Best practices for documenting attestations in package repositories

Attestations documentation should:

1. Help package consumers understand attestations, including what they are, how they work, and what their limitations are
2. Help package consumers understand attestations within the context of other security features and practices
3. Direct package maintainers towards workflows that generate attestations

For attestation documentation to be most successful, we recommend documentation authors follow seven principles:

1. Use key terms consistently

To ensure clarity and consistency for attestation consumers, use key terms as defined below. Adhering to these standards prevents user confusion across packaging ecosystems.

Attestation

- **Definition:** A statement of signed facts (or metadata) about a software artifact
- **Do:** Use this as the general term for a signed claim. It is the foundation for more specific terms like "Build Provenance Attestation"

Build Provenance

- **Definition:** A verifiable trail of evidence for a package's origin. It answers *where* a package came from and *how* it was built
- **Do:** Clarify this term in headings or text when the concept might be new to users
- **Example:** *"Verifying build provenance (package origin)"*

Build Provenance Attestation

- **Definition:** A specific type of attestation that describes how, when, and where a package was built
- **Do:** Clarify the relationship between attestations and build provenance using this term
- **Example:** *“A build provenance attestation describes how, when, and where a package was built.”*

Transparency Log Entry

- **Definition:** The tamper-proof record of the build provenance attestation saved to a public ledger
- **Do:** Use ""“transparency log entry” to describe how an attestation is recorded
- **Example:** *"Each attestation is recorded in a public ledger as transparency log entry, which is a permanent, auditable record that cannot be changed."*

Verified

- **Definition:** The process that a package repository takes when accepting an attestation from a build platform, ensuring the attestation's source matches the maintainer's configuration
- **Do:** Use "verified" to describe the repository's automated check during package upload
- **Example:** *"PyPI verifies this attestation at upload time, confirming that the identity matches what the maintainer previously configured for Trusted Publishing."*

Verified

- **Definition:** The process a user follows to cryptographically check that an attestation is legitimate
- **Do:** Use the term "verified" (and its variations) for the end-user's action
- **Example:** *"To manually verify a PyPI artifact against its provenance object, the* `pypi-attestations` *CLI tool can be used"*

2. Structure documentation around user roles

Package consumers and maintainers require (and have) different levels of knowledge about security concepts and features, including attestations.

Content for package *consumers* should be separated from content for package *maintainers*, surfacing relevant and actionable information to each audience.

Structure the Information Architecture around user roles, using clear, top-level sections like "Using This Repository" (for consumers) and "Publishing Packages" (for maintainers).

3. Provide content to help package *consumers* understand attestations within the context of other security features

Attestations are only one part of a larger security story. To be effective, documentation must place them in context, helping users build a holistic mental model of risk.

Documentation for package consumers should explain how security practices and features work together as a layered defense. It should be structured around the fundamental questions a user has about risk:

1. **Can I trust the source code?**
   Explaining signals like project activity, vulnerability scanning, etc.
2. **Where did this package come from and how was it made?**
   Explaining build provenance and attestations
3. **Did I download the same file that is hosted on the package repository?**
   Explaining checksums and integrity

4. Warn package consumers that packages are not secure by default

Some package consumers hold the incorrect assumption that because a package is published on an official repository it is secure and/or trustworthy.

To combat this, include disclaimers on pages for package consumers, directing users towards security documentation. Example from PyPI:

> *Warning: Installing packages from PyPI means running third-party code, which carries inherent security risks. Before installing any package, we strongly recommend reading our Package security guide.*

5. Normalize secure-by-default publishing workflows

Documentation should be opinionated in favor of security. The most secure method for any task should be presented as the canonical, recommended path.

When writing documentation about publishing packages, prominently feature the most secure, automated publishing method that generates attestations. This should be the first and most detailed workflow described, establishing it as the modern standard.

Where applicable, clearly and explicitly state that using the recommended workflow confers the benefit of generating provenance attestations *automatically*. This makes adopting the secure workflow more compelling.

Legacy methods that do not produce attestations should be explicitly marked as "deprecated" or "for advanced use cases" to guide users away from them.

6. Explain common security attacks and how they are mitigated

Transparently list the threats the user faces (e.g., Typosquatting, Account Takeover) and what they can do to protect themselves. Explain platform-level features that mitigate common attacks, including the role of Trusted Publishing and attestations.

Provide clear reporting channels for users concerned about the security of an individual package or the package repository as a whole.

7. Be transparent about limitations and set clear expectations

Documentation must be upfront about the scope and limitations of the security practices or features it describes.

State clearly that an attestation proves *build provenance* (where a file came from) but does not vouch for the *code's safety* (that the code is free of bugs or vulnerabilities). Explain that trusting an attestation means implicitly trusting the tooling and authority that issued it.

Be transparent about the readiness of downstream tooling. Where applicable, highlight that automated attestation verification is not yet ready in common package managers.