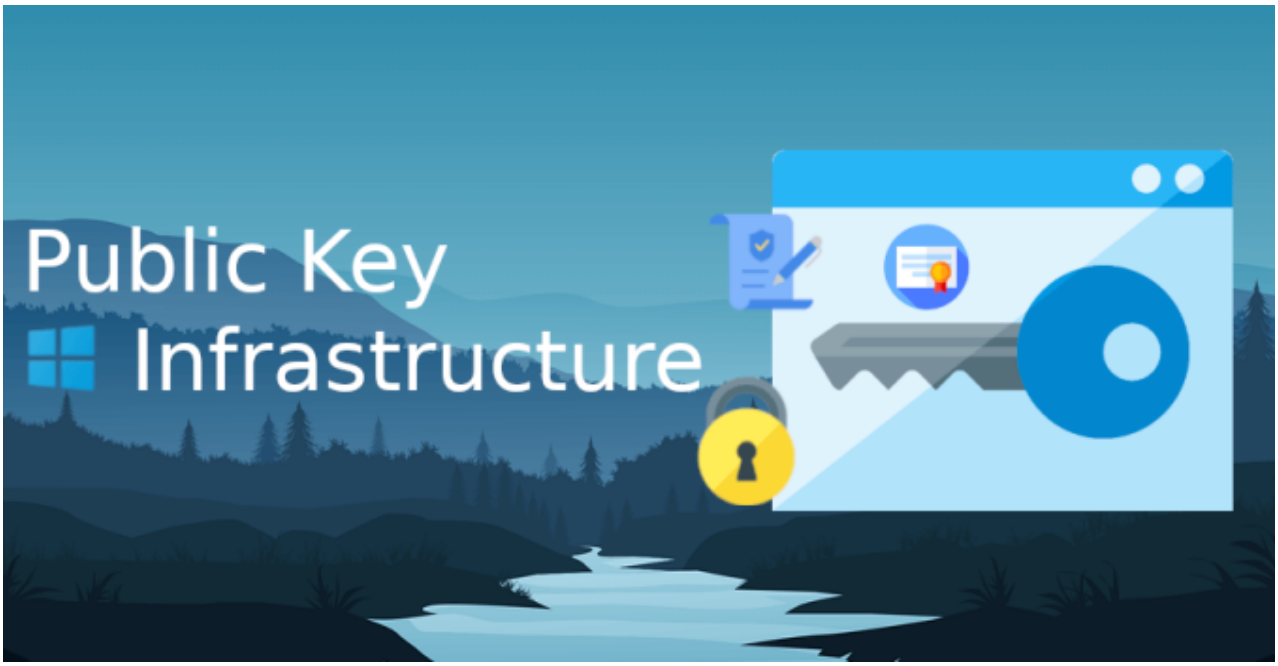
PKI – Part 6: Demystifying the CAPolicy.inf file

michaelwaterman.nl/2025/05/18/pki-part-6-demystifying-the-capolicy-inf-file

Michael Waterman May 18, 2025



Ever heard of the capolicy.inf file? It's like a digital instruction manual for a Certificate Authority (CA) server. It pre-configures the CA and has a say in how certificates are set up or renewed. In other words, it's a behind-the-scenes helper that ensures everything is governed with digital certificates. In this blog post, I'll break down what this file does and why it matters in plain and simple terms.

Usage of capolicy.inf in PKI

Now that I've been introduced to the `capolicy.inf` file, the configuration file that shapes how your CA behaves, it's time to look at what it actually does under the hood. While it may seem like a simple text file, its role is anything but minor.

In this part of the blog, we'll break down how `capolicy.inf` influences key aspects of your Public Key Infrastructure (PKI): from certificate validity and policy OIDs to CRL distribution points and renewal behavior. It's not magic, it's configuration. But when used correctly, it becomes one of the most powerful tools to keep your CA setup clean, predictable, and aligned with your security requirements.

Let's take a practical look at how this file works, and why it deserves more attention than it usually gets.

Definition and purpose of capolicy.inf

So, what exactly is this capolicy.inf file? Well, think of it as the brain behind a Certificate Authority (CA) server. Its job is to hold a set of instructions that tell the CA how to operate. These instructions include details like how long certificates are valid for, what they can be used for, and who's allowed to get them.

In simple terms, the capolicy.inf file is like a rulebook for the CA. Without it, the CA might issue certificates all willy-nilly, which could lead to chaos and security risks. With the capolicy.inf file, everything is clear and organized, making sure the CA does its job properly.

Where to locate the file

Now that we understand the capolicy.inf file's role, let's talk about where to find it. Here's the twist: by default, this file doesn't exist! You need to created it, think of it as crafting your own set of rules for your Certificate Authority (CA).

To get started, you'll need to create the capolicy.inf file before you even think about configuring the certificate authority role. Here's how:

1. *File Creation*: Open a text editor, like Notepad, and create a new text file.
2. *File Location*: Save this text file in the "*C:\Windows*" directory on your server. This location is crucial because it's where your CA server will look for the capolicy.inf file when it starts issuing certificates.
3. *File Encoding*: Ensure that the file is saved with ANSI encoding. This encoding ensures that the file format is compatible with the CA server's expectations.

Once you've completed these steps, you'll have your very own capolicy.inf file in the right place, ready to guide your CA server in issuing certificates securely and in line with your defined policies. It's your rulebook, and you're in control.

Watch Out: Don't let Notepad save it as .txt!

When creating `capolicy.inf` using Notepad, make sure the file extension is truly `.inf`, not `.txt`. This is a very common mistake, especially on systems where file extensions are hidden by default.

How to do it right:

1. Open Notepad.
2. Click **File > Save As**.
3. In the **File name** field, enter `"capolicy.inf"` **with quotes**.
4. In the **Save as type** dropdown, select **All Files**.
5. Save it to "*C:\Windows*".

> Why the quotes?
> If you just type `capolicy.inf` without quotes, Notepad may silently save it as `capolicy.inf.txt`, and your CA **will ignore it**.

This tiny mistake can cause hours of confusion, especially since there's no warning or error during CA installation. The installation just proceeds with defaults settings.

Root CA – capolicy.inf explained line by line

So, let's dive in, below is a sample capolicy.inf file used to configure a Root Certification Authority (CA). This file provides critical settings that influence how certificates are issued and renewed. Let's break it down line by line so you understand exactly what each section and directive means.

```
[Version]
Signature="$Windows NT$"

[Certsrv_Server]
RenewalKeyLength=4096
RenewalValidityPeriod=Years
RenewalValidityPeriodUnits=10
CRLDeltaPeriod=Days
CRLDeltaPeriodUnits=0
CNGHashAlgorithm=SHA256
AlternateSignatureAlgorithm=0

[CRLDistributionPoint]
[AuthorityInformationAccess]

[Extensions]
2.5.29.19= critical,CA=true
```

File Signature

```
[Version]
Signature="$Windows NT$"
```

This line identifies the file as a valid configuration file for Windows Certificate Services. It must be present at the top of every `capolicy.inf` file.

[Certsrv_Server]

```
[Certsrv_Server]
RenewalKeyLength=4096
```

Specifies the key length in bits for the CA's next renewal. In this case, when the CA certificate is renewed, a 4096-bit key will be generated, offering strong cryptographic protection.

```
RenewalValidityPeriod=Years
RenewalValidityPeriodUnits=10
```

Defines the default validity period for the CA certificate renewal. Here, it's set to 10 years. These settings do not affect issued certificates, only the CA certificate itself.

```
CRLDeltaPeriod=Days
CRLDeltaPeriodUnits=0
```

Disables delta CRLs by setting the unit to zero. Delta CRLs are typically used to publish only changes since the last base CRL, but for Root Ca's not very practical as they don't revoke CA's that often (hopefully).

```
CNGHashAlgorithm=SHA256
```

Specifies the hash algorithm used for signing the CA certificate and CRLs. SHA256 is the modern standard that ensures strong security while remaining compatible with most systems. More on CNGHashAlgorith down below.

```
AlternateSignatureAlgorithm=0
```

Disables the use of alternate signature algorithms (such as dual signing with SHA1 and SHA2), which were primarily used for backward compatibility. Setting this to 0 ensures the CA uses a single, modern signature algorithm.

[CRLDistributionPoint]

```
[CRLDistributionPoint]
```

This section would normally define one or more URLs where clients can download the Certificate Revocation List (CRL). If no values are specified (as in this case), it means the CRL distribution points will ***not*** be included in the Root CA certificate.

This is common practice for Root CAs, and here's why:

- *Trust is established differently*: The Root CA certificate is not validated using a CRL like subordinate certificates are. Instead, it's ***trusted explicitly*** because it's placed in the Trusted Root Certification Authorities store on clients.
- *No online availability expected*: Since the Root CA is offline for security reasons, publishing an online URL in the certificate (like an HTTP-based CRL Distribution Point) would point to a service that doesn't exist. This could cause unnecessary delays or errors during validation attempts.

- *Subordinate CAs handle revocation*: Clients typically rely on CRLs from subordinate CAs to verify the status of end-entity certificates. The Root CA only needs to revoke its own subordinate CAs, and even that is handled out-of-band or manually updated.

In short, omitting CDPs in the Root CA certificate avoids confusion and aligns with its offline and explicitly trusted nature.

[AuthorityInformationAccess]

```
[AuthorityInformationAccess]
```

This section is used to define the Authority Information Access (AIA) extension, which typically includes:

- *CA Issuers URL*: A pointer to where the issuing CA's certificate can be downloaded.
- *OCSP responder location*: For real-time certificate status checking.

In this example, the section is left empty, meaning no AIA extension will be added to the Root CA certificate. This is not a mistake, it's best practice for Root CAs.

Here's why:

- *Root certificates don't have issuers*: A Root CA is self-signed, meaning it issues its own certificate. There is no higher CA to reference, so a CA Issuers URL would serve no purpose.
- *Trust is pre-established*: Clients trust the Root CA because it's already installed in their Trusted Root Certification Authorities store, not because they fetched it online via an AIA location.
- *No OCSP needed*: Root CAs don't typically publish OCSP endpoints because they don't issue end-entity certificates directly and are offline by design.

Including an AIA section in a Root CA certificate can lead to broken URLs or unnecessary network lookups that provide no value. So, keeping this section empty reinforces the offline, explicitly trusted role of the Root CA.

[Extensions]

```
[Extensions]
2.5.29.19= critical,CA=true
```

This defines the "Basic Constraints extension:

- `critical` marks this extension as mandatory for certificate validation.
- `CA=true` indicates that this certificate is intended to be a Certification Authority.

Without this extension, a certificate cannot function as a CA, and clients would reject it when attempting to validate a trust chain. Please note that a default installation of a Microsoft CA already includes these settings, I just like to be explicit.

capolicy.inf for Enterprise CAs, key differences

Now that we've broken down the configuration for an Root CA, let's look at a second example, this time for an Enterprise CA. These are typically Subordinate CAs that issue certificates to users, computers, and services within an organization.

Instead of explaining every line again, we'll focus on the differences (the delta) between this Enterprise CA configuration and the previous Root CA example. This approach keeps things concise and helps you understand when and why specific settings differ based on the CA's role.

```
[Version]
Signature="$Windows NT$"

[Certsrv_Server]
RenewalKeyLength=2048
LoadDefaultTemplates=0
CNGHashAlgorithm=SHA256
AlternateSignatureAlgorithm=0

[PolicyStatementExtension]
Policies=CorpPolicy

[CorpPolicy]
OID=1.3.6.1.4.1.55468.1.1
Notice=This Certification Authority is an internal resource. Certificates issued
by this CA are for internal use only. See the Certification Practices Statement
for more information
URL=http://trust.corp.michaelwaterman.nl/cps/cps.html

[CRLDistributionPoint]
URL=http://trust.corp.michaelwaterman.nl/crl/Corp-Enterprise-CA.crl

[AuthorityInformationAccess]
URL=http://trust.corp.michaelwaterman.nl/crl/Corp-Enterprise-CA.crt
[Extensions]
2.5.29.19= critical,CA=true,pathlength=0
```

RenewalKeyLength

```
RenewalKeyLength=2048
```

For the Enterprise CA, the renewal key length is set to 2048 bits, which is often sufficient for CAs with shorter certificate lifetimes and internal scope. Root CAs, by contrast, typically use 4096-bit keys for long-term trust anchors. See my blog on key lengths for more explanation.

Certificate Templates

```
LoadDefaultTemplates=0
```

This setting prevents the CA from automatically loading default certificate templates upon installation. It gives administrators more control over which templates are published and issued, especially useful in tightly regulated environments.

Policy Statement Extension

```
[PolicyStatementExtension]
Policies=CorpPolicy

[CorpPolicy]
OID=1.3.6.1.4.1.55468.1.1
Notice=...
URL=http://trust.corp.michaelwaterman.nl/cps/cps.html
```

Enterprise CAs often include detailed policy information using custom OIDs and notices. This helps organizations formalize and communicate their internal PKI policies, and point to documents like a Certification Practice Statement (CPS). See my blog on object identifiers for a more detailed explanation on this specific topic.

Side step, [PolicyStatementExtension] on a Root CA

In the `capolicy.inf` file, the `[PolicyStatementExtension]` section is used to include Certificate Policies in a CA certificate. But when it comes to a Root CA, this section is usually omitted, and that's by design.

Let's break down ***why*** this is the case and what can go wrong if you include it anyway.

Root CAs don't enforce Policies, they delegate trust

A Root CA's job is to establish trust and delegate certificate issuance to subordinate CAs. It doesn't issue end-entity certificates directly (or shouldn't, in best practice setups), and therefore, it doesn't need to assert certificate issuance policies.

Including a policy on the Root CA certificate tells clients:

> "Any certificate that chains to this Root must match this policy OID."

And that's the problem.

The Pitfall: Policy constraints are inherited down the chain

If you include a policy in your Root CA like:

```
[PolicyStatementExtension]
Policies= RootPolicy

[RootPolicy]
OID= 1.3.6.1.4.1.99999.1.1
```

…then all subordinate CAs and end-entity certificates must ***also*** include this policy OID (or a mapped equivalent), or validation may fail on systems that enforce policy chaining (e.g., Windows, ADCS with smartcard logon).

This behavior is defined in RFC 5280 under certificate policy processing.

Example of things going wrong:

- Your Root CA includes a custom policy OID.
- Your Subordinate CA is configured with its own policy OID for issuing certificates.
- Clients try to validate an end-user certificate, but the policy OIDs don't match the one in the Root.
- Result: Certificate is rejected due to policy mismatch.

This is extremely difficult to debug, especially in environments like smartcard logon or automated TLS validation.

Best practice for Root CAs

Do not define [PolicyStatementExtension] **in the Root CA's capolicy.inf**.

Keep the Root CA certificate minimal and clean, it should only contain what's strictly necessary:

- Signature algorithm
- Validity
- Basic Constraints (`CA=true`)
- (Optional) Name constraints (in specific cases)

Let Subordinate CAs handle policies that reflect issuance purposes (like code signing, authentication, etc.). Now let's continue with the capolicy.inf file for the Enterprise CA.

CRL Distribution Point

```
[CRLDistributionPoint]
URL=http://trust.corp.michaelwaterman.nl/crl/Corp-Enterprise-CA.crl
```

Unlike the Root CA, the Enterprise CA ***must*** provide a working URL for CRL distribution. Clients need this to check revocation status for end-entity certificates in real-time. Since the Enterprise CA is online, the URL points to an HTTP-accessible location.

Authority Information Access

```
[AuthorityInformationAccess]
URL=http://trust.corp.michaelwaterman.nl/crl/Corp-Enterprise-CA.crt
```

This adds the CA Issuers AIA extension, pointing to a location where clients can retrieve the issuing CA certificate to build a complete trust chain. It's essential for subordinate CAs that aren't already trusted by the client.

Basic Constraints: Path Length

```
[Extensions]
2.5.29.19= critical,CA=true,pathlength=0
```

This limits the CA from issuing certificates to another subordinate CA. The `pathlength=0` restriction means it can only issue end-entity certificates, reinforcing strict hierarchy and control. ***Pay real good attention to the latter setting, you can not change this afterwards!***

Understanding 2.5.29.32.0: The "Any Policy" Trap

When configuring the `capolicy.inf` file, you might come across or be tempted to use the following snippet:

```
[PolicyStatementExtension]
Policies= AllIssuancePolicy
Critical= FALSE

[AllIssuancePolicy]
OID= 2.5.29.32.0
```

At first glance, this might seem harmless or even helpful, but let's take a closer look at what it really means, and why using it without fully understanding the consequences can lead to weakened trust in your PKI.

What is 2.5.29.32.0?

This Object Identifier (OID) comes from RFC 5280 and represents the special value **"*anyPolicy*"**. When included in a certificate, it tells consuming systems:

> "This certificate is valid under *any* certificate policy."

That might sound convenient, and it is, but in the worst way. You're essentially ***removing all policy-based restrictions***.

Why is this risky?

Including 2.5.29.32.0 can have the following negative impacts:

- *Bypasses policy enforcement*: Systems that rely on certificate policies (like smartcard logon, code signing, or government ID frameworks) may allow certificates that otherwise wouldn't pass policy checks.
- *Breaks compliance*: For organizations bound by standards like eIDAS, ETSI, or WebTrust, this OID is explicitly disallowed because it fails to establish a concrete policy trust anchor.
- *Causes unintended chaining*: Some clients may interpret this as "match-all," potentially allowing certificates to validate under unexpected or unauthorized policy chains.

Real-World example: Smartcard logon fails silently

Let's say your environment requires smartcard logon certificates to include a specific policy OID, e.g., `1.3.6.1.4.1.311.20.2.2`. If your issuing CA has `2.5.29.32.0` in its own policy extension, some clients might silently accept certificates ***without the required policy***, resulting in authentication issues that are extremely difficult to troubleshoot.

Best Practice

Avoid using 2.5.29.32.0 in production CAs. Instead:

1. Define your own policy OIDs (either internally or via a registered enterprise OID root).
2. Include meaningful notices and a CPS URL in the PolicyStatementExtension.
3. Only include policy OIDs that reflect actual issuance restrictions.

Important Note: capolicy.inf ≠ Initial Setup

Some settings in `capolicy.inf`, such as `RenewalKeyLength`, `RenewalValidityPeriod`, and `CNGHashAlgorithm`, only apply when the CA certificate is being renewed, not during the initial CA installation.

During CA Setup:

- The initial key length, hash algorithm, and cryptographic provider are defined through the setup wizard or command-line tools like `certutil` or `Install-AdcsCertificationAuthority`.
- At this point, the `capolicy.inf` file is partially ignored for those specific cryptographic settings.
- For example: `RenewalKeyLength=4096` does **not** affect the initial key. You'd have to explicitly choose 4096-bit during setup.

During CA Renewal:

When you later renew the CA certificate (e.g., after 10 years), `capolicy.inf` settings like:

- RenewalKeyLength
- RenewalValidityPeriod
- CNGHashAlgorithm

…do take effect, and control how the new key pair and certificate are generated.

Why this matters

If you assumed that placing `RenewalKeyLength=4096` in `capolicy.inf` before installing your CA would result in a 4096-bit key, you might end up with a weaker 2048-bit key, and you won't know until it's too late. The only way to enforce this at setup is through:

- The CA installation wizard (GUI)
- PowerShell (e.g. `Install-AdcsCertificationAuthority -KeyLength 4096`)
- Certutil-based scripted deployments

Does CNGHashAlgorithm actually do anything initially?

Yes, but only under the right conditions.

The CNGHashAlgorithm setting in `capolicy.inf` only applies if your CA is configured to use a CNG (Cryptography Next Generation) provider, such as the "*Microsoft Software Key Storage Provider*".

In that case, this setting determines which hashing algorithm will be used when signing the CA certificate, for example, SHA256 instead of older options like `SHA1`. However, if you're using a legacy CSP (Cryptographic Service Provider) such as the "*Microsoft Strong Cryptographic Provider*", then CNGHashAlgorithm is ignored , even if it's present in the file. CSPs don't honor this setting and instead use default hash algorithms depending on system configuration and key size.

Tip: If you want to guarantee use of SHA256 or higher, use PowerShell to install the CA with explicit parameters like:

```
Install-AdcsCertificationAuthority-HashAlgorithmName SHA256 -CryptoProviderName
"Microsoft Software Key Storage Provider"
```

* More on installing your Ca's in the blog posts to come!

Do's and don'ts for using capolicy.inf

The `capolicy.inf` file plays a subtle but critical role in how your Certificate Authority behaves. It's easy to overlook, but a poorly configured file can introduce serious misconfigurations in your PKI hierarchy.

This chapter outlines some practical Do's and Don'ts, with a focus on security, maintainability, and avoiding common pitfalls.

Best practices for working with capolicy.inf

Do create and document your file before CA installation

The `capolicy.inf` file must be present in "*C:\Windows*" ***before*** installing or renewing a CA. If you forget, Windows will proceed with defaults, and those aren't always secure or compliant with your organization's needs.

Do use version control or change tracking

Even though it's "just a text file," treat `capolicy.inf` like a configuration item. Track changes using version control or include comments with dates and responsible persons:

Do test in a lab before deploying to production

Especially when defining CRL or AIA URLs, it's critical to test the impact of these settings in a non-production environment. A broken URL can lead to certificate validation failures across the entire organization.

Do align settings with your CA's role

- *Root CA*: Keep the file minimal, no CRL/AIA URLs, no policy extensions.
- *Subordinate CA*: Include policy, CRL and AIA extensions based on internal requirements.

Common pitfalls and what to avoid

Don't reuse the same file for Root and Subordinate CAs

Each CA has a different purpose and trust model. Reusing the same file can introduce incorrect extensions or policies that break chain validation.

Don't include 2.5.29.32.0 (anyPolicy) in production

As explained earlier, this bypasses all policy enforcement and weakens your PKI security model. Define real OIDs that reflect issuance intent.

Don't forget to use ANSI encoding

Some tools (especially older ones) expect `capolicy.inf` to be ANSI-encoded. Using UTF-8 with BOM or Unicode can cause parsing issues during CA installation.

Don't guess OIDs, document them

If you're defining your own OIDs for certificate policies, make sure they are:

- Unique to your organization
- Documented with purpose and use case
- Mapped correctly to templates or issuance logic

Security precautions

- Avoid hardcoding internal server names or IPs in CRL/AIA URLs, use DNS aliases like `http://trust.contoso.com/crl/`.
- Don't expose unnecessary URLsin Root CA certificates, especially for offline CAs. If the URL doesn't exist or respond, clients may pause or fail during validation.
- Validate your `capolicy.inf` with tools like `certutil -dump` after CA creation or renewal, to check which extensions were applied.

Handling sensitive configurations

While the `capolicy.inf` file doesn't contain secrets (like private keys), it does influence certificate chain behavior and trust. Treat it with care:

- Limit who can edit the file (e.g., only senior PKI admins).
- Back up before making changes, especially prior to a renewal.
- Use comments to explain why a change was made, not just what was changed.

Conclusion

The `capolicy.inf` file may be small and easy to overlook, but its impact on your Public Key Infrastructure (PKI) is anything but minor. It acts as the hidden blueprint that silently shapes how your Certification Authority behaves, from key sizes and validity periods to CRL distribution and policy enforcement.

We've seen how Root CAs benefit from a minimalist approach, while Enterprise CAs require more detailed configuration to support certificate chaining and client validation. We've also explored how certain choices, like including 2.5.29.32.0 or misusing policy extensions, can unintentionally weaken the trust model your PKI is built upon.

Whether you're spinning up a test CA or deploying a production-grade hierarchy, the key takeaway is this:

> ***Treat the capolicy.inf file with the same care and intention as your CA's private key.***
> A misstep here won't just cause technical glitches, it can undermine trust, compliance, and security.

Take the time to plan, test, document, and review. It's not just a text file.

References

Robinharwood. (n.d.). *Prepare the CAPolicy.inf file*. Microsoft Learn. https://learn.microsoft.com/en-us/windows-server/networking/core-network-guide/cncg/server-certs/prepare-the-capolicy-inf-file