Qwick Cert Security White Paper: Digest Proxy Signing

Executive Summary

Qwick Cert is a developer-experience platform for Windows code signing (Authenticode) built on Microsoft's Azure Trusted Signing infrastructure. It eliminates the complexity of Azure portal configuration, credential management, and certificate lifecycle operations by wrapping them into a single CLI command and a web dashboard.

At the core of Qwick Cert's architecture is **Digest Proxy signing** — a model where private keys and Azure credentials never leave the server. Instead of distributing secrets to developer machines or CI runners, the CLI computes a SHA-256 hash of the binary locally and sends only the 32-byte digest to the Qwick Cert API. The server authenticates the request, signs the digest via Azure Trusted Signing's REST API, assembles a PKCS#7/CMS signature, and returns it. The CLI then embeds the signature and an RFC 3161 timestamp into the PE file. At no point do signing credentials exist outside the server.

This architecture provides a fundamentally stronger security posture than traditional code signing approaches — including direct Azure Trusted Signing integration — while delivering a dramatically simpler developer experience.

The Problem with Traditional Code Signing

Windows code signing has been a persistent pain point for software teams. The root issues span security, cost, and operational complexity:

Private Keys on Developer Machines

Traditional Authenticode signing requires a PFX file or hardware token (USB HSM) containing the private key. These long-lived credentials sit on developer machines, shared drives, or CI servers — often for months or years. A single compromised machine can lead to signed malware distributed under your organization's identity. Supply chain attacks like SolarWinds and Codecov have shown the devastating consequences of compromised signing infrastructure.

EV Certificate Management

Extended Validation (EV) certificates cost $300–700/year from traditional Certificate Authorities, require physical hardware tokens, and involve multi-day validation processes. Certificate renewal is manual and error-prone. When a team member leaves, revoking their access to the signing key is often impossible without rotating the entire certificate.

Shared Secrets in CI/CD

Automating code signing in CI/CD pipelines means storing signing credentials as pipeline secrets. These secrets are often long-lived, shared across projects, and accessible to anyone with repository access. Azure service principal secrets, PFX passwords, or API keys sit in GitHub Actions secrets, Azure DevOps variable groups, or Jenkins credential stores — each one a potential attack vector.

Supply Chain Risk

The combination of long-lived credentials, shared access, and limited audit trails creates a significant supply chain risk. Organizations often cannot answer basic questions: Who signed this binary? When? From what machine? Was the signing key compromised between version 2.1 and 2.3?

How Digest Proxy Signing Works

Digest Proxy signing separates the hash computation (which requires access to the binary) from the signing operation (which requires access to credentials). This separation is the key architectural insight that eliminates credential exposure.

Step-by-Step Flow

1. CLI Computes Authenticode Digest The developer runs `qwick sign ./app.exe`. The CLI reads the PE (Portable Executable) file and computes the Authenticode digest — a SHA-256 hash of the binary's signable content, following the PE/COFF specification. This computation happens entirely on the developer's machine. The result is a 32-byte hash value.

The Authenticode digest specifically excludes the checksum field, the certificate table entry, and the certificate data directory from the hash computation, ensuring the signature can be embedded into the binary without invalidating itself.

2. Hash Sent to Qwick Cert API The 32-byte SHA-256 hash is sent to the Qwick Cert API (`POST /api/v1/sign-digest`) over TLS. The request includes the hash, the digest algorithm identifier, and authentication credentials (API key or OAuth token). The actual binary never leaves the developer's machine.

3. Server Authenticates and Authorizes The Qwick Cert server validates the request: - Authenticates the caller (API key, OAuth token, or CLI session token) - Checks organization membership and signing permissions - Validates the request against any configured signing policies - Records the signing operation in the audit log with user identity, IP address, timestamp, and file hash

4. Server Signs via Azure Trusted Signing The server calls Azure Trusted Signing's REST API with the digest. Azure performs the cryptographic signing operation using the organization's certificate profile and returns the raw RSA signature along with the certificate chain. The server then assembles these components into a complete PKCS#7/CMS (Cryptographic Message Syntax) signature blob following the Authenticode specification.

This step uses the organization's Azure credentials, which are stored encrypted in Supabase Vault on the server. These credentials never leave the server environment.

5. Signed PKCS#7 Returned to CLI The assembled PKCS#7 signature blob is returned to the CLI over TLS. This blob contains the signature value, the signing certificate, the certificate chain, and authenticated attributes — everything needed to embed a valid Authenticode signature.

6. CLI Embeds Signature and Timestamp The CLI embeds the PKCS#7 signature into the PE file's certificate table (WIN_CERTIFICATE structure). It then obtains an RFC 3161 timestamp from Microsoft's timestamp authority (`timestamp.acs.microsoft.com`) and countersigns the signature. The timestamped signature ensures the binary remains valid even after the signing certificate expires (certificates issued by Azure Trusted Signing have a 72-hour validity window, rotated automatically).

Architecture Diagram

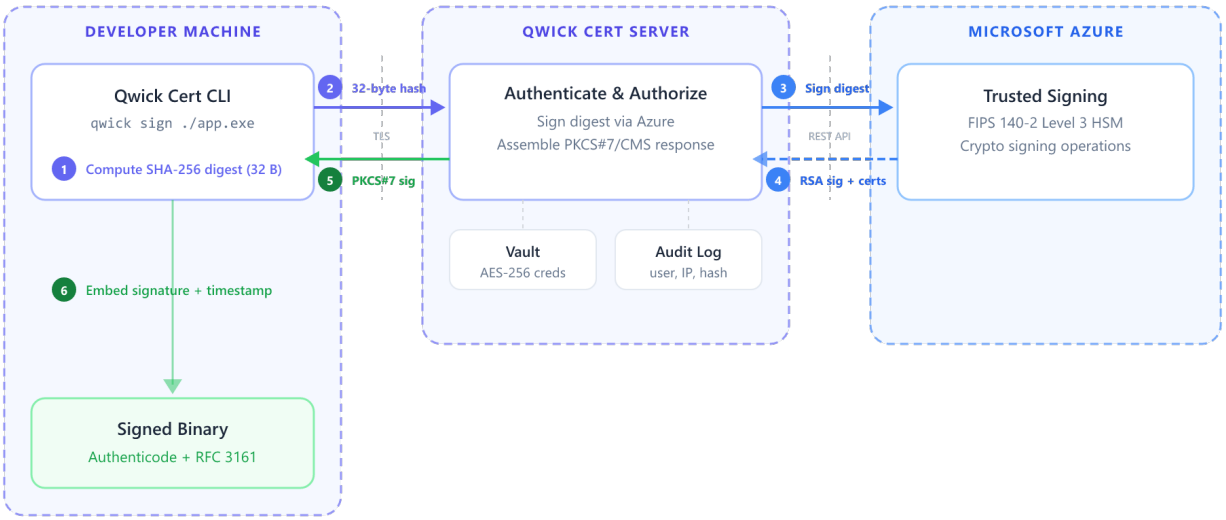

Figure 1: Digest Proxy signing architecture — six numbered steps showing the flow from CLI to server to Azure and back

Security Properties

Zero Credential Exposure

Azure service principal credentials, client secrets, and certificate material never leave the Qwick Cert server. Unlike direct Azure integration where developers or CI systems need `AZURE_CLIENT_ID`, `AZURE_TENANT_ID`, and `AZURE_CLIENT_SECRET` environment variables, the Digest Proxy model keeps all credentials server-side, encrypted at rest in Supabase Vault (AES-256).

A compromised developer machine or CI runner cannot be used to perform unauthorized signing operations because it has no access to Azure credentials. The worst an attacker can obtain is the Qwick Cert API key or OAuth token, which is scoped to the Qwick Cert platform and subject to rate limiting, signing policies, and full audit logging.

Minimal Data in Transit

Only the 32-byte SHA-256 digest travels between the developer machine and the server — not the binary, not credentials, not certificates. This minimizes the attack surface and bandwidth requirements. The digest alone reveals nothing about the binary's contents beyond its hash value, and is useless without the corresponding private key.

Server-Side Audit Trail

Every signing operation is recorded server-side with: - User identity (who initiated the signing) - Timestamp (when it happened) - Client IP address (where it came from) - File digest (what was signed) - Organization and project context - Certificate profile used - Success or failure status

This audit trail is tamper-resistant because it lives server-side, not on developer machines where logs can be modified or deleted.

No Client-Side Azure SDK Dependency

The CLI does not require the Azure SDK, `DefaultAzureCredential`, or any Azure-specific configuration on developer machines. This eliminates an entire category of configuration errors, SDK version conflicts, and credential management issues. The CLI is a pure HTTP client that speaks to the Qwick Cert API.

Rate Limiting and Signing Policies

The server enforces rate limits and signing policies before forwarding requests to Azure. Organizations can configure: - Maximum signatures per hour/day - Allowed file types - Required approval for production signing - IP allowlists - Time-of-day restrictions

These controls are impossible to implement with direct Azure Trusted Signing integration, where the developer has direct access to the signing API.

Qwick Cert vs. Native Azure Setup

	Native Azure Trusted Signing	Qwick Cert (Digest Proxy)
Setup	2–8 hours (Portal, RBAC, app reg, Key Vault)	5 minutes (setup wizard)
Dev creds	Yes — service principal secrets required	No — only Qwick Cert API key
CI/CD creds	Yes — `AZURE_CLIENT_*` env vars	API key only (no Azure secrets)
Cred lifetime	Months or years (manually managed)	N/A — creds never leave server
Access control	Azure RBAC (complex)	Simple roles: admin, signer, viewer
Audit trail	Azure Monitor (extra config and cost)	Built-in, per-signature, included
CI/CD	Custom scripting per provider	One-line CLI or GitHub Action
Policies	Not available	Built-in rules engine
Batch signing	Manual scripting per file	Glob patterns + parallel workers
Timestamping	Manual configuration	Automatic (RFC 3161)
Cert mgmt	Azure-managed (you configure profiles)	Fully managed (zero configuration)
SDK deps	Azure SDK + signing DLib on every machine	None (pure HTTP client)
Cost	$9.99/mo (Azure) + engineering time	$9.99/mo (Azure) + Qwick Cert subscription (see pricing)

Compliance and Trust

Authenticode Standard Compliance

All signatures produced through Qwick Cert are standard Authenticode signatures, indistinguishable from those produced by `signtool.exe` or any other Authenticode-compatible signing tool. They use the same PE/COFF signature format, the same PKCS#7/CMS structure, and the same certificate chain validation that Windows has used since Windows XP.

RFC 3161 Timestamping

Every signature includes a countersignature from Microsoft's RFC 3161 timestamp authority (`timestamp.acs.microsoft.com`). This ensures signatures remain valid long after the signing certificate expires. Azure Trusted Signing certificates have a 72-hour validity window and rotate automatically — without timestamping, signatures would become invalid within three days. With timestamping, they remain valid for the lifetime of the timestamp authority's certificate (typically 10+ years).

Microsoft-Backed Signing Infrastructure

The actual signing operation is performed by Microsoft's Azure Trusted Signing service. Qwick Cert does not operate its own Certificate Authority or manage private keys. The signing certificate, private key, and certificate chain are all managed by Microsoft within Azure's FIPS 140-2 Level 3 certified Hardware Security Modules (HSMs). Qwick Cert is a developer-experience layer on top of this infrastructure, not a replacement for it.

Data Handling

- **Binary content**: Never leaves the developer's machine. Only the 32-byte SHA-256 digest is transmitted.
- **Azure credentials**: Encrypted at rest in Supabase Vault (AES-256). Never transmitted to clients.
- **Signing audit logs**: Retained server-side for the lifetime of the organization's account.
- **API authentication**: OAuth 2.0 tokens or API keys, scoped to the Qwick Cert platform.

For questions about Qwick Cert's security model, contact security@qwickcert.com.