Code Signing Certificates:
Protecting Software Distribution

Every time an operating system displays a warning about an unrecognized publisher, or silently allows an application to install without any friction at all, a code signing certificate is somewhere behind that decision. This article covers how code signing actually works, why it matters more than ever given the current software supply chain threat landscape, and what organizations distributing software need to get right.

What Code Signing Actually Verifies

A code signing certificate allows a software publisher to cryptographically sign an executable, script, or software package, creating a signature that proves two things to anyone running it: that the code genuinely came from the claimed publisher, and that it has not been altered or tampered with since it was signed. Operating systems, browsers, and security software use this signature to determine how much to trust the software, ranging from a smooth installation experience to prominent warnings or outright blocking for unsigned or improperly signed code.

Why Long-Term Validity Reasoning Differs Here

Code signing certificates present a genuinely different validity consideration than TLS server certificates, discussed elsewhere in this series in the context of short-lived certificates becoming the industry norm. Software signed today may still be running, unmodified, years or even decades after its original release, and a naive approach where the signature simply becomes invalid once the certificate itself expires would break the ability to verify old, legitimately signed software indefinitely. This is why code signing relies heavily on trusted timestamping, where a signature is stamped with a cryptographically verified time at the moment of signing, allowing verification systems to confirm the code was signed while the certificate was still valid, even long after that certificate has since expired.

Extended Validation for Code Signing

Many platforms, including Microsoft's SmartScreen reputation system, give considerably smoother installation experiences to software signed with an Extended Validation code signing certificate, which requires the same rigorous organizational identity verification covered elsewhere in this series applied specifically to a software publisher. EV code signing certificates are also frequently required to be stored on a hardware token or hardware security module rather than as a plain file, specifically to prevent the kind of private key theft that has, in real incidents, allowed attackers to sign malware with a legitimate, stolen code signing key.

The Real Cost of a Compromised Code Signing Key

A stolen code signing private key is a uniquely damaging compromise, since it allows an attacker to sign malicious software that inherits the full trust a legitimate publisher's signature carries, potentially bypassing security warnings entirely and getting distributed to every user who trusts that publisher's signed software. Several well-documented supply chain attacks have involved exactly this pattern, malware signed with a compromised legitimate certificate, which is precisely why hardware-backed key storage and strict internal access controls around code signing keys are considered essential rather than optional best practice for any organization distributing software at scale.

Integrating Code Signing Into Build Pipelines

Modern software development increasingly automates code signing as part of the CI/CD build and release pipeline, discussed in a broader context elsewhere in this series, signing every build artifact automatically before it is published or distributed. This automation needs to be designed carefully so the signing key itself, particularly for EV certificates requiring hardware-backed storage, is accessed securely through a dedicated signing service or hardware security module integration rather than exposed directly within the build environment where it could be extracted by a compromised build server or a malicious dependency.

Code Signing for AI Models and AI-Generated Artifacts

As organizations increasingly distribute AI models, model weights, and AI-generated software artifacts, the same code signing principles are being extended to these new categories of distributed content, providing provenance verification for a model file in much the same way code signing has long verified traditional software. This is a genuinely emerging area, but the underlying logic transfers directly: anyone downloading and running a model or an AI-generated script benefits from the same cryptographic assurance of authenticity and integrity that code signing has provided for traditional executables for decades.

The Countdown Is Already Running: 200 Days, 100 Days, 47 Days

Every certificate conversation in 2026 eventually arrives at the same clock, and it is worth closing on it here. The CA/Browser Forum's Ballot SC-081v3 is not a proposal under discussion; it is an approved, already-in-motion schedule. Maximum public TLS certificate lifetimes fall from 398 days to 200 days on March 15, 2026. They fall again to 100 days on March 15, 2027. By March 15, 2029, they drop to just 47 days, with domain validation itself needing to be re-proven roughly every 10 days.

01

Translate that into operational terms and the picture gets stark quickly. An organization currently renewing certificates a few times a year will be handling renewal events on the order of every couple of weeks by the end of this countdown, across every endpoint it operates. Manual tracking, calendar reminders, and a spreadsheet somebody checks once a month will not survive contact with that cadence. What has always been an occasional chore is becoming a continuous, automated operation, whether an organization plans for it or not.

02

Code signing certificates themselves still need to be renewed under the shrinking public lifetime schedule below even though trusted timestamping preserves the validity of already-signed code, which means the signing certificate renewal process, distinct from the signed artifacts it produced, needs the same automation discipline as any other certificate approaching a 47-day maximum

03

The 200-day, 100-day, and 47-day milestones are not distant hypotheticals; the first has already arrived. Organizations that build the automation loop now, generating keys, vaulting them securely, brokering issuance across Certificate Authorities through APIs, and rebinding certificates to live endpoints without manual intervention, will meet each deadline without disruption. Organizations that wait will be rebuilding their certificate operations under deadline pressure, with far less room for error and far less time to get it right. The countdown is the call to action. The only real decision left is whether to automate on your own schedule, or on the CA/Browser Forum's.