Code Signing

Revision

Version 3
2/18/22 1:17 PM

Author

Charles Wilson

Abstract

This document describes the process used to sign code and other artifacts.

Group / Owner

DevOps / Information Systems Security Developer

Motivation

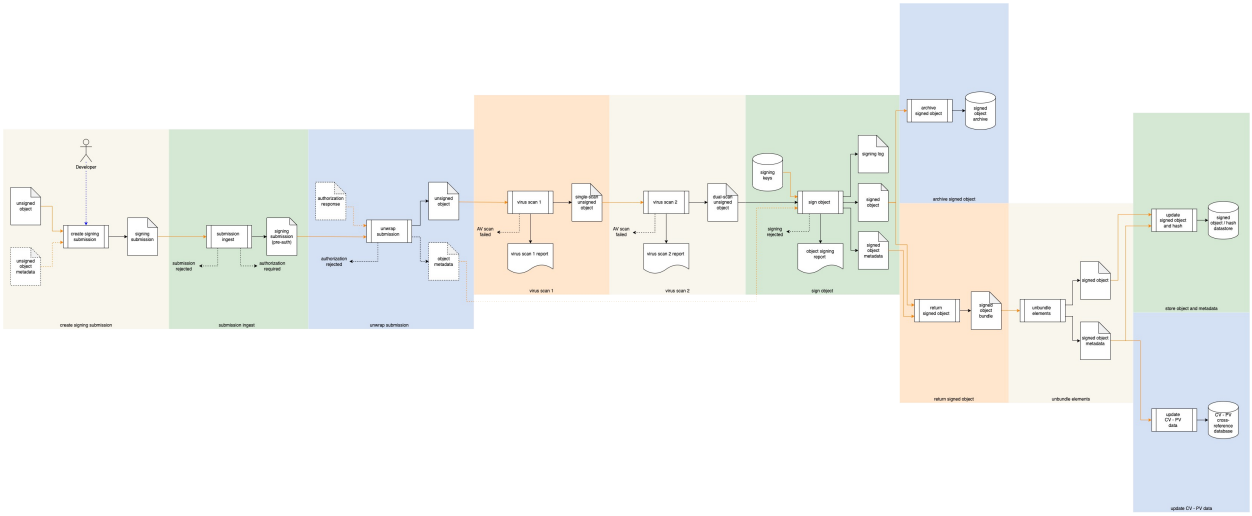
This document is motivated by the need to adopt best practices regarding management of source code, specifically code signing, to allow for certification of compliance to standards such as **ISO 21434** and **ISO 26262**.

License

Overview

Code signing is a critical element of the process of ensuring overall system cybersecurity. Although code signing may be performed manually, it is not recommended to take this approach. The security of the signing keys as well as general access to the process are two areas which need to be thoroughly considered. The greater the level of security of the signing system itself, the greater the certainty that no inappropriate use has been made of it.

The following diagram illustrates the process to be used:

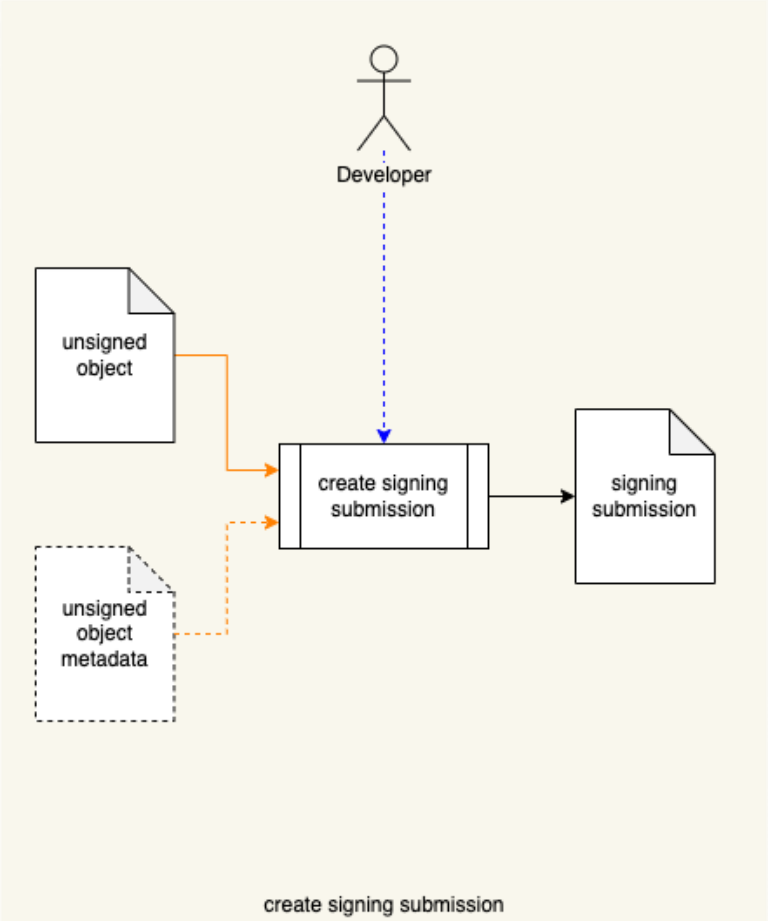


Note: The signing process outlined in this document is a representative the handling of a single object type for a single target platform. In practice, a diversity of both object type and target platform exists. It is presumed that the reader is aware of the implications of this and will emplace those additional activities needed to accommodate this diversity.

Process

Create Signing Submission

Inputs	Unsigned object Unsigned object metadata (optional)
Outputs	Signing submission
Participants	Developer (optional)

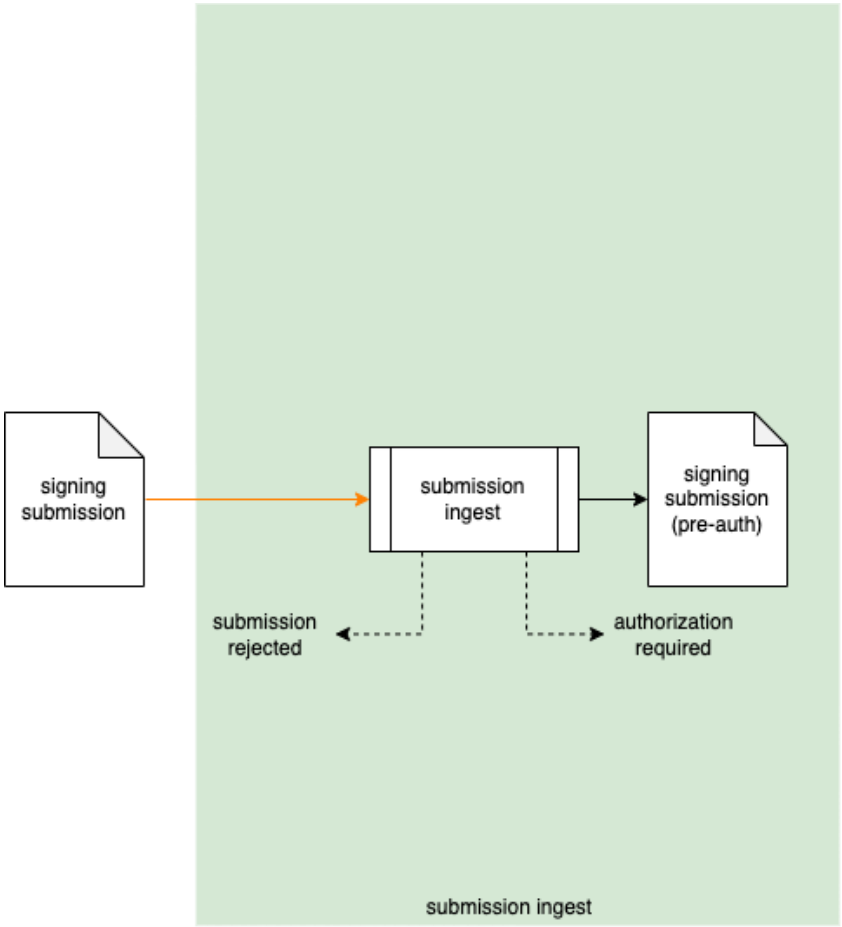


The **unsigned object** and optionally **unsigned object metadata** are used to create a **signing submission**. This process may be fully automated or initiated manually by a developer.

Note: Typically, metadata is required in order to sign objects targeting embedded systems.

Submission Ingest

Inputs	Signing submission
Outputs	Signing submission (pre-auth)
Participants	**none**



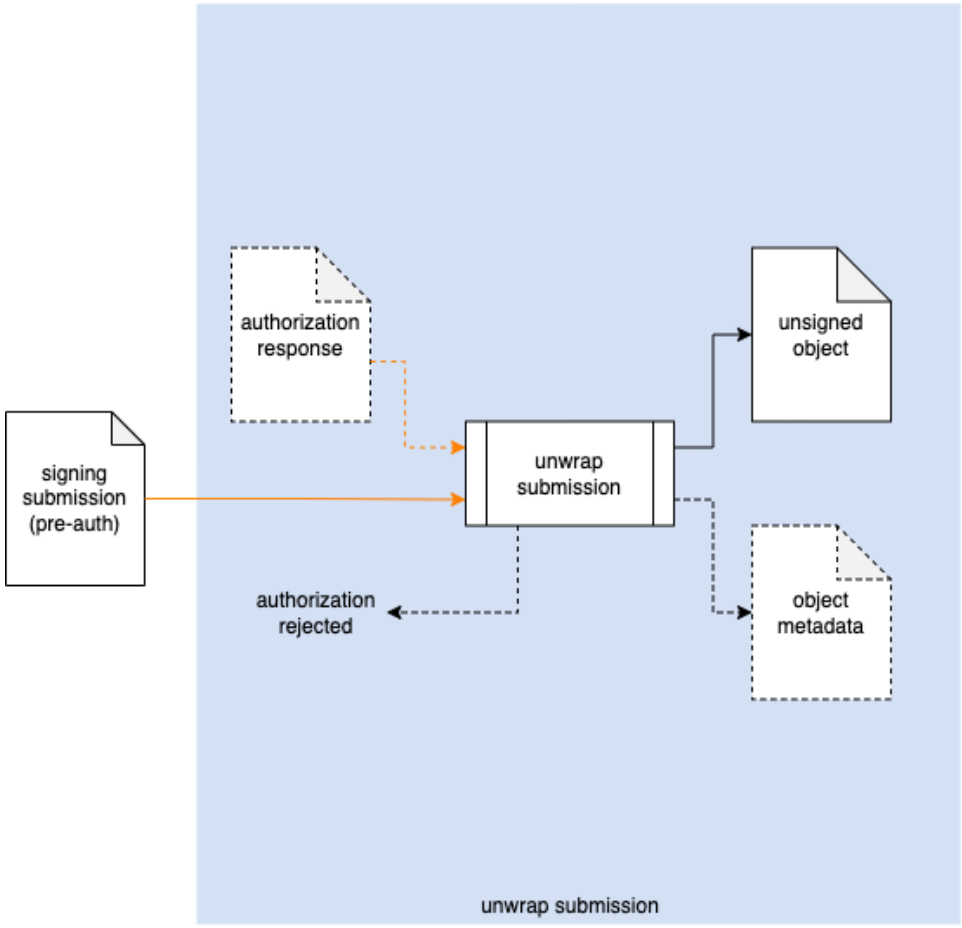
The **signing submission** is ingested by the signing system. If the submission is ill-formed, a **submission rejected** notification is generated. If a secondary authorization is required, an **authorization required** notification is generated. A **signing submission (pre-auth)** is generated.

Note: In the case where no additional authorization is required, the submission is flagged as being authorized.

Note: The **authorization required** notification is used to trigger a system by which authorization for the signing activity is solicited. This may take the form of a URL confirmation link in an email or other similar authorization mechanism that the signing system can use to gate the signing process.

Unwrap Submission

Inputs	Signing submission (pre-auth) Authorization response (optional)
Outputs	Unsigned object Object metadata (optional)
Participants	**none**

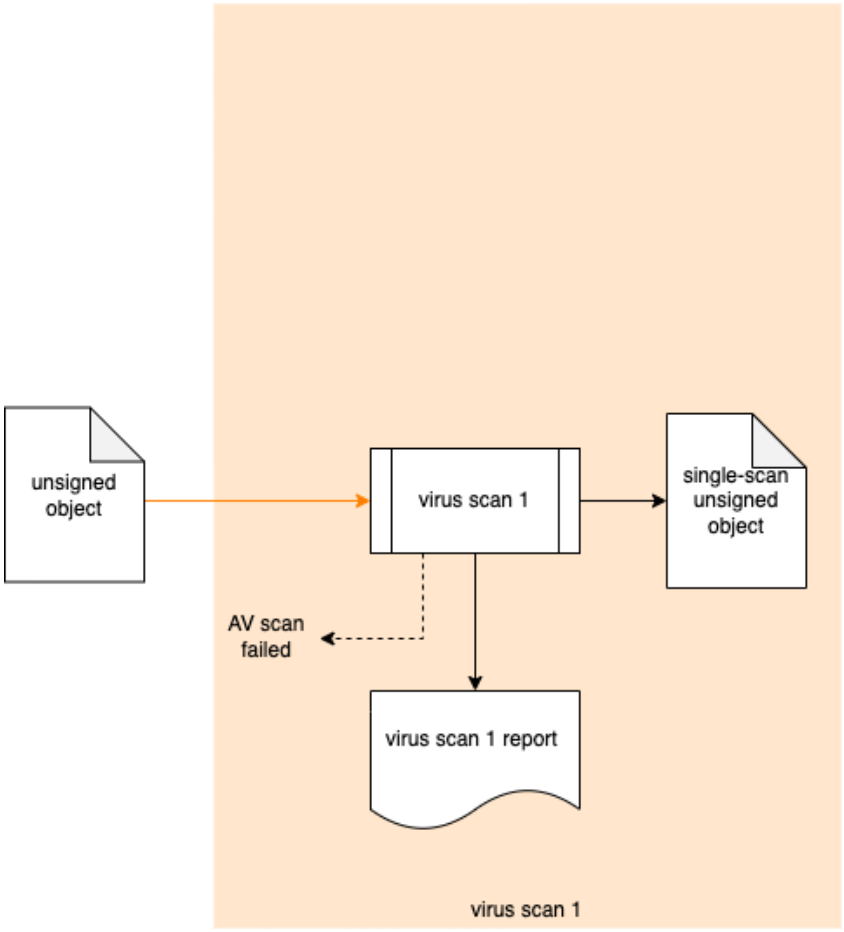


The **signing submission (pre-auth)** is unwrapped if the submission is authorized for signing. This is either because the submission has been flagged as authorized in the previous step or if the **authorization response** is in the affirmative. If the authorization response is in the negative, an **authorization rejected** notification is generated. If authorized, the unwrapped **unsigned object** and optionally **object metadata** artifacts are generated.

Note: The signing system must be capable of gating on the authorization response for this activity to take place.

Virus Scan 1

Inputs	Unsigned object
Outputs	Single-scan unsigned object Virus scan 1 report
Participants	**none**

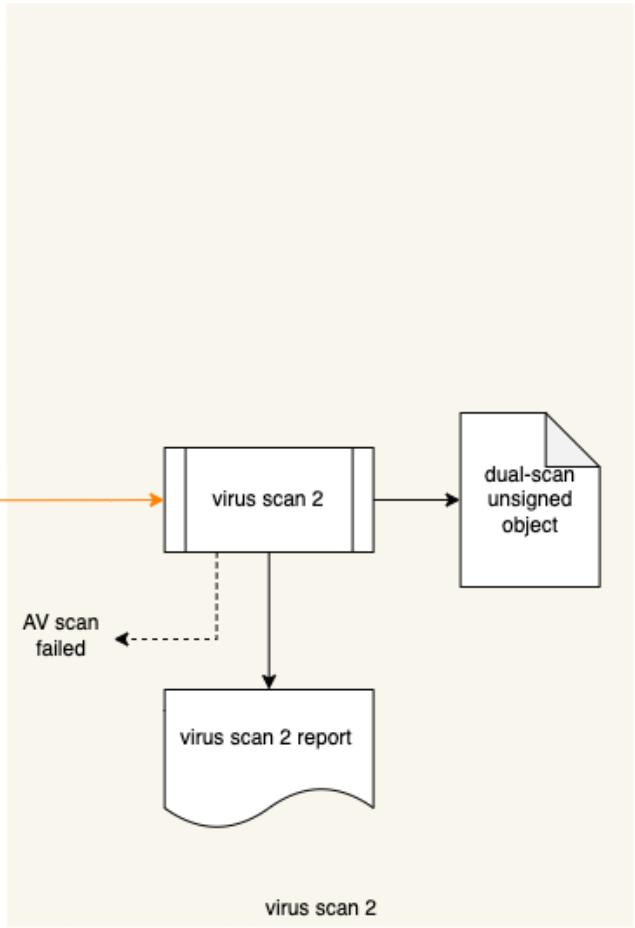


The first of two virus scans is performed on the **unsigned object**. If the scan fails, an **AV scan failed** notification is generated. If the scan succeeds, a **single-scan unsigned object** is generated (passed along). A **virus scan 1 report** is generated.

Note: The virus scans are performed by two different virus scanning applications in order to gain greater coverage.

Virus Scan 2

Inputs	Single-scan unsigned object
Outputs	Dual-scan unsigned object Virus scan 2 report
Participants	**none**

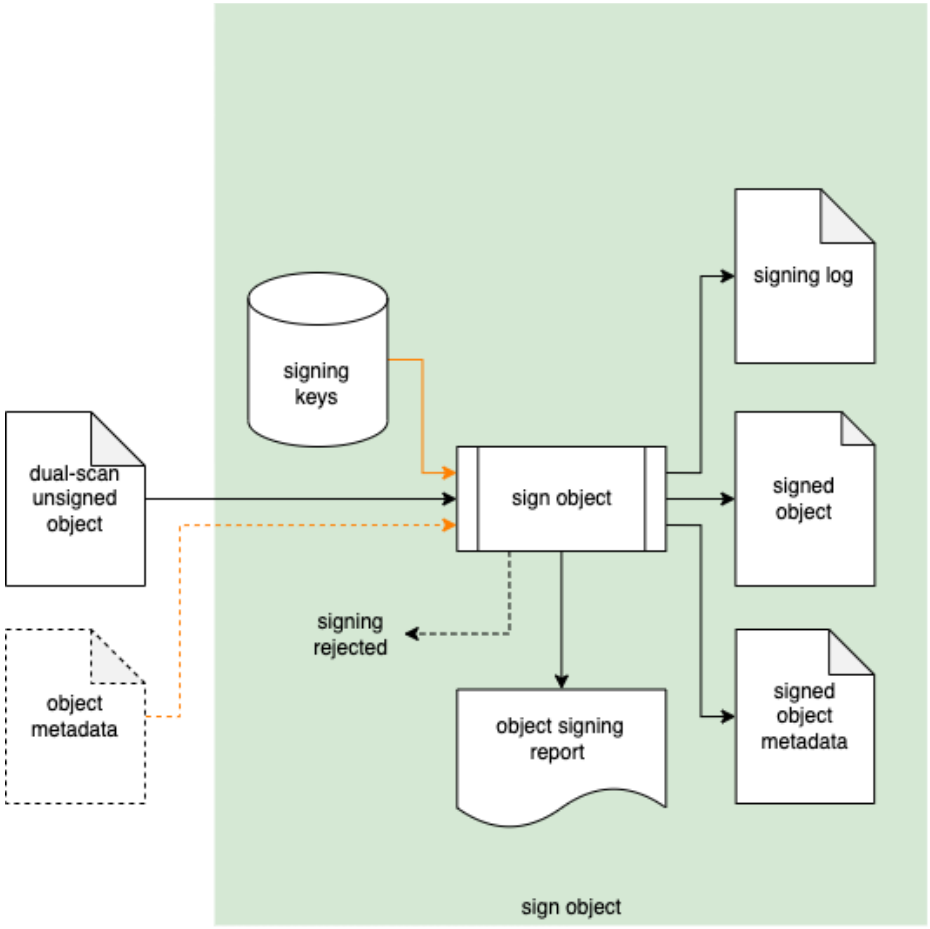


The second virus scan is performed on the **single-scan unsigned object**. If the scan fails, an **AV scan failed** notification is generated. If the scan succeeds, a **dual-scan unsigned object** is generated (passed along). A **virus scan 2 report** is generated.

Note: The virus scans are performed by two different virus scanning applications in order to gain greater coverage.

Sign Object

Inputs	Dual-scan unsigned object Object metadata (optional) Signing keys
Outputs	Signing log Signed object Signed object metadata Object signing report
Participants	**none**



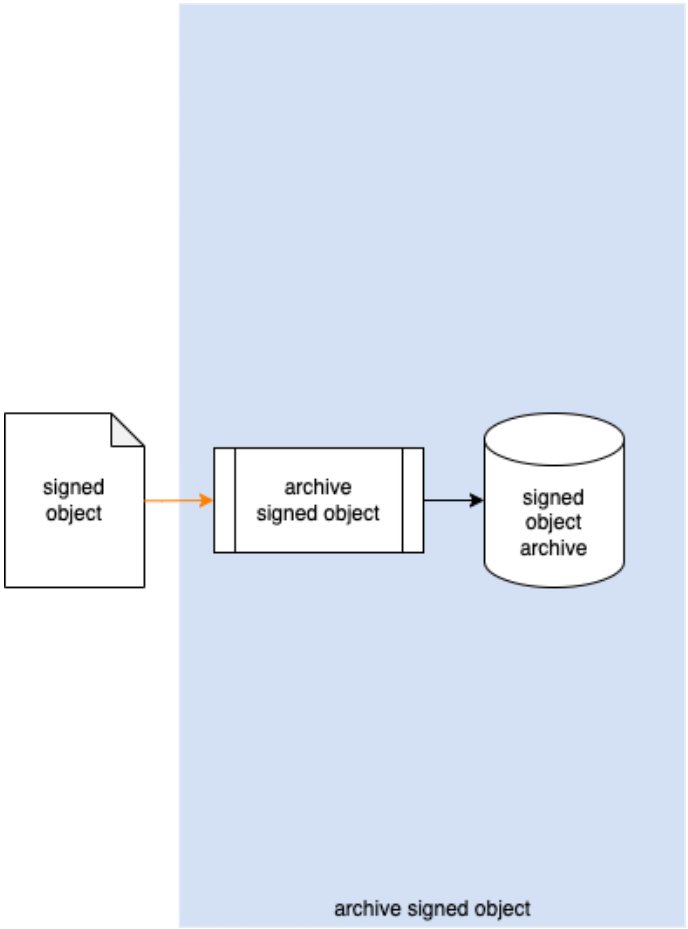
Using the appropriate signing key from the **signing keys** datastore and possible information provided by the **object metadata**, the **dual-scan unsigned object** is signed. If the signing is unsuccessful, a **signing rejected** notification is generated. If successful, a **signing object** and **signed object metadata** artifacts are generated. The **signing log** is updated. An **object signing report** is generated.

Note: There are multiple reasons for a signing failure. These include (but are not limited to) issues with the signing key selected, signing key lifetime, and details of the **object metadata**.

Note: The object metadata may include (but not be limited to) information such as hash, certificate, and CRC.

Archived Signed Object

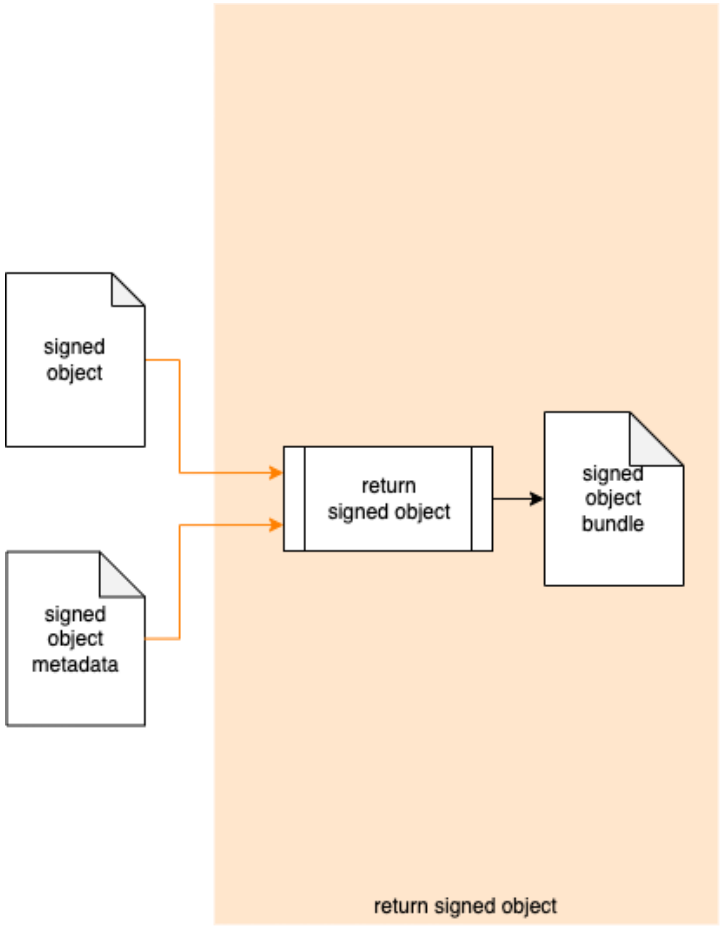
Inputs	Signed object
Outputs	Signed object archive
Participants	**none**



The **signed object** is copied into the **signed object archive** for auditing purposes.

Return Signed Object

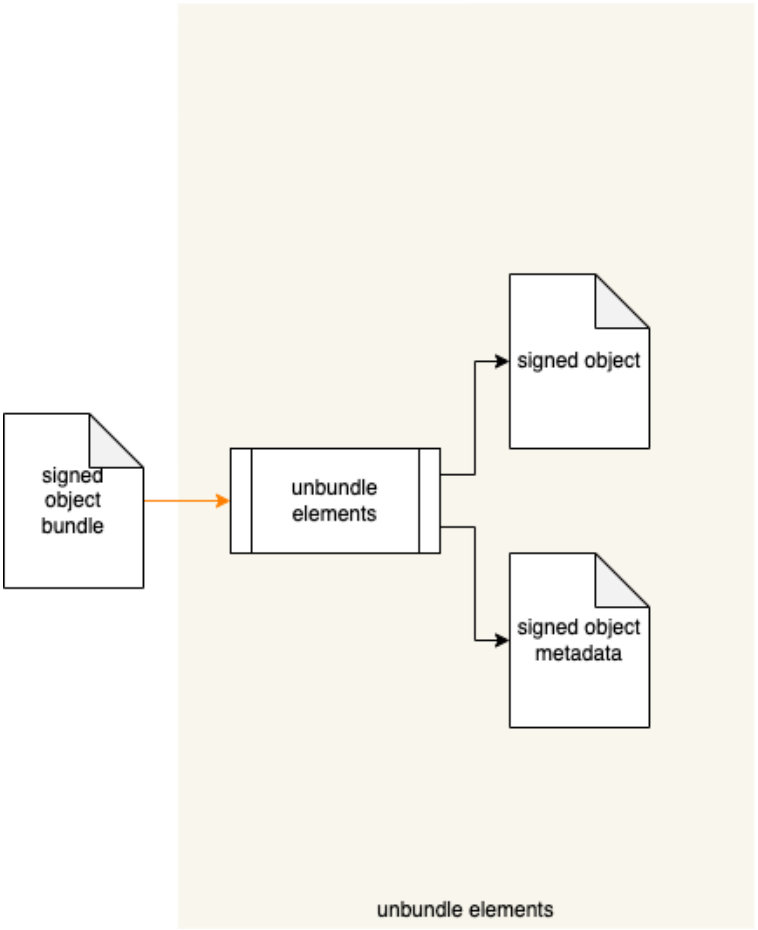
Inputs	Signed object Signed object metadata
Outputs	Signed object bundle
Participants	**none**



The **signed object** and **signed object metadata** artifacts are grouped together into a **signed object bundle** for return to the submitter.

Unbundle Elements

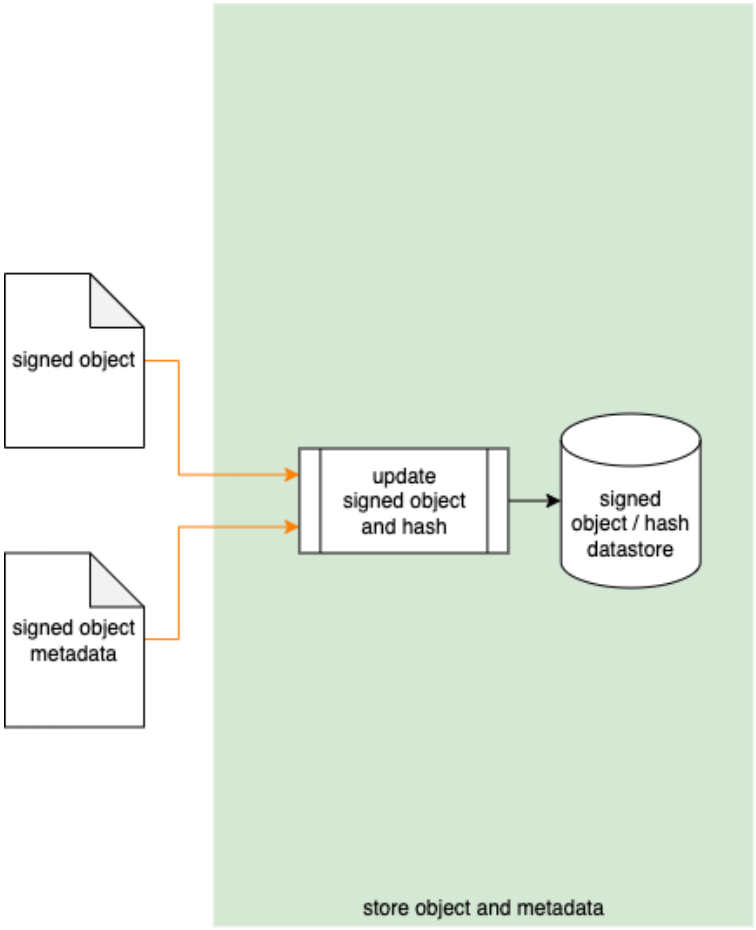
Inputs	Signed object bundle
Outputs	Signed object Signed object metadata
Participants	**none**



The returned **signed object bundle** is unbundled into the **signed object** and **signed object metadata** artifacts.

Store Object and Metadata

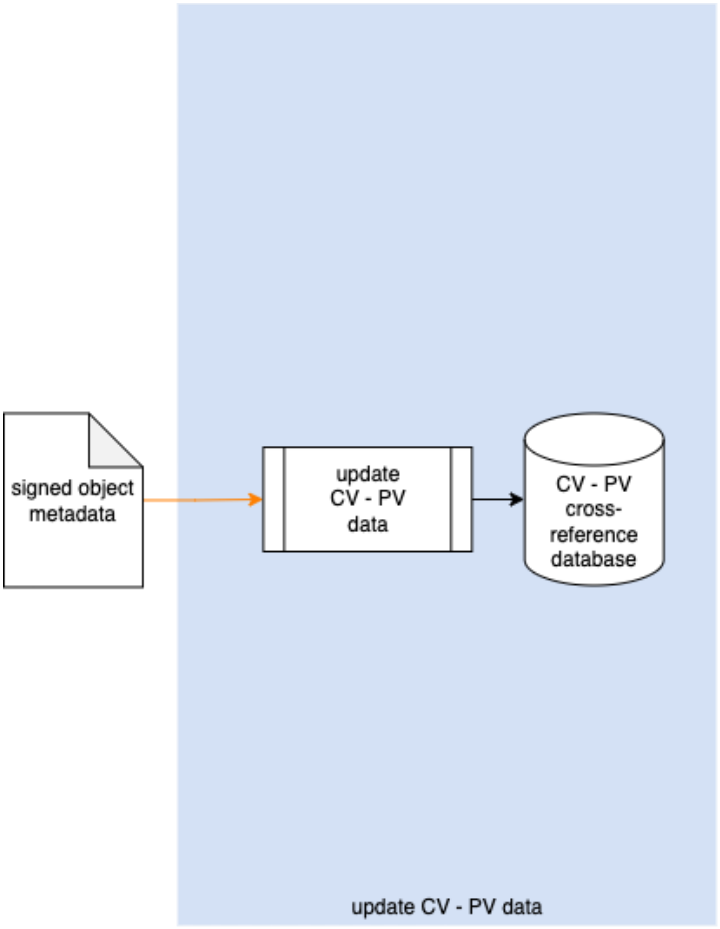
Inputs	Signed object Signed object metadata
Outputs	Signed object / hash datastore
Participants	**none**



The **signed object** and **signed object metadata** are associated and copied into the **signed object / hash datastore**.

Update CV-PV Data

Inputs	Signed object metadata
Outputs	CV – PV cross-reference database
Participants	none



The **signed object metadata** is used to update the object's entry in the **Component/Version – Product/Version (CV-PV) cross-reference database** [5].

References

1. **Code Protection Plan** (AVCDL secondary document)
2. **Code Signing Best Practices**
   https://download.microsoft.com/download/a/f/7/af7777e5-7dcd-4800-8a0a-b18336565f5b/best_practices.doc
3. **Security Considerations for Code Signing**
   https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.01262018.pdf
4. **Protecting Software Integrity Through Code Signing**
   https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=925977
5. **Component / Version - Product / Version Cross-reference Document** (AVCDL secondary document)