

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.

Architecture allégée : la clé privée de la CA est stockée sur une Yubikey 5 NFC dédiée (slot PIV), pas sur disque. Un backup d'urgence (fichier classique chiffré) est conservé dans Bitwarden pour les cas de perte matérielle.

Compatibilité Windows

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

Rôle	Système	Notes
Poste de signature (utilisation Yubikey CA)	Windows / Linux / macOS	OpenSSH + OpenSC ou ykcs11
Poste technicien (utilisation Yubikey perso)	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

Pour signer avec la Yubikey CA depuis Windows : Yubico fournit `ykcs11.dll` (inclus dans `yubico-piv-tool`) qui peut être utilisé comme provider PKCS#11 pour `ssh-keygen -D`. Marche aussi.

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 :

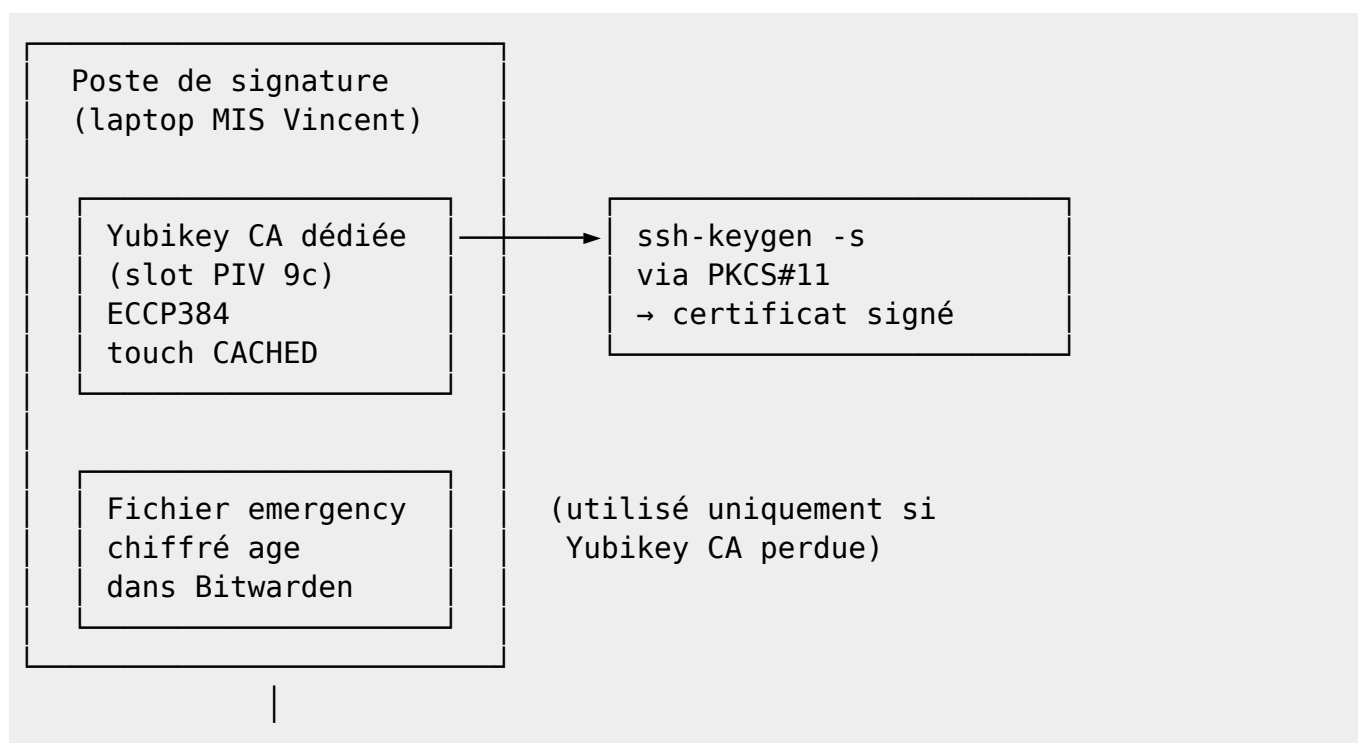
- Les **clés DSA ne sont plus supportées du tout**, même avec les options de compat (HostKeyAlgorithms, PubkeyAcceptedAlgorithms). Aucun impact pour nous (on n'utilise que Ed25519/ECCP384), mais à savoir si tu te connectes à de très vieux équipements.
- Le support FIDO2 (libfido2) est plus mature qu'en Bookworm.
- Les algorithmes post-quantum (mlkem768x25519-sha256) sont disponibles par défaut pour le key exchange.

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.

Vue d'ensemble de l'architecture



↓ déploiement via Ansible
▼

Serveurs clients

/etc/ssh/mis-users-ca.pub
(200+ VMs)

/etc/ssh/mis-revoked-keys

← contient 2 lignes :

1. Yubikey CA pub
2. Emergency CA pub

← KRL pour révocations

Le principe :

- La CA quotidienne vit sur Yubikey PIV → clé privée non-extractible, touch + PIN obligatoires
- La CA emergency est un fichier classique chiffré avec age, stocké dans Bitwarden
- Les **deux** clés publiques sont déployées sur les serveurs comme CAs valides
- En cas de perte de la Yubikey CA, on déchiffre l'emergency, on s'en sert pour signer en attendant la nouvelle Yubikey, puis on révoque (rolling rotation)

Phase 1 : Préparation du poste de signature

Étape 1.1 : Matériel requis

- **1x Yubikey 5 NFC dédiée à la CA** (~50 €). À **séparer** de tes Yubikeys SSH d'authentification quotidienne. Cette Yubikey reste rangée la plupart du temps, branchée uniquement pour signer.
- Ton laptop MIS comme poste de signature (Linux, macOS ou Windows)

Étape 1.2 : Installation des outils

Sur Debian / Ubuntu / WSL2 :

```
sudo apt update
sudo apt install -y \
  openssh-client \
  yubikey-manager \
  opensc \
  age
```

Sur macOS (Homebrew) :

```
brew install yubikey-manager opensc age openssh
```

Sur Windows :

```
# Yubikey Manager : https://www.yubico.com/support/download/yubikey-manager/
# Installer aussi yubico-piv-tool qui contient ykcs11.dll
# OpenSSH inclus avec Windows 10/11 (sinon : Add-WindowsCapability)
```

```
# age : winget install FiloSottile.age
```

Étape 1.3 : Localisation du provider PKCS#11

Cette information est cruciale pour les commandes de signature. Note le chemin selon ton OS :

OS	Chemin PKCS#11 (OpenSC)
Debian/Ubuntu (amd64)	/usr/lib/x86_64-linux-gnu/opensc-pkcs11.so
Debian/Ubuntu (arm64)	/usr/lib/aarch64-linux-gnu/opensc-pkcs11.so
macOS (Intel)	/usr/local/lib/opensc-pkcs11.so
macOS (Apple Silicon)	/opt/homebrew/lib/opensc-pkcs11.so
Windows	C:\Program Files\OpenSC Project\OpenSC\pkcs11\opensc-pkcs11.dll

Vérification rapide qu'il existe :

```
# Linux/macOS
ls -la $PKCS11_PATH
```

Pour la suite du document, on utilisera la variable \$PKCS11 à adapter à ton chemin :

```
export PKCS11=/usr/lib/x86_64-linux-gnu/opensc-pkcs11.so
```

Étape 1.4 : Initialisation des PINs de la Yubikey CA

⚠ **Yubikey CA branchée.** Vérifie que c'est la bonne (pas une Yubikey SSH du quotidien) :

```
ykman info
# Doit afficher le serial number de la Yubikey CA
```

PIN PIV (8 caractères minimum, sera demandé à chaque session de signature) :

```
ykman piv access change-pin
# Default PIN: 123456
# New PIN: [choisir 8+ chars]
```

PUK (utilisé pour débloquer si trop d'échecs PIN, à conserver précieusement) :

```
ykman piv access change-puk
# Default PUK: 12345678
# New PUK: [générer 8 chars aléatoires]
```

Management Key (utilisée pour les opérations admin PIV, on la protège avec le PIN) :

```
ykman piv access change-management-key --algorithm AES256 --generate --
```

```
protect
```

```
# --protect = stocke la management key chiffrée par le PIN
```

☐ Notes à conserver dans Bitwarden, entrée "MIS Yubikey CA" :

- Serial number de la Yubikey
- PIN PIV
- PUK
- (la management key est protégée par le PIN, pas besoin de la stocker à part)

Phase 2 : Génération des CAs

Étape 2.1 : CA principale sur Yubikey (users)

```
# Yubikey CA branchée
mkdir -p ~/mis-ssh-ca && cd ~/mis-ssh-ca
chmod 700 .

# Génération de la paire ECCP384 dans le slot 9c (Digital Signature)
ykman piv keys generate \
  --algorithm ECCP384 \
  --pin-policy ONCE \
  --touch-policy CACHED \
  9c \
  mis-users-ca-yubi-cert.pub
# → demande le PIN
# → demande le touch
```

Décortiquons les options :

Option	Valeur	Sens
--algorithm	ECCP384	Courbe elliptique NIST P-384 (compromis sécurité/perf)
--pin-policy	ONCE	PIN demandé 1x par session (après débranchement = re-demandé)
--touch-policy	CACHED	Touch valide 15s après un précédent touch
9c	slot	Slot "Digital Signature" - sémantique appropriée

Alternative plus stricte : --touch-policy ALWAYS exige un touch à **chaque** signature. Plus sûr mais pénible pour signer plusieurs certifs d'affilée (enrôlement de plusieurs Yubikeys d'un nouveau tech). CACHED est un bon compromis.

Étape 2.2 : Certificat self-signed PIV

PIV exige qu'un certificat X.509 soit présent dans le slot, même s'il ne sert à rien dans notre usage SSH :

```
ykman piv certificates generate \
```

```
--subject "CN=MIS Users CA,O=Make IT Simple,C=BE" \  
--valid-days 3650 \  
9c \  
mis-users-ca-yubi-cert.pub  
# → demande le PIN
```

Étape 2.3 : Export de la clé publique au format SSH

C'est cette clé publique qui sera déployée sur tous les serveurs comme **CA principale acceptée** :

```
ssh-keygen -D $PKCS11 -e > mis-users-ca-yubi.pub  
  
# Inspection  
cat mis-users-ca-yubi.pub  
# Doit afficher : ecdsa-sha2-nistp384 AAAA...  
  
# Renommer avec un commentaire explicite (facultatif)  
sed -i 's|$| MIS Users CA - Yubikey|' mis-users-ca-yubi.pub
```

Étape 2.4 : Génération de la CA emergency (container éphémère)

La CA emergency est une clé privée Ed25519 classique, mais qui ne doit **jamais exister en clair sur ton laptop** après génération. La meilleure approche : un container Docker éphémère sans réseau, qui ne laisse aucune trace après destruction.

Pourquoi pas juste sur ton laptop ? Parce qu'entre la génération en clair et le chiffrement age, la clé existe en clair sur ton disque. Avec un container sans réseau, aucun process tiers (même un éventuel malware) ne peut exfiltrer pendant cette fenêtre.

Génération via Docker :

```
# Lancement d'un container Debian 13 minimal, SANS RÉSEAU  
docker run --rm -it \  
--network none \  
--hostname offline-ca-gen \  
-v ~/mis-ssh-ca:/output \  
debian:trixie-slim bash
```

Dans le container (aucun accès Internet possible) :

```
# Installation des outils (depuis le cache APT du container, pas Internet)  
# Si l'image n'a pas les paquets, faire docker build préalable (voir note  
ci-dessous)  
apt-get install -y openssh-client age coreutils 2>/dev/null || echo "Image  
sans cache, voir note"  
  
cd /tmp
```

```
# Génération Ed25519 classique
ssh-keygen -t ed25519 -a 200 \
  -f mis-users-ca-emergency \
  -C "MIS Users CA - EMERGENCY USE ONLY" \
  -N ''

# Chiffrement avec age - passphrase en interactif (PAS en argument !)
age --passphrase \
  -o mis-users-ca-emergency.age \
  mis-users-ca-emergency
# → entrer une passphrase TRÈS robuste (Diceware 6-8 mots recommandé)

# Effacement sécurisé de la version en clair
shred -u mis-users-ca-emergency

# Copie des fichiers finaux vers l'hôte via le volume monté
cp mis-users-ca-emergency.age /output/
cp mis-users-ca-emergency.pub /output/

# Sortie du container → destruction automatique (--rm)
exit
```

Note : si `debian:trixie-slim` n'a pas les paquets en cache, prépare une image avec les outils inclus une fois pour toutes :

```
# À faire une seule fois, sur ton hôte
cat > /tmp/Dockerfile <<'EOF'
FROM debian:trixie-slim
RUN apt-get update && \
  apt-get install -y --no-install-recommends \
  openssh-client age coreutils && \
  rm -rf /var/lib/apt/lists/*
EOF

docker build -t mis-ca-gen:latest -f /tmp/Dockerfile /tmp/

# Puis utiliser cette image (en remplaçant debian:trixie-slim) :
docker run --rm -it --network none -v ~/mis-ssh-ca:/output mis-ca-gen:latest
bash
```

Variante Podman rootless (si tu préfères ne pas dépendre d'un daemon Docker root) :

```
podman run --rm -it --network none -v ~/mis-ssh-ca:/output:Z mis-ca-gen:latest bash
```

Résultat sur ton hôte :

```
ls -la ~/mis-ssh-ca/
# mis-users-ca-emergency.age      ← clé privée DOUBLEMENT chiffrable (age + Bitwarden)
# mis-users-ca-emergency.pub     ← clé publique (pas un secret)
```

```
# mis-users-ca-yubi.pub          ← clé publique de la Yubikey CA
# mis-users-ca-yubi-cert.pub     ← cert X.509 PIV (inutile pour SSH)
```

Étape 2.5 : CA pour les hôtes (optionnelle pour démarrer)

Même approche pour la CA d'hôtes, si tu décides de la mettre en place :

- Soit sur la **même Yubikey** dans le slot 9a (Authentication) ou 9d (Key Management)
- Soit avec un fichier classique chiffré, plus simple pour démarrer

Comme tu as décidé de ne pas faire les certifs machines tout de suite, on saute cette étape pour le moment. À reprendre quand tu te lanceras dans les certifs d'hôtes.

Étape 2.6 : Stratégie de stockage des secrets (double chiffrement)

Principe : la clé emergency est protégée par **deux couches** indépendantes :

```
Clé privée Ed25519 (claire)
    ↓ couche 1 : chiffrement age (passphrase forte)
Fichier .age (illisible sans passphrase age)
    ↓ couche 2 : attachment dans Bitwarden (chiffré par master password)
Coffre Bitwarden (illisible sans master password)
```

Pour que cette défense en profondeur fonctionne, **les deux secrets doivent être stockés à des endroits différents** : sinon une seule compromission donne accès à tout.

Ce qui va dans Bitwarden

Entrée **"MIS SSH CA - Master Backup"** :

Notes (champ texte de l'entrée) :

PROCÉDURE DE RÉCUPÉRATION D'URGENCE

1. Récupérer la passphrase age :
 - Coffre bureau MIS, enveloppe scellée "CA Emergency"
 - OU coffre backup (maison Vincent / coffre bancaire)
2. Télécharger mis-users-ca-emergency.age depuis cette entrée
3. Sur container éphémère :

```
docker run --rm -it --network none -v $PWD:/work mis-ca-gen bash
cd /work
age --decrypt mis-users-ca-emergency.age > recovery-key
chmod 600 recovery-key
# utiliser pour signer (voir Phase 9 du runbook)
```

```
shred -u recovery-key
exit
```

INFOS YUBIKEY CA

```
Serial number      : [serial à compléter]
PIN PIV           : [PIN à compléter]
PUK               : [PUK à compléter]
Algorithme        : ECCP384
Slot              : 9c
Touch policy      : CACHED
PIN policy         : ONCE
```

FINGERPRINTS (vérification d'intégrité)

```
Yubikey CA pub    : [ssh-keygen -l -f mis-users-ca-yubi.pub]
Emergency CA pub  : [ssh-keygen -l -f mis-users-ca-emergency.pub]
```

Attachments (fichiers attachés à l'entrée) :

Fichier	Sensibilité	Pourquoi dans Bitwarden
mis-users-ca-emergency.age	DOUBLEMENT chiffré	Clé privée emergency, inutilisable sans la passphrase age
mis-users-ca-emergency.pub	Publique (pas un secret)	Pratique pour la déployer et la vérifier
mis-users-ca-yubi.pub	Publique (pas un secret)	Pratique pour le déploiement
mis-users-ca-deploy.pub	Publique	Concaténation des 2 pubs prête à déployer

Ce qui ne doit JAMAIS aller dans Bitwarden :

- ❑ La passphrase age (sinon couche 1 + couche 2 = même point de défaillance)
- ❑ La clé privée emergency en clair
- ❑ Un export PKCS#12 de la Yubikey (impossible techniquement, mais à ne pas chercher à faire)

Ce qui va sur papier (hors numérique)

La **passphrase age** uniquement. À écrire à la main sur papier :

- **Localisation principale** : coffre physique au bureau MIS, dans une enveloppe scellée "CA Emergency - Vincent uniquement"
- **Localisation backup** : coffre physique à la maison de Vincent (ou coffre bancaire), même enveloppe

Format recommandé : passphrase Diceware 6-8 mots, plus facile à recopier sans erreur qu'une chaîne aléatoire.

Exemple : correct horse battery staple winter coffee (à remplacer évidemment).

Approche avancée (optionnelle) : Shamir Secret Sharing avec `ssss-split` pour découper la passphrase en N parts dont K nécessaires. Exemple : 3 parts, 2 nécessaires, distribuées entre Vincent + associé + avocat de confiance. Sécurité maximale, mais à mettre en place quand tout le reste est rodé.

Backup hors-Bitwarden additionnel

En complément (ceinture-bretelles) :

- Clé USB chiffrée LUKS dans coffre physique avec une copie de tout `~/mis-ssh-ca/`
- À refaire à chaque rotation/renouvellement de CA
- La passphrase LUKS est encore un secret différent (toujours pas dans Bitwarden)

Étape 2.7 : Test de récupération (CRITIQUE)

⚠ **Avant de tout ranger dans le coffre et de considérer le setup terminé**, tu dois faire un test de récupération complet. C'est l'étape qui te garantit que le jour J, la procédure fonctionnera vraiment.

```
# 1. Dans un dossier temporaire de test
mkdir /tmp/test-recovery && cd /tmp/test-recovery

# 2. Copier le .age depuis Bitwarden (téléchargement de l'attachment)
# OU depuis ~/mis-ssh-ca/ si encore dispo
cp ~/mis-ssh-ca/mis-users-ca-emergency.age .

# 3. Aller récupérer le papier dans le coffre, lire la passphrase
# (Geste réel, pas une simulation : tu dois vérifier que la passphrase
# est bien lisible et que tu peux la recopier sans erreur)

# 4. Déchiffrer en utilisant la passphrase du papier
age --decrypt mis-users-ca-emergency.age > recovery-key
# → entrer la passphrase telle qu'écrite sur papier

# 5. Vérifier que la clé déchiffrée est bien la bonne
ssh-keygen -y -f recovery-key > recovered.pub
diff recovered.pub ~/mis-ssh-ca/mis-users-ca-emergency.pub
# Sortie attendue : VIDE (les deux fichiers sont identiques)
# Si différents → la passphrase a été mal recopiée OU le .age est corrompu

# 6. Nettoyage immédiat
shred -u recovery-key recovered.pub
cd .. && rm -rf /tmp/test-recovery
```

Si le test réussit, tu as validé :

- La passphrase est correctement écrite sur papier (lisible, sans erreur)

- Le fichier .age dans Bitwarden n'est pas corrompu
- Tu sais exécuter la procédure (le geste compte autant que le savoir)
- Les fingerprints correspondent

Si le test échoue, NE PAS continuer. Identifier le problème (passphrase mal recopiée ? mauvais fichier .age ?) et régénérer si nécessaire.

À refaire annuellement dans le cadre du drill de récupération (voir checklist en fin de doc).

Phase 3 : Bootstrap d'un serveur client

Étape 3.1 : Préparation du fichier de CAs publiques

Sur ton poste de signature, prépare un fichier qui contient les **deux** clés publiques (Yubikey + emergency) que tu vas déployer sur tous les serveurs :

```
cd ~/mis-ssh-ca
cat mis-users-ca-yubi.pub mis-users-ca-emergency.pub > mis-users-ca-deploy.pub

cat mis-users-ca-deploy.pub
# Doit contenir 2 lignes :
# ecdsa-sha2-nistp384 AAAA... MIS Users CA - Yubikey
# ssh-ed25519 AAAA... MIS Users CA - EMERGENCY USE ONLY
```

C'est ce fichier `mis-users-ca-deploy.pub` qui sera distribué sur tout le parc. OpenSSH accepte plusieurs CAs dans un même `TrustedUserCAKeys`, une par ligne.

Étape 3.2 : Déploiement sur un serveur client

Sur un serveur client (Debian 13 provisionné via cloud-init) :

```
# Copie du fichier de CAs
scp ~/mis-ssh-ca/mis-users-ca-deploy.pub mis-admin@srv01.acme.lan:/tmp/

# Côté serveur, déploiement
ssh mis-admin@srv01.acme.lan
sudo install -o root -g root -m 0644 /tmp/mis-users-ca-deploy.pub
/etc/ssh/mis-users-ca.pub

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

# Configuration sshd
sudo tee -a /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
sudo sshd -t && sudo systemctl reload ssh
```

À industrialiser via Ansible (voir Phase 7).

Phase 4 : Enrôlement d'une Yubikey technicien

Étape 4.1 : Génération sur le poste du technicien

Sur Windows (PowerShell, Yubikey du tech 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 \
  -f $env:USERPROFILE\.ssh\id_ed25519_sk_yubi1 \
  -C "vincent@yubi-primary"
```

Sur Linux/macOS :

```
ssh-keygen -t ed25519-sk -O resident -O application=ssh:mis -O verify-
required \
  -f ~/.ssh/id_ed25519_sk_yubi1 \
  -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 Yubikey

Le tech envoie sa .pub au poste de signature. **Yubikey CA branchée** :

```
cd ~/mis-ssh-ca
export PKCS11=/usr/lib/x86_64-linux-gnu/opensc-pkcs11.so

# Variables
```

```
PUBKEY=/tmp/vincent-yubi-primary.pub
CERT_ID="vincent-yubi-primary-$(date +%Y)"
PRINCIPALS="mis-admin,vincent"
VALIDITY="+52w"
SERIAL=1001
```

```
ssh-keygen -s mis-users-ca-yubi.pub \
  -D $PKCS11 \
  -I "$CERT_ID" \
  -n "$PRINCIPALS" \
  -V "$VALIDITY" \
  -z $SERIAL \
  -O clear \
  -O no-port-forwarding \
  -O no-agent-forwarding \
  -O permit-pty \
  "$PUBKEY"
```

```
# → demande le PIN PIV (1x par session grâce à --pin-policy ONCE)
# → demande le touch sur la Yubikey CA
# → fichier vincent-yubi-primary-cert.pub généré à côté de la .pub
```

Note importante : avec `-D $PKCS11`, l'argument `-s` n'est **pas** la clé privée (qui est sur la Yubikey) mais la clé **publique** de la CA. C'est elle qui permet à `ssh-keygen` d'identifier laquelle utiliser sur le token.

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 :

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

Doit afficher : Signing CA (ECDSA SHA256 :... = ta Yubikey CA), Principals, Valid from/to.

Étape 4.4 : Test de connexion

```
ssh -i ~/ .ssh/id_ed25519_sk_yubi1 mis-admin@srv01.acme.lan
# → PIN FIDO2 demandé (côté Yubikey du tech)
# → touch Yubikey du tech
```

```
# → connecté
```

Étape 4.5 : Enrôlement de la Yubikey de backup

Répéter Étape 4.1 à 4.4 avec la deuxième Yubikey du tech. Utiliser :

- Serial différent (-z 1002)
- Identifiant unique (vincent-yubi-backup-2026)

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

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

Phase 5 : Révocation d'une clé

Étape 5.1 : Mise à jour de la KRL

Yubikey CA branchée sur le poste de signature :

```
cd ~/mis-ssh-ca

# 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 via Yubikey
ssh-keygen -k -f mis-revoked-keys \
  -s mis-users-ca-yubi.pub \
  -D $PKCS11 \
  -u /tmp/krl-spec.txt
# -u = update (ajoute au KRL existant)
```

```
# → demande PIN + touch
```

Étape 5.2 : Déploiement de la KRL via Ansible

```
ansible-playbook -i inventory.yml mis-update-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

Option A : Manuel

Refaire l'étape 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, déclenche une demande de signature (qui attend que tu branches la Yubikey CA + valides physiquement)
4. Envoie le nouveau `-cert.pub` au tech

Phase 7 : Industrialisation Ansible

Playbook de déploiement initial CA + KRL

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

```
---
- name: Deploy MIS SSH CA on all client servers
  hosts: all_clients
  become: true
  vars:
    ca_pub_path: files/mis-users-ca-deploy.pub # contient les 2 CAs
    (Yubikey + emergency)
    krl_path: files/mis-revoked-keys
  tasks:
```

```
- name: Deploy MIS users CA public keys
  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

playbooks/mis-update-krl.yml :

```
---
- 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.**

C'est un parachute **indépendant** de la CA SSH (ni Yubikey, ni emergency CA). Le compte break-glass utilise une clé SSH classique, pour le cas où **toute** la PKI CA serait inaccessible (perte Yubikey + Bitwarden corrompu + papier coffre perdu, scénario apocalypse).

```
# 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) - entrée distincte de la CA
# - 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 :

```
virt-customize -a debian-13-genericcloud-amd64.qcow2 \
  --run-command 'useradd -m -s /bin/bash mis-breakglass' \
  --ssh-inject mis-breakglass:file:/path/to/mis-breakglass.pub \
  --run-command 'echo "mis-breakglass ALL=(ALL) NOPASSWD:ALL" >
/etc/sudoers.d/mis-breakglass' \
  ...
```

Phase 9 : Procédure de récupération d'urgence

Cas 1 : Yubikey CA perdue / cassée, emergency disponible

Workflow :

1. Acheter une nouvelle Yubikey 5 NFC dédiée à la CA
2. Déchiffrer temporairement l'emergency :

```
```bash
```

```
cd ~/mis-ssh-ca
age --decrypt mis-users-ca-emergency.age > /tmp/emergency-key
chmod 600 /tmp/emergency-key
```

- **Utiliser l'emergency CA pour signer une nouvelle CA Yubikey** (chaîne de confiance temporaire) :
  ```bash
 # Préparer la nouvelle Yubikey (Étapes 1.4 + 2.1 à 2.3)
 # Tu obtiens mis-users-ca-yubi-NEW.pub
  ```

- Mettre à jour 'mis-users-ca-deploy.pub' pour inclure la **nouvelle** Yubikey CA + l'emergency (toujours) :
  ```bash
 cat mis-users-ca-yubi-NEW.pub mis-users-ca-emergency.pub > mis-users-ca-deploy.pub
  ```

- Déployer via Ansible sur tout le parc
- Révoquer l'ancienne Yubikey CA via KRL (signature avec la NEW Yubikey) :
  ```bash
 cat > /tmp/krl-revoke-old-ca.txt <<'EOF'
 # Révocation de l'ancienne CA Yubikey perdue
 sha256: SHA256:<fingerprint de mis-users-ca-yubi-OLD.pub>
 EOF

 ssh-keygen -k -f mis-revoked-keys \
 -s mis-users-ca-yubi-NEW.pub \
 -D $PKCS11 \
 -u /tmp/krl-revoke-old-ca.txt
  ```

- **Effacement sécurisé** du fichier emergency en clair :
  ```bash
 shred -u /tmp/emergency-key
  ```

- Mise à jour de l'entrée Bitwarden avec le nouveau serial Yubikey
```

Important : tous les certificats utilisateurs signés par l'ancienne Yubikey CA sont **toujours valides**

(puisque l'ancienne CA est encore dans le déploiement jusqu'à révocation, et que la révocation cible le pub de la CA, pas les certifs signés). Pour invalider tous les certifs émis par l'ancienne CA, supprimer son .pub du déploiement et redéployer.

Cas 2 : Tout est perdu sauf break-glass

Scénario apocalypse : Yubikey CA perdue + Bitwarden inaccessible + papier coffre détruit.

1. Connexion sur chaque serveur via SSH avec mis-breakglass :

```
```bash
```

```
ssh -i ~/.bitwarden-recovered/mis-breakglass mis-breakglass@srv01.acme.lan
```
```

- Régénération complète d'une nouvelle CA (depuis zéro, Phase 2)
- Déploiement manuel du nouveau 'mis-users-ca.pub' sur chaque serveur via 'mis-breakglass'
- Re-signature de toutes les Yubikeys techniciens
- Rotation du break-glass aussi (puisque'il a servi)

À tester : faire un drill annuel de récupération sur un environnement de staging pour valider que la procédure marche **avant** d'en avoir vraiment besoin.

Annexes

A. Commandes utiles de diagnostic

```
# Inspecter un certificat utilisateur
ssh-keygen -L -f id_ed25519_sk_yubil-cert.pub

# Voir l'état PIV de la Yubikey CA
ykman piv info

# Voir le slot 9c (où vit la CA)
ykman piv keys metadata 9c

# 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_yubil-
cert.pub
# Output: "id_ed25519_sk_yubil-cert.pub: ok" ou "REVOKED"

# Côté serveur, voir les connexions par certificat
journalctl -u ssh | grep "Accepted publickey.*ID"
```

```
# Lister les clés publiques visibles via PKCS#11 (avec Yubikey CA branchée)
ssh-keygen -D $PKCS11 -e

# Tester une signature sans toucher à un fichier
echo "test" | ssh-keygen -Y sign -n test -f mis-users-ca-yubi.pub -D $PKCS11
```

B. Format complet d'une signature ssh-keygen avec Yubikey

```
ssh-keygen -s CA_PUBLIC_KEY \
-D PKCS11_PROVIDER \
-I CERT_IDENTIFIER \
-n PRINCIPAL1,PRINCIPAL2 \
-V "+VALIDITY" \
-z SERIAL \
-O OPTION1 \
-O OPTION2 \
USER_PUBLIC_KEY.pub
```

| Option | Description |
|--------------------------------|--|
| -s | Chemin vers la clé publique de la CA (quand -D est utilisé) |
| -D | Path vers le module PKCS#11 (opensc-pkcs11.so) |
| -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 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 (sections CERTIFICATES et KEY REVOCATION LISTS)
- Documentation officielle Yubico FIDO2 SSH : <https://developers.yubico.com/SSH/>
- Documentation Yubico PIV : <https://developers.yubico.com/PIV/Guides/SSH//with//PIV//and//PKCS11.html>
- Documentation age : <https://age-encryption.org/>
- Format des certificats SSH (technique) : PROTOCOL.certkeys dans le code OpenSSH

Checklist de mise en place

Phase 0 - Matériel et logiciels

- Yubikey 5 NFC dédiée à la CA commandée
- Yubikeys 5 NFC techniciens commandées (2 par tech minimum)
- Bitwarden ready avec entrée "MIS SSH CA - Master Backup"
- Coffre physique identifié pour passphrase + clé USB de backup
- OpenSSH + yubikey-manager + opensc + age installés sur le poste de signature

Phase 1-2 - CA

- PINs PIV de la Yubikey CA configurés (PIN + PUK + management key)
- CA principale générée sur Yubikey (slot 9c, ECCP384)
- Clé publique Yubikey CA exportée (mis-users-ca-yubi.pub)
- Image Docker mis-ca-gen préparée
- Emergency CA générée dans container éphémère + chiffrée avec age
- Bitwarden : entrée "MIS SSH CA - Master Backup" avec attachments
- Passphrase age écrite sur papier dans coffre bureau MIS
- Passphrase age (copie) dans coffre backup (maison ou bancaire)
- Backup additionnel sur clé USB LUKS dans coffre
- **Test de récupération validé** (Étape 2.7) avant rangement définitif

Phase 3 - Bootstrap parc

- Compte break-glass mis-breakglass créé et clé déployée partout (indépendant CA)
- Playbook Ansible mis-ssh-ca-deploy.yml testé sur 1-2 serveurs
- Fichier mis-users-ca-deploy.pub contient les 2 CAs (Yubikey + emergency)
- Déploiement progressif sur le parc complet

Phase 4 - Techniciens

- Premières Yubikeys enrôlées (toi en priorité)
- Tests de connexion validés
- Configuration ~/.ssh/config documentée pour les techs

Phase 5+ - Industrialisation

- Module Odo makeitsimple_ssh_ca (modèles + wizards)
- Intégration Semaphore pour déploiement KRL
- Cron de notification d'expiration des certs
- **Drill de récupération annuel** (cas 1 emergency) testé en staging

Document maintenu par Make IT Simple - dernière révision : 2026

From:
<https://wiki.makeitsimple.be/> - **makeITsimple wiki**

Permanent link:
<https://wiki.makeitsimple.be/doku.php?id=linux:ca-ssh&rev=1780494795>

Last update: **2026/06/03 13:53**

