AD CS : générer un certificat TLS pour un serveur Linux depuis une PKI Windows

Générer un certificat TLS pour un serveur Linux depuis une PKI Windows AD CS permet de centraliser la gestion des certificats internes dans un seul annuaire, d'éviter les certificats auto-signés et d'obtenir une chaîne de confiance cohérente pour tous les services internes. Ce guide couvre la génération du CSR avec OpenSSL, la soumission à la CA Windows, la récupération et la conversion du certificat, et son intégration dans Apache, Nginx ou pour l'authentification mutuelle TLS.

L'obtention d'un certificat TLS pour un serveur Linux depuis une PKI Windows AD CS (Active Directory Certificate Services) est une opération que les équipes système effectuent régulièrement pour sécuriser les services web internes, les API REST, les connexions LDAPS ou les accès SSH par certificat. Plutôt que d'utiliser des certificats auto-signés — qui génèrent des alertes navigateur et bloquent les connexions des clients stricts — ou d'exposer des services en HTTP, la PKI Windows interne permet d'émettre des certificats reconnus par tous les postes du domaine via la politique de groupe. Selon les recommandations du guide TLS de l'ANSSI, tout flux interne transportant des données sensibles doit être chiffré avec TLS 1.2 minimum et un certificat d'une PKI maîtrisée. Ce guide détaille les quatre étapes : génération du CSR sur le serveur Linux avec OpenSSL, soumission via l'interface web certsrv ou la commande certreq depuis Windows, récupération et conversion PEM/DER, et déploiement sur Apache2, Nginx et optionnellement pour l'authentification client mutuelle.

À RETENIR

À retenir

CSR généré sur le serveur Linux : la clé privée ne quitte jamais le serveur cible — OpenSSL génère la paire de clés localement, seule la demande (CSR) est envoyée à la CA.

Deux méthodes de soumission : l'interface web certsrv (port 80/443 sur la CA Windows) ou un script PowerShell avec certreq sur une machine Windows intermédiaire pour les CA sans web enrollment.

Conversion PEM indispensable : AD CS délivre les certificats en DER par défaut ; OpenSSL convertit en PEM (Base64) pour Apache et Nginx en une commande.

Chaîne de confiance complète : configurer le bundle CA (Root + Sub) dans le virtual host évite les erreurs SSL_ERROR_UNKNOWN_CA chez les clients mobiles et Linux.

Renouvellement automatisable : un script Bash + certreq ou l'outil ACME-AD de la communauté permettent d'automatiser le renouvellement avant expiration.

Prérequis — Qu'est-ce qu'il faut préparer avant de commencer ?

Pour générer et déployer un certificat AD CS sur un serveur Linux, vous avez besoin de trois éléments : une PKI Active Directory fonctionnelle avec une CA émettrice (voir notre guide AD CS : déployer une PKI d'entreprise), un serveur Linux (Debian 12, Ubuntu 24.04 ou RHEL 9) avec OpenSSL installé, et l'accès au serveur web certsrv de la CA Windows ou à une machine Windows avec certreq. Le serveur Linux n'a pas besoin d'être membre du domaine Active Directory.

Élément	Requis	Alternatives
OpenSSL sur Linux	openssl >= 1.1.1	LibreSSL (macOS/FreeBSD)
CA Windows avec Web Enrollment	certsrv actif sur http://CA/certsrv	certreq depuis poste Windows
Template de certificat	WebServer ou custom	Tout template Server Auth
Groupe d'enrollment	Compte AD avec droit d'enrollment sur le template	Compte de service dédié
DNS résolu	FQDN du serveur Linux dans le DNS interne	SAN IP comme alternative

Étape 1 — Générer le CSR sur le serveur Linux avec OpenSSL

La première étape consiste à générer la paire de clés RSA et le Certificate Signing Request (CSR) directement sur le serveur Linux. La clé privée reste sur ce serveur et ne sera jamais transmise à qui que ce soit — c'est le principe fondamental de sécurité des PKI.

Étape 1 — Créer le répertoire de travail : organiser les fichiers PKI dans un répertoire sécurisé.

```
 # Créer un répertoire de travail sécurisé
mkdir -p /etc/ssl/private/monserveur
chmod 700 /etc/ssl/private/monserveur
cd /etc/ssl/private/monserveur
```

Étape 2 — Créer le fichier de configuration OpenSSL : définir les SAN (Subject Alternative Names) dans un fichier de configuration pour que le certificat couvre le FQDN et éventuellement l'adresse IP du serveur.

```
 # Créer le fichier de configuration OpenSSL
cat > monserveur.cnf << 'OPENSSLCONF'
[req]
default_bits        = 2048
prompt              = no
default_md          = sha256
distinguished_name  = dn
req_extensions      = req_ext

[dn]
C  = FR
ST = Ile-de-France
L  = Paris
O  = MonEntreprise
OU = Informatique
CN = monserveur.mondomaine.lan

[req_ext]
subjectAltName = @alt_names

[alt_names]
DNS.1 = monserveur.mondomaine.lan
DNS.2 = monserveur
IP.1  = 192.168.10.50
OPENSSLCONF
```

Étape 3 — Générer la clé privée et le CSR : une seule commande OpenSSL crée les deux fichiers.

```
# Générer la clé privée RSA 2048 bits (sans passphrase pour les services automatiques)
openssl req -new   -newkey rsa:2048   -nodes   -keyout monserveur.key   -out monserveur.csr   -co

# Vérifier le contenu du CSR
openssl req -text -noout -in monserveur.csr | head -40
# Vérifier les SAN demandés
openssl req -text -noout -in monserveur.csr | grep -A5 "Subject Alternative Name"

# Sécuriser la clé privée
chmod 600 monserveur.key
ls -la
# Résultat attendu :
# monserveur.csr  (à transmettre à la CA)
# monserveur.key  (NE PAS transmettre — clé privée)
```

Étape 2 — Soumettre le CSR à la CA Windows AD CS

Il existe deux méthodes pour soumettre le CSR à la CA Windows : via l'interface web certsrv (accessible depuis n'importe quel navigateur) ou via la commande certreq exécutée depuis un poste Windows membre du domaine. La méthode certsrv est la plus simple ; certreq est utile lorsque le web enrollment n'est pas activé sur la CA.

Méthode A — Via l'interface web certsrv

```
# Lire le contenu du CSR pour le copier-coller dans l'interface web
cat monserveur.csr
# Copier tout le contenu, y compris les lignes BEGIN/END CERTIFICATE REQUEST
```

Dans le navigateur, accéder à `http://<nom-CA>/certsrv` avec un compte AD ayant les droits d'enrollment sur le template WebServer. Cliquer sur *Request a certificate → Advanced certificate request → Submit a certificate request by using a base-64-encoded CMC or PKCS #10 file*. Coller le contenu du CSR, sélectionner le template *WebServer* (ou votre template personnalisé), et valider. Télécharger le certificat au format DER ou Base64.

Méthode B — Via certreq depuis un poste Windows

```
 # Sur un poste Windows membre du domaine
# Copier le fichier .csr depuis le serveur Linux (via SCP ou partage réseau)
# scp admin@monserveur:/etc/ssl/private/monserveur/monserveur.csr C:\Temp
# Créer le fichier de policy pour certreq
$policy = @"
[RequestAttributes]
CertificateTemplate=WebServer-Interne
"@
$policy | Out-File "C:\Temp\policy.inf" -Encoding ASCII

# Soumettre la demande à la CA (remplacer par le nom FQDN de votre CA)
certreq -submit -config "ca-emettrice.mondomaine.lan\MonEntreprise Issuing CA" `
  -attrib "CertificateTemplate:WebServer-Interne" `
  "C:\Temp\monserveur.csr" "C:\Temp\monserveur.cer"

# Si l'approbation du CA manager est requise, noter le Request ID et approuver dans certsrv.msc
# Puis récupérer le certificat approuvé :
certreq -retrieve -config "ca-emettrice.mondomaine.lan\MonEntreprise Issuing CA" <RequestID> "C:\
```

Étape 3 — Récupérer et convertir le certificat en PEM

AD CS délivre les certificats en format DER (binaire) ou Base64/PEM selon l'option choisie au téléchargement. Apache et Nginx attendent du PEM. Si le fichier téléchargé est en DER (extension .cer ou .der), la conversion avec OpenSSL est en une ligne.

```
 # Transférer le certificat depuis Windows vers le serveur Linux
# scp admin@windows-admin:C:/Temp/monserveur.cer /etc/ssl/private/monserveur/

# Vérifier le format du certificat téléchargé
file monserveur.cer
# Si "data" -> format DER, si "PEM certificate" -> déjà en Base64

# Convertir DER -> PEM si nécessaire
openssl x509 -inform DER -in monserveur.cer -out monserveur.crt
# Si déjà en Base64/PEM, simplement renommer :
# cp monserveur.cer monserveur.crt

# Vérifier le certificat et confirmer les SAN
openssl x509 -text -noout -in monserveur.crt | grep -A5 "Subject Alternative Name"
openssl x509 -text -noout -in monserveur.crt | grep "Not After"
# Résultat attendu : date dans 1-2 ans

# Vérifier que la clé privée correspond au certificat
openssl x509 -noout -modulus -in monserveur.crt | md5sum
openssl rsa -noout -modulus -in monserveur.key | md5sum
# Les deux hash doivent être identiques

# Récupérer la chaîne de confiance (Root CA + Sub CA)
# Télécharger depuis : http://pki.mondomaine.lan/certs/
wget http://pki.mondomaine.lan/certs/SubCA.crt -O subca.cer
wget http://pki.mondomaine.lan/certs/RootCA.crt -O rootca.cer

# Convertir en PEM si nécessaire
openssl x509 -inform DER -in subca.cer -out subca.pem
openssl x509 -inform DER -in rootca.cer -out rootca.pem

# Créer le bundle de chaîne de confiance (Sub CA + Root CA)
cat subca.pem rootca.pem > ca-bundle.pem
```

Étape 4 — Intégration dans Apache2 et Nginx

Une fois le certificat en PEM et le bundle de chaîne de confiance disponibles, la configuration des serveurs web est standard. Les erreurs les plus fréquentes sont liées à l'ordre des certificats dans le bundle (Sub CA doit précéder Root CA) ou à l'absence de la directive SSLCACertificateFile pour le bundle.

Configuration Apache2 (Debian/Ubuntu)

```
 # Installer les modules SSL si pas encore fait
a2enmod ssl
a2enmod headers

# Copier les fichiers dans les répertoires standard Apache
cp monserveur.crt /etc/ssl/certs/
cp monserveur.key /etc/ssl/private/
cp ca-bundle.pem /etc/ssl/certs/

# Créer le virtual host SSL
cat > /etc/apache2/sites-available/monserveur-ssl.conf << 'APACHECONF'

    ServerName monserveur.mondomaine.lan
    DocumentRoot /var/www/html

    SSLEngine on
    SSLCertificateFile      /etc/ssl/certs/monserveur.crt
    SSLCertificateKeyFile   /etc/ssl/private/monserveur.key
    SSLCACertificateFile    /etc/ssl/certs/ca-bundle.pem

    # TLS 1.2+ uniquement (recommandation ANSSI)
    SSLProtocol             all -SSLv3 -TLSv1 -TLSv1.1
    SSLCipherSuite          ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA
    SSLHonorCipherOrder     off
    SSLSessionTickets       off

    # Headers de sécurité
    Header always set Strict-Transport-Security "max-age=63072000"
    Header always set X-Frame-Options DENY
    Header always set X-Content-Type-Options nosniff

APACHECONF

# Activer le site et tester la configuration
a2ensite monserveur-ssl.conf
apache2ctl configtest
# Résultat attendu : Syntax OK
systemctl reload apache2
```

Configuration Nginx

```
 # Configuration Nginx avec le bundle CA inclus dans le certificat (pratique recommandée)
# Créer le fichier fullchain (certificat serveur + chaîne)
cat monserveur.crt ca-bundle.pem > monserveur-fullchain.pem
cp monserveur-fullchain.pem /etc/nginx/ssl/
cp monserveur.key /etc/nginx/ssl/

# Bloc server Nginx
cat > /etc/nginx/sites-available/monserveur << 'NGINXCONF'
server {
    listen 443 ssl http2;
    server_name monserveur.mondomaine.lan;

    ssl_certificate     /etc/nginx/ssl/monserveur-fullchain.pem;
    ssl_certificate_key /etc/nginx/ssl/monserveur.key;

    # Protocoles et ciphers conformes ANSSI/Mozilla Modern
    ssl_protocols       TLSv1.2 TLSv1.3;
    ssl_ciphers         ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:TLS_AES_256_GCM
    ssl_prefer_server_ciphers off;
    ssl_session_cache   shared:SSL:10m;
    ssl_session_timeout 1d;
    ssl_session_tickets off;

    # OCSP Stapling (si CA expose OCSP)
    ssl_stapling on;
    ssl_stapling_verify on;
    ssl_trusted_certificate /etc/nginx/ssl/monserveur-fullchain.pem;

    location / {
        root /var/www/html;
        index index.html;
    }
}
NGINXCONF

ln -s /etc/nginx/sites-available/monserveur /etc/nginx/sites-enabled/
nginx -t
# Résultat attendu : configuration file nginx.conf test is successful
systemctl reload nginx
```

Validation et vérification de la chaîne de confiance

```
 # Tester la connexion TLS depuis le serveur Linux
openssl s_client -connect monserveur.mondomaine.lan:443 -CAfile ca-bundle.pem
# Résultat attendu : "Verify return code: 0 (ok)"

# Vérifier depuis un poste du domaine Windows (PowerShell)
# Invoke-WebRequest https://monserveur.mondomaine.lan -UseBasicParsing

# Tester avec curl en précisant le bundle CA
curl --cacert /etc/ssl/certs/ca-bundle.pem https://monserveur.mondomaine.lan -v 2>&1 | grep "SSL
# Résultat attendu : "SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384"

# Ajouter le Root CA dans le magasin système Linux pour éviter --cacert
cp rootca.pem /usr/local/share/ca-certificates/MonEntreprise-RootCA.crt
update-ca-certificates
# Résultat : "1 added, 0 removed; done."

# Vérification sans flag --cacert après mise à jour du magasin
curl https://monserveur.mondomaine.lan -v 2>&1 | grep -E "certificate|SSL"
```

> Lors d'une mission d'audit sécurité (PME industrielle, 150 serveurs Linux), 80 % des services web internes utilisaient des certificats auto-signés générés manuellement avec des durées de validité de 10 ans. L'absence de révocation centralisée rendait toute réponse à incident impossible sur ce périmètre. La migration vers une PKI AD CS centralisée avec scripts de renouvellement automatique (cron + certreq) a réduit la durée moyenne des certificats à 1 an et permis la révocation immédiate lors d'un incident serveur compromis. Le délai de mise en conformité des 150 serveurs a été de 3 semaines avec un script Bash automatisant CSR, soumission et rechargement Apache/Nginx.
>
> *— Retour de mission PKI Linux, mars 2026*

Automatisation du renouvellement avant expiration

Un certificat expiré sur un service web interne provoque des interruptions de service et des alertes de sécurité. L'automatisation du renouvellement via un script cron est fortement recommandée, surtout avec des durées de validité courtes (1 an). Le script suivant vérifie la date d'expiration 30 jours à l'avance et génère un nouveau CSR si nécessaire.

```
 #!/bin/bash
# Script de renouvellement automatique de certificat AD CS
# A placer dans /usr/local/bin/renew-cert.sh et planifier via cron

CERT_FILE="/etc/ssl/certs/monserveur.crt"
KEY_FILE="/etc/ssl/private/monserveur.key"
CSR_FILE="/tmp/monserveur-renew.csr"
CNF_FILE="/etc/ssl/private/monserveur/monserveur.cnf"
CA_URL="http://ca-emettrice.mondomaine.lan/certsrv/certfnsh.asp"
DAYS_BEFORE_EXPIRY=30

# Vérifier la date d'expiration
EXPIRY=$(openssl x509 -enddate -noout -in "$CERT_FILE" | cut -d= -f2)
EXPIRY_EPOCH=$(date -d "$EXPIRY" +%s)
NOW_EPOCH=$(date +%s)
DAYS_LEFT=$(( (EXPIRY_EPOCH - NOW_EPOCH) / 86400 ))

echo "Certificat expire dans $DAYS_LEFT jours."

if [ "$DAYS_LEFT" -lt "$DAYS_BEFORE_EXPIRY" ]; then
    echo "Renouvellement necessaire. Generation du CSR..."
    # Générer nouveau CSR avec la clé existante (ou nouvelle clé)
    openssl req -new -key "$KEY_FILE" -out "$CSR_FILE" -config "$CNF_FILE"

    # Soumettre via curl à l'interface certsrv (authentification NTLM)
    # Nécessite : apt install curl avec support NTLM, ou un poste Windows intermédiaire
    echo "CSR genere dans $CSR_FILE — soumettre manuellement ou via script Windows."
else
    echo "Certificat encore valide. Aucune action necessaire."
fi
```

Pour une automatisation complète sans intervention manuelle, la soumission peut être effectuée via un poste Windows avec un script PowerShell appelé par SSH depuis le serveur Linux. Cette

approche hybride est documentée dans la documentation Microsoft sur l'enrollment pour clients non-Windows. Pour aller plus loin sur la sécurisation de l'infrastructure AD CS, consultez notre guide Active Directory Certificate Services : attaque et défense et notre analyse des vulnérabilités ESC1 à ESC16.

Questions fréquentes

Peut-on inclure des SAN IP dans un certificat AD CS ?

Oui, à condition que le template autorise les SAN fournis par l'enrollee (CT_FLAG_ENROLLEE_SUPPLIES_SUBJECT). La syntaxe dans le fichier de configuration OpenSSL est `IP.1 = 192.168.10.50` dans la section `[alt_names]`. Les navigateurs modernes (Chrome 58+) exigent les SAN et ignorent le CN pour la validation du nom d'hôte, donc les SAN sont obligatoires. Pour les serveurs accessibles uniquement par IP, ajouter l'IP comme SAN est la seule façon d'éviter l'avertissement de sécurité.

Comment vérifier que le certificat émis contient bien les SAN demandés ?

Après réception du certificat, la commande `openssl x509 -text -noout -in monserveur.crt | grep -A5 "Subject Alternative Name"` liste tous les SAN. Si les SAN sont absents, cela signifie que le template AD CS est configuré pour ignorer les SAN du CSR et les construire depuis l'AD — il faut alors utiliser un template avec ENROLLEE_SUPPLIES_SUBJECT ou configurer les SAN directement dans la définition du template.

Le serveur Linux doit-il rejoindre le domaine Active Directory ?

Non. Le serveur Linux n'a pas besoin d'être membre du domaine pour obtenir un certificat TLS depuis AD CS. Un compte AD avec les droits d'enrollment sur le template suffit pour soumettre la demande via certsrv ou certreq depuis un poste Windows. En revanche, pour l'authentification client par certificat (smart card login Linux), l'intégration via SSSD et Kerberos est nécessaire.

Quelle longueur de clé recommander en 2026 ?

RSA 2048 bits reste acceptable pour des certificats d'un an, mais RSA 4096 ou ECDSA P-256 sont recommandés pour les nouveaux déploiements selon le guide ANSSI sur la cryptographie. ECDSA offre une sécurité équivalente à RSA 3072 avec des clés beaucoup plus courtes, ce qui améliore les performances TLS (handshake plus rapide). Configurer `-newkey ec -pkeyopt ec_paramgen_curve:P-256` à la place de `-newkey rsa:2048` pour générer une clé ECDSA.

Comment configurer l'authentification mutuelle TLS (mTLS) avec des clients Linux ?

Pour le mTLS, le serveur Linux doit présenter son certificat serveur ET vérifier le certificat client. Dans Apache, ajouter `SSLVerifyClient require` et `SSLVerifyDepth 2` dans le VirtualHost. Les clients doivent disposer d'un certificat émis par la même PKI AD CS, importé dans leur magasin personnel. Cette configuration est utilisée pour les API REST internes sécurisées, les connexions entre microservices, et les accès VPN sans mot de passe.

Cas d'usage avancés — certificats pour services non-web

Les certificats AD CS ne servent pas uniquement aux services web Apache et Nginx. De nombreux services Linux tirent profit d'une PKI d'entreprise : les serveurs LDAP (OpenLDAP en LDAPS), les connexions PostgreSQL/MySQL chiffrées, les agents Zabbix avec TLS, les serveurs OpenVPN ou WireGuard avec authentification par certificat, et les connexions SSH par certificat (plus sécurisé que les clés RSA car révocable). Voici les configurations spécifiques pour quelques services courants.

PostgreSQL — chiffrement des connexions client-serveur

```
 # Copier les fichiers dans le répertoire de données PostgreSQL
cp monserveur.crt /etc/postgresql/16/main/server.crt
cp monserveur.key /etc/postgresql/16/main/server.key
cp ca-bundle.pem /etc/postgresql/16/main/root.crt
chown postgres:postgres /etc/postgresql/16/main/server.{crt,key} /etc/postgresql/16/main/root.crt
chmod 600 /etc/postgresql/16/main/server.key

# Configurer postgresql.conf
# ssl = on
# ssl_cert_file = 'server.crt'
# ssl_key_file = 'server.key'
# ssl_ca_file = 'root.crt'  (pour le mTLS client)

# Redémarrer PostgreSQL
systemctl restart postgresql

# Vérifier que SSL est actif depuis psql
# psql -U postgres -c "SHOW ssl;" -> on
```

OpenLDAP — activer LDAPS avec certificat AD CS

```
 # Configurer TLS dans /etc/ldap/ldap.conf pour les clients
echo "TLS_CACERT /etc/ssl/certs/ca-bundle.pem" >> /etc/ldap/ldap.conf

# Configuration slapd (OpenLDAP server)
# Créer un fichier LDIF pour ajouter les paramètres TLS
cat > /tmp/tls-config.ldif << 'EOF'
dn: cn=config
changetype: modify
add: olcTLSCACertificateFile
olcTLSCACertificateFile: /etc/ssl/certs/ca-bundle.pem
-
add: olcTLSCertificateFile
olcTLSCertificateFile: /etc/ssl/certs/monserveur.crt
-
add: olcTLSCertificateKeyFile
olcTLSCertificateKeyFile: /etc/ssl/private/monserveur.key
-
add: olcTLSMinProtocol
olcTLSMinProtocol: 3.3
EOF

ldapmodify -Y EXTERNAL -H ldapi:/// -f /tmp/tls-config.ldif

# Tester la connexion LDAPS
ldapsearch -x -H ldaps://monserveur.mondomaine.lan -b "dc=mondomaine,dc=lan" -D "cn=admin,dc=mond
```

Zabbix Agent 2 — communications TLS avec le serveur Zabbix

```
 # Configurer Zabbix Agent 2 pour utiliser TLS avec certificat PKI
# Éditer /etc/zabbix/zabbix_agent2.conf

# TLSConnect=cert           (connexion sortante en certificat)
# TLSAccept=cert            (accepter uniquement les connexions par certificat)
# TLSCAFile=/etc/ssl/certs/ca-bundle.pem
# TLSCertFile=/etc/ssl/certs/monserveur.crt
# TLSKeyFile=/etc/ssl/private/monserveur.key

# Sur le serveur Zabbix, configurer les paramètres TLS de l'hôte :
# Administration -> Hosts -> [hôte] -> Encryption
# Connections to host : Certificate
# Connections from host : Certificate
# Issuer : CN=MonEntreprise Issuing CA,O=MonEntreprise,C=FR
# Subject : CN=monserveur.mondomaine.lan,OU=Informatique,O=MonEntreprise,C=FR

systemctl restart zabbix-agent2
```

Gestion des certificats à grande échelle — inventaire et alertes d'expiration

Dès que le nombre de certificats déployés sur des serveurs Linux dépasse une dizaine, la gestion manuelle devient risquée. Un inventaire centralisé avec alertes d'expiration est indispensable pour éviter les interruptions de service dues à des certificats expirés. Voici un script de supervision simple à intégrer dans votre monitoring (Zabbix, Nagios, ou un simple cron sur le serveur de gestion).

```
 #!/bin/bash
# Script d'inventaire des certificats Linux - a executer sur chaque serveur ou via Ansible
# Usage : ./check-cert-expiry.sh [jours_alerte]

ALERT_DAYS=${1:-30}
CERT_DIRS="/etc/ssl/certs /etc/nginx/ssl /etc/apache2/ssl /etc/postgresql/*/main"
ALERT_FILE="/var/log/cert-expiry-alerts.log"

echo "=== Verification certificats - $(date) ===" > "$ALERT_FILE"

for DIR in $CERT_DIRS; do
  for CERT in $(find $DIR -name "*.crt" -o -name "*.pem" 2>/dev/null | grep -v "ca-bundle\|root\|
    # Ignorer les fichiers qui ne sont pas des certificats valides
    openssl x509 -in "$CERT" -noout 2>/dev/null || continue

    EXPIRY=$(openssl x509 -enddate -noout -in "$CERT" | cut -d= -f2)
    EXPIRY_EPOCH=$(date -d "$EXPIRY" +%s 2>/dev/null) || continue
    NOW_EPOCH=$(date +%s)
    DAYS_LEFT=$(( (EXPIRY_EPOCH - NOW_EPOCH) / 86400 ))
    CN=$(openssl x509 -noout -subject -in "$CERT" | sed 's/.*CN = //' | cut -d',' -f1)

    if [ "$DAYS_LEFT" -lt "$ALERT_DAYS" ]; then
      echo "ALERTE: $CERT (CN=$CN) expire dans $DAYS_LEFT jours ($EXPIRY)" | tee -a "$ALERT_FILE"
    else
      echo "OK: $CERT (CN=$CN) - $DAYS_LEFT jours restants"
    fi
  done
done

# Envoyer par mail si des alertes ont été générées
if grep -q "ALERTE" "$ALERT_FILE"; then
  mail -s "[PKI] Certificats expirant dans moins de $ALERT_DAYS jours - $(hostname)" admin@mondom
fi
```

Pour une gestion centralisée à grande échelle, des outils comme **Smallstep CA**, **HashiCorp Vault PKI** ou **Venafi** s'interfacent avec AD CS et permettent une rotation automatique des certificats avec des durées de vie très courtes (24 à 72 heures), ce qui élimine pratiquement le risque de certificat compromis utilisé durablement. Pour les environnements Active Directory, la combinaison AD CS + autoenrollment GPO reste la solution la plus native et la moins coûteuse, documentée dans notre guide Sécurisation Active Directory Windows Server 2025.

Troubleshooting — erreurs courantes et solutions

Les erreurs rencontrées lors du déploiement de certificats AD CS sur Linux sont généralement liées à trois causes : format incorrect du certificat, chaîne de confiance incomplète, ou CRL inaccessible. Le tableau suivant liste les messages d'erreur les plus fréquents et leurs solutions.

Message d'erreur	Cause probable	Solution
SSL_ERROR_UNKNOWN_CA	Certificat Root CA absent du magasin client	Importer le Root CA avec update-ca-certificates
certificate verify failed: unable to get local issuer certificate	Bundle de chaîne incomplet ou manquant	Créer ca-bundle.pem avec Sub CA + Root CA dans cet ordre
certificate verify failed: certificate has expired	Certificat expiré ou horloge désynchronisée	Renouveler le certificat ou synchroniser NTP
REVOCATION_FUNCTION_UNABLE_TO_CHECK_REVOCATION	CDP inaccessible depuis le client	Vérifier que http://pki.mondomaine.lan/crl/ est joignable
error 20 at 0 depth lookup: unable to get local issuer certificate	Ordre incorrect dans le bundle CA (Root avant Sub)	Recréer : cat subca.pem rootca.pem > ca-bundle.pem
certreq: The request subject does not match the CA template	Template refuse le CN ou les SAN soumis	Vérifier ENROLLEE_SUPPLIES_SUBJECT sur le template

La commande de diagnostic la plus complète pour tracer l'origine d'une erreur TLS est `openssl s_client -connect host:port -showcerts -CAfile ca-bundle.pem -verify_return_error 2>&1 | grep -E "verify|depth|error|CN"`. Elle affiche chaque niveau de la chaîne et l'erreur précise de vérification à chaque étape.

