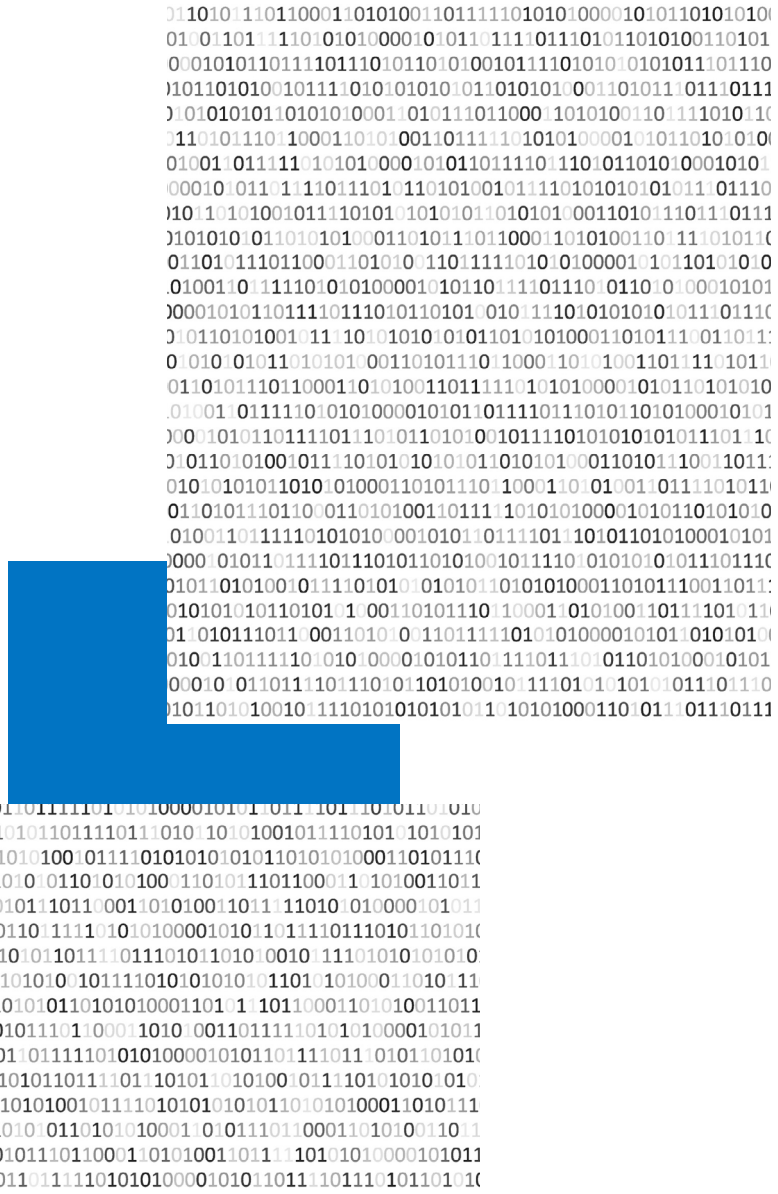
POLICY GUIDE

A GUIDE TO AN EFFECTIVE CODE AND SOFTWARE SIGNING POLICY



digicert®

SOFTWARE AND CODE SIGNING: THE IMPORTANCE OF A STRONG, DOCUMENTED SIGNING POLICY

Code signing is a critical step in the process of building and protecting software—but many developer and engineering teams struggle with implementing consistent signing practices, largely because there's a misconception that signing that interferes with the time-to-market goals of agile development practices. Additionally, our research suggests many software development teams operate on generalized guidelines and implied common practices rather than a comprehensive, documented policy governing signing processes.

As the need for mature signing requirements increases, organizations and development teams must craft and operate robust signing policies and processes to protect software. By creating and documenting a formal policy for signing code, you enable your developers and engineers to follow the best practices necessary for securing the software you build.

This guide will help you formulate a strategy that combines software, process, and policy to achieve end-to-end security in your signing practices. To create this guide, we surveyed and interviewed dozens of DevOps experts, team leads, and software security professionals. Hailing from organizations of different sizes, industries, and geographies, these experts helped us build a useful signing baseline that can be applied to virtually any software development team in any organization.

How to use this guide

Part I introduces key concepts you can use to create the foundation of a signing policy based on the needs of your team. The focus here is developing an understanding of how code signing helps you accomplish what's best for the software your team creates.

Part II outlines the creation of a written policy, including practices and procedures considered critical by the leading software security and DevOps experts we consulted.

At the end of the document, you will find a list you can use to track the creation of your own policy.

PART I

HOW TO THINK ABOUT GOOD SOFTWARE SECURITY

Development teams need a clear vision of their goal

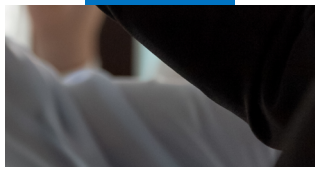
What is the mission of your software development team?

The answer to this question may not be as obvious as you think. Too often, the mission of the organization or other teams and business units stand in for the team mission, and while the team mission should serve the mission of the organization, they are not interchangeable. Just as finance cannot operate at its best using the principles and guidelines of human resources, and human resources cannot operate using a sales system, the software development team cannot operate at its best under the principles and guidelines of other teams or the broad umbrella of the entire organization.

A software development team is best served by working from and toward a strong, clear mission statement that defines how the team operates so everyone does the right thing for the software—especially when it comes to security. When the mission is defined using the parameters and function of the software development team, this drives a shift from the reactive work of meeting short-term business or sales goals to the proactive work of meeting long-term goals.

As you read through this guide, think about your own software development process. If you haven't already, write your own software development team mission statement. As you think and write, remember, a strong mission statement does not define how the team works or what the organization or customer can expect as residual benefits. Instead, a strong mission statement defines what is best for the software you're building and why it matters.

A good mission statement should be simple, memorable, and jargon-free. A good rule to follow: avoid the use of commas or lists.

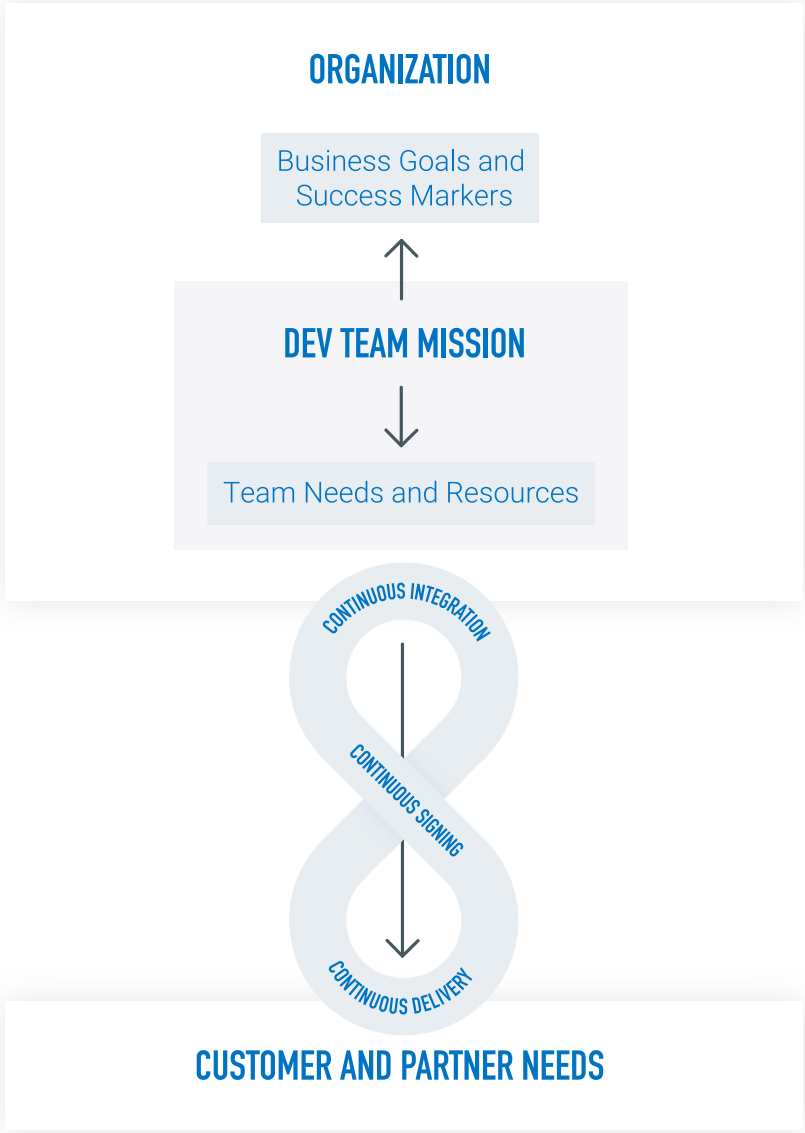


Security grows out of a strong mission

Security, and especially signing, in software development has traditionally been slow and laborious. The extra steps involved in properly securing code and software have, countless times, interfered with meeting short-term business goals. Developers and engineers want what's right for their software, but signing—unless mandated by a platform or software supply chain—falls under the definition of a proactive, long-term goal. Oftentimes, software development, particularly agile development, is organized along lines that can lead to sacrificing long term goals in favor of shorter, immediate needs and market response.

That's where a strong software development mission statement makes the difference. A strong mission statement understands all the steps and resources needed in service of completing the software development mission. This mission statement works inward to meet the needs of the development team. It also works up the organizational structure to meet business goals and success markers. And finally, it works outward to meet the needs of partners and customers using the software.

With a strong software development mission statement, signing and all other forms of security become an integral part of a proactive, long-term development process that allows developers and engineers to meet short-term business objectives by delivering signed code with integrity on time. Any policy guide you write should stand on the foundation of your software development mission statement.



UNDERSTANDING CULTURE AND ROLES

When it comes to software signing policy, your guide should be more than a rulebook. The best guides explain how software signing helps development teams do what's right for the software they build. In the context we've begun outlining already, a strong software signing policy helps every member of the team understand how to correctly use signing to meet goals and accomplish the team's mission.

To achieve this, it's important to understand the relationship between roles on the team, and how these roles define the communication and implementation of the signing policy. When the policy is clear, and those using the policy understand how it benefits them and the overall effort of the software development process, we find that code security becomes part of the culture of a team, rather than a step in a checklist, or a point of tension between managers and developers or engineers.

Start with these three tenets:

1

Rules are for administrators, processes are for developers and engineers

It's important to have a written policy defining when and how signing should be completed, but these rules should be foundational. Administrators and managers must ensure that developers and engineers operate based on policy processes that speak to how they use signing in their daily work.

2

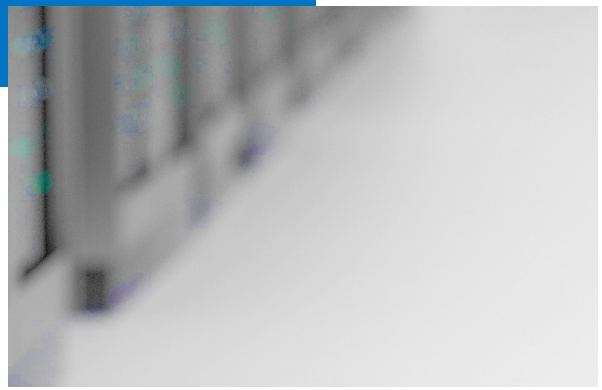
Signing should be enabled, not simply mandated

Most people want to do what's right most of the time. Developers and engineers want to do what's right for the code they create. Your software signing policy shouldn't operate as a policing tool for enforcing security rules or compliance mandates. Instead, it should make it easy for developers and engineers to protect and stand behind their hard work. Signing policies should enable good security practices.

3

Communication is crucial

A policy that is confusing, opaque, unrealistic, or punitive interferes with accomplishing the mission. Make sure your software signing policy is clear, logical, easy to understand and implement, and that everyone using the policy has regular refreshes on how to complete signing as expected. And remember—communication goes two ways. An effective policy should be reviewed and updated regularly using feedback from everyone who puts the policy into action.



RECOGNIZING POSSIBLE POINTS OF FAILURE

Humans are good at simplification. When something goes wrong, we tend to look for a single culprit—one unnoticed error, one person's mistake, one moment of negligence, or simply a single aberration. The truth, though, is that failure is often a complex chain—or even a web—of factors leading to a tipping point.

While today's news is full of headlines talking about software supply chain security breaches caused by one lazy employee or one malicious hacker, the truth is that vulnerabilities in software development are complex and all too common. This complexity has recently grown as more and more software is developed and deployed in more places. Without a solution for complexity, the system multiplies vulnerabilities until it reaches a point where individual errors are so numerous, finding and exploiting one is simply an inevitable matter of time.

If we continue to operate on the idea that negligence, laziness, or the rare human error is responsible for security failures, we guarantee the number and scope of software attacks will increase. While it's true that sometimes something just goes wrong, most of the vulnerabilities in software security can be traced back to systemic failures in policies and processes, and those failures are usually the result of bad information, unrealistic expectations, oversight, and cumbersome or nonexistent tools.

Security failures are not typically the result of simple human error. They are the result of a systemic failure to provide software development teams with the tools and guidelines they need to do what's right for the software they build.

GOOD SECURITY STARTS AND ENDS WITH GOOD SECURITY CULTURE

A strong software signing policy is the written definition of good security culture. A culture that empowers the software development team to do what's right for the software focuses on a strong software development mission and collaborative processes and enablement rather than top-down rules and enforcement. It also considers security as systemic and facilitates good security practices with excellent tools.

WEAK CULTURE

- Short-term, reactive posture
- Top-down
- Enforcement mindset
- Unrealistic individual responsibility
- Manual tools

STRONG CULTURE

- Mission-based, proactive posture
- Reciprocal
- Enablement mindset
- Systemic support
- Automated tools

"Security is key to every individual within a company. Use curiosity and create a deep desire of non-stop learning to create a culture of security."

Jason Sabin, Chief Technology Officer, DigiCert

PART II

WRITING A SOFTWARE SIGNING POLICY

Gathering expertise

By this point, you've had a chance to think about how you want to define the parameters of your software signing policy and good security culture. You have a vision in mind, and you've written a clear team mission statement. Now it's time to fill in the specifics and write a policy that you can communicate to your team. The next step involves determining how your team, with its unique mission, processes, needs, and tools, fits into the larger organization.

In our work with software developers around the world, in hundreds of industries, we've found that software signing policy is often informed by other teams and divisions within an organization. When it comes to writing a software signing policy, these are some of the most reported input sources:

- CISO
- SecOps/InfoSec
- Risk Management
- Product Management
- Compliance
- Customers and partners
- QA and Site Reliability

When consulting with resources, it's helpful to consider the following:

1. **Inclusion**
What can they contribute that will help you achieve a proactive, and mature, security posture when it comes to signing? Which of their policies and processes help you accomplish your mission?

2. **Reciprocity**
How do your policies and processes fit with their policies and processes? How do you work together while ensuring your signing policy and process is not compromised on the way to achieving your mission?

3. **Action**
How can you enable your resources so they provide input that's actionable and timely? Which inputs are vital, and which are impractical or purely informational?

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 write a signing policy that shrinks the threat landscape and better protects your code.

FOUNDATIONS OF SOFTWARE SIGNING USAGE

Individual signing system configurations and team structures may vary between organizations, but in our research, development teams reported several common components that make for a strong software signing policy, regardless of team size, type of software, or the industry. These components act as the foundation of signing best practices.

Key issuance

While this is a function of key management, our research shows this is the most important concern for DevOps professionals. When establishing management protocols, set controls and procedures for who can issue keys and when. Team members should know when it is appropriate to issue keys, according to their role. You'll also want to manage which types of keys are issued, as well as the attributes associated with them. This helps avoid issuing new keys with weak algorithms that introduce attack vectors.

Key management

Set controls and procedures for user roles. Who manages keys? How is key management divided according to role within the team, organization, or stage of production? These procedures should include location tracking for all keys throughout the organization.

Key storage

Keys must be protected. Set controls and procedures for key storage while in use and not in use. This should include HSMs, tokens, USBs, and key access using multi-factor authentication. Team members must know how to avoid losing or improperly storing keys.

Signing permissions

Team members should know who has permission to sign, and when it is appropriate to sign. They should also know when signing permission is not granted and how to request signing if permission is not a part of their role.

Key usage

How keys are used, or not used, is a crucial part of correct signing procedures. Keys should be used at the right time by the right people

Preventing key sharing

Key sharing is a common practice. It is also one of the most dangerous practices. Team members should never share keys, even inside repositories or internal servers, systems, and networks. Keys should be uniquely issued, controlled, and stored by the appropriate individual, according to their role.

Continuous Signing

Code and software signing should never be treated like an afterthought or a tedious compliance step. Every CI/CD pipeline should include CS—Continuous Signing—so code security is practiced correctly and consistently.

DON'T OVERLOOK INTERNAL SIGNING

There's a truth in software security: best practices and platform enforcement tend to go hand-in-hand. We see more consistent code signing when software is headed for deployment in app stores, operating systems, and regulated browsers. If software requires signing to run, that code gets signed. Yet, these requirements are not esoteric or arbitrary. Signing really does 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.

So, why does software signing quickly turn into a rarely exercised option when it's not required for deployment? If we know that signing, as a best practice, is a real-world safety measure, shouldn't software be signed even when it's not bound for a platform with signing enforcement?

Internal publishing's false sense of security

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, though, 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.

Solve by automating

If one truth in software security is that enforcement drives signing, the other truth is that slow, labor-intensive security steps are antithetical to the DevOps philosophy. Given the amount of protection surrounding the average organization's internal systems, it makes sense that developers and engineers would skip impeding steps when it seems like the risk is low. While not as damaging as hacking or tampering, the slow and laborious nature of manual code signing is often an impairment itself.

The solution is automation, coupled with accountability. Systems that take advantage of Continuous Signing facilitate the processes that protect your software, no matter where it's built or deployed, and that makes it easier for developers to sign the right way, every time.

Modern, managed software signing tools make secure signing easier. "These tools let you add signing into your pipeline with no overhead, so you gain another level of security. It's an example of belts-and-suspenders thinking in action."

Wade Choules, VP of Engineering at DigiCert

EDUCATION, TRAINING AND FEEDBACK

Now that you've created your software signing policy, you need a system that trains and refreshes your developers and engineers. This straightforward curriculum will serve as a template for teaching your team how to practice secure software signing.

Training

These components should be a part of your initial training system:

- Software development team vision and mission
- Software security stakeholders
- An introduction to your policies
- Why is signing as important as any other step in the development process?
- Signing roles within the organization
- Central tenets of the signing process
- Who, when, and how to sign
- Proper key usage and key attributes
- When to time stamp
- Key security and management
- How to report a problem

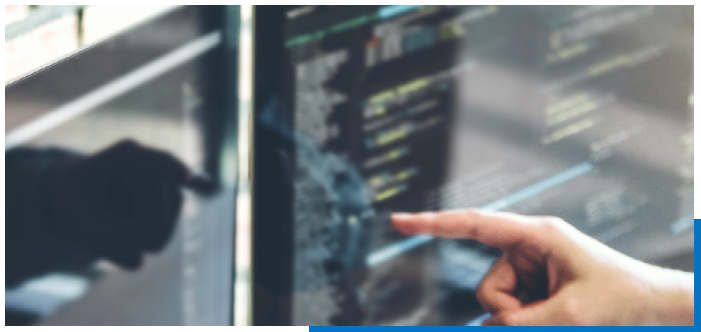
In addition to these components, make sure your training system includes a check for understanding. It's also a good idea to offer new team members a space to offer feedback or ask questions that may not be covered by the training. Use feedback and questions to build a more robust version of the training module.

Education

Ongoing education helps reinforce knowledge and deepen understanding. It's a good idea to offer regular policy and process reviews. What's more, it's helpful to place context around the value of steps in their process. Consider the following:

Regular reviews of signing policy and process, with a venue for feedback and concerns from team members. This can be an assigned training module, but it can also be a hosted meeting with a Q&A, or another space for discussion and interaction.

Occasional updates and reminders regarding the real-world importance of software signing. Discussion forums regarding software security. A dedicated space (like an email account or private slack channel) for suggestions or concerns amongst team members related to software security.



COMMUNITY IS CRUCIAL

The very nature of security can make it feel like sharing information opens us to attack. But as the supply chain grows and complexity increases, collaboration and information sharing is only becoming more vital to software security the world around.

With attacks increasing in number and scale, it's important that software developers and security professionals share best practices and compare notes on what works when it comes to software signing policies. Individual security configurations and intellectual property should always be kept private, but any non-compromising information that safeguards your software may also safeguard someone else in the supply chain, and vice versa.

Once you've developed and put into practice your own software signing policy guide, consider connecting with other professionals in your network to discuss how you've enabled stronger security through your own policies and processes. Listen to what they are doing and inspire adoption of stronger security postures throughout your community. No single organization can afford not to take a proactive approach to security. Every team in every organization must protect its code so the entire chain is secure.

WRITING A SOFTWARE SIGNING POLICY

1 **Create a software development team mission statement**

Establish your vision. Your mission statement should define what is best for the software and why it matters.

2 **Consider how roles play a part in defining and implementing your policy**

Signing policies should be built for reciprocity and collaboration, and they should be clear and easy to understand and practice.

3 **Map out the role of signing in a systemic landscape of good security practice**

Make sure Continuous Signing is enabled, not enforced, throughout the entire CI/CD pipeline. The system should make signing best practices a simple, integrated part of the software development process.

4 **Gather input from internal and external sources**

Use the expertise and needs of internal stakeholders and your customers to help you shape a signing practice that enables mature security capabilities in excellent software delivered on time. Consider the forensics of failure as a guide for preventing future security mistakes.

5 **Document the availability and use of resources and tools**

Audit your signing processes and the tools developers and engineers use to sign. Where possible, consider implementing automation to expedite signing and mitigate supply chain attacks.

6 **Define the use and practice of signing and signing components (external and internal)**

Consider the daily build process and how signing code fits. Key issuance, management, usage, storage, and roles. Be sure your team members know they should never share keys.

7 **Gather feedback from administrators, managers, developers, and engineers**

Before publishing and deploying your policy, check with the people who will use the policy as a guide. Incorporate feedback, so policies align with the daily software build requirements of each stakeholder.

8 **Develop an education and training program with regular reviews**

Onboarding should include training on software signing policy and use, and team members should regularly review signing procedures to ensure security remains strong.

9 **Regularly review your mission and policy guide, making updates and changes as needed**

Organizational goals change, projects shift, teams grow, shrink, multiply. Regular reviews of your policy—which should include feedback from developers and engineers—will ensure that changes are incorporated so policies and procedures remain current, clear, and useful.

10 **Look for opportunities to strengthen your security and the security of others in the supply chain through information sharing and collaboration.**

When one organization's signing capability matures, the entire world of software can become more secure. Sharing ideas about implementing best practices helps protect organizations, the supply chain, and end users.

DigiCert® Software Trust Manager delivers Continuous Signing that can be automated, end-to-end for your CI/CD pipeline. It delivers tracking, auditing, and strong but simple centralized key management. And thanks to our flexible architecture and APIs, DigiCert Software Trust Manager easily integrates with any development environment—including yours.

If you're interested in learning more about automating your code and software signing, contact us to speak with a technical consultant.
PKI_Info@digicert.com

DIGICERT AND THE OS OF TRUST

At DigiCert, building a better way to secure the connected world is the single-minded pursuit that goes all the way back to our roots. We built the OS of Trust so that managing TLS/SSL, identities, servers, networks, keys, code, signatures, documents and IoT devices easier and more reliable. We've earned the trust of 88% of the Fortune 500, 97 of the 100 top global banks, and for 81% of all global e-commerce, all backed by the most five-star service and support reviews in the industry. You're using PKI every day. Shouldn't you have an operating system to make it work harder for you?