SUIT
Internet-Draft
Intended status: Standards Track
Expires: 4 September 2025

B. Moran
Arm Limited
K. Takayama
SECOM CO., LTD.

3 March 2025

Software Update for the Internet of Things (SUIT) Manifest Extensions for Multiple Trust Domains

draft-ietf-suit-trust-domains-10

Abstract

~~This specification describes extensions to the SUIT Manifest format for use in deployments with multiple trust domains.~~ A device has more than one trust domain when it enables delegation of different rights to mutually distrusting entities for use for different purposes or Components in the context of firmware or software update. This specification describes extensions to the Software Update for the Internet of Things (SUIT) Manifest format for use in deployments with multiple trust domains.

Commenté [MB1]: Link with the IM/Compatibility (rfc9124):

REQ.USE.MFST.COMPONENT.

**Discuss

How the spec satisifies the following:

Satisfies: USER_STORY.OVERRIDE (Section 4.4.3), USER_STORY.COMPONENT (Section 4.4.4)

Commenté [MB2]: Found a WGLC in 2023 against -02 but there is no mention in the wiretup what were the issues to have this now published.

Failed to find other records, including in the history

Revied diff vs -02: but it would be good if we had a summary of the changes (the side-to-side diff is verbose as most of the changes are minor ones): I tagged at least very few changes to the CDDL, removal of teh delegation part, and more examples.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Devices that go beyond single-signer update require more complex rules for deploying software updates. For example, devices may require:

* ~~software~~ Components from multiple software signing authorities.

* a mechanism to remove an unneeded Component

* single-object Dependencies

* a partly encrypted Manifest so that distribution does not reveal

Commenté [MB3]: Was this introduced in some other documents?

I failed to find this use in the suit manifest spec, rfc9019, etc.

Commenté [MB4]: Expecting to see explanation of «more complex»

Commenté [MB5]: Given the definition of «Component» is

«* Component: An updatable logical block of the Firmware, Software, configuration, or data of the Recipient. »

Commenté [MB6]: This is just an elaboration of the definition of «beyond single-signer». I expect this list to exemplify the complexity

Commenté [MB7]: Isn't this applicable even for the single-signer?

Commenté [MB8]: Idem as the comment about single-signer private information

* installation performed by a different execution mode than payload fetch

Commenté [MB9]: Should be defined first.

Because of the more complex use cases that are typically ~~targetted~~targeted by devices implementing this specification, the applicable device class is typically Class 2+ and often isolation level Is8, for example Arm TrustZone for Cortex-M, as described in [I-D.ietf-iotops-7228bis].

Commenté [MB10]: Should this be normative as this define the device to which the spec is applicable?

Dependency Manifests enable several additional use cases. In particular, they enable ~~two or more~~ entities who are trusted for different privileges to coordinate. This can be used in many scenarios. For example:

Commenté [MB11]: I failed to find this in other SUIT spec I checked.

Commenté [MB12]: Not needed

* A device may contain a processor in its radio in addition to the primary processor. These two processors may have separate Software with separate signing authorities. Dependencies allow the Software for the primary processor to reference a Manifest signed by a different authority.

Commenté [MB13]: ??

* A network operator may wish to provide local caching of Update Payloads. The network operator overrides the URI of a Payload by providing a dependent Manifest that references the original Manifest, but replaces its URI.

a mis en forme : Surlignage

* A device operator provides a device with some additional configuration. The device operator wants to test their configuration with each new Software version before releasing it. The configuration is delivered as a binary in the same way as a Software Image. The device operator references the Software Manifest from the Software author in their own Manifest which also defines the configuration.

Commenté [MB14]: What is a device operator? Is it the vendor? Else? Please consider adding a definition or supply a reference.

* An Author wants to entrust a Distributor to provide devices with firmware decryption keys, but not permit the Distributor to sign code. Dependencies allow the Distributor to deliver a device's decryption information without also granting code signing authority.

* A Trusted Application Manager (TAM) wants to distribute personalisation information to a Trusted Execution Environment in addition to a Trusted Application (TA), but does not have code signing authority. Dependencies enable the TAM to construct an update containing the personalisation information and a dependency on the TA, but leaves the TA signed by the TA's Author.

Commenté [MB15]: References, please

By using Dependencies, Components such as Software, configuration, and other Resource data authenticated by different Trust Anchors can be delivered to devices.

These mechanisms are not part of the core Manifest specification, but they are needed for more advanced use cases, such as the architecture described in [I-D.ietf-teep-architecture].

Commenté [MB16]: Add a reference

Commenté [MB17]: Update to rfc9397.

This specification extends the SUIT Manifest specification

Commenté [MB18]: Fall a little bit short. A summary of the extensions would be helpful.

~~(~~[I-D.ietf-suit-manifest]~~)~~.

2. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

~~Additionally, t~~The following terminology is used throughout this document:

Commenté [MB19]: I would avoid redundant terms but refer to draft-ietf-suit-manifest#section 2.

Only new terms should be listed here.

- SUIT: Software Update for the Internet of Things~~, also the IETF working group for this standard~~.

- Payload: A piece of information to be delivered. Typically, Firmware/Software, configuration, or Resource data such as text or images.

Commenté [MB20]: To whom?

- Resource: A piece of information that is used to construct a Payload.

- Manifest: A Manifest is a bundle of metadata about one or more Components for a device, where to find them, and the devices to which they apply.

- Envelope: A container with the Manifest, an authentication wrapper with cryptographic information protecting the Manifest, authorization information, and severable elements (~~see~~ Section 5.1 of [I-D.ietf-suit-manifest]).

Commenté [MB21]: There are more than 60 occurrences of such in the document. Please simple delete this. A reference is there to be checked/viewed/seen/etc. :-)

- Update: One or more Manifests that describe one or more Payloads.

- Update Authority: The owner of a cryptographic key used to sign Updates, trusted by Recipients.

- Recipient: The system that receives and processes a Manifest.

- Manifest Processor: A component of the Recipient that consumes Manifests and executes the Commands in the Manifest.

- Component: An updatable logical block of the Firmware, Software, configuration, or data of the Recipient.

- Component Set: A group of interdependent Components that must be updated simultaneously.

- Command: A Condition or a Directive.

- Condition: A test for a property of the Recipient or its Components.

- Directive: An action for the Recipient to perform.

- Trusted Invocation: A process by which a system ensures that only trusted code is executed, for example secure boot or launching a Trusted Application.

- A/B Images: Dividing a Recipient's storage into two or more bootable Images, at different offsets, such that the active Image can write to the inactive Image(s).

- Record: The result of a Command and any metadata about it.

- Report: A list of Records.

- Procedure: The process of invoking one or more sequences of Commands.

- Update Procedure: A superset of Staging Procedure and Installation Procedure.

- Staging Procedure: A procedure that fetches dependencies and images referenced by an Update and stores them to a Staging Area.

- Installation Procedure: A procedure that installs dependencies and images stored in a Staging Area; copying (and optionally, transforming them) into an active Image storage location.

- Invocation Procedure: A Procedure in which a Recipient verifies Dependencies and Images, loading Images, and invokes one or more Image.

- Staging Area: A Component or group of Components that are used for transient storage of Images between fetch and installation. Images in this area are opaque, except for use by the Installation Procedure.

- Software: Instructions and data that allow a Recipient to perform a useful function.

- Firmware: Software that is typically changed infrequently, stored in nonvolatile memory, and small enough to apply to [I-D.ietf-iotops-7228bis] Class 0-2 devices.

- Image: Information that a Recipient uses to perform its function, typically Firmware/Software, configuration, or Resource data such as text or images. Also, a Payload, once installed is an Image.

- Slot: One of several possible storage locations for a given Component, typically used in A/B Image systems

- Abort: An event in which the Manifest Processor immediately halts execution of the current Procedure. It creates a Record of an error Condition.

- Trust Anchor: A Trust Anchor, as defined in [RFC6024], represents an authoritative entity via a public key and associated data. The public key is used to verify digital signatures, and the associated data is used to constrain the types of information for which the Trust Anchor is authoritative.

3. Changes to SUIT Workflow Model

The use of the features presented for use with multiple trust domains

Commenté [MB22]: (meta comment) Approaching this from OPS and how this can be put into effect, an effort is needed provide a short overview of the various pieces being grafted. I'm not asking to have that in this document, but the WG can consider having such an overview «somewhere». If you already have, please share it LOUDLY. Thank you.

Commenté [MB23]: This is not introduced yet!

requires some augmentations of the workflow presented in the SUIT Manifest specification ([I-D.ietf-suit-manifest]):

One additional assumption is added for the Update Procedure:

* All Dependency Manifests must be present before any Payload is fetched.

One additional assumption is added to the Invocation Procedure:

* All Dependencies must be validated prior to loading.

Steps 3 and 5 are added to the expected installation workflow of a Recipient:

1. Verify the signature of the Manifest.

2. Verify the applicability of the Manifest.

3. Resolve Dependencies.

4. Fetch Payload(s).

5. Verify Candidate.

6. Install Payload(s).

In addition, when multiple Manifests are used for an Update, each Manifest's steps occur in a lockstep fashion; all Manifests have Dependency resolution performed before any Manifest performs a Payload fetch, etc.

4. Changes to Manifest Metadata Structure

To accommodate the additional metadata needed to enable these features, the Envelope and Manifest ~~have several~~require new elements ~~added~~.

The Envelope gains one more element~~s~~: Integrated Dependencies. The Common metadata section in the Manifest also gains a list of Dependencies.

The new metadata structure is shown below.

```
+-------------------------+
| Envelope                |
+-------------------------+
| Authentication Block    |
| Manifest           --------------> +-------------------------------+
| Severable Elements      |          | Manifest                      |
| Human-Readable Text     |          +-------------------------------+
| CoSWID                  |          | Structure Version             |
| Integrated Dependencies |          | Sequence Number               |
| Integrated Payloads     |          | Reference to Full Manifest    |
+-------------------------+    +------ Common Structure              |
                               | +---- Command Sequences             |
+-------------------------+    | |   | Digests of Envelope Elements  |
```

Commenté [MB24]: Please call out where the «other» assumptions are defined? I suspect that you meant 4.2 of the manifest? If so, please say so. If not, I need to understand ;-)

Commenté [MB25]: Only one?always?

Commenté [MB26]: DISCUSS--Putting aside ⅗, the list does not mirror all the items in 4.2 of the manifest.

I'd like to chech this is on purpose and understand why.

Commenté [MB27]: check

Commenté [MB28]: DISCUSS==Do we need to tag this as updating the manifest as it amends it?

Commenté [MB29]: Please say this is an update of the figure in Section 4.2 if the manifest spec.

Commenté [MB30]: Where is this defined?

```
   | Common Structure         | <--+ |   +-------------------------------+
   +--------------------------+      |
   | Dependency Indices       |      +-> +------------------------+
   | Component IDs            |          | Command Sequence       |
   | Common Command Sequence ---------> +------------------------+
   +--------------------------+          | List of ( pairs of (   |
                                         |   * command code       |
                                         |   * argument /         |
                                         |      reporting policy |
                                         | ))                     |
                                         +------------------------+
```

5. Dependencies

A Dependency is another SUIT_Envelope that describes additional Components.

Commenté [MB31]: Cite where this is defined.

As described in Section 1, Dependencies enable several common use cases.

Commenté [MB32]: Does not bring much, even when repeated.

5.1. Changes to Required Checks

This section augments the definitions in Required Checks (Section 6.2~~)~~ of [I-D.ietf-suit-manifest].

More checks are required when handling Dependencies. By default, any signature of a Dependency MUST be verified. However, there are some exceptions to this rule: where a device supports only one level of access (no ACLs defining which authorities have access to different Components/Commands/Parameters), it MAY choose to skip signature verification of Dependencies, since they are verified by digest. Where a device differentiates between trust levels, such as with an ACL, it MAY choose to defer the verification of signatures of Dependencies until the list of affected Components is known so that it can skip redundant signature verifications. For example, if a dependent's signer has access rights to all Components specified in a Dependency, then that Dependency does not require a signature verification. Similarly, if the signer of the dependent has full rights to the device, according to the ACL, then no signature verification is necessary on the Dependency.

Commenté [MB33]: How/where these are supplied?

Can we have examples or a reference where these examples are provided?

Components that should be treated as Dependency Manifests are identified in the suit-common metadata. See Section 5.2 for details.

If the Manifest contains more than one Component and/or Dependency, each Command sequence MUST begin with a Set Component Index Command.

Commenté [MB34]: Not only for this this specific statement, can we include a reminder about what to do when a mandatory condition is not met, I guess the process will abort.

If a Dependency is specified, then the Manifest processor MUST perform the following checks:

1. The dependent MUST populate all Command sequences for the current Procedure (Update or Invoke).

Commenté [MB35]: Can we please add a pointer where this command sequence is defined?

2. At the end of each section in the dependent: The corresponding section in each Dependency has been executed.

If the interpreter does not support Dependencies and a Manifest specifies a Dependency, then the interpreter MUST Abort.

If a Recipient supports groups of interdependent Components (a Component Set), then it SHOULD verify that all Components in the Component Set are specified by a single Manifest and all its Dependencies that together:

1. have sufficient permissions imparted by their signatures

2. specify a digest and a Payload for every Component in the Component Set.

The single dependent Manifest is sometimes called a Root Manifest.

5.2. Changes to Manifest Structure

This section augments the Manifest Structure (Section 8.4~~)~~ in [I-D.ietf-suit-manifest].

5.2.1. Manifest Component ID

In complex systems, it may not always be clear where the Root Manifest should be stored; this is particularly complex when a system has multiple, independent Root Manifests. The Manifest Component ID resolves this contention. The manifest-component-id is intended to be used by the Root Manifest. When a Dependency Manifest also declares a Component ID, the Dependency Manifest's Component ID is overridden by the Component ID declared by the dependent.

Commenté [MB36]: Is?

The following CDDL describes the Manifest Component ID:

Commenté [MB37]: DISCUSS: RFC 8610 should be normative

BTW, not only specific to this document, but it would be useful to have a representation similar to tRFC8340 for abstract YANG data structure (8791).

```
$$SUIT_Manifest_Extensions //=
    (suit-manifest-component-id => SUIT_Component_Identifier)
```

5.2.2. SUIT_Dependencies Manifest Element

The suit-common section, as described in Section 8.4.5 of [I-D.ietf-suit-manifest], ~~Section 8.4.5~~ is extended with a map of Component indices that indicate a Dependency Manifest. The keys of the map are the Component indices and the values of the map are any extra metadata needed to describe those Dependency Manifests.

Because some operations treat Dependency Manifests differently from other Components, it is necessary to identify them. SUIT_Dependencies identifies which Components from suit-components (~~see~~ Section 8.4.5 of [I-D.ietf-suit-manifest]) are to be treated as Dependency Manifest Envelopes. SUIT_Dependencies is a map of Components, referenced by Component Index. ~~Optionally, a~~A Component prefix or other metadata may be delivered with the Component index. The CDDL for suit-dependencies is shown below:

Commenté [MB38]: Redundant with «may» right after

```
$$SUIT_Common-extensions //= (
    suit-dependencies => SUIT_Dependencies
)
SUIT_Dependencies = {
    + uint => SUIT_Dependency_Metadata
}
```

```
SUIT_Dependency_Metadata = {
    ? suit-dependency-prefix => SUIT_Component_Identifier
    * $$SUIT_Dependency_Extensions
}
```

If no extended metadata is needed for an extension, SUIT_Dependency_Metadata is an empty map (this is the same encoding size as a null). SUIT_Dependencies MUST be sorted according to CBOR canonical encoding.

Commenté [MB39]: Add an authoritative RFC?

The Components specified by SUIT_Dependency will contain a Manifest Envelope that describes a Dependency of the current Manifest. The Manifest is identified, but the Recipient should expect an Envelope when it acquires the Dependency. This is because the Manifest is the one invariant element of the Envelope, where other elements may change by countersigning, adding authentication blocks, or severing elements.

Commenté [MB40]: DISCUSS: I don't see this in the CDDL

When executing suit-condition-image-match over a Component that is designated in SUIT_Dependency, the digest MUST be computed over just the bstr-wrapped SUIT_Manifest contained in the Manifest Envelope designated by the Component Index. This enables a Dependency reference to uniquely identify a particular Manifest structure. This is identical to the digest that is present as the first element of the suit-authentication-block in the Dependency's Envelope. The digest is calculated over the Manifest structure to ensure that removing a signature from a Manifest does not break Dependencies due to missing signature elements. This is also necessary to support the trusted intermediary use case, where an intermediary re-signs the Manifest, removing the original signature, potentially with a different algorithm, or trading COSE_Sign for COSE_Mac.

Commenté [MB41]: Where those are defined?

The suit-dependency-prefix element contains a SUIT_Component_Identifier (~~see ~~Section 8.4.5.1 of [I-D.ietf-suit-manifest]). This specifies the scope at which the Dependency operates. This allows the Dependency to be forwarded on to a Component that is capable of parsing its own Manifests. It also allows one Manifest to be deployed to multiple dependent Recipients without those Recipients needing consistent Component hierarchy. This element is OPTIONAL for Recipients to implement.

A Dependency prefix can be used with a Component's identifier. This allows complex systems to understand where Dependencies need to be applied.

The Dependency prefix can be used in one of two ways. The first simply prepends the prefix to all Component Identifiers in the Dependency.

A Dependency prefix can also be used to indicate when a Dependency Manifest needs to be processed by a secondary Manifest processor, as described in Section 5.4.1.

Commenté [MB42]: Group both cases

5.3. Changes to Abstract Machine Description

Commenté [MB43]: See update dicussion

This section augments the Abstract Machine Description (Section 6.4) in [I-D.ietf-suit-manifest]. With the addition of Dependencies, some

Commenté [MB44]: Please check as there is no 6.4 in this doc. I guess you meant 6.4 of manifest.

changes are necessary to the abstract machine, outside the typical scope of added Commands. These changes alter the behaviour of an existing Command and way that the parser processes Manifests:

* Five new Commands are introduced:

  - Set Parameters

  - Process Dependency

  - Is Dependency

  - Dependency Integrity

  - Unlink

Commenté [MB45]: Add a reference to Section 5.6?

* Dependency Manifests are also Components. All Commands may target Dependency Manifests as well as Components, with one exception: process Dependency. Commands defined outside of this ~~draft~~ document and [I-D.ietf-suit-manifest] MAY have additional restrictions.

Commenté [MB46]: The normative language is weird here. Consider: «Additonnal restrictions may be added by future commands».

* Dependencies are processed in lockstep with the Root Manifest. This means that every Dependency's current Command sequence must be executed before a dependent's later Command sequence may be executed. For example, every Dependency's Dependency Resolution step MUST be executed before any dependent's Payload fetch step.

Commenté [MB47]: This is an example, s/MUST/must

* When a Manifest Processor supports multiple independent Components, they MAY have shared Dependencies.

Commenté [MB48]: I don't think the normative langiage use is apprpriate here

* When a Manifest Processor supports shared Dependencies, it MUST support reference counting of those Dependencies.

Commenté [MB49]: !! That'is?

I see «reference count» used in some other parts, though.

* When reference counting is used, Components MUST NOT be overwritten. The Manifest Uninstall section must be called, then the component MUST be Unlinked.

5.4. Processing Dependencies

As described in Section 5.1, each Manifest must invoke each of its Dependencies' sections from the corresponding section of the dependent. Any changes made to Parameters by the Dependency persist in the dependent.

When a Process Dependency Command is encountered, the Manifest processor:

1. Checks whether the map of Dependencies contains an entry for the current Component Index. If not present, it causes an immediate Abort.

2. Checks whether the Dependency has been the target of a Dependency integrity check. If not, it causes an immediate Abort.

3. Loads the specified Component as a Dependency Manifest Envelope.

Commenté [MB50]: What does that mean concretely? Failed to find an elaboration of this in the doc.

4. Authenticates the Dependency Manifest.

5. Executes the common-sequence section of the Dependency Manifest.

6. Executes the section of the Dependency Manifest that corresponds to the currently executing section of the dependent.

If the specified Dependency does not contain the current section, Process Dependency succeeds immediately.

The interpreter also performs the checks described in Section 5.1 to ensure that the dependent is processing the Dependency correctly.

5.4.1. Multiple Manifest Processors

Commenté [MB51]: I feel like some of the text in this section should be provided early in the document. Please think about it.

When a system has multiple trust domains, each domain might require independent verification of authenticity or security policies. Trust domains might be divided by separation technology such as Arm TrustZone, Intel SGX, or another Trusted Execution Environment (TEE) technology. Trust domains might also be divided into separate processors and memory spaces, with a communication interface between them.

For example, an application processor may have an attached communications module that contains a processor. The communications module might require metadata signed by a specific Trust Authority for regulatory approval. This may be a different Trust Authority than the application processor.

When there are two or more trust domains, a Manifest processor might be required in each. The first Manifest processor is the normal Manifest processor as described for the Recipient in Section 6 of [I-D.ietf-suit-manifest]. The second Manifest processor only executes sections when the first Manifest processor requests it. An API interface is provided from the second Manifest processor to the first. This allows the first Manifest processor to request a limited set of operations from the second. These operations are limited to: setting Parameters, inserting an Envelope, and invoking a Manifest Command Sequence. The second Manifest processor declares a prefix to the first, which tells the first Manifest processor when it should delegate to the second. These rules are enforced by underlying separation of privilege infrastructure, such as TEEs, or physical separation.

When the first Manifest processor encounters a Dependency prefix, that informs the first Manifest processor that it should provide the second Manifest processor with the corresponding Dependency Envelope. This is done when the Dependency is fetched. The second Manifest processor immediately verifies any authentication information in the Dependency Envelope. When a Parameter is set for any Component that matches the prefix, this Parameter setting is passed to the second Manifest processor via an API. As the first Manifest processor works through the Procedure (set of Command sequences) it is executing, each time it sees a Process Dependency Command that is associated with the prefix declared by the second Manifest processor, it uses the API to ask the second Manifest processor to invoke that Dependency section instead.

This mechanism ensures that the two or more Manifest processors do not need to trust each other, except in a very limited case. When Parameter setting across trust domains is used, it must be very carefully considered. Only Parameters that do not have an effect on security properties should be allowed. The Dependency Manifest MAY control which Parameters are allowed to be set by using the Override Parameters Directive. The second Manifest processor MAY also control which Parameters may be set by the first Manifest processor by means of an ACL that lists the allowed Parameters. For example, a URI may be set by a dependent without a substantial impact on the security properties of the Manifest.

a mis en forme : Surlignage

5.5. Dependency Resolution

The Dependency Resolution Command Sequence is a container for the Commands needed to acquire and process the Dependencies of the current Manifest. All Dependency Manifests SHOULD be fetched before any Payload is fetched to ensure that all Manifests are available and authenticated before any of the (larger) Payloads are acquired.

5.6. Added and Modified Commands

All Commands are modified in that they can also target Dependencies. However, Set Component Index has a larger modification.

Command Name	Semantic of the Operation
Set Parameters	current.params[k] := v if not k in current.params for-each k,v in arg
Process Dependency	exec(current[common]); exec(current[current-segment])
Dependency Integrity	verify(current, current.params[image-digest])
Is Dependency	assert(current exists in Dependencies)
Unlink	unlink(current)

Table 1: New Commands

5.6.1. suit-directive-set-parameters

Similar to suit-directive-override-parameters, suit-directive-set-parameters allows the Manifest to configure behavior of future Directives by changing Parameters that are read by those Directives. Set Parameters is for use when Dependencies are used because it allows a Manifest to modify the behavior of its Dependencies.

Commenté [MB52]: Cite 8.4.10.3 of manifest (idem for other similar constructs)

Commenté [MB53]: This explains why this is useful for dependency, but does not explain it can't be sued for other contexts.

Available Parameters are defined in Section 8.4.8 of [I-D.ietf-suit-manifest]~~, section 8.4.8~~.

If a Parameter is already set, suit-directive-set-parameters will skip setting the Parameter to its argument. This allows dependent

Manifests to change the behavior of a Manifest, a Dependency that wishes to enforce a specific value of a Parameter MAY use suit-directive-override-parameters instead.

suit-directive-set-parameters does not specify a reporting policy.

5.6.2. suit-directive-process-dependency

Execute the Commands in the common section of the current Dependency, followed by the Commands in the equivalent section of the current Dependency. For example, if the current section is "Payload Fetch," this will execute "Common metadata" in the current Dependency, then "Payload Fetch" in the current Dependency. Once this is complete, the Command following suit-directive-process-dependency will be processed.

If the current Component index does not have an entry in the suit-dependencies map, then this Command MUST Abort.

If the current Component index has not been the target of a suit-condition-dependency-integrity, then this Command MUST Abort.

If the current Component is True, then this Directive applies to all Dependencies. If the current section is "Common metadata," then the Command sequence MUST Abort.

Commenté [MB54]: May factorize the abort part and list below the conditions as bullet list

When SUIT_Process_Dependency completes, it forwards the last status code that occurred in the Dependency.

5.6.3. suit-condition-is-dependency

Check whether the current Component index is present in the Dependency list. If the current Component is in the Dependency list, suit-condition-is-dependency succeeds. Otherwise, it fails. This can be used along with “component-id = True” to act on all Dependencies or on all non-Dependency Components. ~~See~~ Refer to Section 8 for more details.

5.6.4. suit-condition-dependency-integrity

Verify the integrity of a Dependency Manifest. When a Manifest Processor executes suit-condition-dependency-integrity, it performs the following operations:

1. Verify the signature of the Dependency's suit-authentication-wrapper.

2. Compare the Dependency's suit-authentication-wrapper digest to the dependent's suit-parameter-image-digest

3. Verify the Dependency Manifest against the Depedency's suit-authentication-wrapper digest

If any of these steps fails, the Manifest Process MUST immediately Abort.

The Manifest Processor MAY cache the results of these operations for later use from the context of the current Manifest. The Manifest Processor MUST NOT use cached results from any other Manifest context. If the Manifest Processor caches the results of these checks, it MUST eliminate this cache if any Fetch, or Copy operation targets the Dependency Manifest's Component ID.

Commenté [MB55]: Is there any TTL associated with this? I see that you mention at least two conditions to flush out the cache. That's great.

5.6.5. suit-directive-unlink

A manifest processor that supports multiple independent root manifests MUST support suit-directive-unlink. When a Component is no longer needed, the Manifest processor unlinks the Component to inform the Manifest processor that it is no longer needed.

If a Manifest is no longer needed, the Manifest Processor unlinks it. This causes the Manifest Processor to execute the suit-uninstall section of the unlinked Manifest, after which it decrements the reference count of the unlinked Manifest. The suit-uninstall section of a manifest typically contains an unlink of all its dependencies and components.

All components, including Manifests must be unlinked before deletion or overwrite. If the reference count of a component is non-zero, any command that alters that component MUST cause an immediate ~~ABORT~~Abort. Affected commands are:

- suit-directive-copy
- suit-directive-fetch
- suit-directive-write

The unlink Command decrements an implementation-defined reference counter. This reference counter MUST persist across restarts. The reference counter MUST NOT be decremented by a given Manifest more than once, and the Manifest processor must enforce this. The Manifest processor MAY choose to ignore an Unlink Directive depending on device policy.

When the reference counter of a Manifest reaches zero, the suit-uninstall Command sequence is invoked (~~see~~ Section 6).

suit-directive-unlink is OPTIONAL to implement in Manifest processors, but Manifest processors that support multiple independent Root Manifests MUST support suit-directive-unlink.

6. Uninstall

In some systems, particularly with multiple, independent, optional Components, it may be that there is a need to uninstall the Components that have been installed by a Manifest. Where this is expected, the uninstall Command sequence can provide the sequence needed to cleanly remove the Components defined by the Manifest and its Dependencies. In general, the suit-uninstall Command Sequence will contain primarily unlink Directives.

WARNING: This can cause faults where there are loose Dependencies (e.g., version range matching, ~~see~~

[I-D.ietf-suit-update-management]), since a Component can be removed while it is depended upon by another Component. To avoid Dependency faults, a Manifest author MAY use explicit Dependencies where possible, or a Manifest processor MAY track references to loose Dependencies via reference counting in the same way as explicit Dependencies, as described in Section 5.6.5.

The suit-uninstall Command Sequence is not severable, since it must always be available to enable uninstalling.

7. Staging and Installation

In order to coordinate between download and installation in different trust domains, the Update Procedure defined in Section 8.4.6 of [I-D.ietf-suit-manifest], ~~Section 8.4.6~~ is divided into two sub-procedures:

- The Staging Procedure: This procedure is responsible for dependency resolution and acquiring all payloads required for the Update to proceed. It is composed of two command sequences
  - suit-dependency-resolution
  - suit-payload-fetch
- The Installation Procedure: This procedure is responsible for verifying staged components and installing them. It is composed of:
  - suit-candidate-verification
  - suit-install

This extension is backwards compatible when used with a Manifest Processor that supports the Update Procedure but = does not support the Staging Procedure and Installation Procedure: the payload-fetch command sequence already contains suit-condition-image tests for each payload (~~see~~ [I-D.ietf-suit-manifest], ~~section~~ Section 7.3) which means that images are already validated when suit-install is invoked. This makes suit-candidate-verification OPTIONAL to implement and OPTIONAL to parse.

The Staging and Installation Procedures are only required when Staging occurs in a different trust domain to Installation.

7.1. suit-candidate-verification

This command sequence is responsible for verifying that all elements of an update are present and correct prior to installation. This is only required when Installation occurs in a trust domain different from Staging, such as an installer invoked by the bootloader.

8. Creating Manifests

This section details a set of templates for creating Manifests. These templates explain which Parameters, Commands, and orders of

Commands are necessary to achieve a stated goal.

8.1. Dependency Template

The goal of the Dependency template is to obtain, verify, and process a Dependency Manifest as appropriate.

The following Commands are added to the shared sequence:

- Set Component Index Directive (~~see~~ Section 8.4.10.1 of [I-D.ietf-suit-manifest])

- Set Parameters Directive (~~see~~ Section 5.6.1) for digest (~~see~~ Section 8.4.8.6 of [I-D.ietf-suit-manifest]). Note that the digest MUST match the SUIT_Digest in the Dependency's suit-authentication-block (~~see~~ Section 8.3 of [I-D.ietf-suit-manifest]).

The following Commands are placed into the Dependency resolution sequence:

- Set Component Index Directive (~~see~~ Section 8.4.10.1 of [I-D.ietf-suit-manifest])

- Set Parameters Directive (~~see~~ Section 5.6.1) for a URI (~~see~~ Section 8.4.8.10 of [I-D.ietf-suit-manifest])

- Fetch Directive (~~see~~ Section 8.4.10.4 of [I-D.ietf-suit-manifest])

- Dependency Integrity Condition (~~see~~ Section 5.6.4)

- Process Dependency Directive (~~see~~ Section 5.6.2)

Then, the validate sequence contains the following operations:

- Set Component Index Directive (~~see~~ Section 8.4.10.1 of [I-D.ietf-suit-manifest])

- Dependency Integrity Condition (~~see~~ Section 5.6.4)

- Process Dependency Directive (~~see~~ Section 5.6.2)

If any Dependency is declared, the dependent MUST populate all Command sequences for the current Procedure (Update or Invoke).

NOTE: Any changes made to Parameters in a Dependency persist in the dependent.

8.1.1. Integrated Dependencies

An implementer MAY choose to place a Dependency's Envelope in the Envelope of its dependent. The dependent Envelope key for the Dependency Envelope MUST be a text string. The URI for the Dependency MUST match the text string key of the dependent's Envelope key. It is RECOMMENDED to make the text string key a resolvable URI so that a Dependency Manifest that is removed from the Envelope can still be fetched.

8.2. Encrypted Manifest Template

The goal of the Encrypted Manifest template is to fetch and decrypt a Manifest so that it can be used as a Dependency. To use an encrypted Manifest, create a plaintext dependent, and add the encrypted Manifest as a Dependency. The dependent can include very little information.

NOTE: This template also requires the extensions defined in [I-D.ietf-suit-firmware-encryption].

The following Commands are added to the shared sequence:

- Set Component Index Directive (~~see~~ Section 8.4.10.1 of [I-D.ietf-suit-manifest])

- Set Parameters Directive (~~see~~ Section 5.6.1) for digest (~~see~~ Section 8.4.8.6 of [I-D.ietf-suit-manifest]). Note that the digest MUST match the SUIT_Digest in the Dependency's suit-authentication-block (~~see~~ Section 8.3 of [I-D.ietf-suit-manifest]).

The following operations are placed into the Dependency resolution block:

- Set Component Index Directive (~~see~~ Section 8.4.10.1 of [I-D.ietf-suit-manifest])

- Set Parameters Directive (~~see~~ Section 5.6.1) for

  - URI (~~see~~ Section 8.4.8.9 of [I-D.ietf-suit-manifest])

  - Encryption Info (~~See~~ [I-D.ietf-suit-firmware-encryption])

- Fetch Directive (~~see~~ Section 8.4.10.4 of [I-D.ietf-suit-manifest])

- Dependency Integrity Condition (~~see~~ Section 5.6.4)

- Process Dependency Directive (~~see~~ Section 5.6.2)

Then, the validate block contains the following operations:

- Set Component Index Directive (~~see~~ Section 8.4.10.1 of [I-D.ietf-suit-manifest])

- Check Image Match Condition (~~see~~ Section 8.4.9.2 of [I-D.ietf-suit-manifest])

- Process Dependency Directive (~~see~~ Section 5.6.2)

A plaintext Manifest and its encrypted Dependency may also form a composite Manifest (Section 8.1.1).

8.3. Overriding Encryption Info Template

The goal of overriding the Encryption Info template is to separate the role of generating encrypted Payload and Encryption Info with Key-Encryption Key addressing Section 3 of

[I-D.ietf-suit-firmware-encryption].

As an example, this template describes two manifests: - The dependent Manifest created by the Distribution System contains Encryption Info, allowing the Device to generate the Content-Encryption Key. - The dependency Manifest created by the Author contains Commands to decrypt the encrypted Payload using Encryption Info above and to validate the plaintext Payload with SUIT_Digest.

NOTE: This template also requires the extensions defined in [I-D.ietf-suit-firmware-encryption].

The following operations are placed into the Dependency resolution block of dependent Manifest:

* Set Component Index Directive (~~see~~ Section 8.4.10.1 of [I-D.ietf-suit-manifest]) pointing at dependency Manifest

* Set Parameters Directive (~~see~~ Section 5.6.1) for

  - Image Digest (~~see~~ Section 8.4.8.6 of [I-D.ietf-suit-manifest])

  - URI (~~see~~ Section 8.4.8.9 of [I-D.ietf-suit-manifest]) of dependency Manifest

* Fetch Directive (~~see~~ Section 8.4.10.4 of [I-D.ietf-suit-manifest])

* Dependency Integrity Condition (~~see~~ Section 5.6.4)

The following Commands are placed into the Fetch/Install block of dependent Manifest

* Set Component Index Directive (~~see~~ Section 8.4.10.1 of [I-D.ietf-suit-manifest]) pointing at encrypted Payload

* Set Parameters Directive (~~see~~ Section 5.6.1) for

  - URI (~~see~~ Section 8.4.8.9 of [I-D.ietf-suit-manifest])

* Set Component Index Directive (~~see~~ Section 8.4.10.1 of [I-D.ietf-suit-manifest]) pointing at dependency Manifest

* Set Parameters Directive (~~see~~ Section 5.6.1) for

  - Encryption Info (~~See~~ [I-D.ietf-suit-firmware-encryption])

* Process Dependency Directive (~~see~~ Section 5.6.2)

The following Commands are placed into the same block of dependency Manifest:

* Set Component Index Directive (~~see~~ Section 8.4.10.1 of [I-D.ietf-suit-manifest]) pointing at encrypted Payload

* Fetch Directive (~~see~~ Section 8.4.10.4 of [I-D.ietf-suit-manifest])

- Set Component Index Directive (~~see~~ Section 8.4.10.1 of [I-D.ietf-suit-manifest]) pointing at to be decrypted Payload

- Override Parameters Directive (~~see~~ Section 8.4.10.3 of [I-D.ietf-suit-manifest]) for

  - Source Component (~~see~~ Section 8.4.8.11 of [I-D.ietf-suit-manifest]) pointing at encrypted Payload

- Copy Directive (~~see~~ Section 8.4.10.5 of [I-D.ietf-suit-manifest]) consuming the Encryption Info above

The Distribution System can Set the Parameter URI in the Fetch/ Install block of dependent Manifest if it wants to overwrite the URI of encrypted Payload.

Commenté [MB56]: Please check

Because the Author and the Distribution System have different roles and MAY be separate entities, it is highly RECOMMENDED to leverage permissions (~~see~~ Section 9 of [I-D.ietf-suit-manifest]). For example, ~~The~~ the Device can protect itself from attacker who breaches the Distribution System by allowing only the Author's Manifest to modify the Component of (to be) decrypted Payload.

Commenté [MB57]: Weird use of normative language

Commenté [MB58]: Not sure «highly» is needed

8.4. Operating on Multiple Components

In order to produce compact encoding, it is efficient to perform operations on multiple Components simultaneously. Because Dependency Manifests and Component Images are processed at different times, there is a mechanism to distinguish between these elements: suit-condition-is-dependency. This can be used with suit-directive-try-each to perform operations ~~just~~ on Dependency Manifests or ~~just~~ on Component Images.

For example, to fetch all Dependency Manifests, the following Commands are added to the Dependency resolution block:

- Set Component Index Directive (~~see~~ Section 8.4.10.1 of [I-D.ietf-suit-manifest])

- Set Parameters Directive (~~see~~ Section 5.6.1) for a URI (~~see~~ Section 8.4.8.9 of [I-D.ietf-suit-manifest])

- Set Component Index Directive, with argument "True" (~~see~~ Section 8.4.10.1 of [I-D.ietf-suit-manifest])

- Try Each Directive

  - Sequence 0

    - Condition Is Dependency Manifest

    - Fetch

    - Dependency Integrity Condition (~~see~~ Section 5.6.4)

    - Process Dependency
  - Sequence 1 (Empty; no Commands, succeeds immediately)

Another example is to fetch and validate all Component Images. The Image fetch sequence contains the following Commands:

- Set Component Index Directive (~~see~~ Section 8.4.10.1 of [I-D.ietf-suit-manifest])
- Set Parameters Directive (~~see~~ Section 5.6.1) for a URI (~~see~~ Section 8.4.8.9 of [I-D.ietf-suit-manifest])
- Set Component Index Directive, with argument "True" (~~see~~ Section 8.4.10.1 of [I-D.ietf-suit-manifest])
- Try Each Directive
  - Sequence 0
    - Condition Is Dependency Manifest
    - Process Dependency
  - Sequence 1
    - Fetch
    - Condition Image Match

When some Components are "installed" or "loaded" it is more productive to use lists of Component indices rather than Component Index = True. For example, to install several Components, the following Commands should be placed in the Image Install Sequence:

- Set Component Index Directive (~~see~~ Section 8.4.10.1 of [I-D.ietf-suit-manifest])
- Set Parameters Directive (~~see~~ Section 5.6.1) for the Source Component (~~see~~ Section 8.4.8.11 of [I-D.ietf-suit-manifest])
- Set Component Index Directive, with argument containing list of destination Component indices (~~see~~ Section 8.4.10.1 of [I-D.ietf-suit-manifest])
- Copy
- Set Component Index Directive, with argument containing list Dependency Component indices (~~see~~ Section 8.4.10.1 of [I-D.ietf-suit-manifest])
- Process Dependency

9. IANA Considerations

IANA is requested to allocate the following numbers in the listed registries created by draft-ietf-suit-manifest:

9.1. SUIT Envelope Elements

Label	Name	Reference
15	Dependency Resolution	Section 5.5
18	Candidate Verification	Section 7.1

Table 2

9.2. SUIT Manifest Elements

Label	Name	Reference
5	Manifest Component ID	Section 5.2.1
15	Dependency Resolution	Section 5.5
24	Uninstall	Section 6

Table 3

9.3. SUIT Common Elements

Label	Name	Reference
1	Dependencies	Section 5.2.2

Table 4

9.4. SUIT Commands

Label	Name	Reference
7	Dependency Integrity	Section 5.6.4
8	Is Dependency	Section 5.6.3
11	Process Dependency	Section 5.6.2
19	Set Parameters	Section 5.6.1
33	Unlink	Section 5.6.5

Table 5

Commenté [MB59]: Any reason the labels wern't assigned to be consistent with flow use?

10. Security Considerations

This document is about a Manifest format protecting and describing how to retrieve, install, and invoke Images and as such it is part of a larger solution for delivering software updates to devices. A detailed security treatment can be found in the architecture

[RFC9019] and in the information model [RFC9124] documents.

Commenté [MB60]: I was expecting at least a reminder of cons specific to the multi trust domain case. Adding specific pointers where this is discussed in 9124/9019 would be a minimum. Thanks.

11. References

Commenté [MB61]: No Manageability/ops considerations are included, unfortunately.

11.1. Normative References

[I-D.ietf-suit-firmware-encryption]
Tschofenig, H., Housley, R., Moran, B., Brown, D., and K. Takayama, "Encrypted Payloads in SUIT Manifests", Work in Progress, Internet-Draft, draft-ietf-suit-firmware-encryption-23, 29 January 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-suit-firmware-encryption-23>.

[I-D.ietf-suit-manifest]
Moran, B., Tschofenig, H., Birkholz, H., Zandberg, K., and O. Rønningstad, "A Concise Binary Object Representation (CBOR)-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest", Work in Progress, Internet-Draft, draft-ietf-suit-manifest-33, 24 February 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-suit-manifest-33>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

11.2. Informative References

[I-D.ietf-iotops-7228bis]
Bormann, C., Ersue, M., Keränen, A., and C. Gomez, "Terminology for Constrained-Node Networks", Work in Progress, Internet-Draft, draft-ietf-iotops-7228bis-01, 8 January 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-iotops-7228bis-01>.

[I-D.ietf-suit-update-management]
Moran, B. and K. Takayama, "Update Management Extensions for Software Updates for Internet of Things (SUIT) Manifests", Work in Progress, Internet-Draft, draft-ietf-suit-update-management-07, 8 July 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-suit-update-management-07>.

[I-D.ietf-teep-architecture]
Pei, M., Tschofenig, H., Thaler, D., and D. M. Wheeler, "Trusted Execution Environment Provisioning (TEEP) Architecture", Work in Progress, Internet-Draft, draft-ietf-teep-architecture-19, 24 October 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-teep-architecture-19>.

[RFC6024] Reddy, R. and C. Wallace, "Trust Anchor Management Requirements", RFC 6024, DOI 10.17487/RFC6024, October 2010, <https://www.rfc-editor.org/rfc/rfc6024>.

[RFC9019] Moran, B., Tschofenig, H., Brown, D., and M. Meriac, "A Firmware Update Architecture for Internet of Things", RFC 9019, DOI 10.17487/RFC9019, April 2021, <https://www.rfc-editor.org/rfc/rfc9019>.

[RFC9124] Moran, B., Tschofenig, H., and H. Birkholz, "A Manifest Information Model for Firmware Updates in Internet of Things (IoT) Devices", RFC 9124, DOI 10.17487/RFC9124, January 2022, <https://www.rfc-editor.org/rfc/rfc9124>.

Appendix A. A. Full CDDL

Commenté [MB62]: Serialization OK, but a yang structures can be used to model the structure of the manifest.

To be valid, the following CDDL MUST be appended to the SUIT Manifest CDDL. The SUIT CDDL is defined in Appendix A of [I-D.ietf-suit-manifest]

```
$$SUIT_Envelope_Extensions //=
    (suit-delegation => bstr .cbor SUIT_Delegation)
$$SUIT_Envelope_Extensions //= (
    suit-integrated-dependency-key => bstr .cbor SUIT_Envelope)

SUIT_Delegation = [ + [ + bstr .cbor CWT ] ]

CWT = SUIT_Authentication_Block

$$SUIT_Manifest_Extensions //=
    (suit-manifest-component-id => SUIT_Component_Identifier)

$$SUIT_severable-members-extensions //=
    (suit-dependency-resolution => bstr .cbor SUIT_Command_Sequence)

$$SUIT_severable-members-extensions //=
    (suit-candidate-verification => bstr .cbor SUIT_Command_Sequence)

$$unseverable-manifest-member-extensions //=
    (suit-uninstall => bstr .cbor SUIT_Command_Sequence)

suit-integrated-dependency-key = tstr

$$severable-manifest-members-choice-extensions //= (
    suit-dependency-resolution =>
        bstr .cbor SUIT_Command_Sequence / SUIT_Digest)

$$SUIT_Common-extensions //= (
    suit-dependencies => SUIT_Dependencies
)
SUIT_Dependencies = {
    + uint => SUIT_Dependency_Metadata
}
SUIT_Dependency_Metadata = {
    ? suit-dependency-prefix => SUIT_Component_Identifier
    * $$SUIT_Dependency_Extensions
}
```

```
SUIT_Condition //= (
    suit-condition-dependency-integrity, SUIT_Rep_Policy)
SUIT_Condition //= (
    suit-condition-is-dependency, SUIT_Rep_Policy)
SUIT_Directive //= (
    suit-directive-process-dependency, SUIT_Rep_Policy)
SUIT_Directive //= (suit-directive-set-parameters,
    {+ $$SUIT_Parameters})
SUIT_Directive //= (
    suit-directive-unlink, SUIT_Rep_Policy)

suit-manifest-component-id = 5

suit-delegation = 1
suit-dependency-resolution = 15
suit-candidate-verification = 18
suit-uninstall = 24

suit-dependencies = 1

suit-dependency-prefix = 1

suit-condition-dependency-integrity     = 7
suit-condition-is-dependency            = 8
suit-directive-process-dependency       = 11
suit-directive-set-parameters           = 19
suit-directive-unlink                   = 33
```

Appendix B. B. Examples

The following examples demonstrate a small subset of the functionalities in this document.

The examples are signed using the following ECDSA secp256r1 key:

```
-----BEGIN PRIVATE KEY-----
MIGHAgEAMBMGByqGSM49AgEGCCqGSM49AwEHBG0wawIBAQQgApZYjZCUGLM50VBC
CjYStX+09jGmnyJPrpDLTz/hiXOhRANCAASEloEarguqq9JhVxie7NomvqqL8Rtv
P+bitWWchdvArTsfKktsCYExwKNtrNHXi9OB3N+wnAUtszmR23M4tKiW
-----END PRIVATE KEY-----
```

The corresponding public key can be used to verify these examples:

```
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEhJaBGq4LqqvSYVcYnuzaJr6qi/Eb
bz/m4rVlnIXbwK07HypLbAmBMcCjbazR14vTgdzfsJwFLbM5kdtzOLSolg==
-----END PUBLIC KEY-----
```

Each example uses SHA256 as the digest function.

B.1. Example 0: Process Dependency

This example uses functionalities:

- manifest component id
- dependency resolution

- process dependency

The dependency Manifest:

```
/ SUIT_Envelope_Tagged / 107({
  / authentication-wrapper / 2: << [
    << [
      / digest-algorithm-id: / -16 / SHA256 /,
      / digest-bytes: /
h'A2FFB59E9F1A29D20BF655BC1DE909CB7EDD972A6C09D50FC42983778670715E'
    ] >>,
    << / COSE_Sign1_Tagged / 18([
      / protected: / << {
        / algorithm-id / 1: -7 / ES256 /
      } >>,
      / unprotected: / {},
      / payload: / null,
      / signature: /
h'A506F1647E3A9E0F54A07F303443F33E3CFA28520BE1E93C467CD8B14954E460C604A76
23F146D833B6F0A2454095855573C48B18570066FA7472077313E80CE'
    ]) >>
  ] >>,
  / manifest / 3: << {
    / manifest-version / 1: 1,
    / manifest-sequence-number / 2: 0,
    / common / 3: << {
      / dependencies / 1: {
        / component-index / 1: {
          / dependency-prefix / 1: [
            'dependent.suit'
          ]
        }
      },
      / components / 2: [
        ['10']
      ]
    } >>,
    / manifest-component-id / 5: [
      'depending.suit'
    ],
    / invoke / 9: << [
      / directive-set-component-index / 12, 0,
      / directive-override-parameters / 20, {
        / parameter-invoke-args / 23: 'cat 00 10'
      },
      / directive-invoke / 23, 15
    ] >>,
    / dependency-resolution / 15: << [
      / directive-set-component-index / 12, 1,
      / directive-override-parameters / 20, {
        / parameter-image-digest / 3: << [
          / digest-algorithm-id: / -16 / SHA256 /,
          / digest-bytes: /
h'2EEEC4ACEC877EE13D8B52DB16C4390C93E5D84FD9F25AEAE0717B861BE0C4A2'
        ] >>,
        / parameter-image-size / 14: 190,
        / parameter-uri / 21: "http://example.com/dependent.suit"
      },
      / directive-fetch / 21, 2,
      / condition-image-match / 3, 15
    ] >>,
    / install / 20: << [
      / directive-set-component-index / 12, 1,
      / directive-override-parameters / 20, {
        / parameter-image-digest / 3: << [
          / digest-algorithm-id: / -16 / SHA256 /,
          / digest-bytes: /
h'0F02CAF6D3E61920D36BF3CEA7F862A13BB8FB1F09C3F4C29B121FEAB78EF3D8'
        ] >>
      },
      / condition-dependency-integrity / 7, 15,
      / directive-process-dependency / 11, 0,

      / directive-set-component-index / 12, 0,
      / directive-override-parameters / 20, {
        / parameter-content / 18: ' in multiple trust domains'
      },
      / directive-write / 18, 15
    ] >>
  } >>
})
```

Total size of Envelope with COSE authentication object: 373

```
   D86BA2025873825824822F5820A2FFB59E9F1A29D20BF655BC1DE909CB7E
   DD972A6C09D50FC42983778670715E584AD28443A10126A0F65840A506F1
   647E3A9E0F54A07F303443F33E3CFA28520BE1E93C467CD8B14954E460C6
   04A7623F146D833B6F0A2454095855573C48B18570066FA7472077313E80
   CE0358F9A70101020003581CA201A101A101814E646570656E64656E742E
   7375697402818142313005814E646570656E64696E672E73756974095286
   0C0014A11749636174203030203130170F0F5857880C0114A3035824822F
   58202EEEC4ACEC877EE13D8B52DB16C4390C93E5D84FD9F25AEAE0717B86
   1BE0C4A20E18BE157821687474703A2F2F6578616D706C652E636F6D2F64
   6570656E64656E742E737569741502030F1458538E0C0114A1035824822F
   58200F02CAF6D3E61920D36BF3CEA7F862A13BB8FB1F09C3F4C29B121FEA
   B78EF3D8070F0B000C0014A112581A20696E206D756C7469706C65207472
   75737420646F6D61696E73120F
```

The dependent Manifest (fetched from "https://example.com/dependent.suit"):

```
/ SUIT_Envelope_Tagged / 107({
  / authentication-wrapper / 2: << [
    << [
      / digest-algorithm-id: / -16 / SHA256 /,
      / digest-bytes: /
h'0F02CAF6D3E61920D36BF3CEA7F862A13BB8FB1F09C3F4C29B121FEAB78EF3D8'
    ] >>,
    << / COSE_Sign1_Tagged / 18([
      / protected: / << {
        / algorithm-id / 1: -7 / ES256 /
      } >>,
      / unprotected: / {},
      / payload: / null,
      / signature: /
h'D0703EA193E12381A66FFADEF2F0949711CFE05ED2322818D73D19F2BBD91BE5C52F160
4B45C405E96B0642F3D49B2D7C6E3B2C0B40030BDDFBD27AF930B1F8B'
    ]) >>
  ] >>,
  / manifest / 3: << {
    / manifest-version / 1: 1,
    / manifest-sequence-number / 2: 0,
    / common / 3: << {
      / components / 2: [
        ['00']
      ]
    } >>,
    / manifest-component-id / 5: [
      'dependent.suit'
    ],
    / invoke / 9: << [
      / directive-override-parameters / 20, {
        / parameter-invoke-args / 23: 'cat 00'
      },
      / directive-invoke / 23, 15
    ] >>,
    / install / 20: << [
      / directive-override-parameters / 20, {
        / parameter-content / 18: 'hello world'
      },
      / directive-write / 18, 15
    ] >>
  } >>
})
```

Total size of Envelope with COSE authentication object: 190

```
   D86BA2025873825824822F58200F02CAF6D3E61920D36BF3CEA7F862A13B
   B8FB1F09C3F4C29B121FEAB78EF3D8584AD28443A10126A0F65840D0703E
   A193E12381A66FFADEF2F0949711CFE05ED2322818D73D19F2BBD91BE5C5
   2F1604B45C405E96B0642F3D49B2D7C6E3B2C0B40030BDDFBD27AF930B1F
   8B035842A6010102000347A102818142303005814E646570656E64656E74
   2E73756974094D8414A11746636174203030170F14528414A1124B68656C
   6C6F20776F726C64120F
```

B.2. Example 1: Integrated Dependency

* manifest component id

* dependency resolution

* process dependency

* integrated dependency

```
/ SUIT_Envelope_Tagged / 107({
  / authentication-wrapper / 2: << [
    << [
      / digest-algorithm-id: / -16 / SHA256 /,
      / digest-bytes: /
h'6391CBC36495B9C87AC3EC841DB124DABD8D3C9FE2DEEFE16569AFC349E7DDB2'
    ] >>,
    << / COSE_Sign1_Tagged / 18([
      / protected: / << {
        / algorithm-id / 1: -7 / ES256 /
      } >>,
      / unprotected: / {},
      / payload: / null,
      / signature: /
h'517250281E6567FF9DF519CF9D76A440D86DFEB65B505D180D7D794FEC67823FA0E98EB
C526FBC985777EAB4E2FFE813A44F205C015AEB3FA842F33E37B52716'
    ]) >>
  ] >>,
  / manifest / 3: << {
    / manifest-version / 1: 1,
    / manifest-sequence-number / 2: 0,
    / common / 3: << {
      / dependencies / 1: {
        / component-index / 1: {
          / dependency-prefix / 1: [
            'dependent.suit'
          ]
        }
      },
      / components / 2: [
        ['10']
      ]
    } >>,

    / manifest-component-id / 5: [
      'depending.suit'
    ],
    / invoke / 9: << [
      / directive-set-component-index / 12, 0,
      / directive-override-parameters / 20, {
        / parameter-invoke-args / 23: 'cat 00 10'
      },
      / directive-invoke / 23, 15
    ] >>,
    / dependency-resolution / 15: << [
      / directive-set-component-index / 12, 1,
      / directive-override-parameters / 20, {
        / parameter-image-digest / 3: << [
          / digest-algorithm-id: / -16 / SHA256 /,
          / digest-bytes: /
h'2EEEC4ACEC877EE13D8B52DB16C4390C93E5D84FD9F25AEAE0717B861BE0C4A2'
        ] >>,
        / parameter-image-size / 14: 190,
        / parameter-uri / 21: "#dependent.suit"
      },
      / directive-fetch / 21, 2,
      / condition-image-match / 3, 15
    ] >>,
    / install / 20: << [
      / directive-set-component-index / 12, 1,
      / directive-process-dependency / 11, 0,

      / directive-set-component-index / 12, 0,
      / directive-override-parameters / 20, {
        / parameter-content / 18: ' in multiple trust domains'
      },
      / directive-write / 18, 15
    ] >>
  } >>,
  "#dependent.suit":
h'D86BA2025873825824822F58200F02CAF6D3E61920D36BF3CEA7F862A13BB8FB1F09C3F
4C29B121FEAB78EF3D8584AD28443A10126A0F65840D0703EA193E12381A66FFADEF2F094
9711CFE05ED2322818D73D19F2BBD91BE5C52F1604B45C405E96B0642F3D49B2D7C6E3B2C
0B40030BDDFBD27AF930B1F8B035842A6010102000347A102818142303005814E64657065
6E64656E742E73756974094D8414A11746636174203030170F14528414A1124B68656C6C6
F20776F726C64120F'
})
```

   Total size of Envelope with COSE authentication object: 519

   Envelope with COSE authentication object:

```
   D86BA3025873825824822F58206391CBC36495B9C87AC3EC841DB124DABD
   8D3C9FE2DEEFE16569AFC349E7DDB2584AD28443A10126A0F65840517250
   281E6567FF9DF519CF9D76A440D86DFEB65B505D180D7D794FEC67823FA0
   E98EBC526FBC985777EAB4E2FFE813A44F205C015AEB3FA842F33E37B527
   160358BBA70101020003581CA201A101A101814E646570656E64656E742E
   737569740281814231300581 4E646570656E64696E672E73756974095286
   0C0014A11749636174203030203130170F0F5844880C0114A3035824822F
   58202EEEC4ACEC877EE13D8B52DB16C4390C93E5D84FD9F25AEAE0717B86
   1BE0C4A20E18BE156F23646570656E64656E742E73756974150203 0F1458
   288A0C010B000C0014A112581A20696E206D756C7469706C652074727573
   7420646F6D61696E73120F6F23646570656E64656E742E7375697458BED8
   6BA2025873825824822F58200F02CAF6D3E61920D36BF3CEA7F862A13BB8
   FB1F09C3F4C29B121FEAB78EF3D8584AD28443A10126A0F65840D0703EA1
   93E12381A66FFADEF2F0949711CFE05ED2322818D73D19F2BBD91BE5C52F
   1604B45C405E96B0642F3D49B2D7C6E3B2C0B40030BDDFBD27AF930B1F8B
   035842A6010102000347A102818142303005814E646570656E64656E742E
   73756974094D8414A11746636174203030170F14528414A1124B68656C6C
   6F20776F726C64120F
```

Authors' Addresses

   Brendan Moran
   Arm Limited
   Email: brendan.moran.ietf@gmail.com

   Ken Takayama
   SECOM CO., LTD.
   Email: ken.takayama.ietf@gmail.com