



Powered by  Garantir

DEPLOYING A MODERN Code Signing System



Table of Contents

- 3 / Introduction
- 4 / Code Signing Is Mandatory
- 5 / The Stakes Are High
- 6 / A Look Under The Hood
- 8 / Implementation Challenges
- 9 / Requirement #1: Protect Your Signing Keys
- 10 / Requirement #2: Keep The CI/CD Pipeline Moving
- 12 / Requirement #3: Manage Access To Signing Keys
- 14 / Requirement #4: Client Integrations
- 16 / Requirement #5: Plan For A Complex Infrastructure
- 17 / Requirement #6: Simplify Deployment
- 18 / Conclusion
- 19 / GaraSign: Securing & Simplifying Code Signing



// Introduction

Code signing is a critical part of secure software development and delivery. When properly designed, a code signing system ensures the integrity and authenticity of everything from applications to firmware and operating systems.

Because code signing relies on cryptographic processes, private signing keys are mission-critical assets and must be protected with the highest level of security. A compromised key can lead to wide-reaching breaches, making key security a top priority.

At the same time, in a DevOps environment, speed is paramount: code signing must not slow down the CI/CD pipeline. Additionally, a proper code signing solution must integrate with existing enterprise tools and processes for ease of deployment, use, and management.

Every enterprise is unique and must have a flexible code signing solution that meets its specific organizational needs. However, with CNSA 2.0 now guiding federal cryptography standards, software-producing organizations are expected to adopt Post-Quantum Cryptography (PQC) and support hybrid cryptographic schemes. For signing software and firmware under CNSA 2.0, compliant algorithms are preferred by 2025 and required by 2030.

This e-book provides key insights into designing an enterprise code signing system that is highly secure, exceptionally performant, and easy to deploy, manage, and use. While other documents provide high-level recommendations on what a code signing system should do, this book aims to provide detailed guidance on what features to implement, why those features are essential, and how to implement them.

// Code Signing Is Mandatory

Every Enterprise Must Deploy A Code Signing Solution

Some enterprise leaders may be tempted to think that establishing a rigorous and secure code signing system isn't essential or relevant to their organization. However, the reality is that every company that produces software must have a proper code signing solution in place. And, in today's digital world, virtually all companies produce software of some kind, whether it's enterprise applications for external customers, mobile applications for individual users, or internal tools, such as Excel macros and PowerShell scripts.

In some cases, signing code is not optional. For instance, major platforms like Apple App Store and Google Play require all mobile applications to be signed before the applications are made available to end-users for download. If an application is not signed, the platform will reject it. For organizations handling sensitive or regulated data, the bar is even higher. Under CNSA 2.0, software authenticity and cryptographic agility are foundational expectations. As enterprises prepare to adopt Post-Quantum Cryptography (PQC), securing the code signing pipeline is an essential first step.

To be clear, this is just one small example of the necessity of code signing. More significantly, signing code is necessary because it ensures the integrity and authenticity of the software. The inverse of this also holds true: without a secure code signing system in place, there is no scalable and reliable way to verify that software originates from a trusted source.

The following section will illustrate just how disastrous the repercussions of an insecure code signing system can be.

// The Stakes Are High

Improper Code Signing Can Have Severe Consequences

Production code signing keys are extremely sensitive assets. If attackers compromise a code signing key, they can sign malware to disguise their malicious code as legitimate software.

Attacks of this variety can have ramifications of the highest order, from costly downtime and increased insurance costs to reduced trust in the company's brand and even, in some cases, national security consequences.

For instance, the infamous Stuxnet worm, first discovered in 2010, sabotaged over 1,000 Iranian nuclear centrifuges in a sophisticated nation-state attack. Stuxnet managed to bypass anti-malware programs, firewalls, and other security controls because it had been signed with trusted code signing keys, which had been previously stolen from legitimate companies.

In another prominent example, attackers managed to spread malware to over 18,000 private businesses and government offices after injecting malicious code into a SolarWinds product called Orion. This extensive damage was only possible because the attackers had compromised the SolarWinds build process, ensuring that the malware was digitally signed with SolarWinds' production signing key.

While these two high-profile examples may seem like rare occurrences, there have been a number of major cyber attacks related to code signing in recent years. Several well-respected organizations, including [Adobe](#), [ASUS](#), and [Bit9](#), have fallen victim to code signing attacks.

// A Look Under The Hood

Technical Synopsis Of How Code Signing Works

With the importance of code signing abundantly clear, it's worth taking a moment to understand the technical underpinnings of code signing. Generally speaking, the code signing process is invoked by a user, either human or automated, and performed by a signing tool, as described in the four steps listed below.

1. An authorized end-user invokes a signing tool and designates which code to sign.
2. The signing tool reads the code and produces a cryptographic hash of the relevant portions.
3. The signing tool signs the hash with the signing private key.
4. The signing tool embeds the signature into the code.



Figure 1: Code Signing Sequence

While the various code signing tools work in roughly the same manner, they can be invoked from a few different locations for different purposes. A proper code signing solution should support all of these use cases.

Developer Build- A build signed by a developer from their workstation, typically triggered manually via a script, the developer's IDE, or some other local build process. The code that is signed comes directly from the developer's workstation.

DevOps - A build signed by a dedicated build server in a CI/CD pipeline. The code that is signed is pulled from the source code repository, typically any time a developer commits code changes.

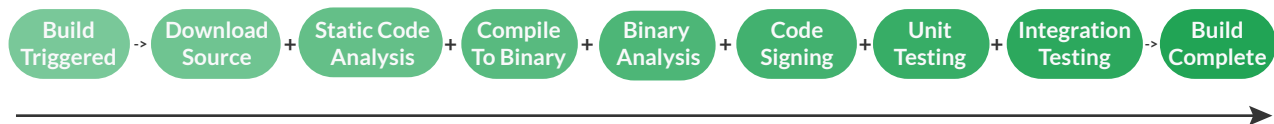


Figure 2: A Typical Build Process

Production Release - A build signed by a dedicated release team with the production code signing key. The code that is signed usually comes from a trusted location such as a dedicated artifact repository or a release branch (or tag) of the source code repository.



// Implementation Challenges

Deploying A Code Signing Solution At Scale Is Not Easy

If code signing is such a fundamental component of secure software development, why don't more companies have a proper solution in place? The answer is simple: designing and implementing a secure code signing system in an enterprise environment is a complex project. All of the following requirements must be met in order for a code signing solution to be useful.

Security

Code signing keys must be protected, but legitimate end-users must be able to use the keys.

Performance

An enterprise code signing solution must be fast and keep the CI/CD pipeline moving.

Access Management

Security leaders must be able to manage access to the signing keys from a single interface.

Integrations

End-users must be able to sign code from all the tools and platforms in their environment.

Flexible

An enterprise solution must function within different organizational units and adapt to new tools.

Deployment

A code signing system must deploy transparently and avoid disruptions to existing processes.

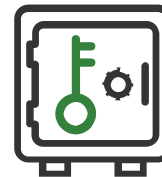
// Requirement #1: Protect Your Signing Keys

Secure All Code Signing Keys With A Hardware Security Module

Cybersecurity authorities strongly recommend that all organizations secure their code signing keys in a hardware security module (HSM). For example, the [NIST white paper on code signing](#) states that the keys should be stored in an HSM.

To be more precise, all code signing keys should be generated, stored, and used while in a non-exportable state in an appropriately certified HSM (e.g., FIPS 140-3, FIPS 140-2, Common Criteria, etc.). This applies to all signing keys, not just the production keys.

Ideally, different HSMs (or at least different logically isolated slots of a single HSM) should be used for different environments (e.g., development, production, etc.). Any legacy keys currently stored in software should be imported to the HSM. After they have been safely imported and backed up, they should be securely deleted from the software. These imported keys should not have their certificates renewed once they expire. Instead, a new key that is generated from within the HSM should be used.



HSM

// Requirement #2: Keep The CI/CD Pipeline Moving

Ensure High Performance With Client-Side Hashing

Once the code signing keys are stored in an HSM, a natural question arises: what is the best way to get the data being signed (i.e., the binaries of the code) to the private key in order to generate the digital signature?

The easiest, and perhaps most common, approach is known as the file upload method. As the name suggests, this technique involves sending all the data to sign over the network to either the HSM or a server with direct access to the HSM, if one is in place. While this method is the most straightforward, it's also slow, inefficient, and potentially insecure, as allowing executable code to be uploaded to a server that has direct access to the HSM exposes unnecessary security risk.



Figure 3: File Upload Method of Code Signing

A better approach is to use a technique known as client-side hashing. Recall that signing data first requires producing a cryptographic hash of the data, which is then used as an input to the private key operation that generates the signature.

Since the private key isn't needed until after the hash is produced, the hash can be generated client-side before any data is sent over the network to the HSM. Hash values are always fixed in length and of a relatively small value, so this approach minimizes the data sent over the network, even when the file being signed is large. The result is significantly faster signing and a reduced attack surface in comparison to the file upload approach.



Figure 4: Client-Side Hashing Method of Code Signing

// Requirement #3: Manage Access To Signing Keys

Enforce Strong Authentication & Authorization

While the HSM protects the keys from being copied, exposed, or otherwise stolen, the code signing system must also ensure that only authorized users can use the keys. Furthermore, those authorized users must only be able to use the keys at appropriate times to sign legitimate code. Since different keys have different sensitivity levels, the code signing system must be able to enforce different security controls based on the user, key, and operation being requested.

Naturally, all users should be strongly authenticated before being allowed to perform any operation. Since code signing can be done by both humans and automated processes, a variety of authentication mechanisms must be supported. Once authenticated, users should only have access to the minimum set of keys and permissions they require in order to perform their duties. Each key should have a policy that defines which security controls are required when the key is being used. At a minimum, a proper code signing system should support the following controls:

Multi-Factor Authentication - Strongly authenticate end-users beyond first form factors with techniques such as FIDO2/WebAuthn, TOTP/HOTP, etc.

Device Authentication - Authenticate the device the user is using via techniques that make use of secure enclaves such as the device's Trusted Platform Module (TPM).

Just-In-Time Access - Activate and disable keys and users, as needed, to restrict when authorized end-users can use the keys to which they have been granted access.

Approval Workflows - Require a quorum of approvers or even multiple tiers of approvers, each having their own quorum size, to approve the signing request before the signature is generated.

Notifications - Send notifications via email (or other enterprise communications systems) when important events occur, such as administrative changes, use of sensitive signing keys, etc.

With these capabilities available, enterprises can apply different policies to different keys, based on their sensitivity. For example, the non-production keys may only require the user to be authenticated and authorized to use the key, whereas the production key may additionally require device and multi-factor authentication as well as a quorum of approvers. The production key may further be kept in a disabled state and only activated once per quarter when official releases must be signed.

While HSMs provide strong security, they don't independently provide all the features listed above. Therefore, it is necessary to place a signing server in front of the HSM that proxies all the signing requests. This dedicated signing server is responsible for implementing all of the functionality listed above and only allowing offloading the signing request to the HSM once all the appropriate checks have been successfully performed.

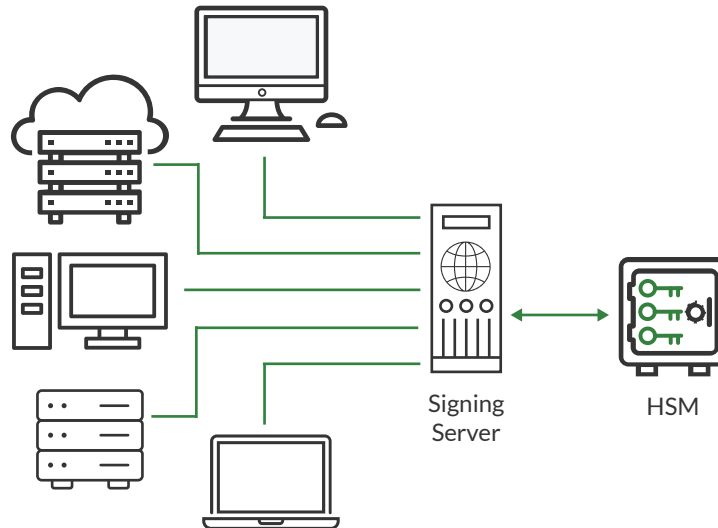


Figure 5: Signing Server Architecture



// Requirement #4: Client Integrations

Prefer Native Client Integrations Over Custom Tools

As you will recall, the first step to sign code is to invoke a signing tool. Organizations have two choices of signing tools to use: Commercial Off The Shelf (COTS) tools or homegrown custom tools. Organizations should always prefer COTS tools, as they are kept up to date with the signature processing requirements by the companies that produce those tools.

Since signing tools are not required to follow any open specification or set of requirements, customers that choose to use homegrown tools are left to reverse engineer signature outputs of COTS tools to try and update their own tools. This is a costly and error-prone process that should be avoided whenever possible.

So, how do you get third-party signing tools to use client-side hashing, support advanced security controls, integrate with external identity providers, all while signing with keys in an HSM? With the help of properly implemented cryptographic service providers.

Most signing tools offload signing (and other related functions) to a cryptographic engine. By default, these signing tools use an engine provided by the device's operating system, which usually assumes the signing key is local to the device running the signing tool. However, it is possible to load custom cryptographic engines (also called "cryptographic service providers," or CSP for short) and instruct the signing tool to use one of these engines for signing, rather than the operating system's default engine. Custom cryptographic engines can integrate with the server-side component of the code signing solution for authentication, signing, and other purposes.

When adopting this approach, it is necessary to create a cryptographic service provider that implements the appropriate APIs for each platform that the enterprise supports. The table below shows some of the cryptographic APIs for commonly used platforms.

| Windows | Java | macOS / iOS | Linux |
|------------------------------------|------------------------------------------------------|----------------|------------------------------|
| Cryptography Next Generation (CNG) | Java Cryptography Architecture / Extension (JCA/JCE) | CryptoTokenKit | PKCS#11, GPG, OpenSSL Engine |

Figure 6: Cryptographic Service Providers For Major Platforms

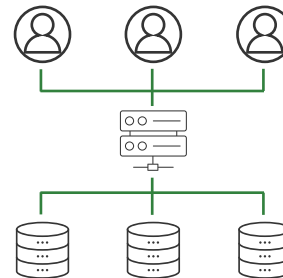
// Requirement #5: Plan For A Complex Infrastructure

Enable Multi-Tenancy & Centralized Management

Enterprise organizations are large and complex. Some development teams will have their own code signing keys, which only they use. Other teams might share signing keys. In order to allow independent teams to self-govern in a consistent way, an effective code signing system should be multi-tenant, preferably with subdomains and hierarchical permissions.

At the same time, enterprises often rely on a hybrid infrastructure consisting of legacy data centers, private clouds, and public clouds, often with multiple cloud providers used for different applications. This complexity must be taken into account as the code signing solution is designed and deployed, with the objective of centralized management in mind.

Centralized management is essential for an enterprise code signing solution to remain secure. If the signing keys are not centrally secured, the enterprise may lose visibility of the keys and therefore jeopardize the ability to audit key usage. In addition to major security risks, this may present compliance challenges.



// Requirement #6: Simplify Deployment

Integrate Seamlessly And Transparently

In order to gain widespread adoption, a code signing solution must deploy and be maintained in a manner that causes the least disruption to existing tools and processes, requires minimal training, and fields few, if any, support calls.

Several of the previous sections described features that help achieve this outcome, such as native client integrations, which ensure that existing tools and processes continue to work, and client-side hashing, which ensures that code signing won't cause a performance bottleneck.

In order to simplify deployment, a proper code signing solution should additionally support the following features:

Single-Sign On (SSO) - Authentication should integrate with existing enterprise infrastructure such as Active Directory or third-party identity providers. By using protocols like Security Assertion Markup Language (SAML), OpenID Connect (OIDC), and Kerberos, users can authenticate with their existing identities and authorize with their current group memberships.

SIEM Integration - Events that occur within the code signing system should be easily visible to the organization. A common way to achieve this is to integrate with the enterprise's Security Information and Event Management (SIEM) system, often by forwarding log entries to the enterprise log management platform.

Self-Service - Large enterprises with different domains are difficult to manage with a single dedicated team. Many enterprises prefer to allow some functions to be administered in a self-service model so as to avoid administrative bottlenecks. A proper code signing system should be flexible enough to allow for self-service where needed but also allow for top-down administration as required.

Enterprise Notifications - While integration with the enterprise SIEM system is necessary, it alone isn't sufficient for all notification requirements. Some notifications must be sent and processed immediately, such as those used for approval workflows. These notifications should be sent via the enterprise notification system. In many cases this is simply email, but in some cases it can be an enterprise chat system such as Slack or Microsoft Teams.

// Conclusion

Protect Your Enterprise With Secure Code Signing

Deploying an enterprise code signing solution is not an easy process. Code signing keys must be secured in a hardware security module (HSM), but doing so may present access, integration, and performance challenges. Additionally, advanced attackers with plenty of resources are motivated to compromise code signing systems and remain undetected.

The HID code signing solution is a SaaS solution supporting cryptographic operations that dramatically simplify the deployment of a secure code signing solution. Architecturally, our SaaS service sits between customer-managed signing clients and the service HSM. The code signing service restricts clients to proxied key access while the keys remain secured and non-exportable in the HSM.

On the client side, the code signing service integrates with all major platforms and tools in your environment, from Microsoft, Apple, and Linux, to GPG, RPM, and OpenSSL.

This makes deploying GaraSign a swift and painless process.

// GaraSign: Securing & Simplifying Code Signing

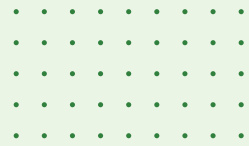
Use GaraSign To Simplify Deployment Of A Code Signing System

Deploying an enterprise code signing solution is not an easy process. Code signing keys must be secured in a hardware security module (HSM), but doing so may present access, integration, and performance challenges. Additionally, advanced attackers with plenty of resources are motivated to compromise code signing systems and remain undetected.

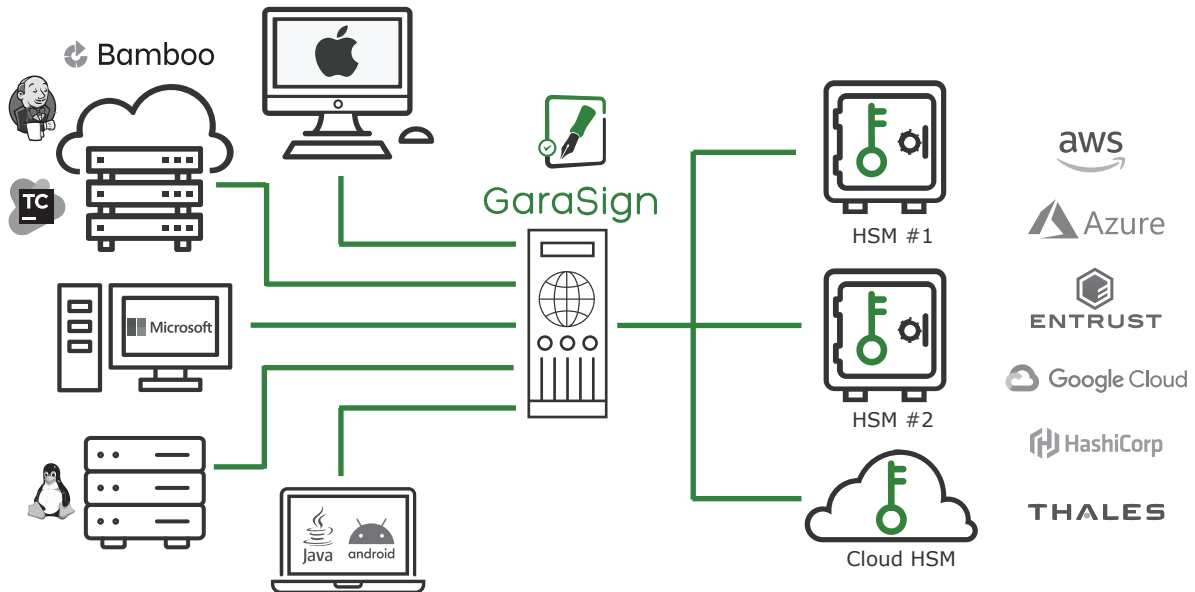
GaraSign is a platform for cryptographic operations that dramatically simplifies the deployment of a secure code signing system. Deployed on customer-managed infrastructure between signing clients and your HSM, GaraSign restricts clients to proxied key access while the keys remain secured and non-exportable in the HSM.

Server-side, GaraSign integrates with multiple HSMs and key managers, including on-premise and cloud-hosted ones, and can even support multiple different cryptographic tokens simultaneously. On the client-side, GaraSign integrates with all of the platforms and tools in your environment, from Microsoft, Apple, and Linux, to GPG, RPM, and OpenSSL, and more.

This makes deploying GaraSign a swift and painless process.



GaraSign implements all the functionality described in this ebook, including client-side hashing, broad client integrations, advanced security controls, automated hash validation, and more. To learn more, [watch this series of demo videos showcasing GaraSign's code signing capabilities](#)





Powered by  Garantir

The text "Powered by" is in a dark grey font. To its right is the Garantir logo, which features a stylized green padlock with a white checkmark inside a small circle on its left side. To the right of the logo is the word "Garantir" in a green, sans-serif font.

+1 (512) 776-9000
hidglobal.com