Setup CA SSH pour Make IT Simple

Guide complet pour mettre en place une autorité de certification SSH dans un contexte MSP, avec support multi-Yubikey et intégration dans l'infrastructure MIS.

—

Compatibilité Windows

Réponse courte : oui, mais avec une distinction importante entre les rôles.

Rôle	Système	Notes
——	———	——-
Machine CA (signature des certifs)	Linux (LXC Proxmox)	OpenSSH côté Linux uniquement pour ce setup
Poste technicien (utilisation Yubikey)	Windows / Linux / macOS	OpenSSH inclus dans Windows 10/11 (1809+)
Serveurs clients (qui acceptent les certifs)	Linux principalement	Windows Server avec OpenSSH for Windows fonctionne aussi

Côté Windows pour les techniciens : - OpenSSH client/server est intégré nativement à Windows 10 (build 1809+) et Windows 11 - Vérification : `Get-WindowsCapability -Online | ? Name -like 'OpenSSH*'` - Installation si absent : `Add-WindowsCapability -Online -Name OpenSSH.Client~~~~0.0.1.0` - Pour les versions récentes d'OpenSSH (recommandé pour FIDO2) : `winget install Microsoft.OpenSSH.Beta` - Le support FIDO2/Yubikey (`ed25519-sk`) fonctionne nativement - L'utilisation des certificats SSH est transparente : il suffit de placer le `-cert.pub` à côté de la clé privée

Côté machine CA : on reste sur Linux (LXC Debian sur Proxmox). C'est plus simple, plus auditable, et compatible avec l'écosystème MIS existant.

—

Notes Debian 13 (Trixie)

Cette documentation cible Debian 13 (codename **Trixie**), sortie le 9 août 2025. Quelques points importants pour notre cas d'usage SSH CA :

Versions embarquées

Composant	Bookworm (Debian 12)	Trixie (Debian 13)
———–	———————	——————–
OpenSSH	9.2p1	10.0p1
systemd	252	257
Kernel	6.1 LTS	6.12 LTS
Python	3.11	3.13

Changements pertinents pour la CA SSH

OpenSSH 10.0 : meilleurs algos par défaut, mais surtout : - Les **clés DSA ne sont plus supportées du tout**, même avec les options de compat (`HostKeyAlgorithms`, `PubkeyAcceptedAlgorithms`). Aucun impact pour nous (on n'utilise que Ed25519/Ed25519-sk), mais

à savoir si tu te connectes à de très vieux équipements (un `apt install openssh-client-ssh1` peut servir pour ces cas-là). - Le support FIDO2 (libfido2) est plus mature qu'en Bookworm. - Les algorithmes post-quantum (mlkem768x25519-sha256) sont disponibles par défaut pour le key exchange.

`/tmp` est maintenant un tmpfs en Trixie (RAM). Aucun impact sur nos workflows (les `.pub` font quelques centaines d'octets), mais à garder en tête pour des scripts qui manipuleraient de gros fichiers temporaires.

`/etc/sysctl.conf` n'est plus lu : utiliser `/etc/sysctl.d/99-mis.conf` à la place pour tes éventuelles personnalisations kernel. Sans impact direct sur le SSH CA, mais bon à savoir pour tes playbooks Ansible.

openssh-server ne lit plus `~/.pam_environment` : si jamais tu avais des configs spécifiques pour des sessions SSH, à migrer vers `~/.bash_profile` ou la conf PAM système. Non pertinent pour notre workflow MIS.

Support à long terme

- Full support jusqu'à **août 2028** - LTS jusqu'à **juin 2030**

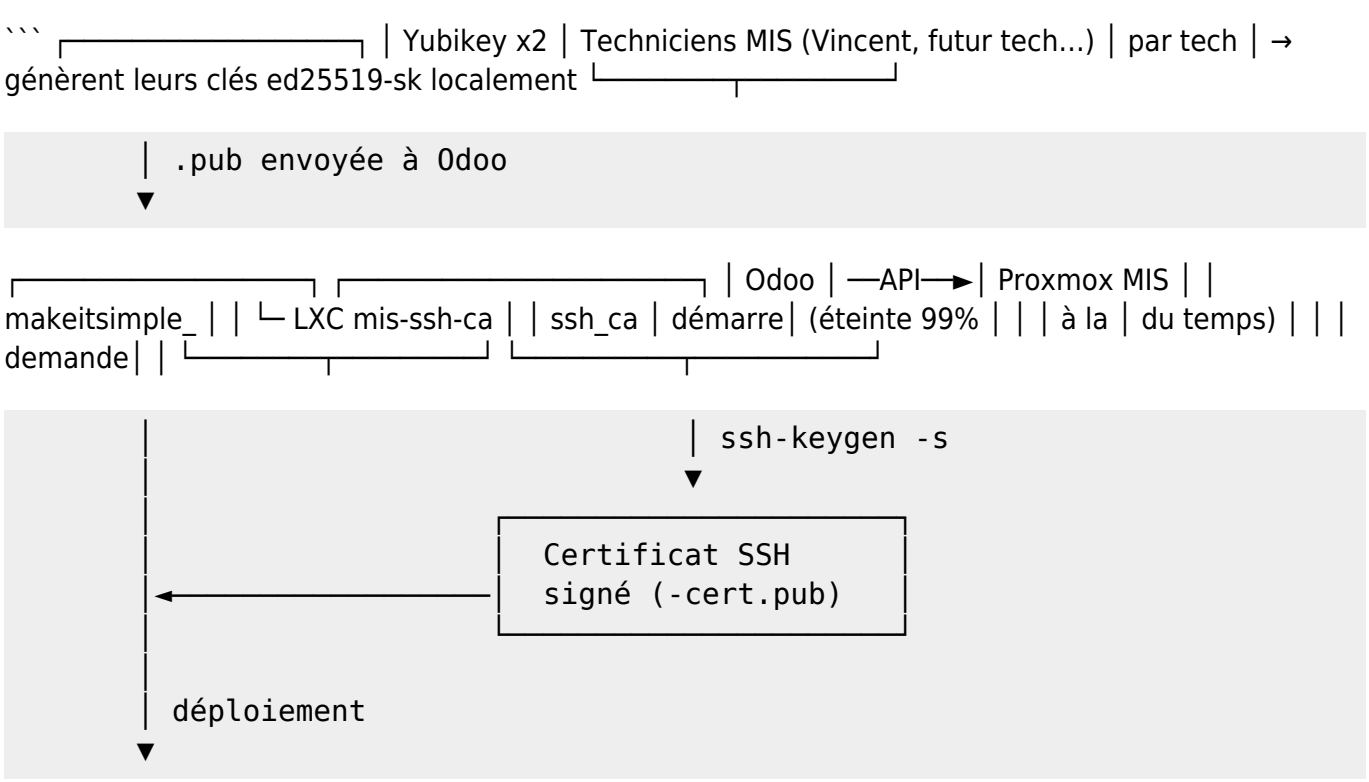
C'est donc une base saine pour démarrer une infra MSP qui doit vivre 4-5 ans sans upgrade majeur.

Images officielles

- Cloud image : https://cloud.debian.org/images/cloud/trixie/latest/debian-13-genericcloud-amd64.qcow2 - Template LXC Proxmox : `debian-13-standard_13.X-Y_amd64.tar.zst` (récupérable via `pveam`)

—

Vue d'ensemble de l'architecture



┌──────────────────┐ │ Serveurs │ │ clients │ /etc/ssh/mis-users-ca.pub │ (200+ VMs) │ /etc/ssh/mis-revoked-keys └──────────────────┘ ```

—

Phase 1 : Préparation de la LXC CA

Étape 1.1 : Création du conteneur

Sur le Proxmox MIS (pas un hyperviseur client) :

```bash # Variables CTID=200 HOSTNAME="mis-ssh-ca" STORAGE="local-zfs" BRIDGE="vmbr10" # VLAN management isolé TEMPLATE="debian-13-standard_13.5-1_amd64.tar.zst" # Vérifier le nom exact disponible avec : # pveam update && pveam available | grep debian-13 # Et télécharger si pas encore présent : # pveam download local debian-13-standard_13.5-1_amd64.tar.zst

# Création LXC pct create $CTID local:vztmpl/$TEMPLATE \

1. -hostname $HOSTNAME \
2. -cores 1 –memory 512 –swap 256 \
3. -rootfs $STORAGE:8 \
4. -net0 name=eth0,bridge=$BRIDGE,ip=10.10.10.50/24,gw=10.10.10.1 \
5. -nameserver 10.10.10.1 \
6. -onboot 0 \
7. -start 0 \
8. -unprivileged 1 \
9. -features nesting=0,keyctl=0 \
10. -password "TEMP_PASSWORD_CHANGE_IMMEDIATELY"

# Démarrage pct start $CTID ```

### Étape 1.2 : Durcissement initial

```bash # Entrer dans la LXC pct enter $CTID

# Hardening de base apt update && apt upgrade -y apt install -y openssh-server fail2ban ufw vim

# Firewall : SSH uniquement depuis le subnet de management ufw default deny incoming ufw default deny outgoing ufw allow from 10.10.10.0/24 to any port 22 proto tcp ufw allow out 53/udp # DNS pour apt ufw allow out 80,443/tcp # apt updates ufw enable

# Utilisateur dédié pour la signature useradd -m -s /bin/bash ca-signer mkdir -p /home/ca-signer/.ssh chmod 700 /home/ca-signer/.ssh

# Désactivation auth password sur SSH sed -i 's/^#\?PasswordAuthentication.*/PasswordAuthentication no/' /etc/ssh/sshd_config sed -i 's/^#\?PermitRootLogin.*/PermitRootLogin no/' /etc/ssh/sshd_config systemctl restart ssh ```

### Étape 1.3 : Stockage chiffré (optionnel mais recommandé)

Pour aller plus loin, la rootfs de la LXC peut être stockée sur un dataset ZFS chiffré côté hôte Proxmox :

```bash # Sur l'hôte Proxmox (à faire AVANT la création de la LXC) zfs create -o encryption=aes-256-
```

gcm -o keyformat=passphrase rpool/encrypted-vault # La passphrase sera demandée à chaque démarrage de l'hôte ```

—

Phase 2 : Génération des CAs

Étape 2.1 : CA pour les utilisateurs (techniciens)

```bash # Connecté en tant que ca-signer dans la LXC mkdir -p ~/mis-ca && cd ~/mis-ca chmod 700 .

ssh-keygen -t ed25519 -a 200 \

1. f mis-users-ca \
2. C “MIS Users CA - Make IT Simple” \
3. N 'PASSPHRASE_TRES_ROBUSTE_ICI'

```

Étape 2.2 : CA pour les hôtes (serveurs)

```bash ssh-keygen -t ed25519 -a 200 \

1. f mis-hosts-ca \
2. C “MIS Hosts CA - Make IT Simple” \
3. N 'AUTRE_PASSPHRASE_TRES_ROBUSTE'

```

Étape 2.3 : Backup chiffré des clés privées

⚠ **Étape critique**. Sans backup, perdre la LXC = perdre tout le contrôle d'accès.

```bash # Création d'une archive chiffrée cd ~/mis-ca tar czf - mis-users-ca mis-hosts-ca | \

```
age -p > mis-ca-backup-$(date +%Y%m%d).tar.gz.age
```

# Demande une passphrase distincte des passphrases des CAs

# À conserver dans : # 1. Bitwarden (attachment de l'entrée “MIS SSH CA - Master Backup”) # 2. Clé USB chiffrée dans coffre physique # 3. (optionnel) Coffre bancaire pour archive papier-QR de la passphrase ```

—

## Phase 3 : Bootstrap d'un serveur client

### Étape 3.1 : Déploiement de la CA users sur le serveur

Sur un serveur client (Debian provisionné via cloud-init) :

```bash # Copie de la clé publique de la CA users scp ca-signer@10.10.10.50:~/mis-ca/mis-users-ca.pub /etc/ssh/mis-users-ca.pub chmod 644 /etc/ssh/mis-users-ca.pub

# Création d'un KRL vide initial ssh-keygen -k -f /etc/ssh/mis-revoked-keys chmod 644 /etc/ssh/mis-revoked-keys

# Configuration sshd cat » /etc/ssh/sshd_config «'EOF'

#

# MIS SSH CA

TrustedUserCAKeys /etc/ssh/mis-users-ca.pub RevokedKeys /etc/ssh/mis-revoked-keys PasswordAuthentication no KbdInteractiveAuthentication no PubkeyAuthentication yes #

EOF

# Test et reload sshd -t && systemctl reload ssh ```

À industrialiser via Ansible (voir Phase 7).

### Étape 3.2 : Certificat d'hôte pour le serveur (recommandé)

Sur la CA :

```bash # Récupération de la clé publique du serveur scp mis-admin@srv01.acme.lan:/etc/ssh/ssh_host_ed25519_key.pub /tmp/srv01.pub

# Signature comme hôte ssh-keygen -s ~/mis-ca/mis-hosts-ca \

1. I “srv01.acme.lan-$(date +%Y)” \
2. h \
3. n “srv01.acme.lan,srv01,192.168.1.50” \
4. V “+104w” \

/tmp/srv01.pub

# Envoi sur le serveur scp /tmp/srv01-cert.pub mis-admin@srv01.acme.lan:/tmp/ ssh mis-admin@srv01.acme.lan “sudo mv /tmp/srv01-cert.pub /etc/ssh/ssh_host_ed25519_key-cert.pub”

# Activation côté serveur ssh mis-admin@srv01.acme.lan “sudo sed -i '/^HostCertificate/d' /etc/ssh/sshd_config && \

```
echo 'HostCertificate /etc/ssh/ssh_host_ed25519_key-cert.pub' | sudo tee -a
/etc/ssh/sshd_config && \
sudo systemctl reload ssh"
```

```

### Étape 3.3 : Configuration côté postes techniciens

Sur **chaque** poste tech (Windows ou Linux), dans `~/.ssh/known_hosts` :

``` @cert-authority *.acme.lan,*.mis.lan,10.10.* ssh-ed25519 AAAAC3… MIS Hosts CA ```
```

(la clé publique correspond au contenu de `mis-hosts-ca.pub`)

À partir de ce moment, plus aucun warning de host key à la première connexion sur un serveur MIS.

—

Phase 4 : Enrôlement d'une Yubikey technicien

Étape 4.1 : Génération sur le poste du technicien

Sur Windows :

```powershell # PowerShell, Yubikey branchée # Définition d'un PIN FIDO2 si pas déjà fait (via Yubikey Manager GUI)

ssh-keygen -t ed25519-sk -O resident -O application=ssh:mis -O verify-required `

1. f $env:USERPROFILE\.ssh\id_ed25519_sk_yubi1 `
2. C “vincent@yubi-primary”

```

Sur Linux/macOS :

```bash ssh-keygen -t ed25519-sk -O resident -O application=ssh:mis -O verify-required \

1. f ~/.ssh/id_ed25519_sk_yubi1 \
2. C “vincent@yubi-primary”

```

Résultat : `id_ed25519_sk_yubi1` (stub) et `id_ed25519_sk_yubi1.pub` (clé publique).

Étape 4.2 : Signature par la CA

Le technicien envoie sa `.pub` à l'admin MIS (via Odoo, mail, ou git interne). Sur la CA :

```bash # Sur la LXC, en tant que ca-signer cd ~/mis-ca

# Variables PUBKEY_PATH=/tmp/vincent-yubi-primary.pub CERT_ID=“vincent-yubi-primary-$(date +%Y)” PRINCIPALS=“mis-admin,vincent” VALIDITY=”+52w“ SERIAL=1001 # incrémenté à chaque signature

ssh-keygen -s mis-users-ca \

1. I “$CERT_ID” \
2. n “$PRINCIPALS” \
3. V “$VALIDITY” \
4. z $SERIAL \
5. O clear \
6. O no-port-forwarding \
7. O no-agent-forwarding \
8. O permit-pty \
```

“$PUBKEY_PATH”

Le fichier vincent-yubi-primary-cert.pub est généré à côté de la .pub ```

Renvoyer le `-cert.pub` au technicien.

Étape 4.3 : Mise en place côté technicien

Le tech place le `-cert.pub` à côté de sa clé :

``` ~/.ssh/ ├── id_ed25519_sk_yubi1 ├── id_ed25519_sk_yubi1.pub └── id_ed25519_sk_yubi1-cert.pub ← nouveau ```

Vérification du contenu du certificat :

```bash ssh-keygen -L -f ~/.ssh/id_ed25519_sk_yubi1-cert.pub ```

Doit afficher : Principals, Valid from/to, Critical Options, Extensions.

Étape 4.4 : Test de connexion

```bash ssh -i ~/.ssh/id_ed25519_sk_yubi1 mis-admin@srv01.acme.lan # → PIN FIDO2 demandé # → touch Yubikey # → connecté ```

Indicateur clé : aucune modification d'`authorized_keys` sur le serveur. Tout passe par le certificat.

Étape 4.5 : Enrôlement de la Yubikey de backup

Répéter Étape 4.1 à 4.4 avec la deuxième Yubikey. Important : utiliser un serial différent (`-z 1002`) et un identifiant unique (`vincent-yubi-backup-2026`).

À la fin, le technicien a 6 fichiers dans `~/.ssh/` :

``` id_ed25519_sk_yubi1 id_ed25519_sk_yubi1.pub id_ed25519_sk_yubi1-cert.pub id_ed25519_sk_yubi2 id_ed25519_sk_yubi2.pub id_ed25519_sk_yubi2-cert.pub ```

Étape 4.6 : Configuration `~/.ssh/config` côté tech

``` Host *.acme.lan acme-*

```
User mis-admin
IdentityFile ~/.ssh/id_ed25519_sk_yubi1
IdentityFile ~/.ssh/id_ed25519_sk_yubi2
IdentitiesOnly yes
CertificateFile ~/.ssh/id_ed25519_sk_yubi1-cert.pub
CertificateFile ~/.ssh/id_ed25519_sk_yubi2-cert.pub
```

Host *.mis.lan

```
User mis-admin
IdentityFile ~/.ssh/id_ed25519_sk_yubi1
IdentityFile ~/.ssh/id_ed25519_sk_yubi2
IdentitiesOnly yes
```
```

```
  CertificateFile ~/.ssh/id_ed25519_sk_yubi1-cert.pub
  CertificateFile ~/.ssh/id_ed25519_sk_yubi2-cert.pub
```

```

SSH essaiera Yubi1 d'abord, puis Yubi2 si la première n'est pas branchée. Aucune intervention nécessaire pour basculer.

—

## Phase 5 : Révocation d'une clé

### Étape 5.1 : Mise à jour de la KRL

Sur la CA, si on perd la Yubikey backup de Vincent :

```bash cd ~/mis-ca

# Création du spec de révocation cat > /tmp/krl-spec.txt «'EOF' # Révocation par serial serial: 1002

# Alternative : par identifiant # id: vincent-yubi-backup-2026

# Alternative : par hash de clé publique # sha256: SHA256:xxxxx EOF

# Génération/mise à jour de la KRL ssh-keygen -k -f mis-revoked-keys -s mis-users-ca -u /tmp/krl-spec.txt # -u = update (ajoute au KRL existant) # Sans -u, le KRL est remplacé

# Vérification du contenu ssh-keygen -Q -f mis-revoked-keys -t krl ```

### Étape 5.2 : Déploiement de la KRL

Via Ansible sur tout le parc (voir Phase 7) :

```bash ansible-playbook -i inventory.yml mis-deploy-krl.yml ```

En quelques minutes, la Yubikey backup n'est plus utilisable sur aucun serveur MIS. **Aucune modification de `sshd_config` ni d'`authorized_keys` n'est requise.**

—

## Phase 6 : Renouvellement des certificats

Les certificats ayant une durée de vie de 1 an, prévoir un workflow de renouvellement :

### Option A : Manuel

Refaire les étapes 4.2 (signature) avec un nouvel identifiant `vincent-yubi-primary-2027` et envoyer le nouveau `-cert.pub` au tech. Le tech remplace son ancien `-cert.pub`.

### Option B : Automatisé via Odoo

Cron Odoo qui scanne les certificats expirant dans 30 jours et : 1. Notifie le tech par mail 2. Propose un wizard de renouvellement 3. Si le tech accepte, regénère un certif via l'API de la CA 4. Envoie le nouveau `-cert.pub` au tech (par mail chiffré ou portail download)
```

—

Phase 7 : Industrialisation Ansible

Playbook de déploiement initial CA + KRL

`playbooks/mis-ssh-ca-deploy.yml` :

```yaml — - name: Deploy MIS SSH CA on all client servers
hosts: all_clients
become: true
vars:
  ca_pub_path: files/mis-users-ca.pub
  krl_path: files/mis-revoked-keys

tasks:
  - name: Deploy MIS users CA public key
    copy:
      src: "{{ ca_pub_path }}"
      dest: /etc/ssh/mis-users-ca.pub
      owner: root
      group: root
      mode: '0644'
    notify: reload sshd
  - name: Deploy current KRL
    copy:
      src: "{{ krl_path }}"
      dest: /etc/ssh/mis-revoked-keys
      owner: root
      group: root
      mode: '0644'
    notify: reload sshd
  - name: Configure sshd for MIS CA
    blockinfile:
      path: /etc/ssh/sshd_config
      marker: "# {mark} MIS CA managed by Ansible"
      block: |
        TrustedUserCAKeys /etc/ssh/mis-users-ca.pub
        RevokedKeys /etc/ssh/mis-revoked-keys
        PasswordAuthentication no
        KbdInteractiveAuthentication no
      validate: 'sshd -t -f %s'
    notify: reload sshd
  - name: Ensure mis-admin user exists
    user:
      name: mis-admin
      groups: sudo
      append: yes
      shell: /bin/bash
      state: present
  - name: Sudo NOPASSWD for mis-admin
```

```
    copy:
      content: "mis-admin ALL=(ALL) NOPASSWD:ALL\n"
      dest: /etc/sudoers.d/mis-admin
      mode: '0440'
      validate: 'visudo -cf %s'

handlers:
  - name: reload sshd
    systemd:
      name: ssh
      state: reloaded
```

```

### Playbook de mise à jour KRL uniquement (rapide)

`playbooks/mis-update-krl.yml` :

```yaml — - name: Update MIS SSH KRL on all client servers

```
hosts: all_clients
become: true

tasks:
  - name: Deploy updated KRL
    copy:
      src: files/mis-revoked-keys
      dest: /etc/ssh/mis-revoked-keys
      mode: '0644'
    notify: reload sshd

handlers:
  - name: reload sshd
    systemd:
      name: ssh
      state: reloaded
```

```

—

## Phase 8 : Compte parachute (break-glass)

⚠ **Ne JAMAIS sauter cette étape.**

Indépendamment de la CA, chaque serveur doit avoir au moins un accès de secours :

```bash # Sur un poste sûr, génération d'une clé classique pour le parachute ssh-keygen -t ed25519 -a 200 -f mis-breakglass -C “MIS BREAK-GLASS - DO NOT USE” # Choisir une passphrase ultra-robuste

# Stocker mis-breakglass (privé) : # - Chiffré dans Bitwarden (attachment) # - Sur clé USB chiffrée dans coffre physique # - Supprimer du disque local après stockage sécurisé

# Garder mis-breakglass.pub pour déploiement ```

Intégrer la clé publique dans le template cloud-init / virt-customize de **tous** les serveurs :

```bash # Dans virt-customize de la golden image virt-customize -a debian-13-genericcloud-amd64.qcow2 \

1. -run-command 'useradd -m -s /bin/bash mis-breakglass' \
2. -ssh-inject mis-breakglass:file:/path/to/mis-breakglass.pub \
3. -run-command 'echo “mis-breakglass ALL=(ALL) NOPASSWD:ALL” > /etc/sudoers.d/mis-breakglass' \

… ```

Le jour où la CA est inaccessible (perte de toutes les Yubikeys, corruption de la LXC, etc.), tu sors la clé du coffre Bitwarden, tu te connectes avec, tu reconstruis la situation.

—

## Phase 9 : Migration future vers Yubikey PIV pour la CA

Quand le workflow sera rodé (6-12 mois), migration de la clé privée de la CA depuis disque vers Yubikey PIV :

```bash # Sur un poste sûr, Yubikey dédiée “CA Master” branchée apt install yubikey-manager opensc

# Génération d'une nouvelle paire ECCP384 dans slot 9c ykman piv keys generate –algorithm ECCP384 –pin-policy ONCE –touch-policy CACHED \

```
9c /tmp/mis-ca-piv.pub
```

Certificat self-signed pour PIV ykman piv certificates generate –subject “CN=MIS Users CA” 9c /tmp/mis-ca-piv.pub

Export au format SSH ssh-keygen -D /usr/lib/x86_64-linux-gnu/opensc-pkcs11.so -e > mis-users-ca-yubi.pub ```

Pour signer ensuite :

```bash ssh-keygen -s mis-users-ca-yubi.pub \

1. D /usr/lib/x86_64-linux-gnu/opensc-pkcs11.so \
2. I “vincent-yubi-primary-2027” \
3. n “mis-admin,vincent” \
4. V ”+52w“ \

vincent-yubi-primary.pub # → PIN PIV demandé # → touch Yubikey # → certificat signé ```

La clé privée de la CA n'existe plus sur disque, elle est uniquement dans la Yubikey “CA Master” (qui doit avoir une jumelle de backup stockée en coffre).

—
```

Annexes

A. Commandes utiles de diagnostic

```bash # Inspecter un certificat utilisateur ssh-keygen -L -f id_ed25519_sk_yubi1-cert.pub

# Voir la KRL active sur un serveur ssh-keygen -Q -f /etc/ssh/mis-revoked-keys -t krl

# Tester un certif spécifique contre la KRL ssh-keygen -Q -f /etc/ssh/mis-revoked-keys -t krl id_ed25519_sk_yubi1-cert.pub # Output: “id_ed25519_sk_yubi1-cert.pub: ok” ou “REVOKED”

# Côté serveur, voir quelles connexions par certificat journalctl -u ssh | grep “Accepted publickey.*ID”

# Lister les clés résidentes sur une Yubikey ssh-keygen -K # Récupère les stubs sur le poste courant depuis la Yubikey ```

### B. Format complet d'une signature ssh-keygen

```bash ssh-keygen -s CA_PRIVATE_KEY \

1. I CERT_IDENTIFIER \
2. n PRINCIPAL1,PRINCIPAL2 \
3. V ”+VALIDITY“ \
4. z SERIAL \
5. O OPTION1 \
6. O OPTION2 \

USER_PUBLIC_KEY.pub ```

| Option | Description |
|---|---|
| ——– | ——— |
| `-s` | Chemin vers la clé privée de la CA |
| `-I` | Identifiant lisible du certificat (visible dans logs sshd) |
| `-n` | Principals (noms d'utilisateurs autorisés), séparés par virgule |
| `-V` | Validité (`+52w`, `+30d`, `+8h`, `20260601:20270601`) |
| `-z` | Serial unique (entier) |
| `-h` | Génère un certificat d'**hôte** (sinon utilisateur par défaut) |
| `-O clear` | Reset les permissions par défaut |
| `-O no-port-forwarding` | Désactive le port forwarding |
| `-O no-agent-forwarding` | Désactive l'agent forwarding |
| `-O no-X11-forwarding` | Désactive X11 |
| `-O permit-pty` | Autorise l'allocation PTY (interactif) |
| `-O force-command=”...“` | Force l'exécution d'une commande spécifique |
| `-O source-address=“10.0.0.0/8”` | Restreint l'origine IP |

### C. Liens utiles

- OpenSSH manual `ssh-keygen(1)` : `man ssh-keygen` - Documentation officielle Yubico FIDO2 SSH : https://developers.yubico.com/SSH/ - Cloud-init reference : https://cloudinit.readthedocs.io/ - Format des certificats SSH (technique) : `PROTOCOL.certkeys` dans le code OpenSSH

—

## Checklist de mise en place

### Phase 0 - Préparation - [ ] Yubikeys commandées (minimum 2 par technicien + 1 vault) - [ ] PINs FIDO2 définis sur chaque Yubikey (via Yubikey Manager) - [ ] Bitwarden ready pour stockage des secrets

### Phase 1-2 - CA - [ ] LXC `mis-ssh-ca` créée et durcie - [ ] CA users générée + passphrase robuste - [ ] CA hosts générée + passphrase robuste - [ ] Backup chiffré des CAs dans Bitwarden + offline - [ ] LXC éteinte par défaut, allumage à la demande

### Phase 3 - Bootstrap parc - [ ] Compte break-glass `mis-breakglass` créé et clé déployée partout - [ ] Playbook Ansible `mis-ssh-ca-deploy.yml` testé sur 1-2 serveurs - [ ] Déploiement progressif sur le parc complet - [ ] Certificats d'hôtes générés pour les serveurs critiques

### Phase 4 - Techniciens - [ ] Premières Yubikeys enrôlées (toi en priorité) - [ ] Tests de connexion validés - [ ] Configuration `~/.ssh/config` documentée pour les techs

### Phase 5+ - Industrialisation - [ ] Module Odoo `makeitsimple_ssh_ca` (modèles + wizards) - [ ] Intégration Semaphore pour déploiement KRL - [ ] Cron de notification d'expiration - [ ] Migration CA vers Yubikey PIV (à 6-12 mois)

—

*Document maintenu par Make IT Simple - dernière révision : 2026*