digicert®

# Practical Guide to Code Signing Policy: Improve your Security, Control, & Governance

POLICY GUIDE

# Practical Guide to Code Signing Policy: Improve your Security, Control, and Governance

**Introduction**

## The Purpose of Code Signing

Code signing adds a tamper-proof seal to software. It allows software users–including browsers, app stores, and operating systems–to verify the identity of the software creator and establish that the software has not been altered since it was signed and released.

Based on Public Key Infrastructure (PKI), code signing is a non-repudiable attestation process, establishing the authority and integrity of the software. In other words, code signing proves, without doubt, where the software came from and that the code is original.

To ensure trust in the signature, organizations must secure their software development lifecycle (SDLC) and software supply chain (SSC). Capturing your rules and practices within a code signing policy establishes a baseline to align code security across all development teams.

## The importance of a strong, documented signing policy

Security regulations and standards that apply to the SDLC and software infrastructure are undergoing rapid changes due to attacks on the software supply chain and the SDLC itself. And new regulations are being added every day.

Whether you are being driven by the CA/B Forum Baseline Requirements, the NIS2 Directive, or PCI DSS industry standards, you'll want to create a single, cohesive policy to address all the regulations that apply to your organization. Remember, it's not just where your company is located that determines applicable regulations. Your software must comply with the regulations of the countries where it is sold and used.

Code signing is a crucial step in the process of building and protecting software that greatly impacts Security and Developer Operations (DevOps) teams. Security/DevSecOps governs what security actions should occur and why they're required. DevOps needs to translate that knowledge into concrete steps, systems settings, and integrations to meet these goals. Often these security measures are seen as disruptive to software delivery. However, embedding the policy tenets into the signing applications and CI/CD platforms enables developers to sign code correctly and smoothly, increasing the speed and efficiency of the release process.

This guide can help bring these groups together and generate the appropriate security actions based on software product risk, software development environments, and security maturity.

## How to use this guide

**Part 1: Gather Information and Expertise**
Code Signing has stakeholders in several different organizations. Preparation and information gathering will help policy writing move faster.

**Part 2: Common Policy Components**
Understand the objectives of each policy section and the practices and procedures considered important by the leading software security and DevOps experts. Scope the level of impact of each section on your current release pipelines.

**Part 3: Write Your Policy**
Here's a sample to jumpstart the creation of your own policy.

**PART 1: Gather Information and Expertise**

# Gathering Expertise

Because this policy is where Security and DevOps overlap, it's important to get input from people in each department, along with Product and Engineering. You'll also want to gather input from related areas that might feel a direct impact from security and software changes in your organization. When it comes to writing a software signing policy, these are some of the most reported input sources:

- CISO
- DevSecOps/SecOps/InfoSec
- DevOps
- Product Security
- Product Management
- QA and Site Reliability
- Compliance
- Risk Management
- Customers and Partners

# Gathering Information

## Software teams and their development tools

**The Value of Forensics**

The best security is proactive. But mistakes offer the opportunity for valuable learning. Conduct regular forensic investigations into any recent security failures or missteps. If you haven't experienced any breaches or lapses yourself, examine others' breaches. Errors and vulnerabilities can help you create a signing policy that shrinks the threat landscape and better protects your code. Be sure your policy creates artifacts to use in investigations and vulnerability remediation.

### Regulations, Directives, and Acts

Identify the internal and external regulations that need to be met. Often standards bodies will provide regulations, frameworks, and additional guidance on how to meet the regulations. Look for government directives (EU NIS2 & US Executive Orders), industry regulations (CA/B Forum & PCI), and regulatory standards (ISO, ENISA & NIST).

### Prevention and Remediation

Code signing not only helps prevent software tampering, but it also generates auditable logs and artifacts that are helpful when remediating issues. Signed SBOMs, build scripts, containers, signature and activity logs, etc. all have useful data when investigating a malware attack or the risk of a new exploitable vulnerability.

**PART 2: Common Components in a Software Signing Policy**

# Scope

Policies are written with a purpose in mind. Be clear about who and what is subject to, and not subject to, the code signing policy. Example elements:

### Include

- **People:** Internal and external developers, signers, reviewers, product security, and auditors
- **Code:** Products you build for External Customers–both free and paid (External Products), Home-built systems (Internal Products)
- **Software Artifacts:** SBOMs, Macros, Libraries, Containers, and Build Scripts

### Exclude

**Third-party Applications:** Software that you use in creation of your software products should not be signed by you (e.g., productivity tools like Microsoft Office or development tools like JFrog). Any respectable third-party software will be digitally signed by its own creators. Other policies should ensure that the organization uses only software signed by a verified creator.

# Access & Privileges

### Authentication

Authentication confirms that the user trying to gain access to the system is the expected person and they are authorized to use the system. Usually there is already an authentication system in place and the policy just states that it needs to be used.

Multi-factor Authentication (MFA) or server-to-server authentication must be used to access and authorize signing per CA/B Forum baseline requirements.[1]

### Role-based Access Control (RBAC)

Access and privileges establish the characteristics of people or servers that should, and should not, participate in code signing activities, as well as which actions they are allowed to take and resources they are allowed to use.

RBAC[2] is an IT security best practice that includes defining roles and giving them distinct privileges, also known as Separation of Duties. Roles should be based on jobs that need to be done, like Submitter, Signer, and Security Officer, and not unrelated groups like Administrator or C-Suite. To prevent single points of failure, all roles should be assigned to more than one user where possible.

Even on a small team, assigning access and privileges to individuals can be a large, logistical challenge. Assigning privileges through roles makes it easier to maintain and scales as the team grows. Consider having roles beyond signers. Who should be able to request, issue, and revoke certificates? Are there roles that should be held by PKI managers, Security, or DevSecOps?

### Mitigating Risk with the Least Privileges Principle[3]

To have tighter control on security, only the minimum set of access and privileges should be given to anyone. This is more important for access to core resources, risky and irreversible actions, and high-risk software. Examples include:

- **Core Resources:** Build servers and HSMs
- **Irreversible Actions:** Keypair deletion and certificate revocation
- **Risky Actions:** Keypair export
- **High-Risk Software:** Code with low-level system access, such as Linux root access

The risk of each software product or project must be assessed based on the potential damage if the software contained malicious code or exploitable vulnerabilities that would result in unauthorized access to data, loss of system functionality, or other failure.

The criticality of software is a common regulatory theme. The term has a legal meaning in the US from Executive Order 14028 issued in May 2021. NIST provides many definitions and supplementary materials to help you determine criticality.

The concept of critical software can be found in other places, such as the EU Cyber Resilience Act, where network software and other critical products are subject to more rules and third-party assessments by a certified body.

For critical software and risky and irreversible actions, code signing policies often use multiple approvers to ensure no single person can make the change.

Another way to limit privileges is to focus access for teams to only the projects and products that they work on. For example, a developer working on infrastructure should not be able to sign code for a release by the mobile apps team. This limits privileges to not only the role someone plays, but the software they produce.

# Certificate and Keypair Management

### Software Environments

Unique code signing keypairs and certificates must be used for non-production and production environments. Test environments that are remotely accessible and process sensitive data must be treated as production environments.

Keypairs and certificates have a 1:1 ratio. At a minimum, you need one set for production signing and one for non-production signing. Theoretically, you could generate a new certificate and keypair for every build, and some teams do. This procedure has the advantage of limiting the exposure of a compromised key.

[1] CA|B Forum MFA requirements are in Baseline Requirements for the Issuance and Management of Publicly Trusted Code Signing Certificates Version 3.9.0–Sections 6.2.7.3 Private key storage for Signing Services and 6.5.1 Specific computer security technical requirements.

[2] As the definition of RBAC by NIST (National Institute of Standards and Technology) states, many regulatory compliance standards (such as NIST SP800-95 and NIST SP 800-53) rely on the proper implementation of RBAC.

[3] See NIST SP 800-53 AC-6 for more information on the Least Privilege Principle.

### Certificate and Keypair Storage

Since June 1, 2023, policies and standards have required that private code signing keys for both Standard/Organization Validation (OV) and Extended Validation (EV) certificates must be generated and stored on secure hardware devices per the CA/B Forum.[4] This requirement means public Certificate Authorities (CAs) can no longer support browser-based key generation and certificate installation or any other process including creating a CSR (Certificate Signing Request) and installing your certificate on a laptop or server. The provisioning method is set when ordering or renewing certificates.

Accepted secure hardware devices

- Hardware Tokens (e.g. USB cryptographic tokens)
- Hardware Security Modules (HSMs)–cloud-based or on-premises–compliant with FIPS 140-2 Level 2 or Common Criteria EAL 4+

Where should you generate, store, and use keys for non-production code? Ideally, you would want to use a FIPS/CC-complaint HSM. Still, less expensive measures, such as a software-based secrets management system, may be acceptable. Hazardous practices like storing keys on insecure file systems or in source code are never acceptable and team members should know never to use them.[5]

### Public vs. Private PKI–Signing Internal Software

Code signing is more consistent when the software's destination is deployment in app stores, operating systems, and regulated browsers. If software requires signing to run, that code gets signed. These requirements are not esoteric or arbitrary. Signing is proven to protect software from corruption, intrusion, and other malicious intent. When platforms enforce signing, they're requiring a practical solution to a threat, not a high-minded ideal.

In most cases, internal software signing is ignored because organizations trust the known developer teams and security surrounding their systems. Threats are lurking in the space beyond firewalls, once code is sent out into the world. Common sense would suggest that inside an organization, software passed between teams or deployed to internal servers for internal use are shielded.

In reality, internal software is vulnerable to attack, too. If someone were to gain access to the system, unsigned code makes an easy target that can be used against the infrastructure of the entire organization. As a best practice, internal signing is not an ideal—it's a practical matter that protects your software and your organization just as much as the protection deployed to keep malicious parties from exploiting it.

Software that is used or accessed outside of your organization should be signed using a CA-backed keypair that can easily be verified through browsers and standard PKI lists.

Software built for your organization's internal use only can use internal PKI, also known as Private PKI, where certificates are chained back to a trusted root, but are issued by your company through an internal Intermediary Certificate Authority (ICA). This is a common situation for software developed and used internally only.

Private PKI certificates and keypairs need to be controlled, managed, and secured, just like their Public PKI counterparts. Some Certificate Lifecycle Management (CLM) systems can manage both Public and Private PKI elements.[6] Administrators must also include the private root's public certificate in internal systems' lists of trusted roots.

### Certificate Inventory and Lifecycle

Maintain an inventory of all code signing certificates, monitoring expiration dates and generating new certificates proactively. In practice, in all but very small organizations, inventory and certificate management can only be done effectively using a CLM system that automates the process. If you are using a minimal number of certificates and keys, such as one for testing and one for production, ensure they are backed up in a secure location.[7]

Certificates must be revoked and reissued as needed and based on the frequency and/or expected lifespan of software releases. Automating certificate renewals ensures that software releases aren't disrupted by expired certificates. The CA/B Forum Code Signing Working Group has been discussing shortening the life of a code signing certificate from about 3 years to 460 days (~1 year 3 months). Automating code signing certificate renewal now will prepare you to handle the eventual change more smoothly.

Certificates and keys should be rotated and/or replaced regularly to maintain security levels. Changing these on a regular basis limits the amount of damage a lost or stolen key can do. Some CLM systems have functionality that can rotate keys automatically.

[4] See CA/Browser Forum's Baseline Requirements for the Issuance and Management of Publicly-Trusted Code Signing Certificates.

[5] Tale of Woe: What happened when Kali Linux lost their repo signing key.

[6] DigiCert has a solution for Public and Private PKI.

[7] See Footnote 4.

### Key Encryption

For the strongest security, encrypt keys at the highest level possible, based on what the resources that use the keys can decrypt. Your policy should clearly state which algorithm to use and the minimum acceptable level of key encryption for each certificate type or use case (e.g., production release for mobile applications.)

NIST has set a 2030 deadline for deprecating widely used, legacy encryption algorithms like RSA, ECDSA, EdDSA, DH, and ECDH. These algorithms are vulnerable to quantum computing and will be fully disallowed by 2035.

With automatic renewals based on profiles, an encryption upgrade can be set via the template and then happen automatically and systematically as each key is replaced. Industry experts and analysts expect companies will need to upgrade their encryption more than once in the next 5 years.

## Code Signing Process

### Security Reviews

Software source code must pass a security review according to the secure Software Development Life Cycle (SDLC) policy.[8]

Each release and associated component must be scanned for malicious indicators and known vulnerabilities. Malware and vulnerable software are as unacceptable in testing and other non-production environments as in production. For this reason, software should be scanned for these problems in the development cycle and in the CI/CD process.

Not all vulnerabilities are worth stopping a build or even a code shipment. Many have low severity, and others are difficult or impossible to exploit. You need to set a vulnerability threshold for when to stop a build or release.

Do not sign software if it contains confirmed malware or an exploitable vulnerability.

### Automated Processes

Best practice is to automate the code signing process as part of the SDLC and CI/CD pipeline. Use an approved Certificate Authority (CA) for production code signing certificates. Ideally, developers should use a unique signing key to sign any code they check-in. The individual signatures must be verified before signing a release.

Multiple users must not share production code signing keys unless activity logging can record and identify actions by unique users, including service users.

Organizations that automate the process can also exploit the security advantages of frequently rotating keys and certificates. Without automation, a "lazy" key rotation policy may be the most practical approach.

### Timestamps

Releases that will be valid after the code signing certificate has expired must include a timestamp from an approved timestamping service to show proof of signing time.

Starting April 15, 2025, Timestamp Authorities (TSA) increased their security with respect to storing certificates and keys, and their minimum level of hash algorithm to SHA-2. Timestamp Certificates also must include the Extended Key Usage (EKU) for Time Stamping. Be sure that your timestamps meet the new CA/B Forum requirements.

### Signing Release Artifacts and Software Bill of Materials (SBOM)

Release artifacts, including the software code branch, SBOM, build scripts, and Vulnerability Exchange Documents, must be signed and stored to help detect tampering with assertions and archives.

Some organizations sign different artifacts with different keys, so a compromised key only causes issues for one element in the build package.

Artifacts should be stored securely and may need methods for sharing them with customers or governing bodies. SBOMs, in particular, may be part of a quote package, product submission, or audit.

### Auditing and Logging

Maintain a comprehensive audit trail, including proof of code reviews, security scans, and code signing key action, access and privileges changes.

Activities must be associated with unique users and service accounts. This can happen through unique keys for every signer or through software sign logs if a group is sharing keys.

There should be a standard procedure on how to share logs with auditors and what file formats to use.

[8] See OWASP for assistance developing a secure SDLC and other useful security policies.

## Crypto-Agility Delivers Preparation and Response to Change

### Preparation and response to incidents and unexpected change

With the number and frequency of software supply chain and SDLC attacks increasing, being prepared to address a change with little notice is a must.

Define and, to the extent possible, automate procedures for documenting, revoking, and reissuing compromised certificates and keypairs.

After following your revoke/re-issue procedure, you will need to resign and redeploy the code and software artifacts that were signed with the compromised key.

Companies who automate these processes use Certificate and Keypair Aliases, and don't use the actual IDs. This allows their CI/CD system to continue running without interruptions or settings adjustment.

Establish processes for upgrading key encryption as a one-time event, in a schedule as certificates expire, and in bulk. These methods address small and large incidents, as well as timed upgrades based on regulations.

### Preparation and Response to Expected Changes

Regulations are constantly changing; however, they often are phased in or provide time to make system changes. Define and, to the extent possible, automate procedures for documenting and making changes progressively. For example, make updates when the certificates expire and need to be reissued.

There are two major changes to be prepared for: shorter lifecycles and changes in encryption algorithms.

Shorter lifecycles have been in discussion for some time, and the recently approved ballot shortening TLS certificate life is the leading indicator it will happen.

On the verge of Post Quantum Cryptography (PQC) breaking most of the encryption algorithms used today, we will see both new complex algorithms and elimination of existing algorithms.

Establish processes for upgrading key encryption as a one-time event, in a schedule as certificates expire, and in bulk. These methods address small and large adjustments, as well as timed upgrades based on regulation rollouts.

## Review and Updates

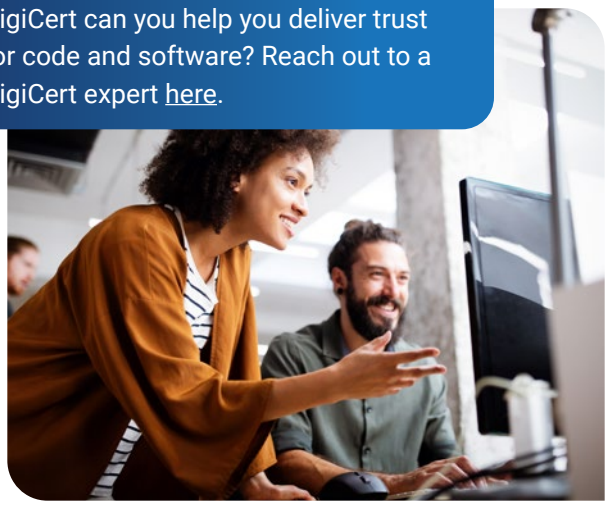
Review policies annually and update as necessary to ensure their effectiveness and alignment with industry standards and organizational changes.

**PART 3: Write Your Software Signing Policy**

Get a head start by downloading this editable policy. It includes places to include the concepts discussed in this guide. Use them together to craft a well thought out software signing policy. **Download the Sample Policy here.**

### DigiCert® Software Trust Manager

Would you like to hear more about how DigiCert can you help you deliver trust for code and software? Reach out to a DigiCert expert here.



## About DigiCert

DigiCert is the world's leading provider of digital trust, enabling individuals and businesses to engage online with the confidence that their footprint in the digital world is secure. DigiCert® ONE, the platform for digital trust, provides organizations with centralized visibility and control over a broad range of public and private trust needs, securing websites, enterprise access and communication, software, identity, content and devices. DigiCert pairs its award winning software with its industry leadership in standards, support and operations, and is the digital trust provider of choice for leading companies around the world. For more information, visit digicert.com