

Why Your Team Needs an API for Code Signing in 2025

The Shift From Local to Remote Signing

If you have managed code signing for a team spread across multiple offices or time zones, you know the friction. The USB token lives on one desk. The private key never leaves the hardware, which is exactly how it should be. But the developer who needs to sign a build at 2 AM has to wait until someone with physical access to that token is awake. This is the problem remote signing solves, and at the heart of that solution is a well-designed api for code signing.

I have spent years working with PKI and public key infrastructure across certificate authorities such as DigiCert, GlobalSign, and Sectigo. The one constant pain point has always been key mobility. You want the security of a hardware security module or a USB token, but you also need your signing operations to happen on demand from any environment. The fix is not to move the private key from the hardware. The fix is to expose the signing operation through an api for code signing that your build pipeline can call from anywhere.

What a Code Signing API Actually Does

Think of it as a thin wrapper around your existing signing hardware. The [api for code signing](#) accepts a hash of your code or PDF, sends it to the hardware token or HSM where the private key lives, and returns the signature. The key never leaves the device. This is the same model that cloud providers like AWS CloudHSM and Azure Key Vault use for key management, but applied to the specific needs of code signing and PDF signing.

For example, a typical workflow looks like this: your CI server builds the application, then calls the remote signing API with a digest of the binary. The API authenticates the request using a token or client certificate, forwards the digest to a connected YubiKey or SafeNet eToken plugged into a signing server, and returns the detached signature. The build server attaches the signature, and the signed artifact is ready for distribution. No one touches the private key. No one has to be in the office.

Why USB Tokens Still Matter

There is a persistent myth that moving signing to the cloud requires abandoning USB tokens. That is not true. Many teams have existing investments in hardware from Thales, SafeNet, or Yubico. They have been told that to enable remote signing they must migrate to a cloud HSM and reissue all their keys. That advice misses a simpler option: keep the USB token, but plug it

into a shared remote signing appliance. The token stays in a secure location, and your team calls an API to use it.

This is exactly the approach that a platform like onesign.sg takes. It lets you keep your existing USB tokens and private keys where they are, while giving your developers, contractors, and release engineers access to signing on demand. The signing operation happens on the hardware. The API simply mediates the request.

Comparing the Options

There are several ways to implement remote code signing. Each has trade-offs:

- **Cloud HSMs** (AWS CloudHSM, Azure Key Vault) offer high scalability and FIPS 140-2 Level 3 certification, but they lock you into a specific cloud provider and often require you to generate new keys inside their infrastructure. You cannot bring your own USB token.
- **Local signing servers with hardware tokens** give you full control. You keep your existing keys and hardware. The downside is you need to manage the server, the network, and the physical security of the token.
- **Dedicated remote signing platforms** (like onesign.sg) combine the best of both worlds. They provide the API layer and secure access controls, but you keep ownership of the token and the private key. The token stays in your office or a colocation facility.

For most teams, the third option is the sweet spot. You get the compliance and audit trail of a centralised signing service without surrendering control of your key material.

Integrating With Your Existing PKI

If your organisation already runs a public key infrastructure, you understand the importance of certificate lifecycle management. A code signing API should integrate with your existing CA workflows. Whether you use DigiCert, GlobalSign, or Sectigo to issue your code signing certificates, the API should accept any certificate that your hardware token holds. It should also log every signing request so you have a clear audit trail for compliance and revocation scenarios.

One detail that often gets overlooked is timestamping. When you sign code, you usually include a timestamp from a trusted authority. The API should handle that automatically, appending a signed timestamp token (TSA) so the signature remains valid even after the certificate expires. Without this, your signed software may stop verifying once the signing certificate is revoked or expires.

PDF Signing Is Not Code Signing, But They Converge

Many teams need to sign both software and PDF documents. The security requirements are similar: protect the integrity of the content, prove the signer identity, and prevent repudiation. The technical implementations differ, however. Code signing uses formats like Microsoft Authenticode or macOS signing, while PDF signing uses PAdES or CAdES standards. A good remote signing platform supports both through the same API, using the same hardware token. This is where platforms like eSign and Adobe Sign come into play on the document side, but for organisations that need both, a unified API saves significant operational overhead.

one signer
Sign Anywhere with Your USB Tokens

Your USB tokens, signing *remotely.*

Turn the tokens you already own into remote signing for PDFs and Windows code. Tokens stay in one Windows box; sign from the browser, REST API, watched folders or CI. Keys never leave the token.

[PAdES](#) [Authenticode](#) [YubiKey · SafeNet · ePass2003](#) [REST API](#)

onesign.sg

Digitally signed by Tan Wei Ming
PAdES · certificate on a USB token in your box · signed from a browser, anywhere

I have seen teams maintain two separate signing solutions: one for code, one for PDFs. It creates duplicate audit trails, separate token management, and twice the operational risk. A single api for code signing that also handles PDF signing is a cleaner architecture. You manage one set of tokens, one set of access controls, and one audit log.

Security Considerations Beyond the Key

Keeping the private key on a hardware security module or USB token is table stakes. The harder part is controlling who can call the signing API and under what conditions. You need strong authentication for API calls, ideally mutual TLS or OAuth with short-lived tokens. You also need rate limiting and anomaly detection. If a build pipeline suddenly requests hundreds of signatures in a minute, that should trigger an alert.

Another layer is the separation of duties. The person who manages the signing hardware should not be the same person who approves signing requests. The API should support

approval workflows where a designated reviewer must approve a signing request before the token actually signs. This is especially important for production builds and release artifacts.

Lessons From Real Deployments

I worked with a fintech company that used YubiKeys for code signing. They had five developers, each with their own key. When one developer left, they had to revoke and reissue that certificate, then redistribute the new key. With a remote signing API, they consolidated all signing into a single HSM-backed token in a secure rack. Each developer still had their own API key, but the private key never moved. Onboarding and offboarding became a matter of updating API credentials, not shipping hardware.

A different client, a software vendor with government contracts, needed to sign PDFs with qualified electronic signatures under EU regulation eIDAS. They were using Thales HSMs locally. The challenge was that their remote developers could not access the HSM from home offices. They set up a remote signing appliance that exposed the HSM through a REST API. The developers could sign PDFs from anywhere without the private key ever leaving the HSM. That deployment took two days to configure and saved weeks of shipping tokens back and forth.

Choosing the Right Approach

There is no one-size-fits-all answer. If you already have an investment in AWS CloudHSM or Azure Key Vault and your compliance team is comfortable with cloud key storage, that may be the right path. If you need to keep your key material in a hardware token on your premises, a remote signing platform that works with your existing USB tokens is the better fit. The key is to decouple the signing operation from the physical location of the token. An API is the abstraction that makes that possible.

Before you commit, ask these questions: Can the API handle both code signing and PDF signing? Does it integrate with your CI/CD pipeline? Can it log and monitor every signing request? Does it support the certificate authorities you use? And most importantly, can you keep your existing hardware without moving the private key?

Final Thoughts

Remote signing is not a futuristic concept. It is a practical necessity for any team that ships software from more than one location. The USB token on a desk is a bottleneck. An api for code signing that unlocks that token for remote use is the solution. It respects the security of your private key while giving your team the flexibility to sign from anywhere. That is the balance every modern development organisation needs to strike.