Jug Summer Camp

-enjoy it-

👨‍🎓 Back to basics – J'ai une 🔑 clé SSH 💻, et maintenant 🤷‍♂️ ?

Speaker : Laurent Grangeau - @laurentgrangeau

Speaker : Ludovic Piot - @lpiot

SERLI DARVA CleverConnect elastic Doctolib MAIF Cloud APIM clever cloud

Qui sommes-nous ?

Laurent Grangeau

Solutions Architect

@ Google

@laurentgrangeau

Ludovic Piot

CTO

@ Duonyx

@lpiot

SSH *késako ?*

fonctionnalités
cas d'usage

SSH origin story

SSH = **S**ecure **SH**ell

Créé en 1995 par 🇫🇮Tatu Ylönen.
v2.0 en 2006 par l'**IETF** (Internet Engineering Task Force).
RFC-4251, 8308, 9141

Protocole de communication entre 2 ordinateurs :
- client (initiateur de la connexion)
- serveur (destinataire de la connexion)

S'appuie sur TCP/22

Standard d'implémentation :
- programme client (**openSSH**, **PuTTY**)
- daemon serveur (**sshd**)

Des compétiteurs émergent :
- MoSH (mobile, réseau intermittent)
- Teleport (approche moderne/Web)

SSH main features

SSH = **S**ecure **SH**ell

Protocole de communication entre 2 ordinateurs :
- client (initiateur de la connexion)
- serveur (destinataire de la connexion)

Sécurisé. Remplace **rlogin**, **telnet**...

Shell =
- 👍 envoyer des commandes shell à exécuter sur le serveur (**ssh**)
  - et récupérer les résultats des sorties standard
- sécuriser FTP (**sFTP**)
- faire transiter des fichiers (**scp**, **rsync**) vers et depuis le serveur
- faire transiter des flux de données du serveur vers le client (**ssh tunneling**)
- faire transiter des flux de données du client vers un tiers via le serveur (**ssh forward**)
- exposer un filesystem distant (**sshfs**)

Performant

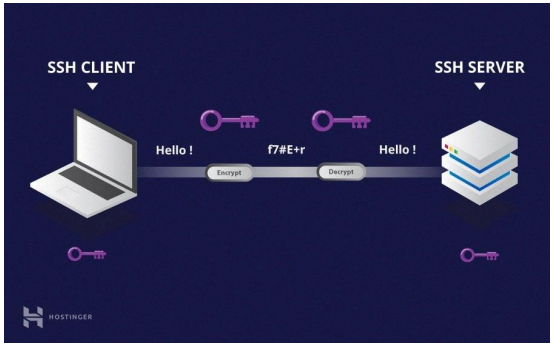
Architecture en couches permet de faire évoluer la force de sécurisation avec le temps

SSH kesako ?

SSH = **S**ecure **SH**ell

Secure =

- négociation des protocoles de chiffrement entre client et serveur
- authentification des client et serveur (par chiffrement **asymétrique**)
- tunnel chiffré des flux de données en transit (par chiffrement **symétrique**)
- identification 😒 du serveur (**sshfp**)
- intégrité des données (par hash des données)



3 couches de sécurité :

transport layer	authentification du serveur confidentialité intégrité
user authentication	l'utilisateur est authentifié par le serveur
connection protocol	tunnel de communication chiffré canaux de communication logiques multiples

cryptographie *koikoubeh ?*

symétrique / asymétrique
RSA, DSA, EDCSA, EdDSA

Mettons-nous bien d'accord !

Chiffrement symétrique

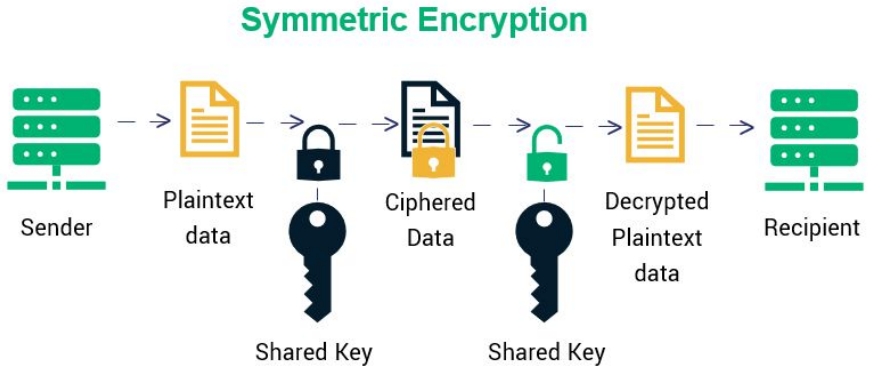
Objectif : transformer la donnée en une donnée chiffrée **irrécupérable sans la clé** de déchiffrement

Une seule clé utilisée
- par l'expéditeur pour chiffrer
- par le destinataire pour déchiffrer

Avantages :
- implémentations relativement simples
- Algo utilisé pour le chiffrement / algo^{-1} pour le déchiffrement
- Perfs équivalentes en chiffrement / déchiffrement
- Algos câblés dans les processeurs

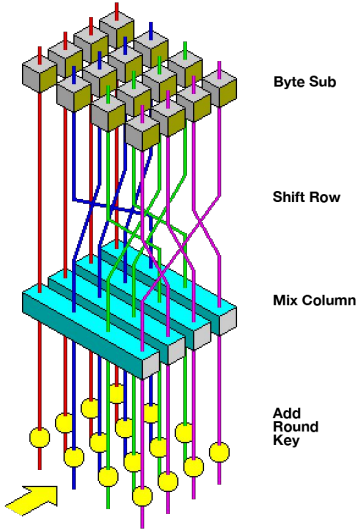
Inconvénient :
- toute la sécurité repose sur un secret… partagé

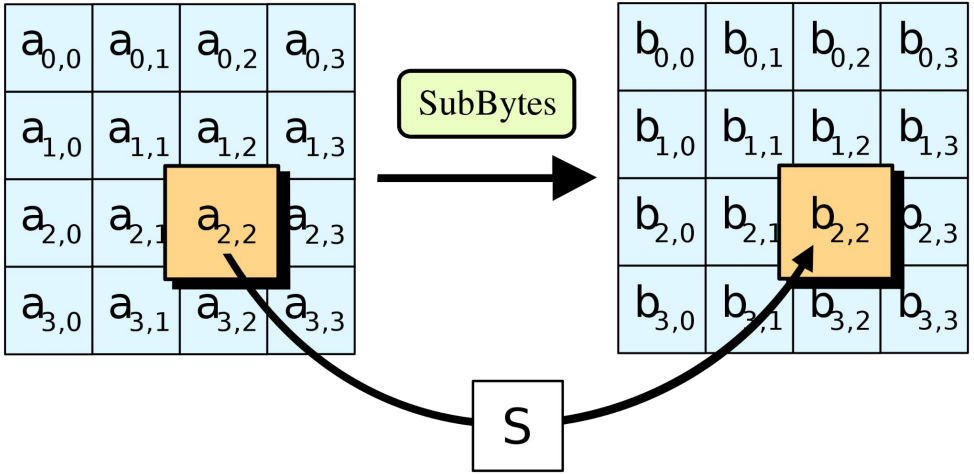
Type d'attaques : brute force



Chiffrement symétrique

Différents algorithmes de chiffrement (**cipher**) : AES, 3DES, TwoFish, ChaCha20...
Différentes familles : stream cipher, block cipher





Basés sur des permutations de blocs avec injection de la clé en multiples instances...

Chiffrement asymétrique

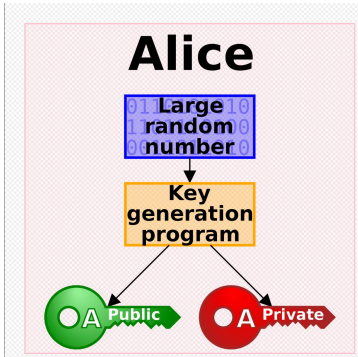
Objectif : **ne pas partager** la clé de chiffrement/déchiffrement

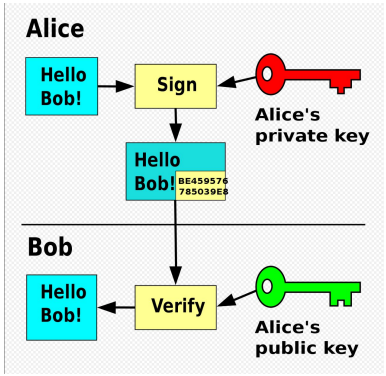
La clé privée n'est jamais partagée
- elle permet de forger une/des clés publiques (paires de clés)
- seule la clé privée peut déchiffrer un ciphertext produit par une clé publique
- la clé publique ne peut pas déchiffrer un ciphertext produit par elle même, ni par la clé privée

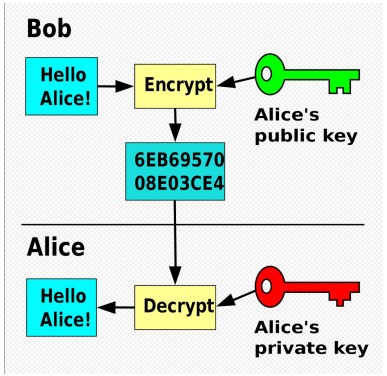
Avantages :
- mieux adapté à la coopération massive autour d'un sujet de connection sécurisée
- la clé privée unique est un moyen d'authentification naturel

Inconvénients :
- algorithmes plus complexes (et plus lents)
- nécessité de diffuser la clé publique
- la fuite de la clé privée réduit la sécurité à néant

Type d'attaque : déduire la clé privée depuis la clé publique







Algos de chiffrement asymétrique

Différents algos :

- RSA (Rivest–Shamir–Adleman)
- DSA (Digital Signature Algorithm)
- ECDSA/EdDSA (Elliptic Curve Digital Signature Algorithm)
- EdDSA (*Edwards-curve Digital Signature Algorithm*)
- ED25519 (twisted *Edwards-curve Digital Signature Algorithm*)

💡Hé oui ! C'est ce que l'on définit quand on produit les paires de clés

RSA
Factorisation de nombres entiers

n = p * q

n : modulus (eq. clé publique)
p, **q** : 2 nombres premiers suffisamment grands et suffisamment éloignés l'un de l'autre (eq. clé privée)

Plus **n** est grand et plus difficile est la déduction de **p** et **q**
Assez adapté à contrer la loi de Moore et la brute force

Aujourd'hui, clés privées de 2048/4096 sont encore valables.

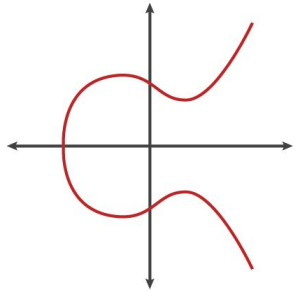
DSA
Problème du logarithme discret
Résolution par exponentiation modulaire

🤫

Pour obtenir un **ciphertext**, on utilise un nombre aléatoire conjointement à la clé privée.

Si ce nombre est découvert, la clé privée est compromise.

La randomisation sur ordinateur n'est pas assez aléatoire.
La brute force a eu raison de cette implémentation. (PS3 encryption keys)



ECDSA / EdDSA / ED25519
Problème du logarithme discret
Résolution par courbe elliptique

Algo très performant
Clés bcp plus petites

Toutes les courbes elliptiques n'offrent pas les mêmes niveaux de sécurité

👎 ECDSA

👍 EdDSA / Ed25519 (Courbe d'Edwards tordue)

Clés formats de stockage

Différents formats de stockage des clés

RFC4716
SSH public key format

IETF standard

Ne stocke que les clés publiques

Assez proche de PEM, mais sans CRC en fin de ligne
Des headers…
Un format de section différent…

PEM
Privacy **E**nhanced **M**ail

Format historique de stockage de clés publique/privée / certificats

Encodage base64.

PKCS8
Public-**K**ey **C**ryptography **S**tandards

Permet de stocker la clé privée sous forme chiffrée ou en clair.

Identifie explicitement l'algo de génération de la clé

Permet de stocker des certificats

Permet de stocker des paires de clés privée/publique

PPK
PuTTY **P**rivate **K**ey

Format propriétaire PuTTY

PuTTY-gen, utilitaire de génération de clés / conversion de format de clés

Permet de stocker la clé privée de manière chiffrée.
Gestion des caractères de fin de ligne (Windows / Linux) 🙄

ssh-keygen demo time!

```
[fedora@guimbarde .ssh]$ ssh-keygen -t rsa -b 4096
-N hellojugsummercamp -f jugsummercamp-rsa-key
Generating public/private rsa key pair.
Your identification has been saved in
jugsummercamp-rsa-key
Your public key has been saved in
jugsummercamp-rsa-key.pub
The key fingerprint is:
SHA256:2ui1Vvf76v/eUPkyW4GN8QY6Z+zC7YX+U8pGZIwWyUs
fedora@guimbarde.thegaragebandofit.com
The key's randomart image is:
+---[RSA 4096]----+
|            . .   |
|             E    |
|            . B   |
|             * @ .|
|        S  + O *.|
|       +  o B +.+|
|      o o. + B+o+|
|     . ...  + *B.|
|      ...    =*BO|
+----[SHA256]-----+
```

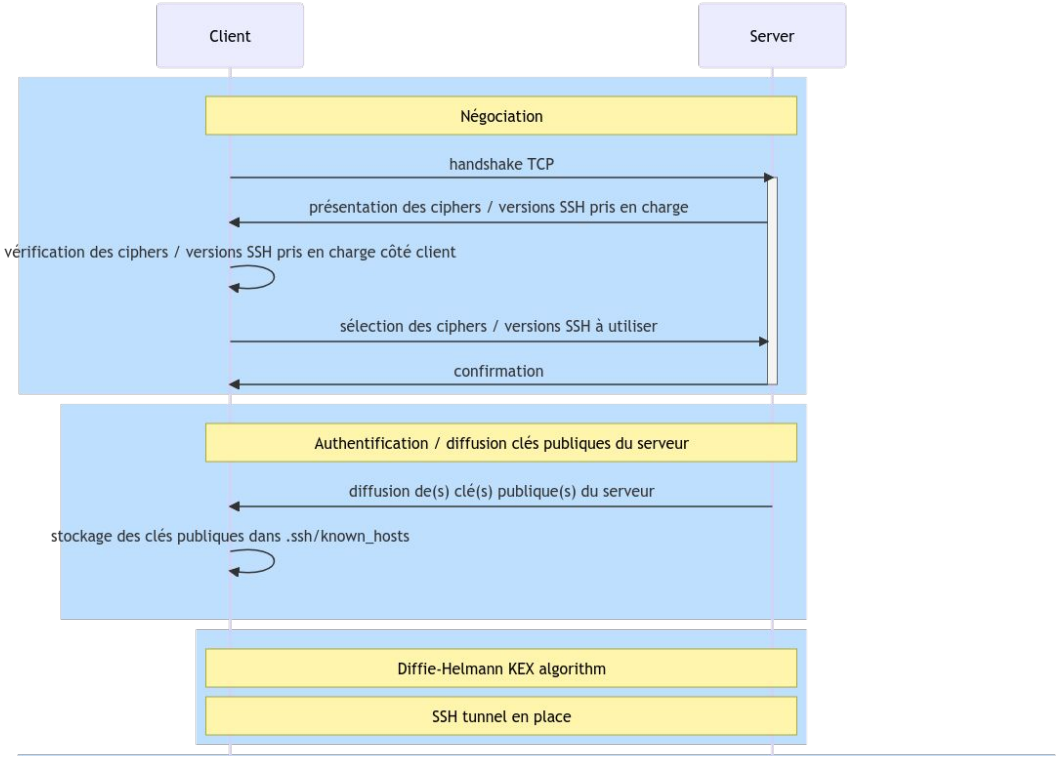
```
[fedora@guimbarde .ssh]$ ssh-keygen -e -f
jugsummercamp-rsa-key.pub -m pem
-----BEGIN RSA PUBLIC KEY-----
MIICCgKCAgEA4WasILNm/wEVvQ3S7Tp0yaqrnxL5X8OGUbTIyZcsy9DyBpfvh8oo
KxLXdafCP55Zz9CKEDTAwpKn14CbzB+w9DUG5/ANigBCSCO1NTIFbCJtlHgxjGpw
k8Kx+I9ytdvkl2kFYMX9PeNpZrs2gacw2+t0ABQoXSsK/khU6UK1tAb8XuM+zfaP
L4D18rlAg2aYGv6z6Q9wfrwzhXvuD3ddnZOyjxp0x80v9rnH9QAV4xgeV4vYQA5K
0M7okWV17jDUzHEluzTKgsjwJKkVQTNn4vox0L5Lq6gn+RaWgVGPLVoDtHSnhs42
gZejrWUIEst6xLIpIWhjpPDIpudnkb0nzv7Nu/N9ZtKIIcsx6sTofP9+PDsf2EFD
0qIJd0t/iOBpS5//AZ2aRyTWHS1DwaYCDOWX2G3We99ionN07ND2PD7h2D31k4h2
YGfQzM09JGarxuPz4WvETk/zGM9uRZR647NQfcwEMkrV6aiLp0fFEDm/p+fJR94L
```

```
[fedora@guimbarde .ssh]$  cat jugsummercamp-rsa-key.pub
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAACAQDhZqwgs2b/ARW9DdLtOnTJqqufEvlfw4ZRtMjJlyzL0PIGl++HyigrEtd1p8I/nlnP0IoQNMDCkqfX
gJvMH7D0NQbn8A2KAEJII7U1MgVsIm2UeDGManCTwrH4j3K12+SXaQVgxf0942lmuzaBpzDb63QAFChdKwr+SFTpQrW0Bvxe4z7N9o8vgPXy
uUCDZpga/rPpD3B+vDOFe+4Pd12dk7KPGnTHzS/2ucf1ABXjGB5Xi9hADkrQzuiRZXXuMNTMcSW7NMqCyPAkqRVBM2fi+jHQvkurqCf5FpaB
UY8tWgO0dKeGzjaBl6OtZQgSy3rEsikhaGOk8Mim52eRvSfO/s27831m0oghyzHqxOh8/348Ox/YQUPSogl3S3+I4GlLn/8BnZpHJNYdLUPB
pgIM5ZfYbdZ732Kic3Ts0PY8PuHYPfWTiHZgZ9DMzT0kZqvG4/Pha8ROT/MYz25FlHrjs1B9zAQyStXpqIunR8UQOb+n58lH3gvwZuumS4IM
YRCJE/IOfMCRVMWRcExnzZ2iivQ8MKnivQFeTdwXx15fMLdZfjkb1M5R9sV4sXSiznE1PfZdC9Iyc6+FLbprW+eCYk/VlOuME/V8nIAuuy0e
QniJUOrnw9WUK4XKGKcSa3S6Mwo02TcsvOQyNwM7K91PJOs74QBSsxWAwrq2/6Mk8w== fedora@guimbarde.thegaragebandofit.com
```

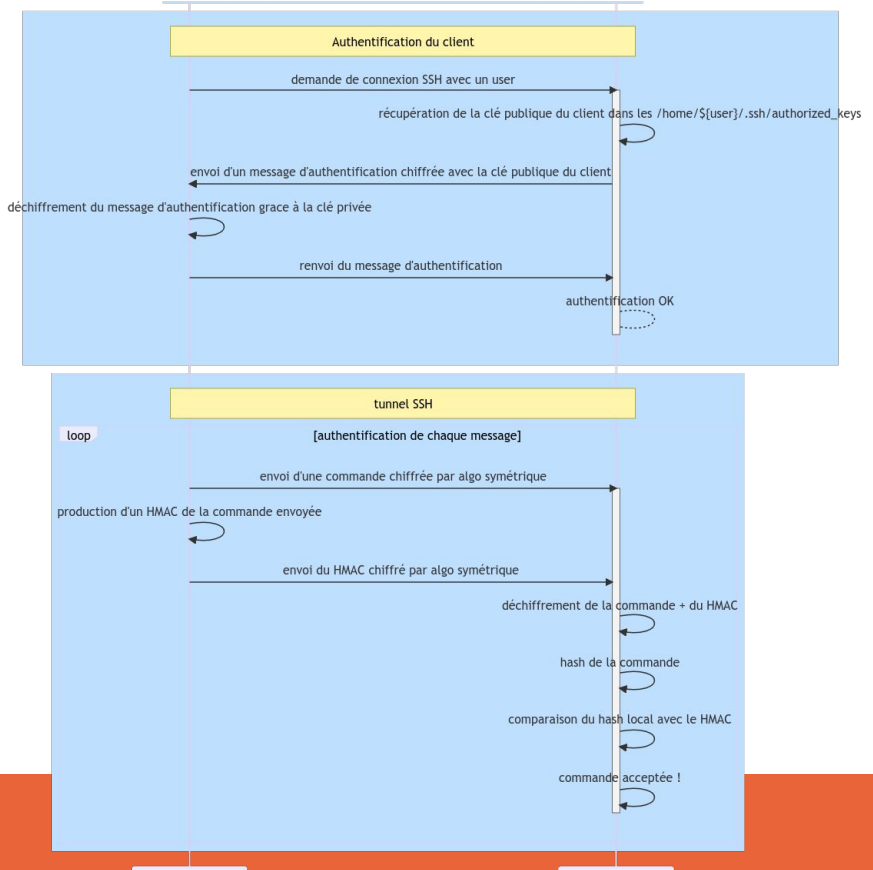
SSH *connexion !*

négociation / connexion
échange de clés publiques
authentification
partage de clé symétrique

Session SSH, séquence d'initialisation



Session SSH, séquence d'initialisation



Partage de la clé symétrique, Diffie-Hellman KEX algorithm

Objectif : générer et partager la clé symétrique utilisée pour le chiffrement symétrique du tunnel

Contrainte : la clé partagée doit être présente des 2 côtés SANS être échangée

Séquence :

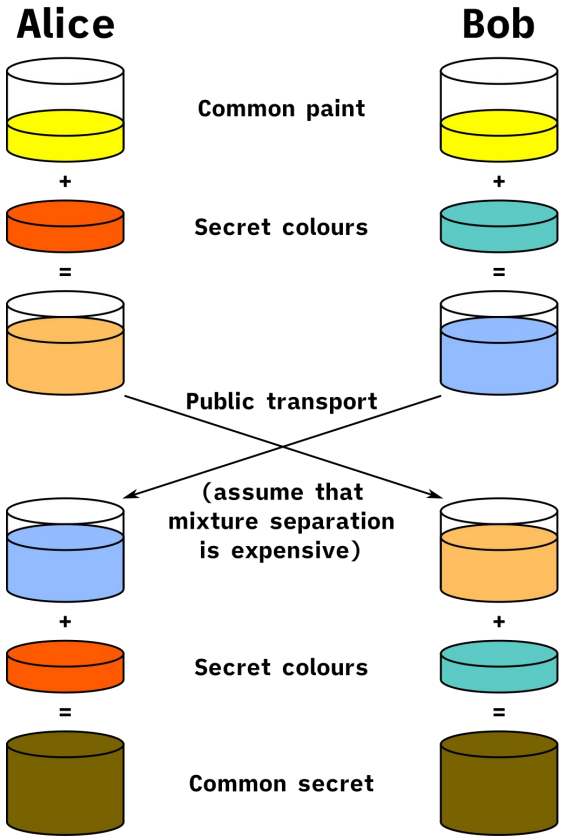
1. Utilisation d'une valeur de départ (très grand nombre premier) partagée
2. Génération de couples de clés asymétriques temporaires
3. Échange des clés publiques temporaires
4. Génération de la clé symétrique partagée en combinant
   a. sa propre clé privée
   b. la clé publique du tiers

Configuration : `RekeyLimit`

Dans la session : ~R pour force le rekey.

SSH REKEY - exemple de log

```
debug1: SSH2_MSG_KEXINIT sent
debug1: rekeying in progress
debug1: SSH2_MSG_KEXINIT received
debug1: kex: algorithm: curve25519-sha256@libssh.org
debug1: kex: host key algorithm: ecdsa-sha2-nistp256
debug1: kex: server->client cipher: chacha20-poly1305@openssh.com MAC: <implicit> compression: none
debug1: kex: client->server cipher: chacha20-poly1305@openssh.com MAC: <implicit> compression: none
debug1: kex: curve25519-sha256@libssh.org need=64 dh_need=64
debug1: kex: curve25519-sha256@libssh.org need=64 dh_need=64
debug1: expecting SSH2_MSG_KEX_ECDH_REPLY
debug1: rekeying in progress
debug1: rekeying in progress
debug1: Server host key: ecdsa-sha2-nistp256 SHA256:E9HuzpVQJ/5DavNIZhWzJrADNj5Ntw69RJ1obXJtlGo
debug1: set_newkeys: rekeying, input 5156 bytes 167 blocks, output 5864 bytes 0 blocks
debug1: rekey after 134217728 blocks
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: set_newkeys: rekeying, input 5168 bytes 0 blocks, output 5864 bytes 0 blocks
debug1: rekey after 134217728 blocks
debug1: SSH2_MSG_NEWKEYS received
```



Gestion des clés, rotation, diffusion, confidentialité , révocation

Clé symétrique

Rotation
Dans le fichier de configuration `/etc/ssh/ssh_config`

```
RekeyLimit 1G 1h
```

Dans la session : `~R` pour forcer le rekey.

SSH REKEY - exemple de log

```
debug1: SSH2_MSG_KEXINIT sent
debug1: rekeying in progress
debug1: SSH2_MSG_KEXINIT received
debug1: kex: algorithm: curve25519-sha256@libssh.org
debug1: kex: host key algorithm: ecdsa-sha2-nistp256
debug1: kex: server->client cipher: chacha20-poly1305@openssh.com MAC: <implicit>
compression: none
debug1: kex: client->server cipher: chacha20-poly1305@openssh.com MAC: <implicit>
compression: none
debug1: kex: curve25519-sha256@libssh.org need=64 dh_need=64
debug1: kex: curve25519-sha256@libssh.org need=64 dh_need=64
debug1: expecting SSH2_MSG_KEX_ECDH_REPLY
debug1: rekeying in progress
debug1: rekeying in progress
debug1: Server host key: ecdsa-sha2-nistp256
SHA256:E9HuzpVQJ/5DavNIZhWzJrADNj5Ntw69RJ1obXJtlGo
debug1: set_newkeys: rekeying, input 5156 bytes 167 blocks, output 5864 bytes 0
blocks
debug1: rekey after 134217728 blocks
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: set_newkeys: rekeying, input 5168 bytes 0 blocks, output 5864 bytes 0 blocks
debug1: rekey after 134217728 blocks
debug1: SSH2_MSG_NEWKEYS received
```

Clés asymétriques

Diffusion
copie de fichiers

Confidentialité

```
chown ${user} ${myprivatekey}
chmod 600 ${myprivatekey}
```

Révocation
suppression de la clé publique
de `/home/${user}/.ssh/known_hosts`
de `/home/${user}/.ssh/authorized_keys`

Rotation
Clés asymétriques 👨‍🔬

ssh, config

Côté client

```
$ cat .ssh/config
Host guimbarde
  Hostname guimbarde.thegaragebandofit.com
  User fedora
  Port 22
  IdentityFile
/root/.ssh/ish-liPadeM2_ovh_ecdsa
  Compression yes
```

Côté serveur

```
$ cat /etc/ssh/ssh_config
# Host *
#   ForwardAgent no
#   ForwardX11 no
#   PasswordAuthentication yes
#   HostbasedAuthentication no
#   GSSAPIAuthentication no
#   GSSAPIDelegateCredentials no
#   GSSAPIKeyExchange no
#   GSSAPITrustDNS no
#   BatchMode no
#   CheckHostIP yes
#   AddressFamily any
#   ConnectTimeout 0
#   StrictHostKeyChecking ask
#   IdentityFile ~/.ssh/id_rsa
#   IdentityFile ~/.ssh/id_dsa
#   IdentityFile ~/.ssh/id_ecdsa
#   IdentityFile ~/.ssh/id_ed25519
#   Port 22
#   Ciphers
aes128-ctr,aes192-ctr,aes256-ctr,aes128-cbc,3des-cbc
#   MACs hmac-md5,hmac-sha1,umac-64@openssh.com
#   EscapeChar ~
#   Tunnel no
#   TunnelDevice any:any
#   PermitLocalCommand no
#   VisualHostKey no
#   ProxyCommand ssh -q -W %h:%p gateway.example.com
#   RekeyLimit 1G 1h
#   UserKnownHostsFile ~/.ssh/known_hosts.d/%k
#
Include /etc/ssh/ssh_config.d/*.conf
```

SSH *fingerprint !*

empreinte de clé publique
et known_host

Empreinte de la clé publique

Chaque clé publique est associée à une empreinte appelé fingerprint

La clé publique est stockée côté client dans un fichier known_host

Objectif : empêcher les attaques de type man-in-the-middle

```
[fedora@guimbarde .ssh]$ ssh fedora@guimbarde.thegaragebandofit.com
The authenticity of host 'guimbarde.thegaragebandofit.com' can't be established.
ED25519 key fingerprint is SHA256:2ui1Vvf76v/eUPkyW4GN8QY6Z+zC7YX+U8pGZIwWyUs.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

Possibilité de stocker le fingerprint de manière centrale dans une zone DNS : DNS SSHFP

```
guimbarde.thegaragebandofit.com. 3600 IN SSHFP 4 2 2ui1Vvf76v/eUPkyW4GN8QY6Z+zC7YX+U8pGZIwWyUs
```

<Name> [<TTL>] [<Classe>] SSHFP <Algorithme> <Type> <Résultat de la Fonction de hachage>
Algorithme : 0=réservé, 1=RSA, 2=DSA, 3=ECDSA, 4=Ed25519
Type : 0=réservé, 1=SHA-1, 2=SHA-256

Pas activé par défaut, lors de la connexion, il faut ajouter l'option -o "VerifyHostKeyDNS ask"

```
[fedora@guimbarde .ssh]$ ssh -o "VerifyHostKeyDNS ask" fedora@guimbarde.thegaragebandofit.com
[...]  Matching host key fingerprint found in DNS.
       Are you sure you want to continue connecting (yes/no)?
```

Vérification du fingerprint au niveau DNS

```
[fedora@guimbarde .ssh]$ ssh-keygen -r guimbarde.thegaragebandofit.com.
guimbarde.thegaragebandofit.com. 3600 IN SSHFP 4 2 2ui1Vvf76v/eUPkyW4GN8QY6Z+zC7YX+U8pGZIwWyUs
```

Empreinte de la clé publique

Si changement du fingerprint

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@ WARNING: POSSIBLE DNS SPOOFING DETECTED!
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
The RSA host key for guimbarde.thegaragebandofit.com remote host has changed,and the key for the corresponding IP address
xxx.yy.xxx.yy is unknown. This could either mean that DNS SPOOFING is happening or the IP address for the host and its host
key have changed at the same time.
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@ WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
2ui1Vvf76v/eUPkyW4GN8QY6Z+zC7YX+U8pGZIwWyUs.
Please contact your system administrator.
Add correct host key in /home/.ssh/known_hosts to get rid of this message.
Offending key in /home/.ssh/known_hosts:1
Keyboard-interactive authentication is disabled to avoid man-in-the-middle attacks.
```

Effacement du fingerprint dans le fichier known_host

```
ssh-keygen -R guimbarde.thegaragebandofit.com
```

Vérification du fingerprint des clés publiques coté client

```
ssh-keygen -lf jugsummercamp-rsa-key.pub

4096 SHA256:2ui1Vvf76v/eUPkyW4GN8QY6Z+zC7YX+U8pGZIwWyUs fedora@guimbarde.thegaragebandofit.com (RSA)
```

Empreinte de la clé publique

Gestion de multiples utilisateurs

```
1/ Ajout de la clé dans le known_host pour chaque utilisateur
2/ -o StrictHostKeyChecking=no (pas recommandé)
3/ ssh-keyscan -H -t rsa guimbarde.thegaragebandofit.com >>
/etc/ssh/ssh_known_hosts
```

SSH *copy-id !*

copie de la clé publique
et authorized_keys

Copie de la clé publique côté serveur

Pour accepter les connexions, il faut que le serveur connaisse la clé publique

Coté serveur, la clé publique est stocké dans le fichier ~/.ssh/authorized_keys

```
[fedora@guimbarde .ssh]$ cat ~/.ssh/jugsummercamp-rsa-key.pub
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDWeSBZZiWaLQkKYwxsNhvEur5HLlymF5A6cGcjEvrQzDzLHqe7/yMaenZ9jMGxJ/et8snq3KyDw7VaQvui
AHsZdES0IhDiAb82XkEn8sd7dvRqMXlnIdGpZUJ33UwOevDfn3N6JGK/6uvuJLTFcz5L/K+6pk06ur9Go1gIseCTtjmBqzpgc3bB+mD/uAfLbBXz
2kJdc1RnvBk8sBxQ9UXYdwGRdEWA6RvkU1mGaOAMLhKDOxjR6rg8JQe0CTeKVFq9JdCs+KIhOiYkyZBMv8qM0s+PJpblexOoivmgPbQRjaI+qdC/
b1QtQzdAPwobFbJeRFT1IrIvKA97YNBtTv8 fedora@guimbarde.thegaragebandofit.com
```

Peut avoir plusieurs types dépendant de ce que vous avez choisi comme algorithme

Plusieurs solutions pour copier la clé
1/ Manuellement
2/ Utilisation de ssh-copy-id

```
[fedora@guimbarde .ssh]$ ssh-copy-id -i ~/.ssh/jugsummercamp-rsa-key.pub fedora@guimbarde.thegaragebandofit.com
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/fedora/.ssh/jugsummercamp-rsa-key.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 fedora@guimbarde.thegaragebandofit.com's password:
Number of key(s) added: 1 Now try logging into the machine, with:   "ssh fedora@guimbarde.thegaragebandofit.com"
and check to make sure that only the key(s) you wanted were added.
```

SSH *agent !*

gestion de multiples clés

Connexion automatique avec ssh-agent

La gestion des clés peut être compliqué (plusieurs 10aine de clés voir 100aine par workstation)

Il faut se rappeler de quelle clé est associé à quels serveurs
Et aussi de chaque passphrase

-> Ca peut être fastidieux

ssh-agent est un helper qui garde la trace des clés d'identité des utilisateurs et de leurs passphrases. L'agent peut ensuite utiliser ces clés pour se connecter à d'autres serveurs sans que l'utilisateur n'ait à saisir à nouveau un mot de passe ou une passphrase. Cela permet de mettre en œuvre une forme d'authentification unique (SSO).

Démarrage de l'agent

```
[fedora@guimbarde .ssh]$ eval "$(ssh-agent -s)"
Agent pid 1383
```

```
Vérification du démarrage de l'agent
[fedora@guimbarde .ssh]$ echo $SSH_AGENT_PID
1383
```

Par défaut, l'agent SSH ajoute les clés suivantes : ~/.ssh/id_rsa, .ssh/id_dsa, ~/.ssh/id_ecdsa, ~/.ssh/id_ed25519, et ~/.ssh/identity

Connexion automatique avec ssh-agent

SSH Agent forwarding
Le protocole SSH met en œuvre du SSH agent forwarding, un mécanisme par lequel un client SSH permet à un serveur SSH d'utiliser l'agent ssh local sur le serveur auquel l'utilisateur se connecte, comme s'il s'agissait d'un agent local.

C'est une forme de SSO transitif

Mais contient des risques ! Toute personne root sur le serveur peut avoir accès à la socket ssh-agent, et donc usurper l'identité de connexion

Il vaut mieux utiliser ProxyJump

```
[fedora@guimbarde .ssh]$ ssh -J bastion.thegaragebandofit.com
fedora@guimbarde.thegaragebandofit.com
```

Séquencement :
1/ SSH va utiliser la clé dans l'agent pour se connecter au bastion
2/ Ensuite, SSHD depuis le bastion va se connecter à guimbarde et va forwarder la connexion à l'agent local
3/ L'agent local va ensuite renégocier le handshake via le bastion

Pas de transfert de clé sur le bastion !

SSH-*add !*

ajout et listing de clés complémentaires

Ajout et listing des clés de connexion

Comment ajouter des clés supplémentaire dans l'agent ?

Avec la commande ssh-add

```
[fedora@guimbarde .ssh]$ ssh-add ~/.ssh/id_ed25519
Identity added: id_ed25519 (fedora@guimbarde.thegaragebandofit.com)
```

Dans le cas ou la clé à une passphrase, ssh-add va lancer la commande ssh-askpass pour avoir la passphrase et la stocker de manière sécurisée.

Lister toutes les identitées

```
[fedora@guimbarde .ssh]$ ssh-add -L
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDWeSBZZiWaLQkKYwxsNhvEur5HLlymF5A6cGcjEvrQzDzLHqe7/yMaenZ9jMGxJ/et8snq3KyD
w7VaQvuiAHsZdES0IhDiAb82XkEn8sd7dvRqMXlnIdGpZUJ33UwOevDfn3N6JGK/6uvuJLTFcz5L/K+6pk06ur9Go1gIseCTtjmBqzpg
c3bB+mD/uAfLbBXz2kJdc1RnvBk8sBxQ9UXYdwGRdEWA6RvkU1mGaOAMLhKDOxjR6rg8JQe0CTeKVFq9JdCs+KIhOiYkyZBMv8qM0s+P
JpblexOoivmgPbQRjaI+qdC/b1QtQzdAPwobFbJeRFT1IrIvKA97YNBtTv8 fedora@guimbarde.thegaragebandofit.com
```

Lister tous les fingerprints

```
[fedora@guimbarde .ssh]$ ssh-add -l
4096 SHA256:2ui1Vvf76v/eUPkyW4GN8QY6Z+zC7YX+U8pGZIwWyUs fedora@guimbarde.thegaragebandofit.com (RSA)
```

SSH *at scale ?*

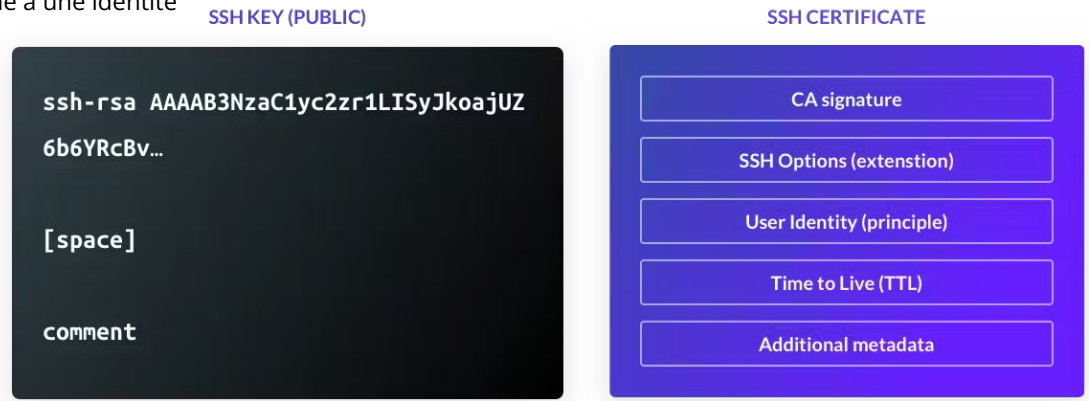
gestion de multiples clés

Gestion at scale

Comment faire pour pouvoir gérer et stocker de manière sécurisée plusieurs 10aine voir 100aine de clés ?

De plus, les clés ne sont pas lié à une identité

La réponse ?
SSH certificate !!



- Les certificats sont liés à l'identité de l'utilisateur.
- Les certificats expirent automatiquement.
- Les certificats peuvent contenir des restrictions SSH, par exemple interdire l'attribution de PTY ou la redirection de ports.
- Les certificats SSH peuvent être synchronisés avec les certificats Kubernetes.
- Les certificats incluent des métadonnées. Cela permet un contrôle d'accès basé sur les rôles.
- Les certificats résolvent les problèmes de TOFU (trust on first use). Les certificats utilisateur et hôte signés par la même autorité de certification établissent la confiance et éliminent le besoin de TOFU.

Gestion at scale

Comment générer des certificats ?

```
[fedora@guimbarde .ssh]$ ssh-keygen -t rsa -b 4096 -f host_ca -C host_ca
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in host_ca.
Your public key has been saved in host_ca.pub.
The key fingerprint is:
SHA256:tltbnMalWg+skhm+VlGLd2xHiVPozyuOPl34WypdEO0 host_ca
The key's randomart image is:
+---[RSA 4096]----+
|              +o.|
|            .+..o|
|           o.o.+ |
|          o o.= E|
|         S  o o=o |
|        ....+ = +.|
|        ..=. %.o.o|
|         *o Oo=.+.|
|        .oo=ooo+..|
+----[SHA256]-----+
```

Une best practice est de générer aussi un CA pour les utilisateurs

```
[fedora@guimbarde .ssh]$ ssh-keygen -t rsa -b 4096 -f user_ca -C user_ca
```

Gestion at scale

Gestion des certificats par machine

```
[fedora@guimbarde .ssh]$ ssh-keygen -f ssh_host_rsa_key -N '' -b 4096 -t rsa

[fedora@guimbarde .ssh]$ ssh-keygen -s host_ca -I guimbarde.thegaragebandofit.com -h -n
guimbarde.thegaragebandofit.com -V +52w ssh_host_rsa_key.pub
Enter passphrase: # the passphrase used for the host CA
Signed host key ssh_host_rsa_key-cert.pub: id "guimbarde.thegaragebandofit.com" serial 0 for
guimbarde.thegaragebandofit.com valid from 2023-06-30T15:00:00 to 2024-06-30T15:01:37
```

-s : CA racine pour signer le certificat (host)
-I : identité du certificat
-h : certificat host
-n : nom de la machine
-V : durée de validité

Ajout du CA dans le know_host :

```
@cert-authority *.thegaragebandofit.com ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAACAQDwiOso0Q4W+KKQ4OrZZ1o1X7g3yWcmAJtySILZSwo1GXBKgurV4jmmBN5RsHetl98QiJq64e8oKX1vGR25
1afalWu0w/iW9jL0isZrPrmDg/p6Cb6yKnreFEaDFocDhoiIcbUiImIWcp9PJXFOK1Lu8afdeKWJA2f6cC4lnAEq4sA/Phg4xfKMQZUFG5sQ/Gj1
StjIXi2RYCQBHFDzzNm0Q5uB4hUsAYNqbnaiTI/pRtuknsgl97xK9P+rQiNfBfPQhsGeyJzT6Tup/KKlxarjkMOlFX2MUMaAj/cDrBSzvSrfOwzk
qyzYGHzQhST/lWQZr4OddRszGPO4W5bRQzddUG8iC7M6U4llUxrb/H5QOkVyvnx4Dw76MA97tiZItSGzRPblU4S6HMmCVpZTwva4LLmMEEIk1lW5
HcbB6AWAc0dFE0KBuusgJp9MlFkt7mZkSqnim8wdQApal+E3p13d0QZSH3b6eB3cbBcbpNmYqnmBFrNSKkEpQ8OwBnFvjjdYB7AXqQqrcqHUqfwk
X8B27chDn2dwyWb3AdPMg1+j3wtVrwVqO9caeeQ1310CNHIFhIRTqnp2ECFGCCy+EDSFNZM+JStQoNO5rMOvZmecbp35XH/UJ5IHOkh9wE5TBYIe
FRUYoc2jHNAuP2FM4LbEagGtP8L5gSCTXNRM1EX2gQ== host_ca
```

Gestion at scale

Gestion des certificats par utilisateur

```
[fedora@guimbarde .ssh]$ ssh-keygen -f user-key -b 4096 -t rsa

[fedora@guimbarde .ssh]$ ssh-keygen -s user_ca -I
fedora@thegaragebandofit.com -n fedora -V +1d user-key.pub
Enter passphrase: # the passphrase used for the user CA
Signed user key user-key-cert.pub: id "fedora@thegaragebandofit.com"
serial 0 for fedora valid from 2020-03-19T16:33:00 to
2020-03-20T16:34:54
```

-s : CA racine pour signer le certificat (user)
-I : identité du certificat
-n : liste des utilisateurs
-V : durée de validité

Gestion at scale

Inspecter un certificat

```
[fedora@guimbarde .ssh]$ ssh-keygen -L -f user-key-cert.pub
user-key-cert.pub:
        Type: ssh-rsa-cert-v01@openssh.com user certificate
        Public key: RSA-CERT SHA256:egWNu5cUZaqwm76zoyTtktac2jxKktj30Oi/ydrOqZ8
        Signing CA: RSA SHA256:tltbnMalWg+skhm+VlGLd2xHiVPozyuOPl34WypdEO0 (using ssh-rsa)
        Key ID: "fedora@thegaragebandofit.com"
        Serial: 0
        Valid: from 2020-03-19T16:33:00 to 2020-03-20T16:34:54
        Principals:
                fedora
        Critical Options: (none)
        Extensions:
                permit-X11-forwarding
                permit-agent-forwarding
                permit-port-forwarding
                permit-pty
                permit-user-rc
```

Gestion at scale

Stockage dans un LDAP

Ajout du schéma dans le LDAP

```
dn: cn=openssh-lpk,cn=schema,cn=config
objectClass: olcSchemaConfig
cn: openssh-lpk
olcAttributeTypes: ( 1.3.6.1.4.1.24552.500.1.1.1.13 NAME 'sshPublicKey'
    DESC 'MANDATORY: OpenSSH Public key'
    EQUALITY octetStringMatch
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.40 )
olcObjectClasses: ( 1.3.6.1.4.1.24552.500.1.1.2.0 NAME 'ldapPublicKey' SUP top AUXILIARY
    DESC 'MANDATORY: OpenSSH LPK objectclass'
    MAY ( sshPublicKey $ uid )
    )
```

Création de la requête de récupération de la clé

```
ldapsearch '(&(objectClass=posixAccount)(uid='"$1"'))' 'sshPublicKey' | sed -n '/^
/{H;d};/sshPublicKey:/x;$g;s/\n *//g;s/sshPublicKey: //gp'
```

Ajout du script dans le démon SSHD

```
AuthorizedKeysCommand /path/to/script
AuthorizedKeysCommandUser nobody
```

Gestion at scale

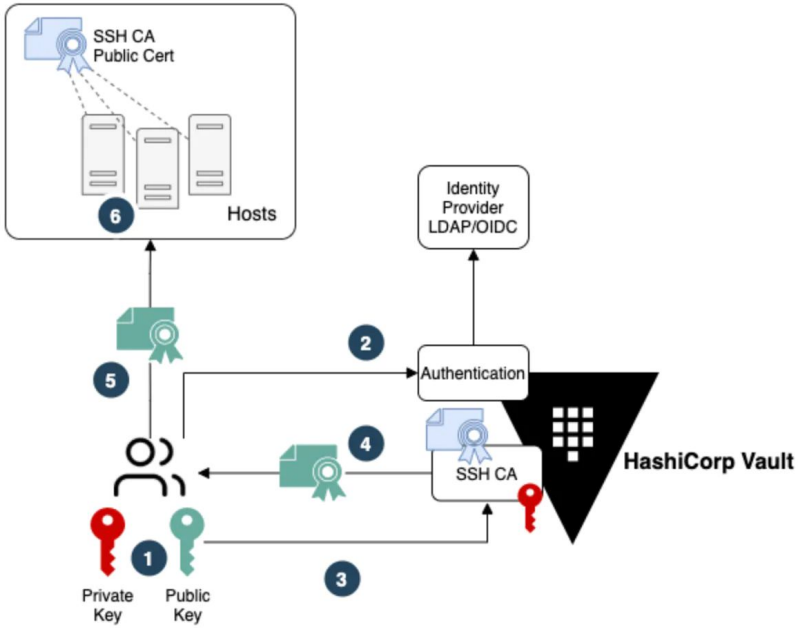
Ajout de la clé dans le LDAP

```
[fedora@guimbarde .ssh]$ ldapmodify -xWD
"uid=fedora,ou=Users,dc=thegaragebandofit,dc=com"
cn: uid=fedora.ou=Groups,dc=thegaragebandofit,dc=com
changetype: modify
add: ldapPublicKey

cn: uid=fedora,ou=Groups,dc=thegaragebandofit,dc=com
changetype: modify
add: sshPublicKey
sshPublicKey: ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDWeSBZZiWaLQkKYwxsNhvEur5HLlymF5A6cGcjEvrQzDzLHqe
7/yMaenZ9jMGxJ/et8snq3KyDw7VaQvuiAHsZdES0IhDiAb82XkEn8sd7dvRqMXlnIdGpZUJ33UwOev
Dfn3N6JGK/6uvuJLTFcz5L/K+6pk06ur9Go1gIseCTtjmBqzpgc3bB+mD/uAfLbBXz2kJdc1RnvBk8s
BxQ9UXYdwGRdEWA6RvkU1mGaOAMLhKDOxjR6rg8JQe0CTeKVFq9JdCs+KIhOiYkyZBMv8qM0s+PJpbl
exOoivmgPbQRjaI+qdC/b1QtQzdAPwobFbJeRFT1IrIvKA97YNBtTv8
fedora@guimbarde.thegaragebandofit.com
```

Gestion at scale

- L'utilisateur crée une paire de clés SSH personnelle.
- L'utilisateur s'authentifie auprès de l'espace de stockage avec ses identifiants de fournisseur d'identité (IDP).
- Une fois authentifié, l'utilisateur envoie sa clé publique SSH à Vault pour signature.
- Vault signe la clé SSH et renvoie le certificat SSH à l'utilisateur.
- L'utilisateur établit une connexion SSH à l'aide du certificat SSH.
- L'hôte vérifie que le certificat SSH du client est signé par l'autorité de certification SSH approuvée et autorise la connexion.



SSH Certificate Authentication Workflow

SSH *et sur le cloud ?*

différentes façons de s’authentifier

Sur le cloud, IaaS

Immutable infra

- Injection des clés dans la *golden* **AMI** (**Packer**, par exemple)
- Jeu de clés par compte / projet / instance group / etc. (**Terraform**, par exemple)
- Configuration du `cloud-init` pour peupler les arborescences `/home/${user}s`

Mutable infra

- gestion via *config. management* (**Ansible**, etc.)
- OS login (**GCP**)

Sur le cloud, IaaS

OS Login

- Gestion automatique du cycle de vie des comptes Linux
- Autorisation fine à l'aide de Google IAM
- Mises à jour automatiques des autorisations
- Possibilité d'importer des comptes Linux existants
- Intégration avec la vérification en deux étapes du compte Google
- Intégration avec la journalisation d'audit

Qu'est-ce que ca fait ?

- Supprime les fichiers authorized_keys de la VM.
- Configure un serveur OpenSSH avec l'option AuthorizedKeysCommand. Cette commande récupère les clés SSH associées au compte utilisateur Linux pour authentifier la tentative de connexion.
- Configure la fonctionnalité NSS (Name Service Switch) pour fournir au système d'exploitation les informations relatives à l'utilisateur OS Login.
- Ajoute un ensemble de configurations PAM (Pluggable Authentication Modules) pour autoriser la connexion de l'utilisateur. Les configurations PAM vérifient les autorisations IAM pour la connexion et l'accès administratif. Ces configurations PAM exécutent également d'autres tâches telles que la configuration du répertoire d'accueil du compte utilisateur Linux.