

npm would like to offer package signatures with post quantum cryptography (PQC) by 2030.

Today, npm packages are signed by several services working together:

- A build system that supports workload identity via OIDC (predominately GitHub Actions, but also GitLab Pipelines, CircleCI, Buildkite, and others)
- The npm CLI runs inside the build system to request the OIDC token and send it to the Sigstore public good instance
- The Sigstore public good instance returns a signature over the binary with included software supply chain metadata (SLSA Build Provenance), together referred to as a “bundle”
- The package and the Sigstore bundle are sent to the npm registry

And then verification of that content follows these steps:

- A consumer (usually a CLI like npm or cosign) downloads the npm package and the associated Sigstore bundle
- The consumer checks the Sigstore public good instance for updated verification material via The Update Framework (TUF)
- The consumer verifies the package with the bundle and the verification material, and optionally evaluates policy against the build provenance

On any given day, there’s roughly 45,000 packages published to npm, 18,000 of which provide signatures this way. npm is the largest package repository in the world, but many other large package repositories use a similar system: for example PyPI, Maven Central, Homebrew, and RubyGems.

There are several challenges to affecting this transition. Here are the items on the critical path:

We need vetted ML-DSA-65 implementations in various languages, as CLIs are often required to be written in the language of the ecosystem they serve: crucially for npm in JavaScript, but also Python, Java, Ruby, Rust, and Go (for Sigstore’s CLI cosign) so that clients can verify each others’ content.

We need Sigstore to finish finalizing its PQC bundle format; this work is well underway and expected to finish soon. The plan is to offer PQC bundles alongside the existing “classical” bundles, and let clients choose what algorithms they accept during verification.

When both of the above are done, we need to add support for verifying the Sigstore PQC bundle format to the npm CLI. Additionally, we would need to figure out how to communicate signing and verification policy to clients. For signing policy, which algorithm(s) should the client use, and for verification policy which algorithm(s) are acceptable for packages that were published in different timeframes. For both types of policy, how do we communicate this information to clients as it changes over time? Typically a major version of npm is released every 18 months, and the most recent major version is supported. We would probably wait at

least 12 months for people to adopt a new version of the npm CLI that supports PQC verification before the registry starts offering PQC content.

The above is sufficient for end-users to send and receive PQC Sigstore bundles. However, for the system to support PQC end-to-end, there are a few other gaps:

Build systems make their workload identity available to clients via OIDC, and so we need the IETF to finish its work on having OIDC tokens support PQC. Once standardized, we would need to coordinate with build systems on a migration strategy. There are many details to work out here, for example would build systems offer both options in parallel for some time as clients make the transition.

Ideally Sigstore key material would be distributed to clients using PQC, and so TUF should support PQC. Sigstore's plan is to provide a PQC-specific TUF repository. This work has not been started, but it seems relatively straightforward. The public key representation and associated signatures would be longer, but this should not be an issue.

Additionally, because Sigstore public good instance is managed by a consortium, the keys used to establish the TUF root of trust are USB hardware devices that are geographically distributed. The Tillitis TKey is a promising option for PQC, but ideally there would be several options that are reasonably priced and widely available.

Lastly, there are organizations (like GitHub) that manage private deployments of Sigstore in different configurations: that supply signatures in X.509 certificates, that include RFC 3161 signed timestamps, or that use managed keys instead of workload identity. It is important to the Sigstore project as a whole that these flexible configurations be maintained, and so there is additional work to be done to ensure client libraries and open source Sigstore servers support these formats. We believe the X.509 certificate can be created on demand via a transform on the PQC bundle content. We need to update the Sigstore RFC 3161 server (as separate infrastructure, no special transition needed), as well as the Sigstore code for working with managed keys, but both of these should be covered by the client and library work listed above.

There are many steps towards transitioning npm and other major package repositories to support PQC signatures, but in collaboration with Sigstore, standards bodies such as the IETF, and industry partners, we believe that this transition can be affected by 2030 if we continue working to resolve the questions listed above.

Zach Steindler is a principal security engineer at GitHub where he works on security the software supply chain for open source projects and enterprise customers. He does security advising with GitHub Actions and npm (which GitHub operates). He also volunteers in the OpenSSF where he chairs the Technical Advisory Council, co-chairs the Security Software Repositories Working Group (which brings together npm, PyPI, Maven Central, and other package repositories to discuss security capabilities), and he co-maintains Sigstore cosign and the sigstore-go client library.