TALL GRASS COMPUTE

BUILD • BREAK • LEARN • REPEAT

Bringing ACME to Microsoft AD CS

Building an Internal Certificate Automation Service for Linux and Other ACME-Capable Systems

A practical guide to bridging standard ACME certificate requests into Microsoft AD CS issuance using NGINX, Kerberos, and acme2certifier — with publication and client validation for internal PKI services.



Tall Grass Compute Lab • October 2026

Bringing ACME to Microsoft AD CS

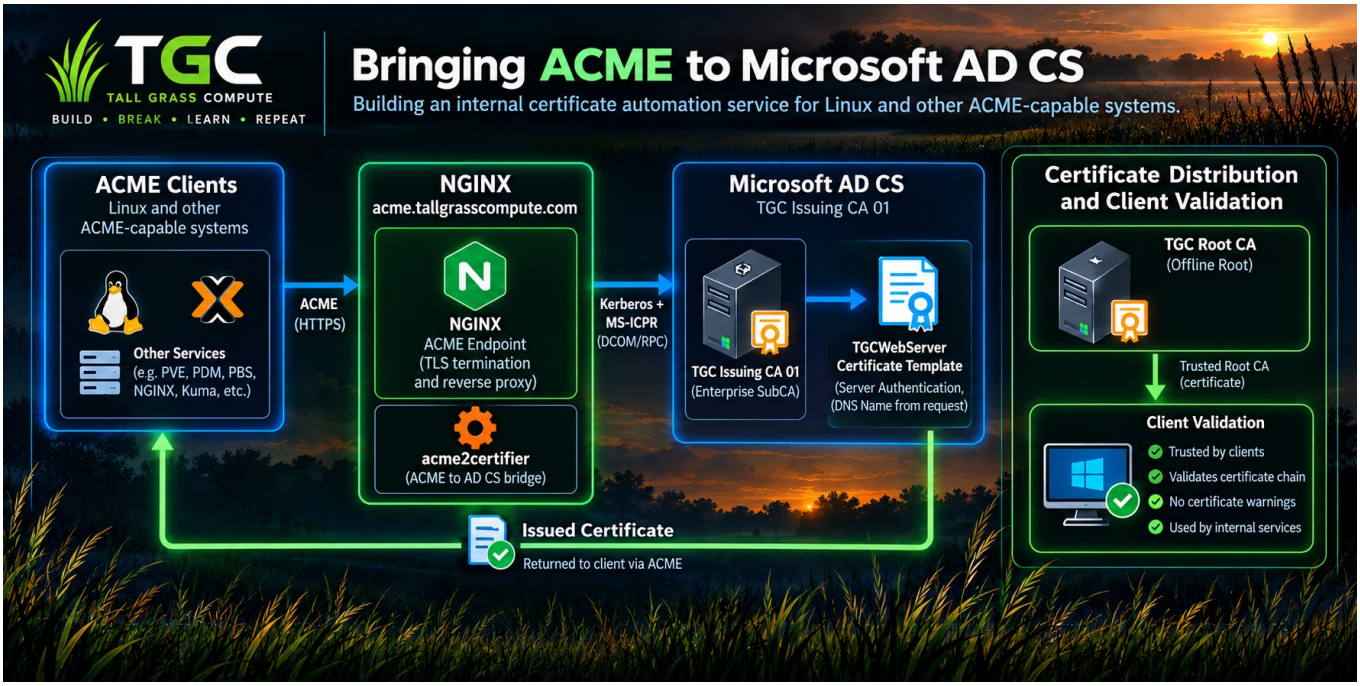
Building an internal certificate automation service for Linux and other ACME-capable systems

Tall Grass Compute • October 2026

The project started with a Linux problem: how do I give systems such as Proxmox, PBS, and NGINX the same kind of automated certificate lifecycle I expect from public ACME services, while keeping Microsoft AD CS as the internal certificate authority?

The answer turned out to be more general than the original problem. By placing an ACME-compatible service in front of AD CS, the resulting architecture can serve any compatible ACME client—not just Linux.

The finished path keeps those responsibilities deliberately separate: ACME clients request and renew certificates through NGINX and acme2certifier, while Microsoft AD CS remains the authority that applies certificate policy and signs the result. The rest of this article shows how that bridge was built, how Proxmox consumed it natively, and why validating the revocation path became just as important as validating issuance. The architecture overview below shows the complete path at a glance.



1. Starting with Microsoft AD CS

Tall Grass Compute already had an internal Microsoft PKI in place before this project started. The design follows a traditional two-tier model: an offline standalone root CA and an online enterprise issuing CA.

TGC-AD-CA01 hosts the TGC Root CA and remains offline during normal operation. Day-to-day issuance belongs to TGC-AD-PKI01, which runs the enterprise subordinate CA TGC Issuing CA 01. The ACME service never needs to interact directly with the offline root.

The certificate template is intentionally narrow. TGC Web Server carries the Server Authentication EKU, allows the requester to supply the subject and SAN, uses a two-year validity period, and is the only template exposed through the final ACME integration.

The environment also separates infrastructure identity from TLS service identity. For example, the Proxmox host is TGC-PVE-01.tallgrasscompute.local, while its web certificate is issued to pve-01.tallgrasscompute.com. Those names are intentionally different: one identifies the underlying infrastructure host; the other identifies the TLS service clients connect to.

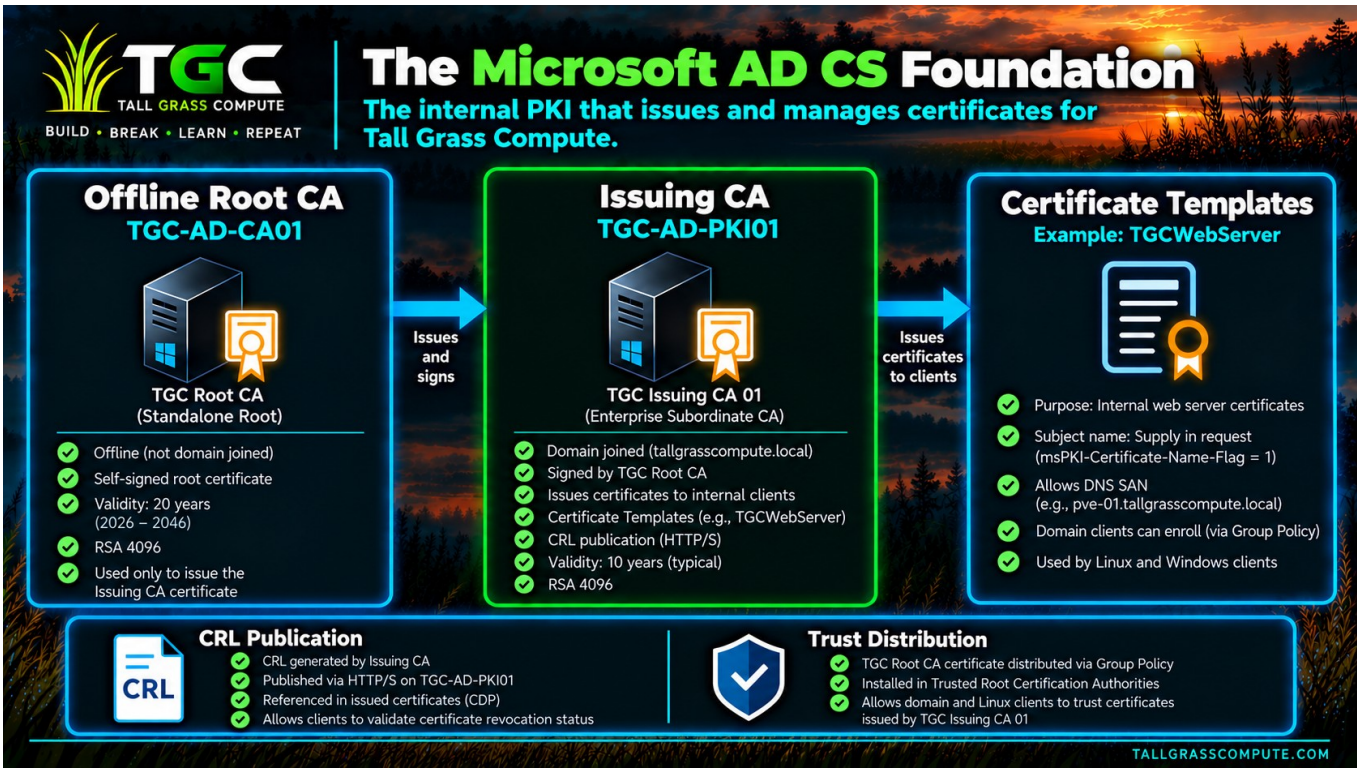


Figure 1 — The Microsoft AD CS foundation: an offline root anchors trust, the online issuing CA handles routine issuance, and TGC Web Server provides the server-certificate policy used by the ACME gateway.

2. Why ACME Was the Missing Piece

Tall Grass Compute was already using ACME to automate public certificates, so the workflow was familiar and proven. The internal PKI was different. Microsoft AD CS provided the trust model I

wanted, but Proxmox, PBS, NGINX, and other Linux-based systems are much more naturally aligned with ACME than with Microsoft certificate-enrollment protocols.

If ACME already worked well for public certificate automation, why not use the same model internally while keeping Microsoft AD CS as the certificate authority?

The goal was not to replace AD CS. It was to put an ACME-compatible interface in front of it.

That way, certificate consumers can continue using the protocol they already understand, while AD CS remains responsible for issuance, policy, trust, and the certificate lifecycle behind the scenes.

3. Building the Bridge Between ACME and AD CS

acme2certifier became the translation layer. From a Proxmox node, the environment looks like a normal ACME service. Behind that endpoint, acme2certifier authenticates to Active Directory with a Kerberos keytab and submits requests to TGC Issuing CA 01 through MS-ICPR/RPC.

acme2certifier uses a dedicated Active Directory service account and Kerberos authentication to submit certificate requests to the issuing CA.

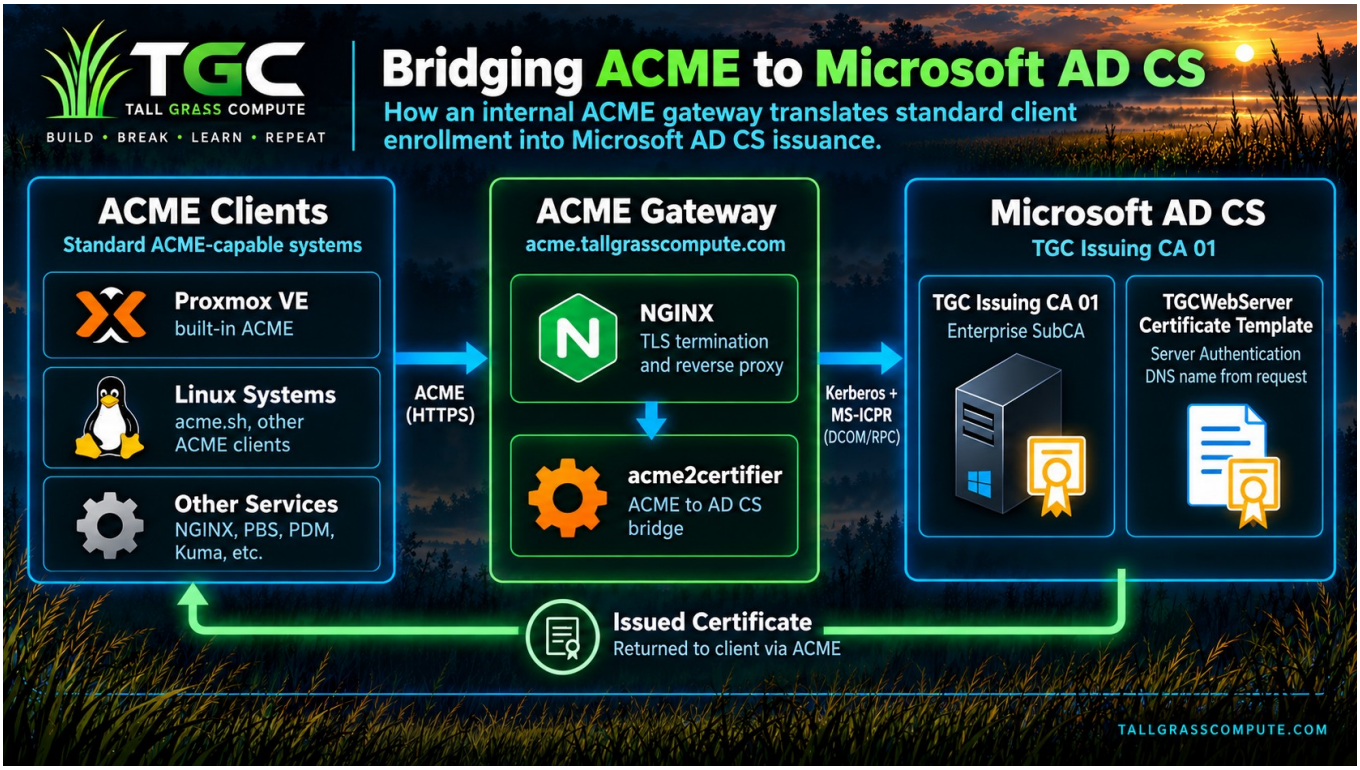


Figure 2 — The protocol bridge: ACME on the client side, NGINX at the service boundary, acme2certifier translating the request, and Kerberos/MS-ICPR carrying enrollment into Microsoft AD CS.

4. Configuring acme2certifier for AD CS

The core Microsoft integration lives in the CAhandler section of acme2certifier. The important pieces are the issuing CA, the TGCWebServer template, Kerberos authentication, and the explicit boundaries around what the gateway is allowed to request.

```
[CAhandler]
handler_module: acme2certifier.cahandlers.msicpr_ca_handler
host: TGC-AD-PKI01.tallgrasscompute.local
target_domain: TALLGRASSCOMPUTE.LOCAL
domain_controller: TGC-AD-DC01.tallgrasscompute.local
ca_name: TGC Issuing CA 01
template: TGCWebServer
allowed_templates: ["TGCWebServer"]
allowed_domainlist: ["tallgrasscompute.com", "*.tallgrasscompute.com"]

use_kerberos: True
krb5_auth_backend: python
krb5_principal: svc-ces@TALLGRASSCOMPUTE.LOCAL
krb5_keytab: /run/secrets/svc-ces.keytab
timeout: 10
```

Those restrictions matter. The bridge is not being handed unrestricted enrollment capability. The template is fixed, and requested names are constrained to the service namespace used by the lab.

NGINX sits in front of the container at https://acme.tallgrasscompute.com. It terminates TLS, applies internal network restrictions, and proxies the ACME service to localhost:22280. HTTP-01 challenge traffic is handled separately under /.well-known/acme-challenge/.

5. Evidence on the Issuing CA

Request ID	Issued Common Name	Requester Name	Binary Certificate	Certificate Template	Serial Number	Certificate Effective Date	Certificate Expiration Date
		TGC\TGCAdmin					
4	TGC-AD-PKI01.tallgrass...	TGC\TGCAdmin	-----BEGIN CERTI...	TGC Web Server (1.3....	7000000004331...	9/30/2026 3:35 PM	9/29/2028 3:35 PM
5	tgc-nginx-01 PKI Client	TGC\TGCAdmin	-----BEGIN CERTI...	TGC Linux PKI Client ...	70000000055cf...	9/30/2026 7:45 PM	9/30/2027 7:45 PM
6	acme.tallgrasscompute....	TGC\svc-ces	-----BEGIN CERTI...	TGC Web Server (1.3....	700000000639c...	10/1/2026 1:35 AM	9/30/2028 1:35 AM
7	pbs.tallgrasscompute.c...	TGC\svc-ces	-----BEGIN CERTI...	TGC Web Server (1.3....	70000000078c4...	10/1/2026 2:31 AM	9/30/2028 2:31 AM
8	proxcenter.tallgrasscom...	TGC\svc-ces	-----BEGIN CERTI...	TGC Web Server (1.3....	7000000008db...	10/1/2026 2:56 AM	9/30/2028 2:56 AM
9	pdm.tallgrasscompute....	TGC\svc-ces	-----BEGIN CERTI...	TGC Web Server (1.3....	70000000096 40...	10/1/2026 3:29 AM	9/30/2028 3:29 AM
10	pve-01.tallgrasscomput...	TGC\svc-ces	-----BEGIN CERTI...	TGC Web Server (1.3....	700000000a324...	10/1/2026 3:48 PM	9/30/2028 3:48 PM
11	pve-02.tallgrasscomput...	TGC\svc-ces	-----BEGIN CERTI...	TGC Web Server (1.3....	700000000be2...	10/1/2026 3:57 PM	9/30/2028 3:57 PM

Figure 3 — Certificates issued by TGC Issuing CA 01. The ACME-backed requests are submitted by TGC\svc-ces and use the TGC Web Server certificate template.

The issuing-CA console provides useful proof that the translation really ends in Microsoft AD CS. Requests for acme.tallgrasscompute.com, PBS, ProxCenter, PDM, PVE-01, and PVE-02 were issued through the TGC Web Server template under the service identity used by the gateway.

6. Using Proxmox as the First ACME Client

Proxmox was a good first client because it already includes native ACME support. There is no custom certificate-deployment script on the node. Proxmox points its ACME account at the internal directory, requests the certificate, completes validation, installs the result, and restarts pveproxy itself.

```
acme: account=tgc-acme
acmedomain0: pve-01.tallgrasscompute.com

Directory:
https://acme.tallgrasscompute.com/directory
```

The important part is what did not change: Proxmox still speaks ACME. The fact that Microsoft AD CS ultimately signs the certificate is an implementation detail behind the ACME gateway.

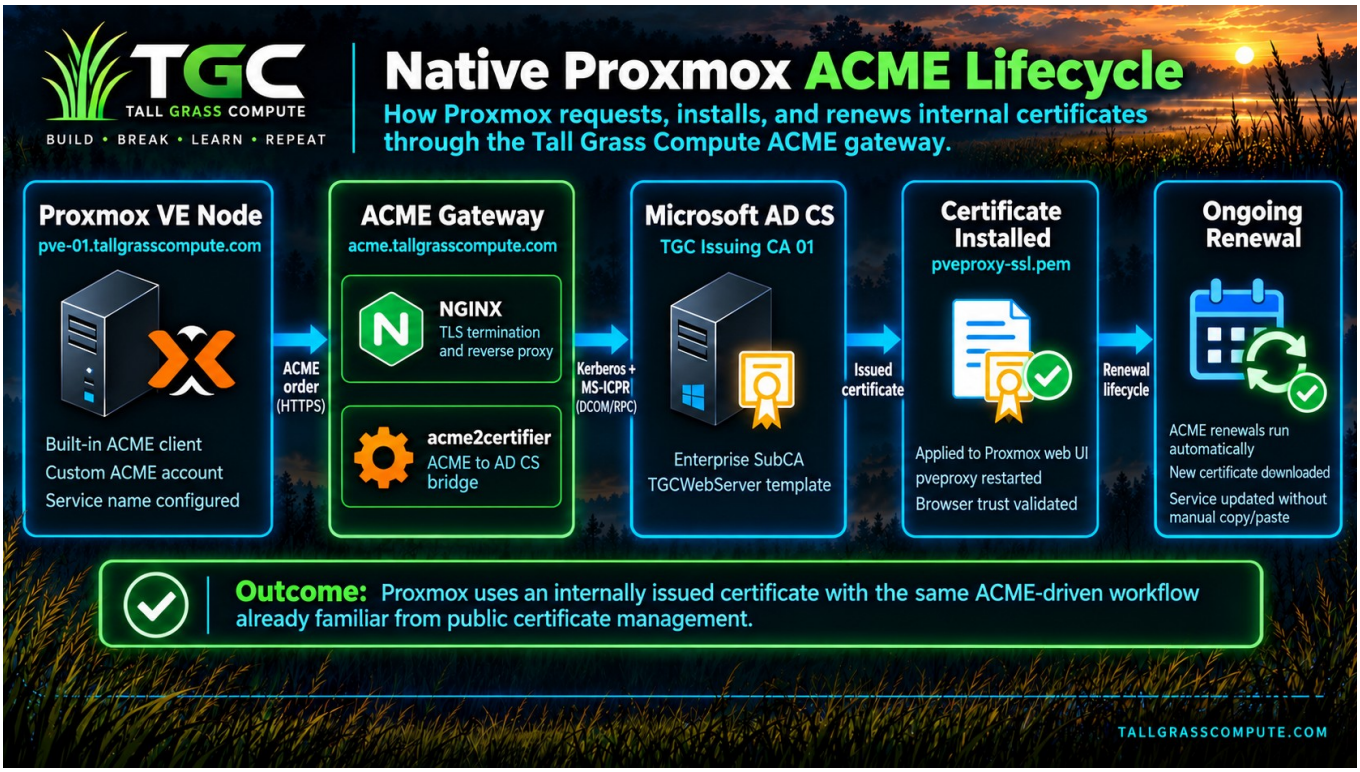


Figure 4 — Native Proxmox ACME lifecycle against the internal AD CS-backed ACME service.

The First Full End-to-End Order

To prove the complete path, I forced a new certificate order on TGC-PVE-01.

The output shows the authorization moving from pending to valid, CSR creation, order finalization, certificate download, installation into pveproxy, and the pveproxy restart.

```
jveith@TGC-PVE-01:~$ sudo pvenode acme cert order --force 1
Loading ACME account details
Placing ACME order
Order URL: https://acme.tallgrasscompute.com/acme/order/07jJhSep5AEj

Getting authorization details from 'https://acme.tallgrasscompute.com/acme/authz/DMb9WnA7fMf6'
The validation for pve-01.tallgrasscompute.com is pending!
Setting up webserver
Triggering validation
Sleeping for 5 seconds
Status is 'valid', domain 'pve-01.tallgrasscompute.com' OK!

All domains validated!

Creating CSR
Checking order status
Order is ready, finalizing order
valid!

Downloading certificate
Setting pveproxy certificate and key
Restarting pveproxy
Task OK
jveith@TGC-PVE-01:~$
```

Figure 5 — A forced ACME order on TGC-PVE-01. Proxmox validates the service name, creates the CSR, downloads the AD CS-issued certificate, installs it, restarts pveproxy, and finishes with Task OK.

This is the point where the architecture stopped being theoretical. A native Proxmox ACME client had completed a normal ACME transaction, while Microsoft AD CS handled the actual certificate issuance behind the scenes.

7. Trust, But Verify

A successful ACME task tells me the client completed its workflow. It does not, by itself, prove that the running service is presenting the certificate I expect or that the complete trust and revocation path works.

That distinction matters because certificate automation has several checkpoints. A successful order proves that the ACME transaction completed; it does not prove that the new certificate was installed, that the live listener is presenting it, or that clients can build and validate the full trust and revocation path. Each of those outcomes needs to be verified independently.

For that reason, I treated runtime verification as a separate phase of the automation rather than an implicit side effect of issuance. The checks needed to answer three different questions: did AD CS issue the expected certificate, did Proxmox actually install and present it, and could a client validate the complete chain and revocation path from the network?

After the forced order, the installed pveproxy certificate and the certificate presented remotely on TCP 8006 had the same SHA-256 fingerprint. That closed the operational loop from order to runtime presentation.

The certificate is not "done" because the order succeeded. It is done when the service is presenting the expected certificate and clients can validate it.

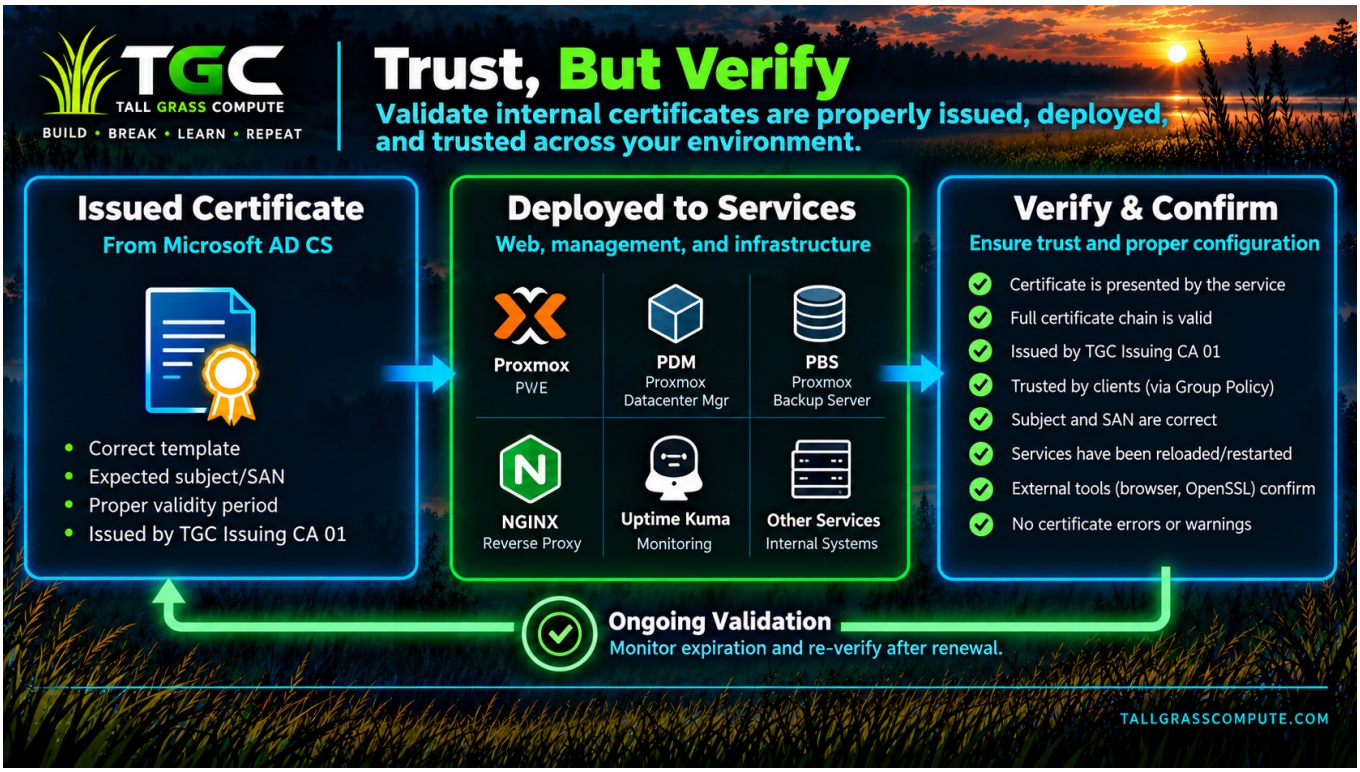


Figure 6 — Validation is layered: successful ACME order, installed certificate, live listener, trusted chain, and revocation-aware verification.

End-to-End Validation with CRL Checking

I then validated the live Proxmox endpoint with the TGC root explicitly trusted and CRL checking enabled. This moved the test beyond "the browser likes it" and into the actual PKI mechanics.

```
jveith@tgc-nginx-01:~$ printf '\n\n' && \
openssl s_client \
  -connect pve-01.tallgrasscompute.com:8006 \
  -servername pve-01.tallgrasscompute.com \
  -CAfile /tmp/tgc-root-ca.pem \
  -crl_download \
  -crl_check \
  </dev/null 2>&1 | \
  grep -E "subject=|issuer=|Verification:|Protocol:|Server public key|Verify return code"


subject=CN=pve-01.tallgrasscompute.com
issuer=DC=local, DC=tallgrasscompute, CN=TGC Issuing CA 01
Verification: OK
Protocol: TLSv1.3
Server public key is 4096 bit
Verify return code: 0 (ok)
jveith@tgc-nginx-01:~$
```

Figure 7 — OpenSSL validates the live Proxmox certificate against the TGC root CA with CRL downloading and CRL checking enabled. The result is Verification: OK and Verify return code: 0 (ok).

The certificate was being served with TLS 1.3, used a 4096-bit RSA key, chained to TGC Issuing CA 01, and passed revocation-aware validation.

That test exposed the most important operational gap in the project.

8. The Part I Almost Missed: CRL Publication

The certificate was valid. The chain was correct. ACME issuance worked. But when I inspected the CRL Distribution Point embedded in the certificate, the CRL being served from pki.tallgrasscompute.com was stale.

The URL was correct and the file existed. The problem was operational: the published issuing-CA CRL had passed its NextUpdate time. That meant certificate issuance could succeed while revocation-aware validation eventually failed.

That discovery changed the finish line again. A complete certificate lifecycle could not stop at successful issuance; the revocation path also had to stay current, reachable, and independently verifiable. I treated CRL publication as a second automation workflow rather than a simple file-copy task.

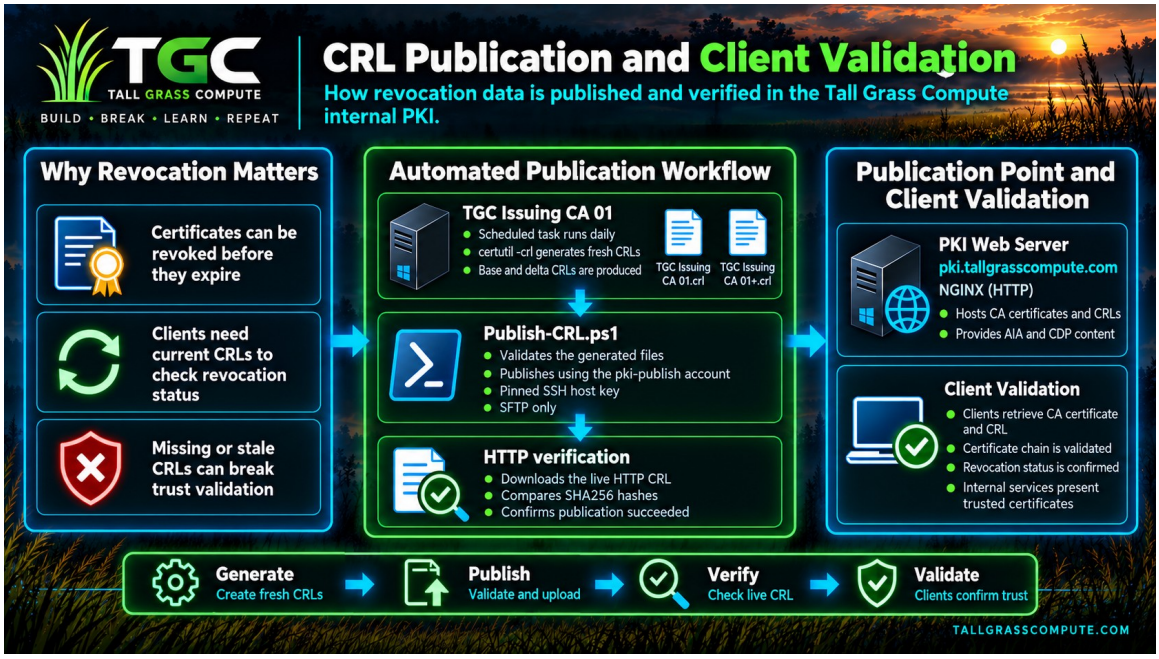


Figure 8 — CRL publication became a second automation workflow: generate on the online issuing CA, publish through a constrained SFTP path, retrieve over HTTP, and verify the published file by SHA-256 hash. The offline root remains manual by design.

The finished job runs on TGC-AD-PKI01 as SYSTEM. It generates fresh CRLs with certutil -crl, transfers them through a dedicated ED25519 key, downloads the HTTP-served copy, compares SHA-256 hashes, logs the result, and exits non-zero if publication verification fails.

The Linux-side key is restricted to the issuing CA source address and forced into SFTP-only operation. The NGINX host key is pinned. The publication account has one job: place PKI publication material in the web root.

Online Issuing CA vs. Offline Root

The issuing CA and root CA deliberately follow different operational models.

TGC Issuing CA 01 is online because it exists to perform routine certificate operations, so automated CRL publication makes sense. The root CA remains offline, and its CRL publication stays controlled and manual.

Online issuing CA: automate routine PKI operations. Offline root CA: preserve the isolation that makes it a root of trust.

9. A Few Paths We Did Not Keep

Before settling on acme2certifier, I tested other approaches, including direct certificate enrollment from Linux and Microsoft CEP/CES. Those experiments proved individual pieces of the workflow, but they also added complexity that did not fit the final goal.

The design became much cleaner once I stopped trying to make every Linux system understand Microsoft enrollment protocols directly. The gateway became the translation layer instead: clients speak ACME, while acme2certifier handles Microsoft-specific enrollment behind the scenes.

Once that direction was clear, the temporary CEP/CES components, the client-authentication template created for that experiment, and other abandoned enrollment tooling were removed.

10. Lessons Learned

Use the protocol the clients already understand.

Proxmox already knows ACME. So do many other infrastructure platforms. The better design was not to teach every client Microsoft PKI enrollment; it was to centralize that translation in one place.

Centralize Microsoft-specific complexity.

Kerberos, MS-ICPR, AD identities, certificate templates, and CA policy all live behind the gateway. Individual clients need only the ACME directory and the service name they are requesting.

Restrict the bridge aggressively.

The service identity, certificate template, DNS namespace, network access, keytab mount, and CRL publication key are all deliberately constrained. Automation connected to a certificate authority should have the smallest useful blast radius.

Issuance is only part of PKI operations.

The stale CRL was the most useful discovery in the build. A certificate can be issued correctly while the revocation infrastructure behind it is unhealthy. AIA and CDP publication are part of the lifecycle, not optional documentation details.

Validate runtime state, not configuration intent.

The final tests checked the certificate actually presented by pveproxy, the chain to the trusted root, and the published CRL. The same philosophy has shown up repeatedly in Tall Grass Compute projects: configuration state tells you what should be true; runtime validation tells you what actually is true.

11. Final Result

The original objective was to automate internal certificates for Linux infrastructure. The finished design goes further: Tall Grass Compute now has an internal ACME service backed by Microsoft AD CS.

From the client side, the experience is ACME. From the PKI side, the authority is still Microsoft AD CS.

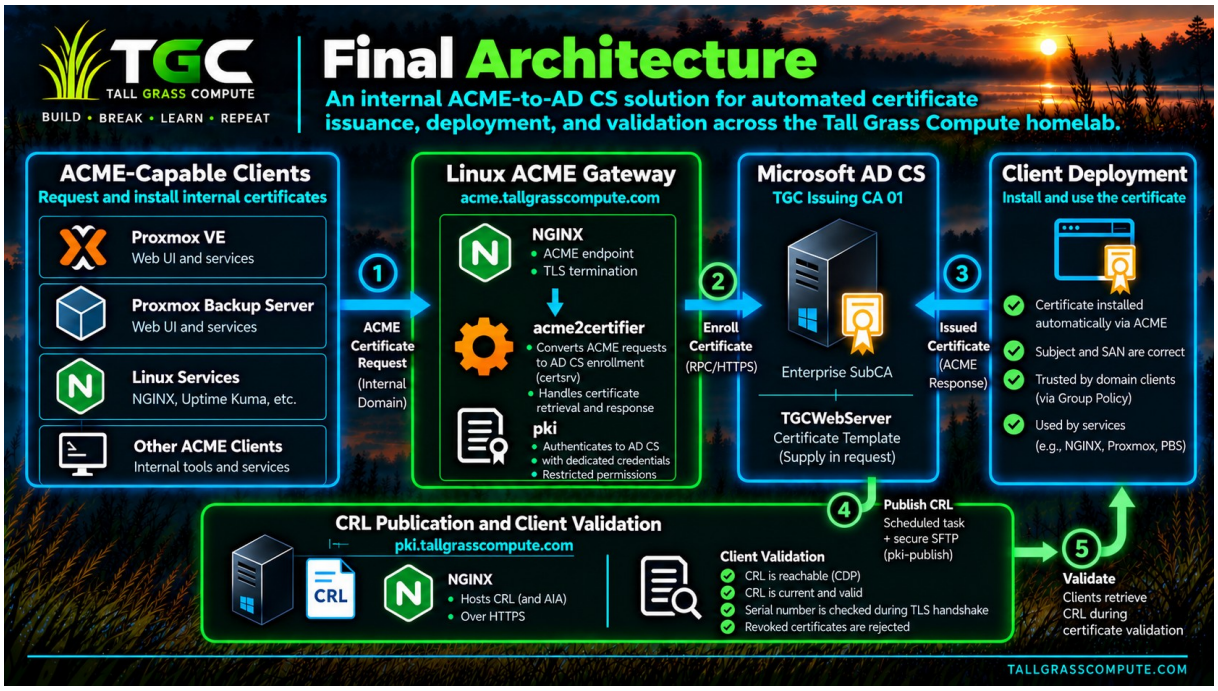


Figure 9 — Final architecture: ACME-capable clients enroll through the internal gateway, Microsoft AD CS remains the issuing authority, certificates are deployed by the clients, and CRL publication closes the validation loop.

Proxmox was the first client used to prove the design, but the architecture is not tied to Proxmox—or even to Linux. Any compatible ACME client that meets the policy and validation requirements can use the same service.

The project started as a Linux certificate problem. It ended as a reusable bridge between two certificate ecosystems already present in the lab.

The final architecture brings the enrollment, deployment, publication, and validation paths together into one internal certificate lifecycle.

Build it. Break it. Find out what actually happens.

That is exactly what happened here. ACME issuance worked. Then deeper validation found a stale CRL. The publication path was automated and hardened. Finally, the complete service was tested again with live chain and revocation checking enabled.

The result is not merely a certificate that can be issued. It is a certificate lifecycle I can trust end to end.

What Comes Next

The architecture is now proven with Proxmox, but the useful part is that nothing about the gateway is Proxmox-specific. The same internal ACME directory can become the enrollment point for other compatible services in the lab as their certificate lifecycles are moved onto the same model.

The next phase is therefore operational rather than architectural: onboard additional internal services, confirm unattended renewal behavior, and keep validating the pieces that live outside the certificate itself - the listener, the trust chain, AIA, and current CRLs. The bridge is built; the job now is to use it consistently.

That is the end state I wanted from the beginning: Microsoft AD CS remains the internal authority, while the systems consuming its certificates get the simple, repeatable ACME experience they already understand.