiel dans la sécurisation des communications réseau, particulièrement dans l'administration à distance des serveurs. Grâce à son **chiffrement robuste** et à ses **méthodes d'authentification avancées**, SSH est aujourd'hui un standard incontournable pour toute connexion à distance nécessitant sécurité et fiabilité. Son utilisation va bien au-delà de l'exécution de commandes, couvrant également des fonctions de **transfert sécurisé de fichiers** et de **tunneling**.

Ce processus garantit non seulement que les données échangées ne puissent être interceptées, mais aussi que l'identité des parties soit vérifiée, rendant SSH indispensable dans des environnements informatiques modernes.

