SUIT
Internet-Draft
Intended status: Standards Track
Expires: December 11, 2020

B. Moran
H. Tschofenig
Arm Limited
H. Birkholz
Fraunhofer SIT
K. Zandberg
Inria

June 09, 2020

A Concise Binary Object Representation (CBOR)-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest

draft-ietf-suit-manifest-07

Abstract

This specification describes the format of a manifest. A manifest is a bundle of metadata about the firmware for an IoT device, where to find the firmware, the devices to which it applies, and cryptographic information protecting the manifest. Firmware updates and secure boot both tend to use sequences of common operations, so the manifest encodes those sequences of operations, rather than declaring the metadata. The manifest also serves as a building block for secure boot.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 11, 2020.

Copyright Notice

Table of Contents

1. Introduction

A firmware update mechanism is an essential security feature for IoT devices to deal with vulnerabilities. While the transport of firmware images to the devices themselves is important there are already various techniques available. Equally important is the inclusion of metadata about the conveyed firmware image (in the form of a manifest) and the use of a security wrapper to provide end-to-end security protection to detect modifications and (optionally) to

make reverse engineering more difficult. End-to-end security allows the author, who builds the firmware image, to be sure that no other party (including potential adversaries) can install firmware updates on IoT devices without adequate privileges. For confidentiality protected firmware images it is additionally required to encrypt the firmware image. Starting security protection at the author is a risk mitigation technique so firmware images and manifests can be stored on untrusted repositories; it also reduces the scope of a compromise of any repository or intermediate system to be no worse than a denial of service.

A manifest is a bundle of metadata about the firmware for an IoT device, where to find the firmware, the devices to which it applies, and cryptographic information protecting the manifest.

This specification defines the SUIT manifest format and it is intended to meet several goals:

- Meet the requirements defined in [I-D.ietf-suit-information-model].

- Simple to parse on a constrained node

- Simple to process on a constrained node

- Compact encoding

- Comprehensible by an intermediate system

- Expressive enough to enable advanced use cases on advanced nodes

- Extensible

The SUIT manifest can be used for a variety of purposes throughout its lifecycle, such as:

- the Firmware Author to reason about releasing a firmware.

- the Network Operator to reason about compatibility of a firmware.

- the Device Operator to reason about the impact of a firmware.

- the Device Operator to manage distribution of firmware to devices.

- the Plant Manager to reason about timing and acceptance of firmware updates.

- the device to reason about the authority & authenticity of a firmware prior to installation.

- the device to reason about the applicability of a firmware.

- the device to reason about the installation of a firmware.

- the device to reason about the authenticity & encoding of a firmware at boot.

Each of these uses happens at a different stage of the manifest lifecycle, so each has different requirements.

It is assumed that the reader is familiar with the high-level firmware update architecture [I-D.ietf-suit-architecture] and the threats, requirements, and user stories in [I-D.ietf-suit-information-model].

A core concept of the SUIT manifest specification are commands. Commands are either conditions or directives used to define the required behavior. Conceptually, a sequence of commands is like a script but the used language is tailored to software updates and secure boot.

The available commands support simple steps, such as copying a firmware image from one place to another, checking that a firmware image is correct, verifying that the specified firmware is the correct firmware for the device, or unpacking a firmware. By using these steps in different orders and changing the parameters they use, a broad range of use cases can be supported. The SUIT manifest uses this observation to heavily optimize metadata for consumption by constrained devices.

While the SUIT manifest is informed by and optimized for firmware update and secure boot use cases, there is nothing in the [I-D.ietf-suit-information-model] that restricts its use to only those use cases. Other use cases include the management of trusted applications in a Trusted Execution Environment (TEE), see [I-D.ietf-teep-architecture].

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.

The following terminology is used throughout this document:

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

- Payload: A piece of information to be delivered. Typically Firmware for the purposes of SUIT.

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

- Manifest: A manifest is a bundle of metadata about the firmware for an IoT device, where to find the firmware, the devices to which it applies, and cryptographic information protecting the manifest.

- Envelope: A container with the manifest, an authentication wrapper, authorization information, and severed fields.

- 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, typically an IoT device, that receives a manifest.

- 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 Execution: A process by which a system ensures that only trusted code is executed, for example secure boot.

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

3. How to use this Document

This specification covers four aspects of firmware update:

- Section 4 describes the device constraints, use cases, and design principles that informed the structure of the manifest.

- Section 6 describes what actions a manifest processor should take.

- Section 7 describes the process of creating a manifest.

- Section 9 specifies the content of the manifest and the envelope.

To implement an updatable device, see Section 6 and Section 9. To implement a tool that generates updates, see Section 7 and Section 9.

The IANA consideration section, see Section 13, provides instructions to IANA to create several registries. This section also provides the CBOR labels for the structures defined in this document.

The complete CDDL description is provided in Appendix A, examples are given in Appendix B and a design rational is offered in Appendix C. Finally, Appendix D gives a summarize of the mandatory-to-implement features of this specification.

4. Background

Distributing firmware updates to diverse devices with diverse trust anchors in a coordinated system presents unique challenges. Devices have a broad set of constraints, requiring different metadata to make appropriate decisions. There may be many actors in production IoT systems, each of whom has some authority. Distributing firmware in such a multi-party environment presents additional challenges. Each party requires a different subset of data. Some data may not be accessible to all parties. Multiple signatures may be required from parties with different authorities. This topic is covered in more depth in [I-D.ietf-suit-architecture]. The security aspects are described in [I-D.ietf-suit-information-model].

4.1. IoT Firmware Update Constraints

The various constraints of IoT devices and the range of use cases that need to be supported create a broad set of urequirements. For example, devices with:

- limited processing power and storage may require a simple representation of metadata.

- bandwidth constraints may require firmware compression or partial update support.

- bootloader complexity constraints may require simple selection between two bootable images.

- small internal storage may require external storage support.

- multiple microcontrollers may require coordinated update of all applications.

- large storage and complex functionality may require parallel update of many software components.

- extra information may need to be conveyed in the manifest in the earlier stages of the device lifecycle before those data items are stripped when the manifest is delivery to a constrained device.

Supporting the requirements introduced by the constraints on IoT devices requires the flexibility to represent a diverse set of possible metadata, but also requires that the encoding is kept simple.

4.2. Update Workflow Model

There are several fundamental assumptions that inform the model of the firmware update workflow:

- Compatibility must be checked before any other operation is performed.

- All dependency manifests should be present before any payload is fetched.

- In some applications, payloads must be fetched and validated prior to installation.

There are several fundamental assumptions that inform the model of the secure boot workflow:

- Compatibility must be checked before any other operation is performed.

- All dependencies and payloads must be validated prior to loading.

- All loaded images must be validated prior to execution.

Based on these assumptions, the manifest is structured to work with a pull parser, where each section of the manifest is used in sequence. The expected workflow for a device installing an update can be broken down into five steps:

1. Verify the signature of the manifest.

2. Verify the applicability of the manifest.

3. Resolve dependencies.

4. Fetch payload(s).

5. Install payload(s).

When installation is complete, similar information can be used for validating and running images in a further three steps:

1. Verify image(s).

2. Load image(s).

3. Run image(s).

If verification and running is implemented in a bootloader, then the bootloader must also verify the signature of the manifest and the applicability of the manifest in order to implement secure boot workflows. The bootloader may add its own authentication, e.g. a MAC, to the manifest in order to prevent further verifications.

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.

5. Severed Fields

Because the manifest can be used by different actors at different times, some parts of the manifest can be removed without affecting later stages of the lifecycle. This is called "Severing." Severing of information is achieved by separating that information from the signed container so that removing it does not affect the signature. This means that ensuring authenticity of severable parts of the manifest is a requirement for the signed portion of the manifest. Severing some parts makes it possible to discard parts of the manifest that are no longer necessary. This is important because it allows the storage used by the manifest to be greatly reduced. For example, no text size limits are needed if text is removed from the manifest prior to delivery to a constrained device.

Elements are made severable by removing them from the manifest, encoding them in a bstr, and placing a SUIT_Digest of the bstr in the manifest so that they can still be authenticated. The SUIT_Digest typically consumes 4 bytes more than the size of the raw digest, therefore elements smaller than (Digest Bits)/8 + 4 should never be severable. Elements larger than (Digest Bits)/8 + 4 may be severable, while elements that are much larger than (Digest Bits)/8 + 4 should be severable.

Because of this, all command sequences in the manifest are encoded in a bstr so that there is a single code path needed for all command sequences.

6. Interpreter Behavior

This section describes the behavior of the manifest interpreter and focuses primarily on interpreting commands in the manifest. However, there are several other important behaviors of the interpreter: encoding version detection, rollback protection, and authenticity verification are chief among these.

6.1. Interpreter Setup

Prior to executing any command sequence, the interpreter or its host application MUST inspect the manifest version field and fail when it encounters an unsupported encoding version. Next, the interpreter or its host application MUST extract the manifest sequence number and perform a rollback check using this sequence number. The exact logic of rollback protection may vary by application, but it has the following properties:

- Whenever the interpreter can choose between several manifests, it MUST select the latest valid, authentic manifest.

- If the latest valid, authentic manifest fails, it MAY select the next latest valid, authentic manifest.

Here, valid means that a manifest has a supported encoding version and it has not been excluded for other reasons. Reasons for excluding typically involve first executing the manifest and may include:

- Test failed (e.g. Vendor ID/Class ID).

- Unsupported command encountered.

- Unsupported parameter encountered.

- Unsupported component ID encountered.

- Payload not available.

- Dependency not available.

- Application crashed when executed.

- Watchdog timeout occurred.

- Dependency or Payload verification failed.

These failure reasons MAY be combined with retry mechanisms prior to marking a manifest as invalid.

Following these initial tests, the interpreter clears all parameter storage. This ensures that the interpreter begins without any leaked data.

6.2. Required Checks

The RECOMMENDED process is to verify the signature of the manifest prior to parsing/executing any section of the manifest. This guards the parser against arbitrary input by unauthenticated third parties, but it costs extra energy when a device receives an incompatible manifest.

A device MAY choose to parse and execute only the SUIT_Common section of the manifest prior to signature verification, if - it expects to receive many incompatible manifests, and - it has power budget that makes signature verification undesirable.

The guidelines in Creating Manifests (Section 7) require that the common section contains the applicability checks, so this section is sufficient for applicability verification. The manifest parser MUST NOT execute any command with side-effects outside the parser (for example, Run, Copy, Swap, or Fetch commands) prior to authentication and any such command MUST result in an error.

Once a valid, authentic manifest has been selected, the interpreter MUST examine the component list and verify that its maximum number of components is not exceeded and that each listed component ID is supported.

For each listed component, the interpreter MUST provide storage for the supported parameters. If the interpreter does not have sufficient temporary storage to process the parameters for all components, it MAY process components serially for each command sequence. See Section 6.5 for more details.

The interpreter SHOULD check that the common section contains at least one vendor ID check and at least one class ID check.

If the manifest contains more than one component, each command sequence MUST begin with a Set Current Component command.

If a dependency is specified, then the interpreter MUST perform the following checks:

1. At the beginning of each section in the dependent: all previous sections of each dependency have been executed.

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 reject the manifest.

6.3. Interpreter Fundamental Properties

The interpreter has a small set of design goals:

1. Executing an update MUST either result in an error, or a verifiably correct system state.

2. Executing a secure boot MUST either result in an error, or a booted system.

3. Executing the same manifest on multiple devices MUST result in the same system state.

NOTE: when using A/B images, the manifest functions as two (or more) logical manifests, each of which applies to a system in a particular starting state. With that provision, design goal 3 holds.

6.4. Abstract Machine Description

The heart of the manifest is the list of commands, which are processed by an interpreter. This interpreter can be modeled as a simple abstract machine. This machine consists of several data storage locations that are modified by commands.

There are two types of commands, namely those that modify state (directives) and those that perform tests (conditions). Parameters are used as the inputs to commands. Some directives offer control flow operations. Directives target a specific component. A component is a unit of code or data that can be targeted by an update. Components are identified by a Component Index, i.e. arrays of binary strings.

The following table describes the behavior of each command. "params" represents the parameters for the current component or dependency.

Command Name	Semantic of the Operation

Check Vendor Identifier	binary-match(component, params[vendor-id])
Check Class Identifier	binary-match(component, params[class-id])
Verify Image	binary-match(digest(component), params[digest])
Set Component Index	component := components[arg]
Override Parameters	params[k] := v for k,v in arg
Set Dependency Index	dependency := dependencies[arg]
Set Parameters	params[k] := v if not k in params for k,v in arg
Process Dependency	exec(dependency[common]); exec(dependency [current-segment])
Run	run(component)
Fetch	store(component, fetch(params[uri]))
Use Before	assert(now() < arg)
Check Component Offset	assert(offsetof(component) == arg)
Check Device Identifier	binary-match(component, params[device-id])
Check Image Not Match	not binary-match(digest(component), params[digest])
Check Minimum Battery	assert(battery >= arg)
Check Update Authorized	assert(isAuthorized())
Check Version	assert(version_check(component, arg))
Abort	assert(0)

Try Each	break if exec(seq) is not error for seq in arg
Copy	store(component, params[src-component])
Swap	swap(component, params[src-component])
Wait For Event	until event(arg), wait
Run Sequence	exec(arg)
Run with Arguments	run(component, arg)

6.5. Serialized Processing Interpreter

Because each manifest has a list of components and a list of components defined by its dependencies, it is possible for the manifest processor to handle one component at a time, traversing the manifest tree once for each listed component. In this mode, the interpreter ignores any commands executed while the component index is not the current component. This reduces the overall volatile storage required to process the update so that the only limit on number of components is the size of the manifest. However, this approach requires additional processing power.

6.6. Parallel Processing Interpreter

Advanced devices may make use of the Strict Order parameter and enable parallel processing of some segments, or it may reorder some segments. To perform parallel processing, once the Strict Order parameter is set to False, the device may fork a process for each command until the Strict Order parameter is returned to True or the command sequence ends. Then, it joins all forked processes before continuing processing of commands. To perform out-of-order processing, a similar approach is used, except the device consumes all commands after the Strict Order parameter is set to False, then it sorts these commands into its preferred order, invokes them all, then continues processing.

Under each of these scenarios the parallel processing must halt:

- Set Parameters.

- Override Parameters.

- Set Strict Order = True.

- Set Dependency Index.

- Set Component Index.

To perform more useful parallel operations, sequences of commands may be collected in a suit-directive-run-sequence. Then, each of these sequences may be run in parallel. Each sequence defaults to Strict Order = True. To isolate each sequence from each other sequence, each sequence must declare a single target component. Set Component Index is not permitted inside this sequence.

6.7. Processing Dependencies

As described in Section 6.2, 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 interpreter loads the dependency identified by the Current Dependency Index. The interpreter first executes the common-sequence section of the identified dependency, then it executes the section of the dependency that corresponds to the currently executing section of the dependent.

The interpreter also performs the checks described in Section 6.2 to ensure that the dependent is processing the dependency correctly.

7. Creating Manifests

Manifests are created using tools for constructing COSE structures, calculating cryptographic values and compiling desired system state into a sequence of operations required to achieve that state. The process of constructing COSE structures and the calculation of cryptographic values is covered in [RFC8152].

Compiling desired system state into a sequence of operations can be accomplished in many ways. Several templates are provided below to cover common use-cases. These templates can be combined to produce more complex behavior.

NOTE: On systems that support only a single component, Set Current Component has no effect and can be omitted.

NOTE: A digest should always be set using Override Parameters, since this prevents a less-privileged dependent from replacing the digest.

7.1. Compatibility Check Template

The compatibility check ensures that devices only install compatible images. In this template all information is contained in the common block and the following sequence of operations are used:

- Set Component Index directive (see Section 9.8.4.1)
- Set Parameters directive (see Section 9.8.4.6) for Vendor ID and Class ID (see Section 9.8.2)
- Check Vendor Identifier condition (see Section 9.8.3.1)
- Check Class Identifier condication (see Section 9.8.3.1)

7.2. Secure Boot Template

This template performs a secure boot operation.

The following operations are placed into the common block:

- Set Component Index directive (see Section 9.8.4.1)
- Override Parameters directive (see Section 9.8.4.7) for Image Digest and Image Size (see Section 9.8.2)

Then, the run block contains the following operations:

- Set Component Index directive (see Section 9.8.4.1)
- Check Image Match condition (see Section 9.8.3.2)
- Run directive (see Section 9.8.4.12)

According to Section 6.4, the Run directive applies to the component referenced by the current Component Index. Hence, the Set Component Index directive has to be used to target a specific component.

7.3. Firmware Download Template

This template triggers the download of firmware.

The following operations are placed into the common block:

- Set Component Index directive (see Section 9.8.4.1)
- Override Parameters directive (see Section 9.8.4.7) for Image Digest and Image Size (see Section 9.8.2)

Then, the install block contains the following operations:

- Set Component Index directive (see Section 9.8.4.1)

- Set Parameters directive (see Section 9.8.4.6) for URI (see Section 9.8.2)

- Fetch directive (see Section 9.8.4.8)

The Fetch directive needs the URI parameter to be set to determine where the image is retrieved from. Additionally, the destination of where the component shall be stored has to be configured. The URI is configured via the Set Parameters directive while the destination is configured via the Set Component Index directive.

7.4. Load from External Storage Template

This directive loads an firmware image from external storage.

The following operations are placed into the load block:

- Set Component Index directive (see Section 9.8.4.1)

- Set Parameters directive (see Section 9.8.4.6) for Component Index (see Section 9.8.2)

- Copy directive (see Section 9.8.4.9)

As outlined in Section 6.4, the Copy directive needs a source and a destination to be configured. The source is configured via Component Index (with the Set Parameters directive) and the destination is configured via the Set Component Index directive.

7.5. Load & Decompress from External Storage Template

The following operations are placed into the load block:

- Set Component Index directive (see Section 9.8.4.1)

- Set Parameters directive (see Section 9.8.4.6) for Component Index and Compression Info (see Section 9.8.2)

- Copy directive (see Section 9.8.4.9)

This example is similar to the previous case but additionally performs decompression. Hence, the only difference is in setting the Compression Info parameter.

7.6. Dependency Template

The following operations are placed into the dependency resolution block:

- Set Dependency Index directive (see Section 9.8.4.2)
- Set Parameters directive (see Section 9.8.4.6) for URI (see Section 9.8.2)
- Fetch directive (see Section 9.8.4.8)
- Check Image Match condition (see Section 9.8.3.2)
- Process Dependency directive (see Section 9.8.4.5)

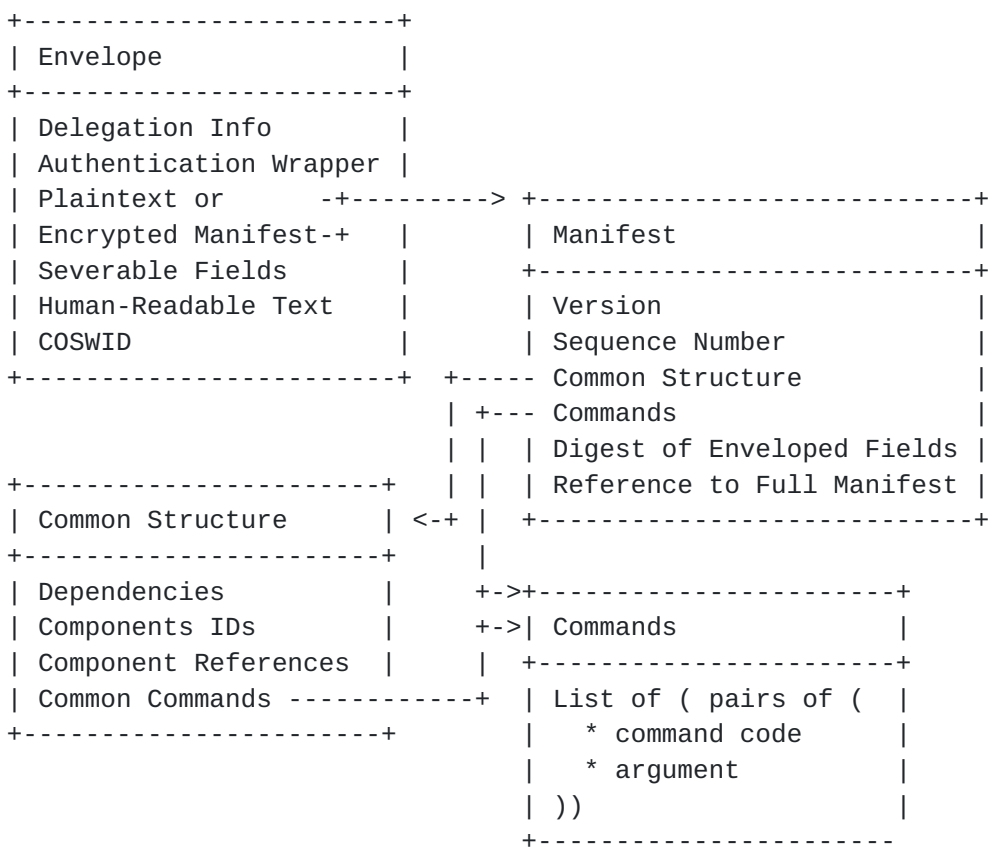
Then, the validate block contains the following operations:

- Set Dependency Index directive (see Section 9.8.4.2)
- Check Image Match condition (see Section 9.8.3.2)
- Process Dependency directive (see Section 9.8.4.5)

NOTE: Any changes made to parameters in a dependency persist in the dependent.

8. Envelope

The diagram below shows high-level structure of the SUIT manifest embedded in the envelope, the top-level structure.



8.1. Authenticated Manifests

The suit-authentication-wrapper contains a list of 1 or more cryptographic authentication wrappers for the core part of the manifest. These are implemented as COSE_Mac_Tagged or COSE_Sign_Tagged blocks. Each of these blocks contains a SUIT_Digest of the manifest. This enables modular processing of the manifest. The COSE_Mac_Tagged and COSE_Sign_Tagged blocks are described in RFC 8152 [RFC8152]. The suit-authentication-wrapper MUST come before any element in the SUIT_Envelope, except for the OPTIONAL suit-delegation, regardless of canonical encoding of CBOR. All validators MUST reject any SUIT_Envelope that begins with any element other than a suit-authentication-wrapper or suit-delegation.

A SUIT_Envelope that has not had authentication information added MUST still contain the suit-authentication-wrapper element, but the content MUST be nil.

For manifests that are only authenticated the envelope MUST contain the plaintext manifest in SUIT_Manifest structure.

8.2. Encrypted Manifests

For encrypted manifest both a SUIT_Encryption_Wrapper and the ciphertext of a manifest is included in the envelope.

When the envelope contains the SUIT_Encryption_Wrapper, the suit-authentication-wrapper MUST authenticate the plaintext of suit-manifest-encrypted. This ensures that the manifest can be stored decrypted and that a recipient MAY convert the suit-manifest-encrypted element to a suit-manifest element.

The SUIT_Manifest structure describes the payload(s) to be installed and any dependencies on other manifests.

The suit-manifest-encryption-info structure contains information required to decrypt a ciphertext manifest and the suit-manifest-encrypted structure contains the ciphertext.

8.3. Delegation Info

The suit-delegation field may carry one or multiple CBOR Web Tokens (CWTs) [RFC8392]. They can be used to perform enhanced authorization decisions.

8.4. Severable Fields

Each of suit-dependency-resolution, suit-payload-fetch, and suit-payload-installation contain the severable contents of the identically named portions of the manifest, described in Section 9.

8.5. Human-Readable Text

suit-text contains all the human-readable information that describes any and all parts of the manifest, its payload(s) and its resource(s). The text section is typically severable, allowing manifests to be distributed without the text, since end-nodes do not require text. The meaning of each field is described below.

Each section MAY be present. If present, each section MUST be as described. Negative integer IDs are reserved for application-specific text values.

CDDL Structure	Description
suit-text-manifest-description	Free text description of the manifest
suit-text-update-description	Free text description of the update
suit-text-vendor-name	Free text vendor name
suit-text-model-name	Free text model name
suit-text-vendor-domain	The domain used to create the vendor-id condition
suit-text-model-info	The information used to create the class-id condition
suit-text-component-description	Free text description of each component in the manifest
suit-text-manifest-json-source	The JSON-formatted document that was used to create the manifest
suit-text-manifest-yaml-source	The yaml-formatted document that was used to create the manifest
suit-text-version-dependencies	List of component versions required by the manifest

8.6. COSWID

suit-coswid contains a Concise Software Identifier. This may be discarded by the Recipient, if not needed.

8.7. Encoding Considerations

The map indices in the envelope encoding are reset to 1 for each map within the structure. This is to keep the indices as small as possible. The goal is to keep the index objects to single bytes (CBOR positive integers 1-23).

Wherever enumerations are used, they are started at 1. This allows detection of several common software errors that are caused by

uninitialised variables. Positive numbers in enumerations are reserved for IANA registration. Negative numbers are used to identify application-specific implementations.

All elements of the envelope must be wrapped in a bstr to minimize the complexity of the code that evaluates the cryptographic integrity of the element and to ensure correct serialization for integrity and authenticity checks.

9. Manifest

The manifest contains:

- a version number (see Section 9.1)
- a sequence number (see Section 9.2)
- a common structure with information that is shared between command sequences (see Section 9.8.1)
- a list of commands that the Recipient should perform (see Section 9.8)
- a reference to the full manifest (see Section 9.3)
- a digest of human-readable text describing the manifest found in the SUIT_Envelope (see Section 9.4)
- a digest of the Concise Software Identifier found in the SUIT_Envelope (see Section 9.5)

Several fields in the Manifest can be either a CBOR structure or a SUIT_Digest. In each of these cases, the SUIT_Digest provides for a severable field. Severable fields are RECOMMENDED to implement. In particular, the human-readable text SHOULD be severable, since most useful text elements occupy more space than a SUIT_Digest, but are not needed by the Recipient. Because SUIT_Digest is a CBOR Array and each severable element is a CBOR bstr, it is straight-forward for a Recipient to determine whether an element has been severed. The key used for a severable element is the same in the SUIT_Manifest and in the SUIT_Envelope so that a Recipient can easily identify the correct data in the envelope.

9.1. suit-manifest-version

The suit-manifest-version indicates the version of serialization used to encode the manifest. Version 1 is the version described in this document. suit-manifest-version is REQUIRED to implement.

9.2. suit-manifest-sequence-number

The suit-manifest-sequence-number is a monotonically increasing anti-rollback counter. It also helps devices to determine which in a set of manifests is the "root" manifest in a given update. Each manifest MUST have a sequence number higher than each of its dependencies. Each Recipient MUST reject any manifest that has a sequence number lower than its current sequence number. It MAY be convenient to use a UTC timestamp in seconds as the sequence number. suit-manifest-sequence-number is REQUIRED to implement.

9.3. suit-reference-uri

suit-reference-uri is a text string that encodes a URI where a full version of this manifest can be found. This is convenient for allowing management systems to show the severed elements of a manifest when this URI is reported by a device after installation.

9.4. suit-text

suit-text is a digest that uniquely identifies the content of the Text that is packaged in the SUIT_Envelope. suit-text is OPTIONAL to implement.

9.5. suit-coswid

suit-coswid is a digest that uniquely identifies the content of the concise-software-identifier that is packaged in the SUIT_Envelope. suit-coswid is OPTIONAL to implement.

9.6. Dependencies

SUIT_Dependency specifies a manifest that describes a dependency of the current manifest.

The suit-dependency-digest specifies the dependency manifest uniquely by identifying a particular Manifest structure. The digest is calculated over the Manifest structure instead of the COSE Sig_structure or Mac_structure. This means that a digest may need to be calculated more than once, however this is necessary 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.

The suit-dependency-prefix element contains a SUIT_Component_Identifier. 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 devices without those devices needing consistent component hierarchy. This element is OPTIONAL.

9.7. SUIT_Component_Reference

The SUIT_Component_Reference describes an image that is defined by another manifest. This is useful for overriding the behavior of another manifest, for example by directing the recipient to look at a different URI for the image or by changing the expected format, such as when a gateway performs decryption on behalf of a constrained device.

9.8. SUIT_Command_Sequence

A SUIT_Command_Sequence defines a series of actions that the Recipient MUST take to accomplish a particular goal. These goals are defined in the manifest and include:

1. Dependency Resolution: suit-dependency-resolution is a SUIT_Command_Sequence to execute in order to perform dependency resolution. Typical actions include configuring URIs of dependency manifests, fetching dependency manifests, and validating dependency manifests' contents. suit-dependency-resolution is REQUIRED to implement and to use when suit-dependencies is present.

2. Payload Fetch: suit-payload-fetch is a SUIT_Command_Sequence to execute in order to obtain a payload. Some manifests may include these actions in the suit-install section instead if they operate in a streaming installation mode. This is particularly relevant for constrained devices without any temporary storage for staging the update. suit-payload-fetch is OPTIONAL to implement.

3. Payload Installation: suit-install is a SUIT_Command_Sequence to execute in order to install a payload. Typical actions include verifying a payload stored in temporary storage, copying a staged payload from temporary storage, and unpacking a payload. suit-install is OPTIONAL to implement.

4. Image Validation: suit-validate is a SUIT_Command_Sequence to execute in order to validate that the result of applying the update is correct. Typical actions involve image validation and manifest validation. suit-validate is REQUIRED to implement. If the manifest contains dependencies, one process-dependency

invocation per dependency or one process-dependency invocation targeting all dependencies SHOULD be present in validate.

5. Image Loading: suit-load is a SUIT_Command_Sequence to execute in order to prepare a payload for execution. Typical actions include copying an image from permanent storage into RAM, optionally including actions such as decryption or decompression. suit-load is OPTIONAL to implement.

6. Run or Boot: suit-run is a SUIT_Command_Sequence to execute in order to run an image. suit-run typically contains a single instruction: either the "run" directive for the bootable manifest or the "process dependencies" directive for any dependents of the bootable manifest. suit-run is OPTIONAL to implement. Only one manifest in an update may contain the "run" directive.

Each of these follows exactly the same structure to ensure that the parser is as simple as possible.

Lists of commands are constructed from two kinds of element:

1. Conditions that MUST be true-any failure is treated as a failure of the update/load/boot

2. Directives that MUST be executed.

Each condition is a command code identifier, followed by Nil.

Each directive is composed of:

1. A command code identifier

2. An argument block or Nil

Argument blocks are defined for each type of directive.

Many conditions and directives apply to a given component, and these generally grouped together. Therefore, a special command to set the current component index is provided with a matching command to set the current dependency index. This index is a numeric index into the component ID tables defined at the beginning of the document. For the purpose of setting the index, the two component ID tables are considered to be concatenated together.

To facilitate optional conditions, a special directive is provided. It runs several new lists of conditions/directives, one after another, that are contained as an argument to the directive. By default, it assumes that a failure of a condition should not indicate

a failure of the update/boot, but a parameter is provided to override this behavior.

9.8.1. suit-common

suit-common encodes all the information that is shared between each of the command sequences, including: suit-dependencies, suit-components, suit-dependency-components, and suit-common-sequence. suit-common is REQUIRED to implement.

suit-dependencies is a list of SUIT_Dependency blocks that specify manifests that must be present before the current manifest can be processed. suit-dependencies is OPTIONAL to implement.

In order to distinguish between components that are affected by the current manifest and components that are affected by a dependency, they are kept in separate lists. Components affected by the current manifest only list the component identifier. Components affected by a dependency include the component identifier and the index of the dependency that defines the component.

suit-components is a list of SUIT_Component blocks that specify the component identifiers that will be affected by the content of the current manifest. suit-components is OPTIONAL to implement, but at least one manifest MUST contain a suit-components block.

suit-dependency-components is a list of SUIT_Component_Reference blocks that specify component identifiers that will be affected by the content of a dependency of the current manifest. suit-dependency-components is OPTIONAL to implement.

suit-common-sequence is a SUIT_Command_Sequence to execute prior to executing any other command sequence. Typical actions in suit-common-sequence include setting expected device identity and image digests when they are conditional (see Section 12 for more information on conditional sequences). suit-common-sequence is RECOMMENDED to implement.

9.8.2. SUIT_Parameters

Many conditions and directives require additional information. That information is contained within parameters that can be set in a consistent way. This allows reduction of manifest size and replacement of parameters from one manifest to the next.

The defined manifest parameters are described below.

+----------------+----------------------------------+---------------+

Name	CDDL Structure	Reference
Vendor ID	suit-parameter-vendor-identifier	Section 9.8.2.1
Class ID	suit-parameter-class-identifier	Section 9.8.2.2
Image Digest	suit-parameter-image-digest	Section 9.8.2.3
Image Size	suit-parameter-image-size	Section 9.8.2.4
Use Before	suit-parameter-use-before	Section 9.8.2.5
Component Offset	suit-parameter-component-offset	Section 9.8.2.6
Encryption Info	suit-parameter-encryption-info	Section 9.8.2.7
Compression Info	suit-parameter-compression-info	Section 9.8.2.8
Unpack Info	suit-parameter-unpack-info	Section 9.8.2.9
URI	suit-parameter-uri	Section 9.8.2.10
Source Component	suit-parameter-source-component	Section 9.8.2.11
Run Args	suit-parameter-run-args	Section 9.8.2.12
Device ID	suit-parameter-device-identifier	Section 9.8.2.13
Minimum Battery	suit-parameter-minimum-battery	Section 9.8.2.14
Update Priority	suit-parameter-update-priority	Section 9.8.2.15
Version	suit-parameter-version	Section

		9.8.2.16
Wait Info	suit-parameter-wait-info	Section 9.8.2.17
URI List	suit-parameter-uri-list	Section 9.8.2.18
Strict Order	suit-parameter-strict-order	Section 9.8.2.19
Soft Failure	suit-parameter-soft-failure	Section 9.8.2.20
Custom	suit-parameter-custom	Section 9.8.2.21

CBOR-encoded object parameters are still wrapped in a bstr. This is because it allows a parser that is aggregating parameters to reference the object with a single pointer and traverse it without understanding the contents. This is important for modularization and division of responsibility within a pull parser. The same consideration does not apply to Directives because those elements are invoked with their arguments immediately

9.8.2.1. suit-parameter-vendor-identifier

A RFC 4122 UUID representing the vendor of the device or component.

9.8.2.2. suit-parameter-class-identifier

A RFC 4122 UUID representing the class of the device or component

9.8.2.3. suit-parameter-image-digest

A fingerprint computed over the image itself encoded in the SUIT_Digest structure.

9.8.2.4. suit-parameter-image-size

The size of the firmware image in bytes.

9.8.2.5. suit-parameter-use-before

An expire date for the use of the manifest encoded as a POSIX timestamp.

9.8.2.6. suit-parameter-component-offset

This parameter sets the offset in a component.

9.8.2.7. suit-parameter-encryption-info

Encryption Info defines the mechanism that Fetch or Copy should use to decrypt the data they transfer. SUIT_Parameter_Encryption_Info is encoded as a COSE_Encrypt_Tagged or a COSE_Encrypt0_Tagged, wrapped in a bstr.

9.8.2.8. suit-parameter-compression-info

Compression Info defines any information that is required for a device to perform decompression operations. Typically, this includes the algorithm identifier. This document defines the use of ZLIB [RFC1950], Brotli [RFC7932], and ZSTD [I-D.kucherawy-rfc8478bis].

Additional compression formats can be registered through the IANA-maintained registry.

9.8.2.9. suit-parameter-unpack-info

SUIT_Unpack_Info defines the information required for a device to interpret a packed format. This document defines the use of the following binary encodings: Intel HEX [HEX], Motorola S-record [SREC], Executable and Linkable Format (ELF) [ELF], and Common Object File Format (COFF) [COFF].

Additional packing formats can be registered through the IANA-maintained registry.

9.8.2.10. suit-parameter-uri

A URI from which to fetch a resource.

9.8.2.11. suit-parameter-source-component

This parameter sets the source component.

9.8.2.12. suit-parameter-run-args

This parameter contains an encoded set of arguments for Run.

9.8.2.13. suit-parameter-device-identifier

A RFC 4122 UUID representing the device or component.

9.8.2.14. suit-parameter-minimum-battery

This parameter sets the minimum battery level in mWh.

9.8.2.15. suit-parameter-update-priority

This parameter sets the priority of the update.

9.8.2.16. suit-parameter-version

Allows to indicate the version numbers of firmware to which the manifest applies, either with a list or with range matching.

9.8.2.17. suit-parameter-wait-info

suit-directive-wait Section 9.8.4.11 directs the manifest processor to pause until a specified event occurs. The suit-parameter-wait-info encodes the parameters needed for the directive.

9.8.2.18. suit-parameter-uri-list

Indicates a list of URIs from which to fetch a resource.

9.8.2.19. suit-parameter-strict-order

The Strict Order Parameter allows a manifest to govern when directives can be executed out-of-order. This allows for systems that have a sensitivity to order of updates to choose the order in which they are executed. It also allows for more advanced systems to parallelize their handling of updates. Strict Order defaults to True. It MAY be set to False when the order of operations does not matter. When arriving at the end of a command sequence, ALL commands MUST have completed, regardless of the state of SUIT_Parameter_Strict_Order. If SUIT_Parameter_Strict_Order is returned to True, ALL preceding commands MUST complete before the next command is executed.

9.8.2.20. suit-parameter-soft-failure

When executing a command sequence inside SUIT_Directive_Try_Each and a condition failure occurs, the manifest processor aborts the sequence. If Soft Failure is True, it returns Success. Otherwise, it returns the original condition failure. SUIT_Parameter_Soft_Failure is scoped to the enclosing

SUIT_Command_Sequence. Its value is discarded when SUIT_Command_Sequence terminates.

9.8.2.21. suit-parameter-custom

This parameter is an extension point for any proprietary, application specific conditions and directives.

9.8.3. SUIT_Condition

Conditions are used to define mandatory properties of a system in order for an update to be applied. They can be pre-conditions or post-conditions of any directive or series of directives, depending on where they are placed in the list. Conditions never take arguments; conditions should test using parameters instead. Conditions include:

Name	CDDL Structure	Reference
Vendor Identifier	suit-condition-vendor-identifier	Section 9.8.3.1
Class Identifier	suit-condition-class-identifier	Section 9.8.3.1
Device Identifier	suit-condition-device-identifier	Section 9.8.3.1
Image Match	suit-condition-image-match	Section 9.8.3.2
Image Not Match	suit-condition-image-not-match	Section 9.8.3.3
Use Before	suit-condition-use-before	Section 9.8.3.4
Component Offset	suit-condition-component-offset	Section 9.8.3.5
Minimum Battery	suit-condition-minimum-battery	Section 9.8.3.6
Update Authorized	suit-condition-update-authorized	Section 9.8.3.7
Version	suit-condition-version	Section 9.8.3.8
Custom Condition	SUIT_Condition_Custom	Section 9.8.3.9

Each condition MUST report a result code on completion. If a condition reports failure, then the current sequence of commands MUST terminate. If a condition requires additional information, this MUST be specified in one or more parameters before the condition is executed. If a Recipient attempts to process a condition that expects additional information and that information has not been set, it MUST report a failure. If a Recipient encounters an unknown condition, it MUST report a failure.

Condition labels in the positive number range are reserved for IANA registration while those in the negative range are custom conditions reserved for proprietary use.

Several conditions use identifiers to determine whether a manifest matches a given Recipient or not. These identifiers are defined to be RFC 4122 [RFC4122] UUIDs. These UUIDs are not human-readable and are therefore used for machine-based processing only.

A device may match any number of UUIDs for vendor or class identifier. This may be relevant to physical or software modules. For example, a device that has an OS and one or more applications might list one Vendor ID for the OS and one or more additional Vendor IDs for the applications. This device might also have a Class ID that must be matched for the OS and one or more Class IDs for the applications.

A more complete example: Imagine a device has the following physical components: 1. A host MCU 2. A WiFi module

This same device has three software modules: 1. An operating system 2. A WiFi module interface driver 3. An application

Suppose that the WiFi module's firmware has a proprietary update mechanism and doesn't support manifest processing. This device can report four class IDs:

1. hardware model/revision

2. OS

3. WiFi module model/revision

4. Application

This allows the OS, WiFi module, and application to be updated independently. To combat possible incompatibilities, the OS class ID can be changed each time the OS has a change to its API.

This approach allows a vendor to target, for example, all devices with a particular WiFi module with an update, which is a very powerful mechanism, particularly when used for security updates.

UUIDs MUST be created according to RFC 4122 [RFC4122]. UUIDs SHOULD use versions 3, 4, or 5, as described in RFC4122. Versions 1 and 2 do not provide a tangible benefit over version 4 for this application.

The RECOMMENDED method to create a vendor ID is: Vendor ID = UUID5(DNS_PREFIX, vendor domain name)

The RECOMMENDED method to create a class ID is: Class ID = UUID5(Vendor ID, Class-Specific-Information)

Class-specific information is composed of a variety of data, for example:

- Model number.

- Hardware revision.

- Bootloader version (for immutable bootloaders).

9.8.3.1. suit-condition-vendor-identifier, suit-condition-class-identifier, and suit-condition-device-identifier

There are three identifier-based conditions: suit-condition-vendor-identifier, suit-condition-class-identifier, and suit-condition-device-identifier. Each of these conditions match a RFC 4122 [RFC4122] UUID that MUST have already been set as a parameter. The installing device MUST match the specified UUID in order to consider the manifest valid. These identifiers MAY be scoped by component.

The Recipient uses the ID parameter that has already been set using the Set Parameters directive. If no ID has been set, this condition fails. suit-condition-class-identifier and suit-condition-vendor-identifier are REQUIRED to implement. suit-condition-device-identifier is OPTIONAL to implement.

9.8.3.2. suit-condition-image-match

Verify that the current component matches the digest parameter for the current component. The digest is verified against the digest specified in the Component's parameters list. If no digest is specified, the condition fails. suit-condition-image-match is REQUIRED to implement.

9.8.3.3. suit-condition-image-not-match

Verify that the current component does not match the supplied digest. If no digest is specified, then the digest is compared against the digest specified in the Component's parameters list. If no digest is specified, the condition fails. suit-condition-image-not-match is OPTIONAL to implement.

9.8.3.4. suit-condition-use-before

Verify that the current time is BEFORE the specified time. suit-condition-use-before is used to specify the last time at which an update should be installed. The recipient evaluates the current time against the suit-parameter-use-before parameter, which must have already been set as a parameter, encoded as a POSIX timestamp, that is seconds after 1970-01-01 00:00:00. Timestamp conditions MUST be evaluated in 64 bits, regardless of encoded CBOR size. suit-condition-use-before is OPTIONAL to implement.

9.8.3.5. suit-condition-component-offset

TBD.

9.8.3.6. suit-condition-minimum-battery

suit-condition-minimum-battery provides a mechanism to test a device's battery level before installing an update. This condition is for use in primary-cell applications, where the battery is only ever discharged. For batteries that are charged, suit-directive-wait is more appropriate, since it defines a "wait" until the battery level is sufficient to install the update. suit-condition-minimum-battery is specified in mWh. suit-condition-minimum-battery is OPTIONAL to implement.

9.8.3.7. suit-condition-update-authorized

Request Authorization from the application and fail if not authorized. This can allow a user to decline an update. Argument is an integer priority level. Priorities are application defined. suit-condition-update-authorized is OPTIONAL to implement.

9.8.3.8. suit-condition-version

suit-condition-version allows comparing versions of firmware. Verifying image digests is preferred to version checks because digests are more precise. The image can be compared as:

- Greater.
- Greater or Equal.
- Equal.
- Lesser or Equal.
- Lesser.

Versions are encoded as a CBOR list of integers. Comparisons are done on each integer in sequence. Comparison stops after all integers in the list defined by the manifest have been consumed OR after a non-equal match has occurred. For example, if the manifest defines a comparison, "Equal [1]", then this will match all version sequences starting with 1. If a manifest defines both "Greater or Equal [1,0]" and "Lesser [1,10]", then it will match versions 1.0.x up to, but not including 1.10.

While the exact encoding of versions is application-defined, semantic versions map conveniently. For example,

- 1.2.3 = [1,2,3].
- 1.2-rc3 = [1,2,-1,3].
- 1.2-beta = [1,2,-2].
- 1.2-alpha = [1,2,-3].
- 1.2-alpha4 = [1,2,-3,4].

suit-condition-version is OPTIONAL to implement.

9.8.3.9. SUIT_Condition_Custom

SUIT_Condition_Custom describes any proprietary, application specific condition. This is encoded as a negative integer, chosen by the firmware developer. If additional information must be provided to the condition, it should be encoded in a custom parameter (a nint) as described in Section 9.8.2. SUIT_Condition_Custom is OPTIONAL to implement.

9.8.4. SUIT_Directive

Directives are used to define the behavior of the recipient. Directives include:

Name	CDDL Structure	Reference
Set Component Index	suit-directive-set-component-index	Section 9.8.4.1
Set Dependency Index	suit-directive-set-dependency-index	Section 9.8.4.2
Abort	suit-directive-abort	Section 9.8.4.3
Try Each	suit-directive-try-each	Section 9.8.4.4
Process Dependency	suit-directive-process-dependency	Section 9.8.4.5
Set Parameters	suit-directive-set-parameters	Section 9.8.4.6
Override Parameters	suit-directive-override-parameters	Section 9.8.4.7
Fetch	suit-directive-fetch	Section 9.8.4.8
Copy	suit-directive-copy	Section 9.8.4.9
Run	suit-directive-run	Section 9.8.4.10
Wait For Event	suit-directive-wait	Section 9.8.4.11
Run Sequence	suit-directive-run-sequence	Section 9.8.4.12
Swap	suit-directive-swap	Section 9.8.4.13

When a Recipient executes a Directive, it MUST report a result code. If the Directive reports failure, then the current Command Sequence MUST terminate.

9.8.4.1. suit-directive-set-component-index

Set Component Index defines the component to which successive directives and conditions will apply. The supplied argument MUST be either a boolean or an unsigned integer index into the concatenation of suit-components and suit-dependency-components. If the following directives apply to ALL components, then the boolean value "True" is used instead of an index. True does not apply to dependency components. If the following directives apply to NO components, then the boolean value "False" is used. When suit-directive-set-dependency-index is used, suit-directive-set-component-index = False is implied. When suit-directive-set-component-index is used, suit-directive-set-dependency-index = False is implied.

9.8.4.2. suit-directive-set-dependency-index

Set Dependency Index defines the manifest to which successive directives and conditions will apply. The supplied argument MUST be either a boolean or an unsigned integer index into the dependencies. If the following directives apply to ALL dependencies, then the boolean value "True" is used instead of an index. If the following directives apply to NO dependencies, then the boolean value "False" is used. When suit-directive-set-component-index is used, suit-directive-set-dependency-index = False is implied. When suit-directive-set-dependency-index is used, suit-directive-set-component-index = False is implied.

Typical operations that require suit-directive-set-dependency-index include setting a source URI, invoking "Fetch," or invoking "Process Dependency" for an individual dependency.

9.8.4.3. suit-directive-abort

Unconditionally fail. This operation is typically used in conjunction with suit-directive-try-each.

9.8.4.4. suit-directive-try-each

This command runs several SUIT_Command_Sequence, one after another, in a strict order. Use this command to implement a "try/catch-try/catch" sequence. Manifest processors MAY implement this command.

SUIT_Parameter_Soft_Failure is initialized to True at the beginning of each sequence. If one sequence aborts due to a condition failure, the next is started. If no sequence completes without condition failure, then suit-directive-try-each returns an error. If a particular application calls for all sequences to fail and still

continue, then an empty sequence (nil) can be added to the Try Each Argument.

9.8.4.5. 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 "fetch payload," this will execute "common" in the current dependency, then "fetch payload" in the current dependency. Once this is complete, the command following suit-directive-process-dependency will be processed.

If the current dependency is False, this directive has no effect. If the current dependency is True, then this directive applies to all dependencies. If the current section is "common," this directive MUST have no effect.

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

9.8.4.6. suit-directive-set-parameters

suit-directive-set-parameters allows the manifest to configure behavior of future directives by changing parameters that are read by those directives. When dependencies are used, suit-directive-set-parameters also allows a manifest to modify the behavior of its dependencies.

Available parameters are defined in Section 9.8.2.

If a parameter is already set, suit-directive-set-parameters will skip setting the parameter to its argument. This provides the core of the override mechanism, allowing dependent manifests to change the behavior of a manifest.

9.8.4.7. suit-directive-override-parameters

suit-directive-override-parameters replaces any listed parameters that are already set with the values that are provided in its argument. This allows a manifest to prevent replacement of critical parameters.

Available parameters are defined in Section 9.8.2.

9.8.4.8. suit-directive-fetch

suit-directive-fetch instructs the manifest processor to obtain one or more manifests or payloads, as specified by the manifest index and component index, respectively.

suit-directive-fetch can target one or more manifests and one or more payloads. suit-directive-fetch retrieves each component and each manifest listed in component-index and manifest-index, respectively. If component-index or manifest-index is True, instead of an integer, then all current manifest components/manifests are fetched. The current manifest's dependent-components are not automatically fetched. In order to pre-fetch these, they MUST be specified in a component-index integer.

suit-directive-fetch typically takes no arguments unless one is needed to modify fetch behavior. If an argument is needed, it must be wrapped in a bstr.

suit-directive-fetch reads the URI or URI List parameter to find the source of the fetch it performs.

The behavior of suit-directive-fetch can be modified by setting one or more of SUIT_Parameter_Encryption_Info, SUIT_Parameter_Compression_Info, SUIT_Parameter_Unpack_Info. These three parameters each activate and configure a processing step that can be applied to the data that is transferred during suit-directive-fetch.

9.8.4.9. suit-directive-copy

suit-directive-copy instructs the manifest processor to obtain one or more payloads, as specified by the component index. suit-directive-copy retrieves each component listed in component-index, respectively. If component-index is True, instead of an integer, then all current manifest components are copied. The current manifest's dependent-components are not automatically copied. In order to copy these, they MUST be specified in a component-index integer.

The behavior of suit-directive-copy can be modified by setting one or more of SUIT_Parameter_Encryption_Info, SUIT_Parameter_Compression_Info, SUIT_Parameter_Unpack_Info. These three parameters each activate and configure a processing step that can be applied to the data that is transferred during suit-directive-copy.

N.B. Fetch and Copy are very similar. Merging them into one command may be appropriate.

suit-directive-copy reads its source from SUIT_Parameter_Source_Component.

9.8.4.10. suit-directive-run

suit-directive-run directs the manifest processor to transfer execution to the current Component Index. When this is invoked, the manifest processor MAY be unloaded and execution continues in the Component Index. Arguments provided to Run are forwarded to the executable code located in Component Index, in an application-specific way. For example, this could form the Linux Kernel Command Line if booting a Linux device.

If the executable code at Component Index is constructed in such a way that it does not unload the manifest processor, then the manifest processor may resume execution after the executable completes. This allows the manifest processor to invoke suitable helpers and to verify them with image conditions.

9.8.4.11. suit-directive-wait

suit-directive-wait directs the manifest processor to pause until a specified event occurs. Some possible events include:

1. Authorization
2. External Power
3. Network availability
4. Other Device Firmware Version
5. Time
6. Time of Day
7. Day of Week

9.8.4.12. suit-directive-run-sequence

To enable conditional commands, and to allow several strictly ordered sequences to be executed out-of-order, suit-directive-run-sequence allows the manifest processor to execute its argument as a SUIT_Command_Sequence. The argument must be wrapped in a bstr.

When a sequence is executed, any failure of a condition causes immediate termination of the sequence.

When suit-directive-run-sequence completes, it forwards the last status code that occurred in the sequence. If the Soft Failure parameter is true, then suit-directive-run-sequence only fails when a directive in the argument sequence fails.

SUIT_Parameter_Soft_Failure defaults to False when suit-directive-run-sequence begins. Its value is discarded when suit-directive-run-sequence terminates.

9.8.4.13. suit-directive-swap

suit-directive-swap instructs the manifest processor to move the source to the destination and the destination to the source simultaneously. Swap has nearly identical semantics to suit-directive-copy except that suit-directive-swap replaces the source with the current contents of the destination in an application-defined way. If SUIT_Parameter_Compression_Info or SUIT_Parameter_Encryption_Info are present, they must be handled in a symmetric way, so that the source is decompressed into the destination and the destination is compressed into the source. The source is decrypted into the destination and the destination is encrypted into the source. suit-directive-swap is OPTIONAL to implement.

10. Access Control Lists

To manage permissions in the manifest, there are three models that can be used.

First, the simplest model requires that all manifests are authenticated by a single trusted key. This mode has the advantage that only a root manifest needs to be authenticated, since all of its dependencies have digests included in the root manifest.

This simplest model can be extended by adding key delegation without much increase in complexity.

A second model requires an ACL to be presented to the device, authenticated by a trusted party or stored on the device. This ACL grants access rights for specific component IDs or component ID prefixes to the listed identities or identity groups. Any identity may verify an image digest, but fetching into or fetching from a component ID requires approval from the ACL.

A third model allows a device to provide even more fine-grained controls: The ACL lists the component ID or component ID prefix that an identity may use, and also lists the commands that the identity may use in combination with that component ID.

11. SUIT Digest Container

RFC 8152 [RFC8152] provides containers for signature, MAC, and encryption, but no basic digest container. The container needed for a digest requires a type identifier and a container for the raw digest data. Some forms of digest may require additional parameters. These can be added following the digest.

The algorithms listed are sufficient for verifying integrity of Firmware Updates as of this writing, however this may change over time.

12. Creating Conditional Sequences

For some use cases, it is important to provide a sequence that can fail without terminating an update. For example, a dual-image XIP MCU may require an update that can be placed at one of two offsets. This has two implications, first, the digest of each offset will be different. Second, the image fetched for each offset will have a different URI. Conditional sequences allow this to be resolved in a simple way.

The following JSON representation of a manifest demonstrates how this would be represented. It assumes that the bootloader and manifest processor take care of A/B switching and that the manifest is not aware of this distinction.

```
{
    "structure-version" : 1,
    "sequence-number" : 7,
    "common" :{
        "components" : [
            [b'0']
        ],
        "common-sequence" : [
            {
                "directive-set-var" : {
                    "size": 32567
                },
            },
            {
                "try-each" : [
                    [
```

```
                        {"condition-component-offset" : "<offset A>"},
                        {
                            "directive-set-var": {
                                "digest" : "<SHA256 A>"
                            }
                        }
                    ],
                    [
                        {"condition-component-offset" : "<offset B>"},
                        {
                            "directive-set-var": {
                                "digest" : "<SHA256 B>"
                            }
                        }
                    ],
                    [{ "abort" : null }]
                ]
            }
        ]
    }
    "fetch" : [
        {
            "try-each" : [
                [
                    {"condition-component-offset" : "<offset A>"},
                    {
                        "directive-set-var": {
                            "uri" : "<URI A>"
                        }
                    }
                ],
                [
                    {"condition-component-offset" : "<offset B>"},
                    {
                        "directive-set-var": {
                            "uri" : "<URI B>"
                        }
                    }
                ],
                [{ "directive-abort" : null }]
            ]
        },
        "fetch" : null
    ]
}
```

13. IANA Considerations

IANA is requested to setup a registry for SUIT manifests. Several registries defined in the subsections below need to be created.

For each registry, values 0-23 are Standards Action, 24-255 are IETF Review, 256-65535 are Expert Review, and 65536 or greater are First Come First Served.

Negative values -23 to 0 are Experimental Use, -24 and lower are Private Use.

13.1. SUIT Directives

Label	Name
12	Set Component Index
13	Set Dependency Index
14	Abort
15	Try Each
16	Reserved
17	Reserved
18	Process Dependency
19	Set Parameters
20	Override Parameters
21	Fetch
22	Copy
23	Run
29	Wait For Event
30	Run Sequence
32	Swap

13.2. SUIT Conditions

Label	Name
1	Vendor Identifier
2	Class Identifier
24	Device Identifier
3	Image Match
25	Image Not Match
4	Use Before
5	Component Offset
26	Minimum Battery
27	Update Authorized
28	Version
nint	Custom Condition

13.3. SUIT Parameters

Label	Name
1	Vendor ID
2	Class ID
3	Image Digest
4	Use Before
5	Component Offset
12	Strict Order
13	Soft Failure
14	Image Size
18	Encryption Info
19	Compression Info
20	Unpack Info
21	URI
22	Source Component
23	Run Args
24	Device ID
26	Minimum Battery
27	Update Priority
28	Version
29	Wait Info
30	URI List
31	Component Index
nint	Custom

13.4. SUIT Text Values

Label	Name
1	Manifest Description
2	Update Description
3	Vendor Name
4	Model Name
5	Vendor Domain
6	Model Info
7	Component Description
8	Manifest JSON Source
9	Manifest YAML Source
10	Component Version Dependencies

13.5. SUIT Algorithm Identifiers

13.5.1. Hash Algorithms

Label	Name
1	SHA224
2	SHA256
3	SHA384
4	SHA512
5	SHA3-224
6	SHA3-256
7	SHA3-384
8	SHA3-512

13.5.2. Unpack Algorithms

Label	Name
1	HEX
2	ELF
3	COFF
4	SREC

14. Security Considerations

This document is about a manifest format describing and protecting firmware images and as such it is part of a larger solution for offering a standardized way of delivering firmware updates to IoT devices. A detailed security treatment can be found in the architecture [I-D.ietf-suit-architecture] and in the information model [I-D.ietf-suit-information-model] documents.

15. Acknowledgements

We would like to thank the following persons for their support in designing this mechanism:

- Milosch Meriac
- Geraint Luff
- Dan Ros
- John-Paul Stanford
- Hugo Vincent
- Carsten Bormann
- Oeyvind Roenningstad
- Frank Audun Kvamtroe
- Krzysztof Chruściński
- Andrzej Puzdrowski
- Michael Richardson
- David Brown
- Emmanuel Baccelli

16. References

16.1. Normative References

[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/info/rfc2119>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

[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/info/rfc8174>.

16.2. Informative References

[COFF] Wikipedia, ., "Common Object File Format (COFF)", 2020, <https://en.wikipedia.org/wiki/COFF>.

[ELF] Wikipedia, ., "Executable and Linkable Format (ELF)", 2020, <https://en.wikipedia.org/wiki/Executable_and_Linkable_Format>.

[HEX] Wikipedia, ., "Intel HEX", 2020, <https://en.wikipedia.org/wiki/Intel_HEX>.

[I-D.ietf-suit-architecture] Moran, B., Tschofenig, H., Brown, D., and M. Meriac, "A Firmware Update Architecture for Internet of Things", draft-ietf-suit-architecture-11 (work in progress), May 2020.

[I-D.ietf-suit-information-model] Moran, B., Tschofenig, H., and H. Birkholz, "An Information Model for Firmware Updates in IoT Devices", draft-ietf-suit-information-model-07 (work in progress), June 2020.

[I-D.ietf-teep-architecture] Pei, M., Tschofenig, H., Thaler, D., and D. Wheeler, "Trusted Execution Environment Provisioning (TEEP) Architecture", draft-ietf-teep-architecture-08 (work in progress), April 2020.

[I-D.kucherawy-rfc8478bis] Collet, Y. and M. Kucherawy, "Zstandard Compression and the application/zstd Media Type", draft-kucherawy-rfc8478bis-05 (work in progress), April 2020.

[RFC1950] Deutsch, P. and J-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, DOI 10.17487/RFC1950, May 1996, <https://www.rfc-editor.org/info/rfc1950>.

[RFC7932] Alakuijala, J. and Z. Szabadka, "Brotli Compressed Data Format", RFC 7932, DOI 10.17487/RFC7932, July 2016, <https://www.rfc-editor.org/info/rfc7932>.

[RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <https://www.rfc-editor.org/info/rfc8392>.

[SREC] Wikipedia, ., "SREC (file format)", 2020, <https://en.wikipedia.org/wiki/SREC_(file_format)>.

A. Full CDDL

In order to create a valid SUIT Manifest document the structure of the corresponding CBOR message MUST adhere to the following CDDL data definition.

```
SUIT_Envelope = {
    ? suit-delegation            => bstr .cbor SUIT_Delegation
    ? suit-authentication-wrapper
        => bstr .cbor SUIT_Authentication_Wrapper / nil,
    $$SUIT_Manifest_Wrapped,
    * $$SUIT_Severed_Fields,
}

SUIT_Delegation = [ + [ + CWT ] ]

CWT = SUIT_Authentication_Block

SUIT_Authentication_Wrapper = [ + bstr .cbor SUIT_Authentication_Block ]

SUIT_Authentication_Block /= COSE_Mac_Tagged
SUIT_Authentication_Block /= COSE_Sign_Tagged
SUIT_Authentication_Block /= COSE_Mac0_Tagged
SUIT_Authentication_Block /= COSE_Sign1_Tagged

$$SUIT_Manifest_Wrapped //= (suit-manifest  => bstr .cbor SUIT_Manifest)
$$SUIT_Manifest_Wrapped //= (
    suit-manifest-encryption-info => bstr .cbor SUIT_Encryption_Wrapper,
    suit-manifest-encrypted       => bstr
)

SUIT_Encryption_Wrapper = COSE_Encrypt_Tagged / COSE_Encrypt0_Tagged

$$SUIT_Severed_Fields //= ( suit-dependency-resolution =>
    bstr .cbor SUIT_Command_Sequence)
$$SUIT_Severed_Fields //= (suit-payload-fetch =>
    bstr .cbor SUIT_Command_Sequence)
$$SUIT_Severed_Fields //= (suit-install =>
    bstr .cbor SUIT_Command_Sequence)
$$SUIT_Severed_Fields //= (suit-text =>
    bstr .cbor SUIT_Text_Map)
$$SUIT_Severed_Fields //= (suit-coswid =>
    bstr .cbor concise-software-identity)

COSE_Mac_Tagged = any
COSE_Sign_Tagged = any
COSE_Mac0_Tagged = any
COSE_Sign1_Tagged = any
```

```
COSE_Encrypt_Tagged = any
COSE_Encrypt0_Tagged = any

SUIT_Digest = [
  suit-digest-algorithm-id : suit-digest-algorithm-ids,
  suit-digest-bytes : bstr,
  ? suit-digest-parameters : any
]

; Named Information Hash Algorithm Identifiers
suit-digest-algorithm-ids /= algorithm-id-sha224
suit-digest-algorithm-ids /= algorithm-id-sha256
suit-digest-algorithm-ids /= algorithm-id-sha384
suit-digest-algorithm-ids /= algorithm-id-sha512
suit-digest-algorithm-ids /= algorithm-id-sha3-224
suit-digest-algorithm-ids /= algorithm-id-sha3-256
suit-digest-algorithm-ids /= algorithm-id-sha3-384
suit-digest-algorithm-ids /= algorithm-id-sha3-512

algorithm-id-sha224 = 1
algorithm-id-sha256 = 2
algorithm-id-sha384 = 3
algorithm-id-sha512 = 4
algorithm-id-sha3-224 = 5
algorithm-id-sha3-256 = 6
algorithm-id-sha3-384 = 7
algorithm-id-sha3-512 = 8

SUIT_Manifest = {
    suit-manifest-version         => 1,
    suit-manifest-sequence-number => uint,
    suit-common                   => bstr .cbor SUIT_Common,
    ? suit-reference-uri          => #6.32(tstr),
    * $$SUIT_Severable_Command_Sequences,
    * $$SUIT_Command_Sequences,
    * $$SUIT_Protected_Elements,
}

$$SUIT_Severable_Command_Sequences //= (suit-dependency-resolution =>
    SUIT_Severable_Command_Sequence)
$$SUIT_Severable_Command_Sequences //= (suit-payload-fetch =>
    SUIT_Severable_Command_Sequence)
$$SUIT_Severable_Command_Sequences //= (suit-install =>
    SUIT_Severable_Command_Sequence)

SUIT_Severable_Command_Sequence =
    SUIT_Digest / bstr .cbor SUIT_Command_Sequence
```

```
$$SUIT_Command_Sequences //= ( suit-validate =>
    bstr .cbor SUIT_Command_Sequence )
$$SUIT_Command_Sequences //= ( suit-load =>
    bstr .cbor SUIT_Command_Sequence )
$$SUIT_Command_Sequences //= ( suit-run =>
    bstr .cbor SUIT_Command_Sequence )

$$SUIT_Protected_Elements //= ( suit-text => SUIT_Digest )
$$SUIT_Protected_Elements //= ( suit-coswid => SUIT_Digest )

SUIT_Common = {
    ? suit-dependencies           => bstr .cbor SUIT_Dependencies,
    ? suit-components             => bstr .cbor SUIT_Components,
    ? suit-dependency-components
        => bstr .cbor SUIT_Component_References,
    ? suit-common-sequence        => bstr .cbor SUIT_Command_Sequence,
}

SUIT_Dependencies         = [ + SUIT_Dependency ]
SUIT_Components           = [ + SUIT_Component_Identifier ]
SUIT_Component_References = [ + SUIT_Component_Reference ]

concise-software-identity = any

SUIT_Dependency = {
    suit-dependency-digest => SUIT_Digest,
    suit-dependency-prefix => SUIT_Component_Identifier,
}

SUIT_Component_Identifier =  [* bstr]


SUIT_Component_Reference = {
    suit-component-identifier => SUIT_Component_Identifier,
    suit-component-dependency-index => uint
}

SUIT_Command_Sequence = [ + (
    SUIT_Condition // SUIT_Directive // SUIT_Command_Custom
) ]

SUIT_Command_Custom = (suit-command-custom, bstr/tstr/int/nil)
SUIT_Condition //= (suit-condition-vendor-identifier, nil)
SUIT_Condition //= (suit-condition-class-identifier,  nil)
SUIT_Condition //= (suit-condition-device-identifier, nil)
SUIT_Condition //= (suit-condition-image-match,       nil)
SUIT_Condition //= (suit-condition-image-not-match,   nil)
SUIT_Condition //= (suit-condition-use-before,        nil)
```

```
SUIT_Condition //= (suit-condition-minimum-battery,   nil)
SUIT_Condition //= (suit-condition-update-authorized, nil)
SUIT_Condition //= (suit-condition-version,           nil)
SUIT_Condition //= (suit-condition-component-offset,  nil)

SUIT_Directive //= (suit-directive-set-component-index,  uint/bool)
SUIT_Directive //= (suit-directive-set-dependency-index, uint/bool)
SUIT_Directive //= (suit-directive-run-sequence,
    bstr .cbor SUIT_Command_Sequence)
SUIT_Directive //= (suit-directive-try-each,
    SUIT_Directive_Try_Each_Argument)
SUIT_Directive //= (suit-directive-process-dependency,   nil)
SUIT_Directive //= (suit-directive-set-parameters,
    {+ SUIT_Parameters})
SUIT_Directive //= (suit-directive-override-parameters,
    {+ SUIT_Parameters})
SUIT_Directive //= (suit-directive-fetch,                nil)
SUIT_Directive //= (suit-directive-copy,                 nil)
SUIT_Directive //= (suit-directive-swap,                 nil)
SUIT_Directive //= (suit-directive-run,                  nil)
SUIT_Directive //= (suit-directive-wait,                 nil)
SUIT_Directive //= (suit-directive-abort,                nil)

SUIT_Directive_Try_Each_Argument = [
    + bstr .cbor SUIT_Command_Sequence,
    nil / bstr .cbor SUIT_Command_Sequence
]

SUIT_Wait_Event = { + SUIT_Wait_Events }

SUIT_Wait_Events //= (suit-wait-event-authorization => int)
SUIT_Wait_Events //= (suit-wait-event-power => int)
SUIT_Wait_Events //= (suit-wait-event-network => int)
SUIT_Wait_Events //= (suit-wait-event-other-device-version
    => SUIT_Wait_Event_Argument_Other_Device_Version)
SUIT_Wait_Events //= (suit-wait-event-time => uint); Timestamp
SUIT_Wait_Events //= (suit-wait-event-time-of-day
    => uint); Time of Day (seconds since 00:00:00)
SUIT_Wait_Events //= (suit-wait-event-day-of-week
    => uint); Days since Sunday

SUIT_Wait_Event_Argument_Other_Device_Version = [
    other-device: bstr,
    other-device-version: [+int]
]

SUIT_Parameters //= (suit-parameter-vendor-identifier => RFC4122_UUID)
SUIT_Parameters //= (suit-parameter-class-identifier => RFC4122_UUID)
```

```
SUIT_Parameters //= (suit-parameter-image-digest
    => bstr .cbor SUIT_Digest)
SUIT_Parameters //= (suit-parameter-image-size => uint)
SUIT_Parameters //= (suit-parameter-use-before => uint)
SUIT_Parameters //= (suit-parameter-component-offset => uint)

SUIT_Parameters //= (suit-parameter-encryption-info
    => bstr .cbor SUIT_Encryption_Info)
SUIT_Parameters //= (suit-parameter-compression-info
    => bstr .cbor SUIT_Compression_Info)
SUIT_Parameters //= (suit-parameter-unpack-info
    => bstr .cbor SUIT_Unpack_Info)

SUIT_Parameters //= (suit-parameter-uri => tstr)
SUIT_Parameters //= (suit-parameter-source-component => uint)
SUIT_Parameters //= (suit-parameter-run-args => bstr)

SUIT_Parameters //= (suit-parameter-device-identifier => RFC4122_UUID)
SUIT_Parameters //= (suit-parameter-minimum-battery => uint)
SUIT_Parameters //= (suit-parameter-update-priority => uint)
SUIT_Parameters //= (suit-parameter-version =>
    SUIT_Parameter_Version_Match)
SUIT_Parameters //= (suit-parameter-wait-info =>
    bstr .cbor SUIT_Wait_Event)

SUIT_Parameters //= (suit-parameter-custom => int/bool/tstr/bstr)

SUIT_Parameters //= (suit-parameter-strict-order => bool)
SUIT_Parameters //= (suit-parameter-soft-failure => bool)

RFC4122_UUID = bstr .size 16

SUIT_Parameter_Version_Match = [
    suit-condition-version-comparison-type:
        SUIT_Condition_Version_Comparison_Types,
    suit-condition-version-comparison-value:
        SUIT_Condition_Version_Comparison_Value
]
SUIT_Condition_Version_Comparison_Types /=
    suit-condition-version-comparison-greater
SUIT_Condition_Version_Comparison_Types /=
    suit-condition-version-comparison-greater-equal
SUIT_Condition_Version_Comparison_Types /=
    suit-condition-version-comparison-equal
SUIT_Condition_Version_Comparison_Types /=
    suit-condition-version-comparison-lesser-equal
SUIT_Condition_Version_Comparison_Types /=
    suit-condition-version-comparison-lesser
```

```
suit-condition-version-comparison-greater = 1
suit-condition-version-comparison-greater-equal = 2
suit-condition-version-comparison-equal = 3
suit-condition-version-comparison-lesser-equal = 4
suit-condition-version-comparison-lesser = 5

SUIT_Condition_Version_Comparison_Value = [+int]

SUIT_Encryption_Info = COSE_Encrypt_Tagged/COSE_Encrypt0_Tagged
SUIT_Compression_Info = {
    suit-compression-algorithm => SUIT_Compression_Algorithms,
    ? suit-compression-parameters => bstr
}

SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_zlib
SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_brotli
SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_zstd

SUIT_Compression_Algorithm_zlib = 1
SUIT_Compression_Algorithm_brotli = 2
SUIT_Compression_Algorithm_zstd = 3

SUIT_Unpack_Info = {
    suit-unpack-algorithm => SUIT_Unpack_Algorithms,
    ? suit-unpack-parameters => bstr
}

SUIT_Unpack_Algorithms /= SUIT_Unpack_Algorithm_Hex
SUIT_Unpack_Algorithms /= SUIT_Unpack_Algorithm_Elf
SUIT_Unpack_Algorithms /= SUIT_Unpack_Algorithm_Coff
SUIT_Unpack_Algorithms /= SUIT_Unpack_Algorithm_Srec

SUIT_Unpack_Algorithm_Hex = 1
SUIT_Unpack_Algorithm_Elf = 2
SUIT_Unpack_Algorithm_Coff = 3
SUIT_Unpack_Algorithm_Srec = 4

SUIT_Text_Map = {SUIT_Text_Keys => tstr}

SUIT_Text_Keys /= suit-text-manifest-description
SUIT_Text_Keys /= suit-text-update-description
SUIT_Text_Keys /= suit-text-vendor-name
SUIT_Text_Keys /= suit-text-model-name
SUIT_Text_Keys /= suit-text-vendor-domain
SUIT_Text_Keys /= suit-text-model-info
SUIT_Text_Keys /= suit-text-component-description
SUIT_Text_Keys /= suit-text-manifest-json-source
SUIT_Text_Keys /= suit-text-manifest-yaml-source
```

```
SUIT_Text_Keys /= suit-text-version-dependencies

suit-delegation = 1
suit-authentication-wrapper = 2
suit-manifest = 3

suit-manifest-encryption-info = 4
suit-manifest-encrypted       = 5

suit-manifest-version = 1
suit-manifest-sequence-number = 2
suit-common = 3
suit-reference-uri = 4
suit-dependency-resolution = 7
suit-payload-fetch = 8
suit-install = 9
suit-validate = 10
suit-load = 11
suit-run = 12
suit-text = 13
suit-coswid = 14

suit-dependencies = 1
suit-components = 2
suit-dependency-components = 3
suit-common-sequence = 4

suit-dependency-digest = 1
suit-dependency-prefix = 2

suit-component-identifier = 1
suit-component-dependency-index = 2

suit-command-custom = nint

suit-condition-vendor-identifier = 1
suit-condition-class-identifier  = 2
suit-condition-image-match       = 3
suit-condition-use-before        = 4
suit-condition-component-offset  = 5

suit-condition-device-identifier        = 24
suit-condition-image-not-match          = 25
suit-condition-minimum-battery          = 26
suit-condition-update-authorized        = 27
suit-condition-version                  = 28

suit-directive-set-component-index      = 12
```

```
suit-directive-set-dependency-index     = 13
suit-directive-abort                    = 14
suit-directive-try-each                 = 15
;suit-directive-do-each                  = 16 ; TBD
;suit-directive-map-filter               = 17 ; TBD
suit-directive-process-dependency       = 18
suit-directive-set-parameters           = 19
suit-directive-override-parameters      = 20
suit-directive-fetch                    = 21
suit-directive-copy                     = 22
suit-directive-run                      = 23

suit-directive-wait                     = 29
suit-directive-run-sequence             = 30
suit-directive-swap                     = 32

suit-wait-event-authorization = 1
suit-wait-event-power = 2
suit-wait-event-network = 3
suit-wait-event-other-device-version = 4
suit-wait-event-time = 5
suit-wait-event-time-of-day = 6
suit-wait-event-day-of-week = 7

suit-parameter-vendor-identifier = 1
suit-parameter-class-identifier  = 2
suit-parameter-image-digest      = 3
suit-parameter-use-before        = 4
suit-parameter-component-offset  = 5

suit-parameter-strict-order      = 12
suit-parameter-soft-failure      = 13
suit-parameter-image-size        = 14

suit-parameter-encryption-info   = 18
suit-parameter-compression-info  = 19
suit-parameter-unpack-info       = 20
suit-parameter-uri               = 21
suit-parameter-source-component  = 22
suit-parameter-run-args          = 23

suit-parameter-device-identifier = 24
suit-parameter-minimum-battery   = 26
suit-parameter-update-priority   = 27
suit-parameter-version           = 28
suit-parameter-wait-info         = 29
suit-parameter-uri-list          = 30
```

```
suit-parameter-custom = nint

suit-compression-algorithm = 1
suit-compression-parameters = 2

suit-unpack-algorithm  = 1
suit-unpack-parameters = 2

suit-text-manifest-description  = 1
suit-text-update-description    = 2
suit-text-vendor-name           = 3
suit-text-model-name            = 4
suit-text-vendor-domain         = 5
suit-text-model-info            = 6
suit-text-component-description = 7
suit-text-manifest-json-source  = 8
suit-text-manifest-yaml-source  = 9
suit-text-version-dependencies  = 10
```

B. Examples

The following examples demonstrate a small subset of the functionality of the manifest. However, despite this, even a simple manifest processor can execute most of these manifests.

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: Secure Boot

Secure boot and compatibility check.

```
{
    / authentication-wrapper / 2:h'81586fd28443a10126a0582482025820655
```

```
  f1230fd3833ca828c18200498fd1cd90656a9a2620c6989921c06623703515840a0416
  20607b7765a51fe0566e5d8fed95491ee6df622132524fdbe67607bf7f2794d7a71dad
  7230d3cab86c5091a226d00061b0a74a01b3d371e07d5b3eca3d4' / [
          h'd28443a10126a0582482025820655f1230fd3833ca828c18200498fd1cd9
  0656a9a2620c6989921c06623703515840a041620607b7765a51fe0566e5d8fed95491
  ee6df622132524fdbe67607bf7f2794d7a71dad7230d3cab86c5091a226d00061b0a74
  a01b3d371e07d5b3eca3d4' / 18([
                  / protected / h'a10126' / {
                      / alg / 1:-7 / "ES256" /,
                  } /,
                  / unprotected / {
                  },
                  / payload / h'82025820655f1230fd3833ca828c18200498fd1c
  d90656a9a2620c6989921c0662370351' / [
                      / algorithm-id / 2 / "sha256" /,
                      / digest-bytes /
  h'"655f1230fd3833ca828c18200498fd1cd90656a9a2620c6989921c0662370351"'
                  ] /,
                  / signature / h'"a041620607b7765a51fe0566e5d8fed95491e
  e6df622132524fdbe67607bf7f2794d7a71dad7230d3cab86c5091a226d00061b0a74a
  01b3d371e07d5b3eca3d4"'
              ]) /
      ] /,
      / manifest / 3:h'a501010201035860a2024481814100045856861 4a40150fa6
  b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab4503582
  48202582000112233445566778899aabbccddeeff0123456789abcdeffedcba9876543
  2100e1987d001f602f60a438203f60c438217f6' / {
          / manifest-version / 1:1,
          / manifest-sequence-number / 2:1,
          / common / 3:h'a2024481814100045856861 4a40150fa6b4a53d5ad5fdfb
  e9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab450358248202582000112
  233445566778899aabbccddeeff0123456789abcdeffedcba98765432100e1987d001f
  602f6' / {
              / components / 2:h'81814100' / [
                  [h'"00"']
              ] /,
              / common-sequence / 4:h'8614a40150fa6b4a53d5ad5fdfbe9de663
  e4d41ffe02501492af1425695e48bf429b2d51f2ab450358248202582000112233445 5
  66778899aabbccddeeff0123456789abcdeffedcba98765432100e1987d001f602f6'
  / [
                  / directive-override-parameters / 20,{
                      / vendor-id /
  1:h'"fa6b4a53d5ad5fdfbe9de663e4d41ffe"' / fa6b4a53-d5ad-5fdf-
  be9d-e663e4d41ffe /,
                      / class-id /
  2:h'"1492af1425695e48bf429b2d51f2ab45"' /
  1492af14-2569-5e48-bf42-9b2d51f2ab45 /,
                      / image-digest / 3:h'8202582000112233445566778899a
```

```
  abbccddeeff0123456789abcdeffedcba9876543210' / [
                          / algorithm-id / 2 / "sha256" /,
                          / digest-bytes /
  h'"00112233445566778899aabbccddeeff0123456789abcdeffedcba9876543210"'
                      ] /,
                      / image-size / 14:34768,
                  } ,
                  / condition-vendor-identifier / 1,F6 / nil / ,
                  / condition-class-identifier / 2,F6 / nil /
              ] /,
          } /,
          / validate / 10:h'8203f6' / [
              / condition-image-match / 3,F6 / nil /
          ] /,
          / run / 12:h'8217f6' / [
              / directive-run / 23,F6 / nil /
          ] /,
      } /,
  }
```

Total size of manifest without COSE authentication object: 118

Manifest:

```
a1035872a501010201035860a2024481814100045856861 4a40150fa6b4a
53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab
450358248202582000112233445566778899aabbccddeeff0123456789ab
cdeffedcba98765432100e1987d001f602f60a438203f60c438217f6
```

Total size of manifest with COSE authentication object: 235

Manifest with COSE authentication object:

```
a202587281586fd28443a10126a0582482025820655f1230fd3833ca828c
18200498fd1cd90656a9a2620c6989921c06623703515840a041620607b7
765a51fe0566e5d8fed95491ee6df622132524fdbe67607bf7f2794d7a71
dad7230d3cab86c5091a226d00061b0a74a01b3d371e07d5b3eca3d40358
72a501010201035860a2024481814100045856861 4a40150fa6b4a53d5ad
5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab450358
248202582000112233445566778899aabbccddeeff0123456789abcdeffe
dcba98765432100e1987d001f602f60a438203f60c438217f6
```

B.2. Example 1: Simultaneous Download and Installation of Payload

Simultaneous download and installation of payload.

```
  {
      / authentication-wrapper / 2:h'81586fd28443a10126a0582482025820815
```

```
  32771898e4ebcccf12c607420eba62b5086192cac4c99692835b58ee62f7b584081592
  1e5148e9b81e79d8be570de6bb42ba2e903c8549f0e13dee4d0ee420d90dd9f8537ebe
  ad3f92b37df703539879129183b0beaf3ba75cacd8a91e075a24e' / [
          h'd28443a10126a0582482025820815327718 98e4ebcccf12c607420eba62b
  5086192cac4c99692835b58ee62f7b5840815921e5148e9b81e79d8be570de6bb42ba2
  e903c8549f0e13dee4d0ee420d90dd9f8537ebead3f92b37df703539879129183b0bea
  f3ba75cacd8a91e075a24e' / 18([
                  / protected / h'a10126' / {
                      / alg / 1:-7 / "ES256" /,
                  } /,
                  / unprotected / {
                  },
                  / payload / h'8202582081532771898e4ebcccf12c607420eba6
  2b5086192cac4c99692835b58ee62f7b' / [
                      / algorithm-id / 2 / "sha256" /,
                      / digest-bytes /
  h'"81532771898e4ebcccf12c607420eba62b5086192cac4c99692835b58ee62f7b"'
                  ] /,
                  / signature / h'"815921e5148e9b81e79d8be570de6bb42ba2e
  903c8549f0e13dee4d0ee420d90dd9f8537ebead3f92b37df703539879129183b0beaf
  3ba75cacd8a91e075a24e"'
              ]) /
      ] /,
      / manifest / 3:h'a501010202035860a2024481814100045856861 4a40150fa6
  b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab4503582
  48202582000112233445566778899aabbccddeeff0123456789abcdeffedcba9876543
  2100e1987d001f602f60958258613a115781b687474703a2f2f6578616d706c652e636
  f6d2f66696c652e62696e15f603f60a438203f6' / {
          / manifest-version / 1:1,
          / manifest-sequence-number / 2:2,
          / common / 3:h'a2024481814100045856861 4a40150fa6b4a53d5ad5fdfb
  e9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab450358248202582000112
  233445566778899aabbccddeeff0123456789abcdeffedcba98765432100e1987d001f
  602f6' / {
              / components / 2:h'81814100' / [
                  [h'"00"']
              ] /,
              / common-sequence / 4:h'8614a40150fa6b4a53d5ad5fdfbe9de663
  e4d41ffe02501492af1425695e48bf429b2d51f2ab450358248202582000112233445 5
  66778899aabbccddeeff0123456789abcdeffedcba98765432100e1987d001f602f6'
  / [
                  / directive-override-parameters / 20,{
                      / vendor-id /
  1:h'"fa6b4a53d5ad5fdfbe9de663e4d41ffe"' / fa6b4a53-d5ad-5fdf-
  be9d-e663e4d41ffe /,
                      / class-id /
  2:h'"1492af1425695e48bf429b2d51f2ab45"' /
  1492af14-2569-5e48-bf42-9b2d51f2ab45 /,
```

```
                    / image-digest / 3:h'8202582000112233445566778899a
  abbccddeeff0123456789abcdeffedcba9876543210' / [
                        / algorithm-id / 2 / "sha256" /,
                        / digest-bytes /
  h'"00112233445566778899aabbccddeeff0123456789abcdeffedcba9876543210"'
                    ] /,
                    / image-size / 14:34768,
                } ,
                / condition-vendor-identifier / 1,F6 / nil / ,
                / condition-class-identifier / 2,F6 / nil /
            ] /,
        } /,
        / install / 9:h'8613a115781b687474703a2f2f6578616d706c652e636f
  6d2f66696c652e62696e15f603f6' / [
            / directive-set-parameters / 19,{
                / uri / 21:'http://example.com/file.bin',
            } ,
            / directive-fetch / 21,F6 / nil / ,
            / condition-image-match / 3,F6 / nil /
        ] /,
        / validate / 10:h'8203f6' / [
            / condition-image-match / 3,F6 / nil /
        ] /,
    } /,
  }
```

Total size of manifest without COSE authentication object: 153

Manifest:

```
a1035895a501010202035860a2024481814100045856861 4a40150fa6b4a
53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab
450358248202582000112233445566778899aabbccddeeff0123456789ab
cdeffedcba98765432100e1987d001f602f60958258613a115781b687474
703a2f2f6578616d706c652e636f6d2f66696c652e62696e15f603f60a43
8203f6
```

Total size of manifest with COSE authentication object: 270

Manifest with COSE authentication object:

```
a202587281586fd28443a10126a0582482025820815327718 98e4ebcccf1
2c607420eba62b5086192cac4c99692835b58ee62f7b5840815921e5148e
9b81e79d8be570de6bb42ba2e903c8549f0e13dee4d0ee420d90dd9f8537
ebead3f92b37df703539879129183b0beaf3ba75cacd8a91e075a24e0358
95a501010202035860a2024481814100045856861 4a40150fa6b4a53d5ad
5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab450358
248202582000112233445566778899aabbccddeeff0123456789abcdeffe
dcba98765432100e1987d001f602f60958258613a115781b687474703a2f
2f6578616d706c652e636f6d2f66696c652e62696e15f603f60a438203f6
```

B.3. Example 2: Simultaneous Download, Installation, and Secure Boot

Compatibility test, simultaneous download and installation, and secure boot.

```
{
    / authentication-wrapper / 2:h'81586fd28443a10126a0582482025820883
90f8988639d8a2cfb6da969fce488333ac5ba77aaf0d66b5623009bbf341158401929f
fd488c455ab40eaf1aa96a7df4a9c16c658221055c3a113232fb81c5751a23a74b5efc
06c459eb47a07028ef3c6a0d9051185dd78899c654249f9070dea' / [
        h'd28443a10126a0582482025820883 90f8988639d8a2cfb6da969fce48833
3ac5ba77aaf0d66b5623009bbf341158401929ffd488c455ab40eaf1aa96a7df4a9c16
c658221055c3a113232fb81c5751a23a74b5efc06c459eb47a07028ef3c6a0d9051185
dd78899c654249f9070dea' / 18([
                / protected / h'a10126' / {
                    / alg / 1:-7 / "ES256" /,
                } /,
                / unprotected / {
                },
                / payload / h'8202582088390f8988639d8a2cfb6da969fce488
333ac5ba77aaf0d66b5623009bbf3411' / [
                    / algorithm-id / 2 / "sha256" /,
                    / digest-bytes /
h'"88390f8988639d8a2cfb6da969fce488333ac5ba77aaf0d66b5623009bbf3411"'
                ] /,
                / signature / h'"1929ffd488c455ab40eaf1aa96a7df4a9c16c
658221055c3a113232fb81c5751a23a74b5efc06c459eb47a07028ef3c6a0d9051185d
d78899c654249f9070dea"'
            ]) /
    ] /,
    / manifest / 3:h'a601010203035860a202448181410004585686 14a40150fa6
b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab4503582
48202582000112233445566778899aabbccddeeff0123456789abcdeffedcba9876543
2100e1987d001f602f60958258613a115781b687474703a2f2f6578616d706c652e636
f6d2f66696c652e62696e15f603f60a438203f60c438217f6' / {
        / manifest-version / 1:1,
        / manifest-sequence-number / 2:3,
        / common / 3:h'a202448181410004585686 14a40150fa6b4a53d5ad5fdfb
```

```
  e9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab45035824820258200011
  233445566778899aabbccddeeff0123456789abcdeffedcba98765432100e1987d001f
  602f6' / {
              / components / 2:h'81814100' / [
                  [h'"00"']
              ] /,
              / common-sequence / 4:h'8614a40150fa6b4a53d5ad5fdfbe9de663
  e4d41ffe02501492af1425695e48bf429b2d51f2ab450358248202582000011223344555
  66778899aabbccddeeff0123456789abcdeffedcba98765432100e1987d001f602f6'
  / [
                  / directive-override-parameters / 20,{
                      / vendor-id /
  1:h'"fa6b4a53d5ad5fdfbe9de663e4d41ffe"' / fa6b4a53-d5ad-5fdf-
  be9d-e663e4d41ffe /,
                      / class-id /
  2:h'"1492af1425695e48bf429b2d51f2ab45"' /
  1492af14-2569-5e48-bf42-9b2d51f2ab45 /,
                      / image-digest / 3:h'8202582000112233445566778899a
  abbccddeeff0123456789abcdeffedcba9876543210' / [
                          / algorithm-id / 2 / "sha256" /,
                          / digest-bytes /
  h'"00112233445566778899aabbccddeeff0123456789abcdeffedcba9876543210"'
                      ] /,
                      / image-size / 14:34768,
                  } ,
                  / condition-vendor-identifier / 1,F6 / nil / ,
                  / condition-class-identifier / 2,F6 / nil /
              ] /,
          } /,
          / install / 9:h'8613a115781b687474703a2f2f6578616d706c652e636f
  6d2f66696c652e62696e15f603f6' / [
              / directive-set-parameters / 19,{
                  / uri / 21:'http://example.com/file.bin',
              } ,
              / directive-fetch / 21,F6 / nil / ,
              / condition-image-match / 3,F6 / nil /
          ] /,
          / validate / 10:h'8203f6' / [
              / condition-image-match / 3,F6 / nil /
          ] /,
          / run / 12:h'8217f6' / [
              / directive-run / 23,F6 / nil /
          ] /,
      } /,
  }
```

Total size of manifest without COSE authentication object: 158

Manifest:

```
a103589aa601010203035860a2024481814100045856861 4a40150fa6b4a
53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab
450358248202582000112233445566778899aabbccddeeff0123456789ab
cdeffedcba98765432100e1987d001f602f60958258613a115781b687474
703a2f2f6578616d706c652e636f6d2f66696c652e62696e15f603f60a43
8203f60c438217f6
```

Total size of manifest with COSE authentication object: 275

Manifest with COSE authentication object:

```
a202587281586fd28443a10126a0582482025820883 90f8988639d8a2cfb
6da969fce488333ac5ba77aaf0d66b5623009bbf341 15840192 9ffd488c4
55ab40eaf1aa96a7df4a9c16c658221055c3a113232fb81c5751a23a74b5
efc06c459eb47a07028ef3c6a0d9051185dd78899c654249f9070dea0358
9aa601010203035860a2024481814100045856861 4a40150fa6b4a53d5ad
5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab450358
248202582000112233445566778899aabbccddeeff0123456789abcdeffe
dcba98765432100e1987d001f602f60958258613a115781b687474703a2f
2f6578616d706c652e636f6d2f66696c652e62696e15f603f60a438203f6
0c438217f6
```

B.4. Example 3: Load from External Storage

Compatibility test, simultaneous download and installation, load from external storage, and secure boot.

```
{
    / authentication-wrapper / 2:h'81586fd28443a10126a0582482025820568
56a72f9ac0ee73b4ea3a866cf2e5c990e8ed8c6056608bc221efd42172b2758402a9d7
573ef6dcf5653b39027fdf87b81adeb0f03122bef0ecf5af9c7d77323c32827230f660
8342b7bf5c125f17148bd67880420ab0d03e235e6ca1d15127499' / [
        h'd28443a10126a0582482025820568 56a72f9ac0ee73b4ea3a866cf2e5c99
0e8ed8c6056608bc221efd42172b2758402a9d7573ef6dcf5653b39027fdf87b81adeb
0f03122bef0ecf5af9c7d77323c32827230f6608342b7bf5c125f17148bd67880420ab
0d03e235e6ca1d15127499' / 18([
                / protected / h'a10126' / {
                    / alg / 1:-7 / "ES256" /,
                } /,
                / unprotected / {
                },
                / payload / h'8202582056856a72f9ac0ee73b4ea3a866cf2e5c
990e8ed8c6056608bc221efd42172b27' / [
                    / algorithm-id / 2 / "sha256" /,
                    / digest-bytes /
h'"56856a72f9ac0ee73b4ea3a866cf2e5c990e8ed8c6056608bc221efd42172b27"'
```

```
                ] /,
                / signature / h'"2a9d7573ef6dcf5653b39027fdf87b81adeb0
  f03122bef0ecf5af9c7d77323c32827230f6608342b7bf5c125f17148bd67880420ab0
  d03e235e6ca1d15127499"'
            ]) /
        ] /,
        / manifest / 3:h'a701010204035865a2024782814100814101045858880c001
  4a40150fa6b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f
  2ab450358248202582000112233445566778899aabbccddeeff0123456789abcdeffed
  cba98765432100e1987d001f602f6095827880c0013a115781b687474703a2f2f65786
  16d706c652e636f6d2f66696c652e62696e15f603f60a45840c0003f60b4b880c0113a
  1160016f603f60c45840c0117f6' / {
           / manifest-version / 1:1,
           / manifest-sequence-number / 2:4,
           / common / 3:h'a2024782814100814101045858880c0014a40150fa6b4a5
  3d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab45035824820
  2582000112233445566778899aabbccddeeff0123456789abcdeffedcba98765432100
  e1987d001f602f6' / {
               / components / 2:h'82814100814101' / [
                   [h'"00"'] ,
                   [h'"01"']
               ] /,
               / common-sequence / 4:h'880c0014a40150fa6b4a53d5ad5fdfbe9d
  e663e4d41ffe02501492af1425695e48bf429b2d51f2ab450358248202582000112233
  445566778899aabbccddeeff0123456789abcdeffedcba98765432100e1987d001f602
  f6' / [
                   / directive-set-component-index / 12,0 ,
                   / directive-override-parameters / 20,{
                       / vendor-id /
  1:h'"fa6b4a53d5ad5fdfbe9de663e4d41ffe"' / fa6b4a53-d5ad-5fdf-
  be9d-e663e4d41ffe /,
                       / class-id /
  2:h'"1492af1425695e48bf429b2d51f2ab45"' /
  1492af14-2569-5e48-bf42-9b2d51f2ab45 /,
                       / image-digest / 3:h'8202582000112233445566778899a
  abbccddeeff0123456789abcdeffedcba9876543210' / [
                            / algorithm-id / 2 / "sha256" /,
                            / digest-bytes /
  h'"00112233445566778899aabbccddeeff0123456789abcdeffedcba9876543210"'
                       ] /,
                       / image-size / 14:34768,
                   } ,
                   / condition-vendor-identifier / 1,F6 / nil / ,
                   / condition-class-identifier / 2,F6 / nil /
               ] /,
           } /,
           / install / 9:h'880c0013a115781b687474703a2f2f6578616d706c652e
  636f6d2f66696c652e62696e15f603f6' / [
```

```
            / directive-set-component-index / 12,0 ,
            / directive-set-parameters / 19,{
                / uri / 21:'http://example.com/file.bin',
            } ,
            / directive-fetch / 21,F6 / nil / ,
            / condition-image-match / 3,F6 / nil /
        ] /,
        / validate / 10:h'840c0003f6' / [
            / directive-set-component-index / 12,0 ,
            / condition-image-match / 3,F6 / nil /
        ] /,
        / load / 11:h'880c0113a1160016f603f6' / [
            / directive-set-component-index / 12,1 ,
            / directive-set-parameters / 19,{
                / source-component / 22:0 / [h'"00"'] /,
            } ,
            / directive-copy / 22,F6 / nil / ,
            / condition-image-match / 3,F6 / nil /
        ] /,
        / run / 12:h'840c0117f6' / [
            / directive-set-component-index / 12,1 ,
            / directive-run / 23,F6 / nil /
        ] /,
    } /,
}
```

Total size of manifest without COSE authentication object: 182

Manifest:

```
a10358b2a701010204035865a2024782814100814101045858880c0014a4
0150fa6b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf42
9b2d51f2ab450358248202582000112233445566778899aabbccddeeff01
23456789abcdeffedcba98765432100e1987d001f602f6095827880c0013
a115781b687474703a2f2f6578616d706c652e636f6d2f66696c652e6269
6e15f603f60a45840c0003f60b4b880c0113a1160016f603f60c45840c01
17f6
```

Total size of manifest with COSE authentication object: 299

Manifest with COSE authentication object:

```
a202587281586fd28443a10126a05824820258205685 6a72f9ac0ee73b4e
a3a866cf2e5c990e8ed8c6056608bc221efd42172b2758402a9d7573ef6d
cf5653b39027fdf87b81adeb0f03122bef0ecf5af9c7d77323c32827230f
6608342b7bf5c125f17148bd67880420ab0d03e235e6ca1d151274990358
b2a701010204035865a2024782814100814101045858880c0014a40150fa
6b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51
f2ab450358248202582000112233445566778899aabbccddeeff01234567
89abcdeffedcba98765432100e1987d001f602f6095827880c0013a11578
1b687474703a2f2f6578616d706c652e636f6d2f66696c652e62696e15f6
03f60a45840c0003f60b4b880c0113a1160016f603f60c45840c0117f6
```

B.5. Example 4: Load and Decompress from External Storage

Compatibility test, simultaneous download and installation, load and decompress from external storage, and secure boot.

```
{
    / authentication-wrapper / 2:h'81586fd28443a10126a058248202582057b
edc0076919ba83908365faf6d205e95c71268d29a94dc5e82698edd3a48225840e0a4d
c500266518742802f2364b65f983175f060c1555d3d0b186f447500ba60c66e3231674
1c3b642c68fed73d47542c3375c0ab72e0f4b94ec392ab398599d' / [
        h'd28443a10126a058248202582057bedc0076919ba83908365faf6d205e95
c71268d29a94dc5e82698edd3a48225840e0a4dc500266518742802f2364b65f983175
f060c1555d3d0b186f447500ba60c66e32316741c3b642c68fed73d47542c3375c0ab7
2e0f4b94ec392ab398599d' / 18([
                / protected / h'a10126' / {
                    / alg / 1:-7 / "ES256" /,
                } /,
                / unprotected / {
                },
                / payload / h'8202582057bedc0076919ba83908365faf6d205e
95c71268d29a94dc5e82698edd3a4822' / [
                    / algorithm-id / 2 / "sha256" /,
                    / digest-bytes /
h'"57bedc0076919ba83908365faf6d205e95c71268d29a94dc5e82698edd3a4822"'
                ] /,
                / signature / h'"e0a4dc500266518742802f2364b65f983175f
060c1555d3d0b186f447500ba60c66e32316741c3b642c68fed73d47542c3375c0ab72
e0f4b94ec392ab398599d"'
            ]) /
    ] /,
    / manifest / 3:h'a701010205035865a2024782814100814101045858880c001
4a40150fa6b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f
2ab450358248202582000112233445566778899aabbccddeeff0123456789abcdeffed
cba98765432100e1987d001f602f6095827880c0013a115781b687474703a2f2f65786
16d706c652e636f6d2f66696c652e62696e15f603f60a45840c0003f60b4d880c0113a
21301160016f603f60c45840c0117f6' / {
        / manifest-version / 1:1,
```

```
        / manifest-sequence-number / 2:5,
        / common / 3:h'a2024782814100814101045858880c0014a40150fa6b4a5
  3d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab45035824820
  2582000112233445566778899aabbccddeeff0123456789abcdeffedcba98765432100
  e1987d001f602f6' / {
            / components / 2:h'82814100814101' / [
                [h'"00"'] ,
                [h'"01"']
            ] /,
            / common-sequence / 4:h'880c0014a40150fa6b4a53d5ad5fdfbe9d
  e663e4d41ffe02501492af1425695e48bf429b2d51f2ab450358248202582000112233
  445566778899aabbccddeeff0123456789abcdeffedcba98765432100e1987d001f602
  f6' / [
                / directive-set-component-index / 12,0 ,
                / directive-override-parameters / 20,{
                    / vendor-id /
  1:h'"fa6b4a53d5ad5fdfbe9de663e4d41ffe"' / fa6b4a53-d5ad-5fdf-
  be9d-e663e4d41ffe /,
                    / class-id /
  2:h'"1492af1425695e48bf429b2d51f2ab45"' /
  1492af14-2569-5e48-bf42-9b2d51f2ab45 /,
                    / image-digest / 3:h'8202582000112233445566778899a
  abbccddeeff0123456789abcdeffedcba9876543210' / [
                        / algorithm-id / 2 / "sha256" /,
                        / digest-bytes /
  h'"00112233445566778899aabbccddeeff0123456789abcdeffedcba9876543210"'
                    ] /,
                    / image-size / 14:34768,
                } ,
                / condition-vendor-identifier / 1,F6 / nil / ,
                / condition-class-identifier / 2,F6 / nil /
            ] /,
        } /,
        / install / 9:h'880c0013a115781b687474703a2f2f6578616d706c652e
  636f6d2f66696c652e62696e15f603f6' / [
            / directive-set-component-index / 12,0 ,
            / directive-set-parameters / 19,{
                / uri / 21:'http://example.com/file.bin',
            } ,
            / directive-fetch / 21,F6 / nil / ,
            / condition-image-match / 3,F6 / nil /
        ] /,
        / validate / 10:h'840c0003f6' / [
            / directive-set-component-index / 12,0 ,
            / condition-image-match / 3,F6 / nil /
        ] /,
        / load / 11:h'880c0113a21301160016f603f6' / [
            / directive-set-component-index / 12,1 ,
```

```
            / directive-set-parameters / 19,{
                / source-component / 22:0 / [h'"00"'] /,
                / compression-info / 19:1 / "gzip" /,
            } ,
            / directive-copy / 22,F6 / nil / ,
            / condition-image-match / 3,F6 / nil /
        ] /,
        / run / 12:h'840c0117f6' / [
            / directive-set-component-index / 12,1 ,
            / directive-run / 23,F6 / nil /
        ] /,
    } /,
}
```

Total size of manifest without COSE authentication object: 184

Manifest:

```
a10358b4a701010205035865a2024782814100814101045858880c0014a4
0150fa6b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf42
9b2d51f2ab450358248202582000112233445566778899aabbccddeeff01
23456789abcdeffedcba98765432100e1987d001f602f6095827880c0013
a115781b687474703a2f2f6578616d706c652e636f6d2f66696c652e6269
6e15f603f60a45840c0003f60b4d880c0113a21301160016f603f60c4584
0c0117f6
```

Total size of manifest with COSE authentication object: 301

Manifest with COSE authentication object:

```
a202587281586fd28443a10126a058248202582057bedc0076919ba83908
365faf6d205e95c71268d29a94dc5e82698edd3a48225840e0a4dc500266
518742802f2364b65f983175f060c1555d3d0b186f447500ba60c66e3231
6741c3b642c68fed73d47542c3375c0ab72e0f4b94ec392ab398599d0358
b4a701010205035865a2024782814100814101045858880c0014a40150fa
6b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51
f2ab450358248202582000112233445566778899aabbccddeeff01234567
89abcdeffedcba98765432100e1987d001f602f6095827880c0013a11578
1b687474703a2f2f6578616d706c652e636f6d2f66696c652e62696e15f6
03f60a45840c0003f60b4d880c0113a21301160016f603f60c45840c0117
f6
```

B.6. Example 5: Compatibility Test, Download, Installation, and Secure Boot

Compatibility test, download, installation, and secure boot.

```
{
```

```
    / authentication-wrapper / 2:h'81586fd28443a10126a0582482025820ecc
  95235f2ab00b9912f8189b213b3e4ade42b792f491644e76004cd2ba87dc8584093952
  6b77d63dac2e138bf074aac757c5f010e8b2cf3ae9fcbba4cafc2d0f81c9ae46bc973c
  c0565410a1cb6bf10d2b3d0a2865392255cc4288d0337af3de837' / [
        h'd28443a10126a0582482025820ecc95235f2ab00b9912f8189b213b3e4ad
  e42b792f491644e76004cd2ba87dc85840939526b77d63dac2e138bf074aac757c5f01
  0e8b2cf3ae9fcbba4cafc2d0f81c9ae46bc973cc0565410a1cb6bf10d2b3d0a2865392
  255cc4288d0337af3de837' / 18([
                / protected / h'a10126' / {
                    / alg / 1:-7 / "ES256" /,
                } /,
                / unprotected / {
                },
                / payload / h'82025820ecc95235f2ab00b9912f8189b213b3e4
  ade42b792f491644e76004cd2ba87dc8' / [
                    / algorithm-id / 2 / "sha256" /,
                    / digest-bytes /
  h'"ecc95235f2ab00b9912f8189b213b3e4ade42b792f491644e76004cd2ba87dc8"'
                ] /,
                / signature / h'"939526b77d63dac2e138bf074aac757c5f010
  e8b2cf3ae9fcbba4cafc2d0f81c9ae46bc973cc0565410a1cb6bf10d2b3d0a28653922
  55cc4288d0337af3de837"'
            ]) /
    ] /,
    / manifest / 3:h'a701010205035865a2024782814100814101045858880c001
  4a40150fa6b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f
  2ab450358248202582000112233445566778899aabbccddeeff0123456789abcdeffed
  cba98765432100e1987d001f602f6085823840c0113a115781b687474703a2f2f65786
  16d706c652e636f6d2f66696c652e62696e094b880c0013a1160116f603f60a45840c0
  003f60c45840c0017f6' / {
        / manifest-version / 1:1,
        / manifest-sequence-number / 2:5,
        / common / 3:h'a2024782814100814101045858880c0014a40150fa6b4a5
  3d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab45035824820
  2582000112233445566778899aabbccddeeff0123456789abcdeffedcba98765432100
  e1987d001f602f6' / {
            / components / 2:h'82814100814101' / [
                [h'"00"'] ,
                [h'"01"']
            ] /,
            / common-sequence / 4:h'880c0014a40150fa6b4a53d5ad5fdfbe9d
  e663e4d41ffe02501492af1425695e48bf429b2d51f2ab450358248202582000112233
  445566778899aabbccddeeff0123456789abcdeffedcba98765432100e1987d001f602
  f6' / [
                / directive-set-component-index / 12,0 ,
                / directive-override-parameters / 20,{
                    / vendor-id /
  1:h'"fa6b4a53d5ad5fdfbe9de663e4d41ffe"' / fa6b4a53-d5ad-5fdf-
```

```
  be9d-e663e4d41ffe /,
                      / class-id /
  2:h'"1492af1425695e48bf429b2d51f2ab45"' /
  1492af14-2569-5e48-bf42-9b2d51f2ab45 /,
                      / image-digest / 3:h'8202582000112233445566778899a
  abbccddeeff0123456789abcdeffedcba9876543210' / [
                          / algorithm-id / 2 / "sha256" /,
                          / digest-bytes /
  h'"00112233445566778899aabbccddeeff0123456789abcdeffedcba9876543210"'
                      ] /,
                      / image-size / 14:34768,
                  } ,
                  / condition-vendor-identifier / 1,F6 / nil / ,
                  / condition-class-identifier / 2,F6 / nil /
              ] /,
          } /,
          / payload-fetch / 8:h'840c0113a115781b687474703a2f2f6578616d70
  6c652e636f6d2f66696c652e62696e' / [
              / directive-set-component-index / 12,1 ,
              / directive-set-parameters / 19,{
                  / uri / 21:'http://example.com/file.bin',
              }
          ] /,
          / install / 9:h'880c0013a1160116f603f6' / [
              / directive-set-component-index / 12,0 ,
              / directive-set-parameters / 19,{
                  / source-component / 22:1 / [h'"01"'] /,
              } ,
              / directive-copy / 22,F6 / nil / ,
              / condition-image-match / 3,F6 / nil /
          ] /,
          / validate / 10:h'840c0003f6' / [
              / directive-set-component-index / 12,0 ,
              / condition-image-match / 3,F6 / nil /
          ] /,
          / run / 12:h'840c0017f6' / [
              / directive-set-component-index / 12,0 ,
              / directive-run / 23,F6 / nil /
          ] /,
      } /,
  }
```

Total size of manifest without COSE authentication object: 178

Manifest:

```
a10358aea701010205035865a2024782814100814101045858880c0014a4
0150fa6b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf42
9b2d51f2ab450358248202582000112233445566778899aabbccddeeff01
23456789abcdeffedcba98765432100e1987d001f602f6085823840c0113
a115781b687474703a2f2f6578616d706c652e636f6d2f66696c652e6269
6e094b880c0013a1160116f603f60a45840c0003f60c45840c0017f6
```

Total size of manifest with COSE authentication object: 295

Manifest with COSE authentication object:

```
a202587281586fd28443a10126a0582482025820ecc95235f2ab00b9912f
8189b213b3e4ade42b792f491644e76004cd2ba87dc85840939526b77d63
dac2e138bf074aac757c5f010e8b2cf3ae9fcbba4cafc2d0f81c9ae46bc9
73cc0565410a1cb6bf10d2b3d0a2865392255cc4288d0337af3de8370358
aea701010205035865a2024782814100814101045858880c0014a40150fa
6b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51
f2ab450358248202582000112233445566778899aabbccddeeff01234567
89abcdeffedcba98765432100e1987d001f602f6085823840c0113a11578
1b687474703a2f2f6578616d706c652e636f6d2f66696c652e62696e094b
880c0013a1160116f603f60a45840c0003f60c45840c0017f6
```

B.7. Example 6: Two Images

Compatibility test, 2 images, simultaneous download and installation, and secure boot.

```
{
    / authentication-wrapper / 2:h'81586fd28443a10126a0582482025820732
5a7d3acf130d161810c4874f275f658970b7bc5a63cda56e9920a4aaba3a3584088cb9
6211bcc4cdb59cb0022cb213017b2d117bac1a5460ae92903acc196282f7888368bf0a
065756e43f53cdbeee367e9523312063e8eaad0889a7cee371859' / [
        h'd28443a10126a05824820258207325a7d3acf130d161810c4874f275f658
970b7bc5a63cda56e9920a4aaba3a3584088cb96211bcc4cdb59cb0022cb213017b2d1
17bac1a5460ae92903acc196282f7888368bf0a065756e43f53cdbeee367e952331206
3e8eaad0889a7cee371859' / 18([
                / protected / h'a10126' / {
                    / alg / 1:-7 / "ES256" /,
                } /,
                / unprotected / {
                },
                / payload / h'820258207325a7d3acf130d161810c4874f275f6
58970b7bc5a63cda56e9920a4aaba3a3' / [
                    / algorithm-id / 2 / "sha256" /,
                    / digest-bytes /
h'"7325a7d3acf130d161810c4874f275f658970b7bc5a63cda56e9920a4aaba3a3"'
                ] /,
                / signature / h'"88cb96211bcc4cdb59cb0022cb213017b2d11
```

```
  7bac1a5460ae92903acc196282f7888368bf0a065756e43f53cdbeee367e9523312063
  e8eaad0889a7cee371859"'
              ]) /
      ] /,
      / manifest / 3:h'a50101020303589da202448181410004589388l4a20150fa6
  b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab450f825
  8308405f614a203582482025820001122334455667788899aabbccddeeff0123456789a
  bcdeffedcba98765432100e1987d058328405f614a203582482025820012345678 9abc
  deffedcba98765432100011223344556778899aabbccddeeff0e1a00012c2201f602f
  6095853860f8258248405f613a115781c687474703a2f2f6578616d706c652e636f6d2
  f66696c65312e62696e58248405f613a115781c687474703a2f2f6578616d706c652e6
  36f6d2f66696c65322e62696e15f603f60a438203f6' / {
          / manifest-version / 1:1,
          / manifest-sequence-number / 2:3,
          / common / 3:h'a202448181410004589388l4a20150fa6b4a53d5ad5fdfb
  e9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab450f8258308405f614a20
  3582482025820001122334455667788899aabbccddeeff0123456789abcdeffedcba987
  65432100e1987d058328405f614a203582482025820012345678 9abcdeffedcba98765
  432100011223344556778899aabbccddeeff0e1a00012c2201f602f6' / {
              / components / 2:h'81814100' / [
                  [h'"00"']
              ] /,
              / common-sequence / 4:h'8814a20150fa6b4a53d5ad5fdfbe9de663
  e4d41ffe02501492af1425695e48bf429b2d51f2ab450f8258308405f614a203582482
  0258200011223344556778899aabbccddeeff0123456789abcdeffedcba9876543210
  0e1987d058328405f614a203582482025820012345678 9abcdeffedcba987654321000
  11223344556778899aabbccddeeff0e1a00012c2201f602f6' / [
                  / directive-override-parameters / 20,{
                      / vendor-id /
  1:h'"fa6b4a53d5ad5fdfbe9de663e4d41ffe"' / fa6b4a53-d5ad-5fdf-
  be9d-e663e4d41ffe /,
                      / class-id /
  2:h'"1492af1425695e48bf429b2d51f2ab45"' /
  1492af14-2569-5e48-bf42-9b2d51f2ab45 /,
                  } ,
                  / directive-try-each / 15,[
                      h'8405f614a203582482025820001122334455667788899aabb
  ccddeeff0123456789abcdeffedcba98765432100e1987d0' / [
                          / condition-component-offset / 5,F6 / nil / ,
                          / directive-override-parameters / 20,{
                              / image-digest / 3:h'820258200011223344556
  6778899aabbccddeeff0123456789abcdeffedcba9876543210' / [
                                  / algorithm-id / 2 / "sha256" /,
                                  / digest-bytes /
  h'"00112233445566778899aabbccddeeff0123456789abcdeffedcba9876543210"'
                              ] /,
                              / image-size / 14:34768,
                          }
```

```
                    ] / ,
                    h'8405f614a203582482025820123456789abcdeffedcba98
  765432100011223344556677889 9aabbccddeeff0e1a00012c22' / [
                        / condition-component-offset / 5,F6 / nil / ,
                        / directive-override-parameters / 20,{
                            / image-digest / 3:h'820258200123456789abc
  deffedcba9876543210001122334455667788 99aabbccddeeff' / [
                                / algorithm-id / 2 / "sha256" /,
                                / digest-bytes /
  h'"0123456789abcdeffedcba9876543210001122334455667788 99aabbccddeeff"'
                                ] /,
                                / image-size / 14:76834,
                            }
                        ] /
                    ] ,
                    / condition-vendor-identifier / 1,F6 / nil / ,
                    / condition-class-identifier / 2,F6 / nil /
                ] /,
            } /,
            / install / 9:h'860f8258248405f613a115781c687474703a2f2f657861
  6d706c652e636f6d2f66696c65312e62696e58248405f613a115781c687474703a2f2f
  6578616d706c652e636f6d2f66696c65322e62696e15f603f6' / [
                / directive-try-each / 15,[
                    h'8405f613a115781c687474703a2f2f6578616d706c652e636f6d
  2f66696c65312e62696e' / [
                        / condition-component-offset / 5,F6 / nil / ,
                        / directive-set-parameters / 19,{
                            / uri / 21:'http://example.com/file1.bin',
                        }
                    ] / ,
                    h'8405f613a115781c687474703a2f2f6578616d706c652e636f6d
  2f66696c65322e62696e' / [
                        / condition-component-offset / 5,F6 / nil / ,
                        / directive-set-parameters / 19,{
                            / uri / 21:'http://example.com/file2.bin',
                        }
                    ] /
                ] ,
                / directive-fetch / 21,F6 / nil / ,
                / condition-image-match / 3,F6 / nil /
            ] /,
            / validate / 10:h'8203f6' / [
                / condition-image-match / 3,F6 / nil /
            ] /,
        } /,
  }
```

Total size of manifest without COSE authentication object: 261

Manifest:

```
a103590100a50101020303589da202448181410004589388l4a20150fa6b
4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2
ab450f8258308405f614a2035824820258200011223344556677889aabb
ccddeeff0123456789abcdeffedcba98765432100e1987d058328405f614
a2035824820258200123456789abcdeffedcba9876543210001122334455
66778899aabbccddeeff0e1a00012c2201f602f6095853860f8258248405
f613a115781c687474703a2f2f6578616d706c652e636f6d2f66696c6531
2e62696e58248405f613a115781c687474703a2f2f6578616d706c652e63
6f6d2f66696c65322e62696e15f603f60a438203f6
```

Total size of manifest with COSE authentication object: 378

Manifest with COSE authentication object:

```
a202587281586fd28443a10126a05824820258207325a7d3acf130d16181
0c4874f275f658970b7bc5a63cda56e9920a4aaba3a3584088cb96211bcc
4cdb59cb0022cb213017b2d117bac1a5460ae92903acc196282f7888368b
f0a065756e43f53cdbeee367e9523312063e8eaad0889a7cee3718590359
0100a50101020303589da202448181410004589388l4a20150fa6b4a53d5
ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab450f
8258308405f614a2035824820258200011223344556677889aabbccddee
ff0123456789abcdeffedcba98765432100e1987d058328405f614a20358
248202582001234567890abcdeffedcba9876543210001122334455667788
99aabbccddeeff0e1a00012c2201f602f6095853860f8258248405f613a1
15781c687474703a2f2f6578616d706c652e636f6d2f66696c65312e6269
6e58248405f613a115781c687474703a2f2f6578616d706c652e636f6d2f
66696c65322e62696e15f603f60a438203f6
```

C. Design Rational

In order to provide flexible behavior to constrained devices, while still allowing more powerful devices to use their full capabilities, the SUIT manifest encodes the required behavior of a Recipient device. Behavior is encoded as a specialized byte code, contained in a CBOR list. This promotes a flat encoding, which simplifies the parser. The information encoded by this byte code closely matches the operations that a device will perform, which promotes ease of processing. The core operations used by most update and trusted execution operations are represented in the byte code. The byte code can be extended by registering new operations.

The specialized byte code approach gives benefits equivalent to those provided by a scripting language or conventional byte code, with two substantial differences. First, the language is extremely high level, consisting of only the operations that a device may perform during update and trusted execution of a firmware image. Second, the

language specifies linear behavior, without reverse branches. Conditional processing is supported, and parallel and out-of-order processing may be performed by sufficiently capable devices.

By structuring the data in this way, the manifest processor becomes a very simple engine that uses a pull parser to interpret the manifest. This pull parser invokes a series of command handlers that evaluate a Condition or execute a Directive. Most data is structured in a highly regular pattern, which simplifies the parser.

The results of this allow a Recipient to implement a very small parser for constrained applications. If needed, such a parser also allows the Recipient to perform complex updates with reduced overhead. Conditional execution of commands allows a simple device to perform important decisions at validation-time.

Dependency handling is vastly simplified as well. Dependencies function like subroutines of the language. When a manifest has a dependency, it can invoke that dependency's commands and modify their behavior by setting parameters. Because some parameters come with security implications, the dependencies also have a mechanism to reject modifications to parameters on a fine-grained level.

Developing a robust permissions system works in this model too. The Recipient can use a simple ACL that is a table of Identities and Component Identifier permissions to ensure that operations on components fail unless they are permitted by the ACL. This table can be further refined with individual parameters and commands.

Capability reporting is similarly simplified. A Recipient can report the Commands, Parameters, Algorithms, and Component Identifiers that it supports. This is sufficiently precise for a manifest author to create a manifest that the Recipient can accept.

The simplicity of design in the Recipient due to all of these benefits allows even a highly constrained platform to use advanced update capabilities.

D. Implementation Conformance Matrix

This section summarizes the functionality a minimal implementation needs to offer to claim conformance to this specification, in the absence of an application profile standard specifying otherwise.

The subsequent table shows the conditions.

Name	Reference	Implementation
Vendor Identifier	Section 9.8.3.1	REQUIRED
Class Identifier	Section 9.8.3.1	REQUIRED
Device Identifier	Section 9.8.3.1	OPTIONAL
Image Match	Section 9.8.3.2	REQUIRED
Image Not Match	Section 9.8.3.3	OPTIONAL
Use Before	Section 9.8.3.4	OPTIONAL
Component Offset	Section 9.8.3.5	OPTIONAL
Minimum Battery	Section 9.8.3.6	OPTIONAL
Update Authorized	Section 9.8.3.7	OPTIONAL
Version	Section 9.8.3.8	OPTIONAL
Custom Condition	Section 9.8.3.9	OPTIONAL

The subsequent table shows the directives.

Name	Reference	Implementation
Set Component Index	Section 9.8.4.1	REQUIRED if more than one component
Set Dependency Index	Section 9.8.4.2	REQUIRED if dependencies used
Abort	Section 9.8.4.3	OPTIONAL
Try Each	Section 9.8.4.4	OPTIONAL
Process Dependency	Section 9.8.4.5	OPTIONAL
Set Parameters	Section 9.8.4.6	OPTIONAL
Override Parameters	Section 9.8.4.7	REQUIRED
Fetch	Section 9.8.4.8	REQUIRED for Updater
Copy	Section 9.8.4.9	OPTIONAL
Run	Section 9.8.4.10	REQUIRED for Bootloader
Wait For Event	Section 9.8.4.11	OPTIONAL
Run Sequence	Section 9.8.4.12	OPTIONAL
Swap	Section 9.8.4.13	OPTIONAL

The subsequent table shows the parameters.

Name	Reference	Implementation
Vendor ID	Section 9.8.2.1	REQUIRED
Class ID	Section 9.8.2.2	REQUIRED
Image Digest	Section 9.8.2.3	REQUIRED
Image Size	Section 9.8.2.4	REQUIRED
Use Before	Section 9.8.2.5	OPTIONAL
Component Offset	Section 9.8.2.6	OPTIONAL
Encryption Info	Section 9.8.2.7	OPTIONAL
Compression Info	Section 9.8.2.8	OPTIONAL
Unpack Info	Section 9.8.2.9	OPTIONAL
URI	Section 9.8.2.10	OPTIONAL
Source Component	Section 9.8.2.11	OPTIONAL
Run Args	Section 9.8.2.12	OPTIONAL
Device ID	Section 9.8.2.13	OPTIONAL
Minimum Battery	Section 9.8.2.14	OPTIONAL
Update Priority	Section 9.8.2.15	OPTIONAL
Version	Section 9.8.2.16	OPTIONAL
Wait Info	Section 9.8.2.17	OPTIONAL
URI List	Section 9.8.2.18	OPTIONAL
Strict Order	Section 9.8.2.19	OPTIONAL
Soft Failure	Section 9.8.2.20	OPTIONAL
Custom	Section 9.8.2.21	OPTIONAL

Authors' Addresses

Brendan Moran
Arm Limited

EMail: Brendan.Moran@arm.com

Hannes Tschofenig
Arm Limited

EMail: hannes.tschofenig@arm.com

Henk Birkholz
Fraunhofer SIT

EMail: henk.birkholz@sit.fraunhofer.de

Koen Zandberg
Inria

EMail: koen.zandberg@inria.fr