SUIT
Internet-Draft
Intended status: Standards Track
Expires: May 7, 2020

B. Moran
H. Tschofenig
Arm Limited
H. Birkholz
Fraunhofer SIT

November 04, 2019

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

draft-ietf-suit-manifest-02

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.

Status of This Memo

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, such as the Lightweight Machine-to-Machine (LwM2M) protocol offering device management of IoT devices. Equally important is the inclusion of meta-data about the conveyed firmware image (in the form of a manifest) and the use of 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. This authorization process is ensured by the use of dedicated symmetric or asymmetric keys installed on the IoT device: for use cases where only integrity protection is required it is sufficient to install a trust anchor on the IoT device. For confidentiality protected firmware images it is additionally required to install either one or multiple symmetric or asymmetric keys on the IoT device. Starting security protection at the author is a risk mitigation technique so firmware images and manifests can be stored on untrusted respositories; it also reduces the scope of a compromise of any repository or intermediate system to be no worse than a denial of service.

It is assumed that the reader is familiar with the high-level firmware update architecture [I-D.ietf-suit-architecture].

The SUIT manifest is heavily optimised for consumption by constrained devices. This means that it is not constructed as a conventional descriptive document. Instead, of describing what an update IS, it describes what a recipient should DO.

While the SUIT manifest is informed by and optimised for firmware update use cases, there is nothing in the [I-D.ietf-suit-information-model] that restricts its use to only

firmware use cases. Software update and delivery of arbitrary data can equally be managed by SUIT-based metadata.

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.

- SUIT: Sofware 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 piece of information that describes one or more payloads, one or more resources, and the processors needed to transform resources into payloads.

- 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 recipient devices.

- Recipient: The system, typically an IoT device, that receives a manifest.

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

- Directive: An action for the Recipient to perform.

- Command: A Condition or a Directive.

- 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).

The map indices in this 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.

CDDL names are hyphenated and CDDL structures follow the convention adopted in COSE [RFC8152]: SUIT_Structure_Name.

3. How to use this document

For information about firmware update in general and the background of the suit manifest, see Section 4. To implement an updatable device, see Section 5 and Section 7. To implement a tool that generates updates, see Section 6 and Section 7.

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].

4.1. Landscape

The various constraints on IoT devices creates a broad set of use-case requirements. 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 processors may require coordinated update of all applications.

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

- mesh networks may require multicast distribution.

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 5 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 3 steps:

1. Verify image(s)

2. Load image(s)

3. Run image(s)

If verification and running is implemented in bootloader, then the

When multiple manifests are used for an update, each manifest's steps occur in a lockstep fashion; all manifests have dependency resolution performed before any manifest performs a payload fetch, etc.

4.3. SUIT Manifest goals

The manifest described in this document is intended to meet several goals, as described below.

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

2. Simple to parse on a constrained node

3. Simple to process on a constrained node

4. Compact encoding

5. Comprehensible by an intermediate system

6. Expressive enough to enable advanced use cases on advanced nodes

7. Extensible

The SUIT manifest can be used for a variety of purposes throughout its lifecycle. The manifest allows:

1. the Firmware Author to reason about releasing a firmware.

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

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

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

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

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

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

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

9. 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.

4.4. SUIT manifest design summary

In order to provide flexible behaviour to constrained devices, while still allowing more powerful devices to use their full capabilities, the SUIT manifest encodes the required behaviour of a Recipient device. Behaviour is encoded as a specialised 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 specialised 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 behaviours in a linearised form, 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 behaviour 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 only manifests authenticated by the appropriate identity have access to operate on a component.

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.

5. Interpreter Behaviour

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

5.1. 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 manifest, 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 (update interpreter)

- Dependency not available (update interpreter)

- Application crashed when executed (bootloader interpreter)

- Watchdog timeout occurred (bootloader interpreter)

- Dependency or Payload verification failed (bootloader interpreter)

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.

5.2. Required Checks

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 (Section 5.4.1). 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 5.5 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.

5.3. 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.

5.4. Abstract Machine Description

The byte code that forms the bulk of the manifest is processed by an interpreter. This interpreter can be modelled as a simple abstract machine. This machine consists of several data storage locations that are modified by commands. Certain commands also affect the machine's behaviour.

Every command that modifies system state targets a specific component. Components are units of code or data that can be targeted by an update. They are identified by Component identifiers, arrays of binary-strings-effectively a binary path. Each component has a corresponding set of configuration, Parameters. Parameters are used as the inputs to commands.

5.4.1. Parameters

Some parameters are REQUIRED to implement. These parameters allow a device to perform core functions.

- Vendor ID

- Class ID

- Image Digest

Some parameters are RECOMMENDED to implement. These parameters are needed for most use-cases.

- Image Size

- URI

Other parameters are OPTIONAL to implement. These parameters allow a device to implement specific use-cases.

- Strict Order

- Soft Failure

- Device ID

- Encryption Info

- Unpack Info

- Source Component

- URI List

- Custom Parameters

5.4.2. Commands

Commands define the behaviour of a device. The commands are divided into two groups: those that modify state (directives) and those that perform tests (conditions). There are also several Control Flow operations.

Some commands are REQUIRED to implement. These commands allow a device to perform core functions

- Check Vendor Identifier (cvid)

- Check Class Identifier (ccid)

- Verify Image (cimg)

- Set Current Component (setc)

- Override Parameters (ovrp)

NOTE: on systems that support only a single component, Set Current Component has no effect.

Some commands are RECOMMENDED to implement. These commands are needed for most use-cases

- Set Current Dependency (setd)

- Set Parameters (setp)

- Process Dependency (pdep)

- Run (run)

- Fetch (getc)

Other commands are OPTIONAL to implement. These commands allow a device to implement specific use-cases.

- Use Before (ubf)

- Check Component Offset (cco)

- Check Device Identifier (cdid)

- Check Image Not Match (nimg)

- Check Minimum Battery (minb)

- Check Update Authorised (auth)

- Check Version (cver)

- Abort (abrt)

- Try Each (try)

- Copy (copy)

- Swap (swap)

- Wait For Event (wfe)

- Run Sequence (srun) mandatory component set

- Run with Arguments (arun)

5.4.3. Command Behaviour

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

Code	Operation
cvid	binary-match(component, params[vendor-id])
ccid	binary-match(component, params[class-id])
cimg	binary-match(digest(component), params[digest])
setc	component := components[arg]
ovrp	params[k] := v for k,v in arg
setd	dependency := dependencies[arg]
setp	params[k] := v if not k in params for k,v in arg
pdep	exec(dependency[common]); exec(dependency[current-segment])
run	run(component)
getc	store(component, fetch(params[uri]))
ubf	assert(now() < arg)
cco	assert(offsetof(component) == arg)
cdid	binary-match(component, params[device-id])
nimg	not binary-match(digest(component), params[digest])
minb	assert(battery >= arg)
auth	assert(isAuthorised())
cver	assert(version_check(component, arg))

abrt	assert(0)
try	break if exec(seq) is not error for seq in arg
copy	store(component, params[src-component])
swap	swap(component, params[src-component])
wfe	until event(arg), wait
srun	exec(arg)
arun	run(component, arg)

5.5. 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.

5.6. 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.

5.7. Processing Dependencies

As described in Section 5.2, 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 Depdendency 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 5.2 to ensure that the dependent is processing the dependency correctly.

6. 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 is covered in [RFC8152] and the calculation of cryptographic values is beyond the scope of this document.

Compiling desired system state into a sequence of operations can be accomplished in many ways, however several templates are provided here to cover common use-cases. Many of these templates can be aggregated to produce more complex behaviour.

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

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

6.1. Manifest Source Material

When a manifest is constructed from a descriptive document, the descriptive document SHOULD be included in the severable text section. This section MAY be pruned from the manifest prior to distribution to a device. The inclusion of text source material enables several use-cases on unconstrained intermediate systems, where small manifest size, low parser complexity, and pull parsing are not required.

An unconstrained system that makes decisions based on the manifest can use the source material instead so that it does not need to execute the manifest.

An unconstrained system that presents data to a user can do so according to typical usage patterns without first executing the manifest, and can trust that information with the same level of confidence as the manifest itself.

A verifier can be constructed to emulate execution the manifest and compare the results of that execution to the source material, providing a check that the manifest performs its stated objectives and that the manifest does not exceed the capabilities of the target device.

6.2. Required Template: Compatibility Check

The compatibility check ensures that devices only install compatible images.

Common: Set Current Component Check Vendor Identifier Check Class Identifier

All manifests MUST contain the compatibility check template, except as outlined below.

If a device class has a unique trust anchor, and every element in its trust chain is unique-different from every element in any other device class, then it MAY include the compatibility check.

If a manifest includes a dependency that performs a compatibility check, then the dependent manifest MAY include the compatibility check.

The compatibility check template contains a data dependency: Vendor Identifier and Class Identifier MUST be set prior to executing the template. One examples of the full template is included below,

however Parameters may be set within a Try-Each block as well. They may also be inherited from a dependent manifest.

- Common:

  o Set Current Component

  o Set Parameters:

    * Vendor ID

    * Class ID

  o Check Vendor Identifier

  o Check Class Identifier

6.3. Use Case Template: XIP Secure Boot

- Common:

  o Set Current Component

  o Override Parameters:

    * Digest

    * Size

- Run:

  o Set Current Component

  o Check Image Match

  o Directive Run

6.4. Use Case Template: Firmware Download

- Common:

  o Set Current Component

  o Override Parameters:

    * Digest

    * Size

- Install:

  o Set Current Component

  o Set Parameters:

    * URI

  o Fetch

6.5. Use Case Template: Load from External Storage

- Load:

  o Set Current Component

  o Set Parameters:

    * Source Index

  o Copy

6.6. Use Case Template Load & Decompress from External Storage

- Load:

  o Set Current Component

  o Set Parameters:

    * Source Index

    * Compression Info

  o Copy

6.7. Use Case Template: Dependency

- Dependency Resolution:

  o Set Current Dependency

  o Set Parameters:

    * URI

  o Fetch

  o Check Image Match

  o Process Dependency

- Validate:

  o Set Current Dependency

  o Check Image Match

  o Process Dependency

For any other section that the dependency has, the dependent MUST invoke Process Dependency.

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

7. Manifest Structure

The manifest is divided into several sections in a hierarchy as follows:

1. The outer wrapper

    1. The authentication wrapper

    2. The manifest

        1. Critical Information

        2. Information shared by all command sequences

            1. List of dependencies

            2. List of payloads

            3. List of payloads in dependencies

            4. Common list of conditions, directives

        3. Dependency resolution Reference or list of conditions, directives

        4. Payload fetch Reference or list of conditions, directives

        5. Installation Reference or list of conditions, directives

6. Verification conditions/directives

7. Load conditions/directives

8. Run conditions/directives

9. Text / Reference

10. COSWID / Reference

3. Dependency resolution conditions/directives

4. Payload fetch conditions/directives

5. Installation conditions/directives

6. Text

7. COSWID / Reference

8. Intermediate Certificate(s) / CWTs

9. Inline Payload(s)

7.1. Severable Elements

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

7.2. Outer wrapper

This object is a container for the other pieces of the manifest to provide a common mechanism to find each of the parts. All elements of the outer wrapper are contained in bstr objects. Wherever the manifest references an object in the outer wrapper, the bstr is included in the digest calculation.

The CDDL that describes the wrapper is below

```
SUIT_Outer_Wrapper = {
    suit-authentication-wrapper   => bstr .cbor
                                     SUIT_Authentication_Wrapper / nil,
    $SUIT_Manifest_Wrapped,
    ? suit-dependency-resolution  => bstr .cbor SUIT_Command_Sequence,
    ? suit-payload-fetch          => bstr .cbor SUIT_Command_Sequence,
    ? suit-install                => bstr .cbor SUIT_Command_Sequence,
    ? suit-text                   => bstr .cbor SUIT_Text_Map,
    ? suit-coswid                 => bstr .cbor COSWID
}

SUIT_Authentication_Wrapper = [ + (COSE_Mac_Tagged / COSE_Sign_Tagged /
                                  COSE_Mac0_Tagged / COSE_Sign1_Tagged)]
SUIT_Encryption_Wrapper = COSE_Encrypt_Tagged / COSE_Encrypt0_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
)
```

All elements of the outer wrapper 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 serialisation for integrity and authenticity checks.

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. The Manifest is authenticated by these blocks in "detached payload" mode. The COSE_Mac_Tagged and COSE_Sign_Tagged blocks are described in RFC 8152 [RFC8152] and are beyond the scope of this document. The suit-authentication-wrapper MUST come first in the SUIT_Outer_Wrapper, regardless of canonical

encoding of CBOR. All validators MUST reject any SUIT_Outer_Wrapper that begins with any element other than a suit-authentication-wrapper.

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

The outer wrapper MUST contain only one of

- a plaintext manifest: SUIT_Manifest

- an encrypted manifest: both a SUIT_Encryption_Wrapper and the ciphertext of a manifest.

When the outer wrapper contains SUIT_Encryption_Wrapper, the suit-authentication-wrapper MUST authenticate the plaintext of suit-manifest-encrypted.

suit-manifest contains a SUIT_Manifest structure, which describes the payload(s) to be installed and any dependencies on other manifests.

suit-manifest-encryption-info contains a SUIT_Encryption_Wrapper, a COSE object that describes the information required to decrypt a ciphertext manifest.

suit-manifest-encrypted contains a ciphertext manifest.

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 7.3.

suit-text contains all the human-readable information that describes any and all parts of the manifest, its payload(s) and its resource(s).

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

7.3. Manifest

The manifest describes the critical metadata for the referenced payload(s). In addition, it contains:

1. a version number for the manifest structure itself

2. a sequence number

3. a list of dependencies

4. a list of components affected

5. a list of components affected by dependencies

6. a reference for each of the severable blocks.

7. a list of actions that the recipient should perform.

The following CDDL fragment defines the manifest.

```
SUIT_Manifest = {
    suit-manifest-version         => 1,
    suit-manifest-sequence-number => uint,
    suit-common                   => bstr .cbor SUIT_Common,
    ? suit-dependency-resolution  => Digest / bstr .cbor SUIT_Command_Sequence,
    ? suit-payload-fetch          => Digest / bstr .cbor SUIT_Command_Sequence,
    ? suit-install                => Digest / bstr .cbor SUIT_Command_Sequence,
    ? suit-validate               => bstr .cbor SUIT_Command_Sequence,
    ? suit-load                   => bstr .cbor SUIT_Command_Sequence,
    ? suit-run                    => bstr .cbor SUIT_Command_Sequence,
    ? suit-text                   => Digest,
    ? suit-coswid                 => Digest / bstr .cbor concise-software-
identity,
}

SUIT_Common = {
    ? suit-dependencies           => bstr .cbor [ + SUIT_Dependency ],
    ? suit-components             => bstr .cbor [ +
SUIT_Component_Identifier ],
    ? suit-dependency-components  => bstr .cbor [ + SUIT_Component_Reference ],
    ? suit-common-sequence        => bstr .cbor SUIT_Command_Sequence,
}
```

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, text SHOULD be severable, since most useful text elements occupy more space than a SUIT_Digest, but are not needed by recipient devices. 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 is been severable. The key used for a severable element is the same in the SUIT_Manifest and in the SUIT_Outer_Wrapper so that a recipient can easily identify the correct data in the outer wrapper.

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

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.

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, 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.

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 10 for more information on conditional sequences). suit-common-sequence is RECOMMENDED.

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 when suit-dependencies is present.

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.

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.

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. If the manifest contains dependencies, one process-dependency invocation per dependency or one process-dependency invocation targeting all dependencies SHOULD be present in validate.

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.

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. Only one manifest in an update may contain the "run" directive.

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

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

7.4. SUIT_Dependency

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

The following CDDL describes the SUIT_Dependency structure.

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

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.

7.5. SUIT_Component_Reference

The SUIT_Component_Reference describes an image that is defined by another manifest. This is useful for overriding the behaviour 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. The following CDDL describes the SUIT_Component_Reference.

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

7.6. Manifest 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.

```
+-----+--------+------------------+-----------+------------------+
```

ID	CBOR Type	Scope	Name	Description
1	boolean	Global	Strict Order	Requires that the manifest is processed in a strictly linear fashion. Set to 0 to enable parallel handling of manifest directives.
2	boolean	Command Segment	Soft Failure	Condition failures only terminate the current command segment.
3	bstr	Component/Global	Vendor ID	A RFC4122 UUID representing the vendor of the device or component
4	bstr	Component/Global	Class ID	A RFC4122 UUID representing the class of the device or component
5	bstr	Component/Global	Device ID	A RFC4122 UUID representing the device or component
6	tstr	Component/Dependency	URI	A URI from which to fetch a resource
7	bstr	Component/Dependency	Encryption Info	A COSE object defining the encryption mode of a resource
8	bstr	Component	Compression Info	The information required to decompress the

				image
9	bstr	Component	Unpack Info	The information required to unpack the image
10	uint	Component	Source Component	A Component Index
11	bstr	Component/Dependency	Image Digest	A SUIT_Digest
12	uint	Component/Dependency	Image Size	Integer size
24	bstr	Component/Dependency	URI List	A CBOR encoded list of ranked URIs
25	boolean	Component/Dependency	URI List Append	A CBOR encoded list of ranked URIs
nint	int/bstr	Custom	Custom Parameter	Application-defined parameter

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 modularisation and division of responsibility within a pull parser. The same consideration does not apply to Conditions and Directives because those elements are invoked with their arguments immediately

7.6.1. 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 parallelise 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.

7.6.2. 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.

7.7. 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

7.8. 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.

SUIT_Parameter_Compression_Info is defined by the following CDDL:

```
SUIT_Compression_Info = {
    suit-compression-algorithm => SUIT_Compression_Algorithms
    ? suit-compression-parameters => bstr
}


SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_gzip
SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_bzip2
SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_deflate
SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_LZ4
SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_lzma
```

7.9. SUIT_Parameter_Unpack_Info

SUIT_Unpack_Info defines the information required for a device to interpret a packed format, such as elf, hex, or binary diff. SUIT_Unpack_Info is defined by the following CDDL:

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

SUIT_Unpack_Algorithms //= SUIT_Unpack_Algorithm_Delta
SUIT_Unpack_Algorithms //= SUIT_Unpack_Algorithm_Hex
SUIT_Unpack_Algorithms //= SUIT_Unpack_Algorithm_Elf
```

7.10. SUIT_Parameters CDDL

The following CDDL describes all SUIT_Parameters.

```
SUIT_Parameters //= (suit-parameter-strict-order => bool)
SUIT_Parameters //= (suit-parameter-soft-failure => bool)
SUIT_Parameters //= (suit-parameter-vendor-id => bstr)
SUIT_Parameters //= (suit-parameter-class-id => bstr)
SUIT_Parameters //= (suit-parameter-device-id => bstr)
SUIT_Parameters //= (suit-parameter-uri => tstr)
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-source-component => uint)
SUIT_Parameters //= (suit-parameter-image-digest => bstr .cbor SUIT_Digest)
SUIT_Parameters //= (suit-parameter-image-size => uint)
SUIT_Parameters //= (suit-parameter-uri-list => bstr .cbor
SUIT_Component_URI_List)
SUIT_Parameters //= (suit-parameter-custom => int/bool/tstr/bstr)

SUIT_Component_URI_List = [ + [priority: int, uri: tstr] ]

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_gzip
SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_bzip2
SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_deflate
SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_LZ4
SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_lzma

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

SUIT_Unpack_Algorithms //= SUIT_Unpack_Algorithm_Delta
SUIT_Unpack_Algorithms //= SUIT_Unpack_Algorithm_Hex
SUIT_Unpack_Algorithms //= SUIT_Unpack_Algorithm_Elf
```

7.11. 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

2. Payload Fetch

3. Payload Installation

4. Image Validation

5. Image Loading

6. Run or Boot

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.

The lists of commands are logically structured into sequences of zero or more conditions followed by zero or more directives. The *logical* structure is described by the following CDDL:

```
Command_Sequence = {
    conditions => [ * Condition],
    directives => [ * Directive]
}
```

This introduces significant complexity in the parser, however, so the structure is flattened to make parsing simpler:

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

Each condition and directive is composed of:

1. A command code identifier

2. An argument block

Argument blocks are defined for each type of command.

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 behaviour.

7.12. 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-conditons of any directive or series of directives, depending on where they are placed in the list. Conditions include:

Condition Code	Condition Name	Argument Type
1	Vendor Identifier	nil
2	Class Identifier	nil
3	Image Match	nil
4	Use Before	Unsigned Integer timestamp
5	Component Offset	Unsigned Integer
24	Device Identifier	nil
25	Image Not Match	nil
26	Minimum Battery	Unsigned Integer
27	Update Authorised	Integer
28	Version	List of Integers
nint	Custom Condition	bstr

Each condition MUST report a success code on completion. If a condition reports failure, then the current sequence of commands MUST terminate. If a recipient encounters an unknown Condition Code, it MUST report a failure.

Positive Condition numbers are reserved for IANA registration. Negative numbers are reserved for proprietary, application-specific directives.

7.12.1. Identifier Conditions

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.

7.12.2. 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.

7.12.3. 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 Components list. If no digest is specified and the component is not present in the Components list, the condition fails. suit-condition-image-not-match is OPTIONAL to implement.

7.12.4. 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. One argument is required, 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.

7.12.5. 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.

7.12.6. suit-condition-update-authorised

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

7.12.7. 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 occured. 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.

The following CDDL describes SUIT_Condition_Version_Argument

```
SUIT_Condition_Version_Argument = [
    suit-condition-version-comparison: SUIT_Condition_Version_Comparison_Types,
    suit-condition-version-comparison: 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]
```

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.

7.12.8. SUIT_Condition_Custom

SUIT_Condition_Custom describes any proprietary, application specific condition. This is encoded as a negative integer, chosen by the firmware developer, and a bstr that encodes the parameters passed to the system that evaluates the condition matching that integer. SUIT_Condition_Custom is OPTIONAL to implement.

7.12.9. Identifiers

Many 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 explicitly NOT human-readable. They are for machine-based matching 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: 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.

7.12.9.1. Creating UUIDs:

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)

7.12.10. SUIT_Condition CDDL

The following CDDL describes SUIT_Condition:

```
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,        uint)
SUIT_Condition //= (suit-condition-minimum-battery,   uint)
SUIT_Condition //= (suit-condition-update-authorised, int)
SUIT_Condition //= (suit-condition-version,
SUIT_Condition_Version_Argument)
SUIT_Condition //= (suit-condition-component-offset,  uint)
SUIT_Condition //= (suit-condition-custom,            bstr)

SUIT_Condition_Version_Argument = [
    suit-condition-version-comparison: SUIT_Condition_Version_Comparison_Types,
    suit-condition-version-comparison: 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_Value = [+int]
```

7.13. SUIT_Directive

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

Directive Code	Directive 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
30	Run Sequence
31	Run with Arguments
32	Swap

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

7.13.1. 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.

The following CDDL describes the argument to suit-directive-set-component-index.

```
SUIT_Directive_Set_Component_Index_Argument = uint/bool
```

7.13.2. 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.

The following CDDL describes the argument to suit-directive-set-dependency-index.

```
SUIT_Directive_Set_Manifest_Index_Argument = uint/bool
```

7.13.3. suit-directive-abort

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

7.13.4. 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.

The following CDDL describes the SUIT_Run_Sequence argument.

```
SUIT_Directive_Run_Sequence_Argument = bstr .cbor SUIT_Command_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.

7.13.5. 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 initialised 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.

The following CDDL describes the SUIT_Try_Each argument.

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

7.13.6. 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.

The argument to suit-directive-process-dependency is defined in the following CDDL.

```
SUIT_Directive_Process_Dependency_Argument = nil
```

7.13.7. suit-directive-set-parameters

suit-directive-set-parameters allows the manifest to configure behaviour 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 behaviour of its dependencies.

Available parameters are defined in Section 7.6.

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 behaviour of a manifest.

The argument to suit-directive-set-parameters is defined in the following CDDL.

```
SUIT_Directive_Set_Parameters_Argument = {+ SUIT_Parameters}
```

N.B.: A directive code is reserved for an optimisation: a way to set a parameter to the contents of another parameter, optionally with another component ID.

7.13.8. 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 7.6.

The argument to suit-directive-override-parameters is defined in the following CDDL.

```
SUIT_Directive_Override_Parameters_Argument = {+ SUIT_Parameters}
```

7.13.9. 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 behaviour. 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 behaviour 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.

The argument to suit-directive-fetch is defined in the following CDDL.

```
SUIT_Directive_Fetch_Argument = nil/bstr
```

7.13.10. 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 behaviour 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.

The argument to suit-directive-copy is defined in the following CDDL.

```
SUIT_Directive_Copy_Argument = nil
```

7.13.11. 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.

7.13.12. 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.

The argument to suit-directive-run is defined in the following CDDL.

```
SUIT_Directive_Run_Argument = nil/bstr
```

7.13.13. suit-directive-wait

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

1. Authorisation

2. External Power

3. Network availability

4. Other Device Firmware Version

5. Time

6. Time of Day

7. Day of Week

The following CDDL defines the encoding of these events.

```
SUIT_Wait_Events //= (suit-wait-event-authorisation => 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_Authorisation = int ; priority
SUIT_Wait_Event_Argument_Power = int ; Power Level
SUIT_Wait_Event_Argument_Network = int ; Network State
SUIT_Wait_Event_Argument_Other_Device_Version = [
    other-device: bstr,
    other-device-version: [+int]
]
SUIT_Wait_Event_Argument_Time = uint ; Timestamp
SUIT_Wait_Event_Argument_Time_Of_Day = uint ; Time of Day (seconds since
00:00:00)
SUIT_Wait_Event_Argument_Day_Of_Week = uint ; Days since Sunday
```

7.13.14. SUIT_Directive CDDL

The following CDDL describes SUIT_Directive:

```
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-run,                  nil)
SUIT_Directive //= (suit-directive-wait,
                    { + SUIT_Wait_Events })
SUIT_Directive //= (suit-directive-run-with-arguments,   bstr)

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

SUIT_Wait_Events //= (suit-wait-event-authorisation => 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_Authorisation = int ; priority
SUIT_Wait_Event_Argument_Power = int ; Power Level
SUIT_Wait_Event_Argument_Network = int ; Network State
SUIT_Wait_Event_Argument_Other_Device_Version = [
    other-device: bstr,
    other-device-version: [+int]
]
SUIT_Wait_Event_Argument_Time = uint ; Timestamp
SUIT_Wait_Event_Argument_Time_Of_Day = uint ; Time of Day (seconds since
00:00:00)
SUIT_Wait_Event_Argument_Day_Of_Week = uint ; Days since Sunday
```

7.14. SUIT_Text_Map

The SUIT_Text_Map contains all text descriptions needed for this manifest. 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.

ID	Name	Summary
1	manifest-description	Free text description of the manifest
2	update-description	Free text description of the update
3	vendor-name	Free text vendor name
4	model-name	Free text model name
5	vendor-domain	The domain used to create the vendor-id (Section 7.12.9.1)
6	model-info	The information used to create the class-id (Section 7.12.9.1)
7	component-description	Free text description of each component in the manifest
8	json-source	The JSON-formated document that was used to create the manifest
9	yaml-source	The yaml-formated document that was used to create the manifest
10	version-dependencies	List of component versions required by the manifest

8. 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.

9. 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. This structure is described by the following CDDL.

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

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

digest-algorithm-ids /= algorithm-id-sha224
digest-algorithm-ids /= algorithm-id-sha256
digest-algorithm-ids /= algorithm-id-sha384
digest-algorithm-ids /= algorithm-id-sha512
digest-algorithm-ids /= algorithm-id-sha3-224
digest-algorithm-ids /= algorithm-id-sha3-256
digest-algorithm-ids /= algorithm-id-sha3-384
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
```

10. 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
      ]
  }
```

11. 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_Outer_Wrapper = {
    suit-authentication-wrapper => bstr .cbor SUIT_Authentication_Wrapper /
nil,
    $$SUIT_Manifest_Wrapped,
    suit-dependency-resolution  => bstr .cbor SUIT_Command_Sequence,
    suit-payload-fetch          => bstr .cbor SUIT_Command_Sequence,
    suit-install                => bstr .cbor SUIT_Command_Sequence,
    suit-text                   => bstr .cbor SUIT_Text_Map,
    suit-coswid                 => bstr .cbor concise-software-identity
}

SUIT_Authentication_Wrapper = [ + (
    COSE_Mac_Tagged /
    COSE_Sign_Tagged /
    COSE_Mac0_Tagged /
    COSE_Sign1_Tagged)
]

SUIT_Encryption_Wrapper = COSE_Encrypt_Tagged / COSE_Encrypt0_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
)

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-dependency-resolution  => SUIT_Digest / bstr .cbor
SUIT_Command_Sequence,
    ? suit-payload-fetch          => SUIT_Digest / bstr .cbor
SUIT_Command_Sequence,
    ? suit-install                => SUIT_Digest / bstr .cbor
SUIT_Command_Sequence,
    ? suit-validate               => bstr .cbor SUIT_Command_Sequence,
    ? suit-load                   => bstr .cbor SUIT_Command_Sequence,
    ? suit-run                    => bstr .cbor SUIT_Command_Sequence,
    ? suit-text                   => SUIT_Digest,
    ? suit-coswid                 => SUIT_Digest / bstr .cbor concise-software-
identity,
}

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 = (nint, bstr)
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,        uint)
SUIT_Condition //= (suit-condition-minimum-battery,   uint)
SUIT_Condition //= (suit-condition-update-authorised, int)
SUIT_Condition //= (suit-condition-version,
SUIT_Condition_Version_Argument)
SUIT_Condition //= (suit-condition-component-offset,  uint)
SUIT_Condition //= (suit-condition-custom,            bstr)

RFC4122_UUID = bstr .size 16

SUIT_Condition_Version_Argument = [
    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_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,                     { +
SUIT_Wait_Events })
SUIT_Directive //= (suit-directive-run-with-arguments,       bstr)

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

SUIT_Wait_Events //= (suit-wait-event-authorisation => 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_Authorisation = int ; priority
SUIT_Wait_Event_Argument_Power = int ; Power Level
SUIT_Wait_Event_Argument_Network = int ; Network State
SUIT_Wait_Event_Argument_Other_Device_Version = [
    other-device: bstr,
    other-device-version: [+int]
]
SUIT_Wait_Event_Argument_Time = uint ; Timestamp
SUIT_Wait_Event_Argument_Time_Of_Day = uint ; Time of Day (seconds since
00:00:00)
SUIT_Wait_Event_Argument_Day_Of_Week = uint ; Days since Sunday

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

```
SUIT_Parameters //= (suit-parameter-soft-failure => bool)
SUIT_Parameters //= (suit-parameter-vendor-id => bstr)
SUIT_Parameters //= (suit-parameter-class-id => bstr)
SUIT_Parameters //= (suit-parameter-device-id => bstr)
SUIT_Parameters //= (suit-parameter-uri => tstr)
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-source-component => uint)
SUIT_Parameters //= (suit-parameter-image-digest => bstr .cbor SUIT_Digest)
SUIT_Parameters //= (suit-parameter-image-size => uint)
SUIT_Parameters //= (suit-parameter-uri-list => bstr .cbor
SUIT_Component_URI_List)
SUIT_Parameters //= (suit-parameter-custom => int/bool/tstr/bstr)

SUIT_Component_URI_List = [ + [priority: int, uri: tstr] ]
SUIT_Priority_Parameter_List = [ + [priority: int, parameters: { +
SUIT_Parameters }] ]

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_gzip
SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_bzip2
SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_lz4
SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_lzma

SUIT_Compression_Algorithm_gzip = 1
SUIT_Compression_Algorithm_bzip2 = 2
SUIT_Compression_Algorithm_deflate = 3
SUIT_Compression_Algorithm_lz4 = 4
SUIT_Compression_Algorithm_lzma = 7

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

SUIT_Unpack_Algorithms /= SUIT_Unpack_Algorithm_Delta
SUIT_Unpack_Algorithms /= SUIT_Unpack_Algorithm_Hex
SUIT_Unpack_Algorithms /= SUIT_Unpack_Algorithm_Elf

SUIT_Unpack_Algorithm_Delta = 1
SUIT_Unpack_Algorithm_Hex = 2
SUIT_Unpack_Algorithm_Elf = 3

SUIT_Text_Map = {int => tstr}

suit-authentication-wrapper = 1
suit-manifest = 2
```

```
suit-manifest-encryption-info = 3
suit-manifest-encrypted       = 4
```

```
suit-manifest-version = 1
suit-manifest-sequence-number = 2
suit-common = 3
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-custom = 6

suit-condition-device-identifier        = 24
suit-condition-image-not-match          = 25
suit-condition-minimum-battery          = 26
suit-condition-update-authorised        = 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-run-with-arguments       = 31
suit-directive-swap                     = 32

suit-wait-event-argument-authorisation = 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-wait-event-authorisation = 8

suit-parameter-strict-order = 1
suit-parameter-soft-failure = 2
suit-parameter-vendor-id = 3
suit-parameter-class-id = 4
suit-parameter-device-id = 5
suit-parameter-uri = 6
suit-parameter-encryption-info = 7
suit-parameter-compression-info = 8
suit-parameter-unpack-info = 9
suit-parameter-source-component = 10
suit-parameter-image-digest = 11
suit-parameter-image-size = 12

suit-parameter-uri-list = 24
suit-parameter-uri-list-append = 25
suit-parameter-prioritised-parameters = 26

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
```

12. 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-----
```


12.1. Example 0:

Secure boot only.

The following JSON shows the intended behaviour of the manifest.

```
{
    "structure-version": 1,
    "sequence-number": 1,
    "run-image": [
        { "directive-set-component": 0 },
        { "condition-image": null },
        { "directive-run": null }
    ],
    "common": {
        "common-sequence": [
            {
                "directive-set-var": {
                    "digest": "00112233445566778899aabbccddeeff"
                              "0123456789abcdeffedcba9876543210",
                    "size": 34768
                }
            }
        ],
        "components": [
            [
                "Flash",
                78848
            ]
        ]
    }
}
```


Converted into the SUIT manifest, this produces:

```
{
    / auth object / 1 : h'd28443a10126a1044874657374206b6579f65840ebec'
                        h'b66cbecb19dcedacf8459c1a22a1453781ba98d8ffb9'
                        h'd4e2912f29d23bac5ae3d51f1ff0c1b1df05e207ca17'
                        h'483a57ede914cf826b73599137881c8364c8',
    / manifest / 2 : h'a401010201035840a2024c818245466c6173684300340104'
                     h'582e8213a20b58248202582000112233445566778899aabb'
                     h'ccddeeff0123456789abcdeffedcba98765432100c1987d0'
                     h'0c47860c0003f617f6' \
    {
        / structure-version / 1 : 1,
        / sequence-number / 2 : 1,
        / common / 3 : h'a2024c818245466c6173684300340104582e8213a20b58'
                       h'248202582000112233445566778899aabbccddeeff0123'
                       h'456789abcdeffedcba98765432100c1987d0' \ {
            / components / 2 : h'818245466c61736843003401' \
            [
                [h'466c617368', h'003401'],
            ],
            / common-sequence / 4 : h'8213a20b58248202582000112233445'
                                    h'66778899aabbccddeeff0123456789ab'
                                    h'cdeffedcba98765432100c1987d0' \ [
                / set-vars / 19, {
                    / digest / 11 : h'8202582000112233445566778899aabb'
                                    h'ccddeeff0123456789abcdeffedcba98'
                                    h'76543210' \
                        [ 2, h'00112233445566778899aabbccddeeff01234567'
                             h'89abcdeffedcba9876543210' ],
                    / size / 12 : 34768,
                },
            ],
        },
        / run-image / 12 : h'860c0003f617f6' \ [
            / set-component-index / 12, 0,
            / condition-image / 3, None,
            / run / 23, None,
        ],
    }
}
```

Total size of outer wrapper without COSE authentication object: 87

Outer:

a201f6025851a401010201035840a2024c818245466c6173684300340104582e8213a20b
58248202582000112233445566778899aabbccddeeff0123456789abcdeffedcba987654
32100c1987d00c47860c0003f617f6

Total size of outer wrapper with COSE authentication object: 172

Signed Outer:


```
a2015854d28443a10126a104487465737420656b6579f65840ebecb66cbecb19dcedacf845
9c1a22a1453781ba98d8ffb9d4e2912f29d23bac5ae3d51f1ff0c1b1df05e207ca17483a
57ede914cf826b73599137881c8364c8025851a401010201035840a2024c818245466c61
736843003401045 82e8213a20b58248202582000112233445566778899aabbccddeeff01
23456789abcdeffedcba98765432100c1987d00c47860c0003f617f6
```


12.2. Example 1:

Simultaneous download and installation of payload.

The following JSON shows the intended behaviour of the manifest.


```
{
    "structure-version": 1,
    "sequence-number": 2,
    "apply-image": [
        { "directive-set-component": 0 },
        {
            "directive-set-var": {
                "uri": "http://example.com/file.bin"
            }
        },
        { "directive-fetch": null }
    ],
    "common": {
        "common-sequence": [
            {
                "directive-set-var": {
                    "digest": "00112233445566778899aabbccddeeff"
                              "0123456789abcdeffedcba9876543210",
                    "size": 34768
                }
            }
        ],
        "components": [
            [
                "Flash",
                78848
            ]
        ]
    }
}
```


Converted into the SUIT manifest, this produces:

```
{
    / auth object / 1 : h'd28443a10126a1044874657374206b6579f65840b531'
                        h'42132ebddbf0c523378d16fc904badc56553e41c6713'
                        h'b758dbd39f47effec5e7a583c418129f456d0aaaa3c4'
                        h'3fe06dd30d664b709edf0ad05b70dad38bc2',
    / manifest / 2 : h'a401010202035840a2024c818245466c6173684300340104'
                     h'582e8213a20b58248202582000112233445566778899aabb'
                     h'ccddeeff0123456789abcdeffedcba98765432100c1987d0'
                     h'095825860c0013a106781b687474703a2f2f6578616d706c'
                     h'652e636f6d2f66696c652e62696e15f6' \
    {
        / structure-version / 1 : 1,
        / sequence-number / 2 : 2,
        / common / 3 : h'a2024c818245466c6173684300340104582e8213a20b58'
                       h'248202582000112233445566778899aabbccddeeff0123'
                       h'456789abcdeffedcba98765432100c1987d0' \ {
            / components / 2 : h'818245466c6173684300340 1' \
            [
                [h'466c617368', h'003401'],
            ],
            / common-sequence / 4 : h'8213a20b582482025820001122334455'
                                    h'66778899aabbccddeeff0123456789ab'
                                    h'cdeffedcba98765432100c1987d0' \ [
                / set-vars / 19, {
                    / digest / 11 : h'8202582000112233445566778899aabb'
                                    h'ccddeeff0123456789abcdeffedcba98'
                                    h'76543210' \
                        [ 2, h'00112233445566778899aabbccddeeff01234567'
                             h'89abcdeffedcba9876543210' ],
                    / size / 12 : 34768,
                },
            ],
        },
        / apply-image / 9 : h'860c0013a106781b687474703a2f2f6578616d70'
                            h'6c652e636f6d2f66696c652e62696e15f6' \ [
            / set-component-index / 12, 0,
            / set-vars / 19, {
                / uri / 6 : http://example.com/file.bin,
            },
            / fetch / 21, None,
        ],
    }
}
```

Total size of outer wrapper without COSE authentication object: 118

Outer:

a201f6025870a401010202035840a2024c818245466c6173684300340104582e8213a20b
58248202582000112233445566778899aabbccddeeff0123456789abcdeffedcba987654
32100c1987d0095825860c0013a106781b687474703a2f2f6578616d706c652e636f6d2f
66696c652e62696e15f6

Total size of outer wrapper with COSE authentication object: 203

Signed Outer:

a2015854d28443a10126a1044874657374206b6579f65840b53142132ebddbf0c523378d
16fc904badc56553e41c6713b758dbd39f47effec5e7a583c418129f456d0aaaa3c43fe0
6dd30d664b709edf0ad05b70dad38bc2025870a401010202035840a2024c818245466c61
73684300340104582e8213a20b58248202582000112233445566778899aabbccddeeff01
23456789abcdeffedcba98765432100c1987d0095825860c0013a106781b687474703a2f
2f6578616d706c652e636f6d2f66696c652e62696e15f6

12.3. Example 2:

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

The following JSON shows the intended behaviour of the manifest.

```
{
    "structure-version": 1,
    "sequence-number": 3,
    "common": {
        "common-sequence": [
            {
                "directive-set-var": {
                    "vendor-id": "fa6b4a53-d5ad-5fdf-be9d-e663e4d41ffe",
                    "class-id": "1492af14-2569-5e48-bf42-9b2d51f2ab45",
                    "digest": "00112233445566778899aabbccddeeff"
                              "0123456789abcdeffedcba9876543210",
                    "size": 34768
                }
            },
            { "condition-vendor-id": null },
            { "condition-class-id": null }
        ],
        "components": [
            [
                "Flash",
                78848
            ]
        ]
    },
    "apply-image": [
        { "directive-set-component": 0 },
        {
            "directive-set-var": {
                "uri": "http://example.com/file.bin"
            }
        },
        { "directive-fetch": null }
    ],
    "run-image": [
        { "directive-set-component": 0 },
        { "condition-image": null },
        { "directive-run": null }
    ]
}
```

Converted into the SUIT manifest, this produces:

```
{
    / auth object / 1 : h'd28443a10126a1044874657374206b6579f658400014'
                        h'750c013f7e1cdbec6f14b99b49195e081d1030508a6b'
                        h'8d271bd99dfb382a7767dc45f20c9943ed22a1eaac9d'
                        h'07a041ec1acfc10ad7e45e6424629ff3e3e5',
    / manifest / 2 : h'a501010203035868a2024c818245466c6173684300340104'
```

```
                     h'58568613a40350fa6b4a53d5ad5fdfbe9de663e4d41ffe04'
                     h'501492af1425695e48bf429b2d51f2ab450b582482025820'
                     h'00112233445566778899aabbccddeeff0123456789abcdef'
                     h'fedcba98765432100c1987d001f602f6095825860c0013a1'
                     h'06781b687474703a2f2f6578616d706c652e636f6d2f6669'
                     h'6c652e62696e15f60c47860c0003f617f6' \
    {
        / structure-version / 1 : 1,
        / sequence-number / 2 : 3,
        / common / 3 : h'a2024c818245466c61736843003401045858613a40350'
                       h'fa6b4a53d5ad5fdfbe9de663e4d41ffe04501492af1425'
                       h'695e48bf429b2d51f2ab450b5824820258200011223344'
                       h'5566778899aabbccddeeff0123456789abcdeffedcba98'
                       h'765432100c1987d001f602f6' \ {
            / components / 2 : h'818245466c61736843003401' \
            [
                [h'466c617368', h'003401'],
            ],
            / common-sequence / 4 : h'8613a40350fa6b4a53d5ad5fdfbe9de6'
                                    h'63e4d41ffe04501492af1425695e48bf'
                                    h'429b2d51f2ab450b5824820258200011'
                                    h'2233445566778899aabbccddeeff0123'
                                    h'456789abcdeffedcba98765432100c19'
                                    h'87d001f602f6' \ [
                / set-vars / 19, {
                    / vendor-id / 3 : h'fa6b4a53d5ad5fdfbe9de663e4d41f'
                                      h'fe',
                    / class-id / 4 : h'1492af1425695e48bf429b2d51f2ab45',
                    / digest / 11 : h'8202582000112233445566778899aabb'
                                    h'ccddeeff0123456789abcdeffedcba98'
                                    h'76543210' \
                        [ 2, h'00112233445566778899aabbccddeeff01234567'
                             h'89abcdeffedcba9876543210' ],
                    / size / 12 : 34768,
                },
                / condition-vendor-id / 1, None,
                / condition-class-id / 2, None,
            ],
        },
        / apply-image / 9 : h'860c0013a106781b687474703a2f2f6578616d70'
                            h'6c652e636f6d2f66696c652e62696e15f6' \ [
            / set-component-index / 12, 0,
            / set-vars / 19, {
                / uri / 6 : http://example.com/file.bin,
            },
            / fetch / 21, None,
        ],
        / run-image / 12 : h'860c0003f617f6' \ [
```

```
            / set-component-index / 12, 0,
            / condition-image / 3, None,
            / run / 23, None,
        ],
    }
}
```

Total size of outer wrapper without COSE authentication object: 167

Outer:

a201f60258a1a501010203035868a2024c818245466c6173684300340104585686 13a403
50fa6b4a53d5ad5fdfbe9de663e4d41ffe04501492af1425695e48bf429b2d51f2ab450b
5824820258200011223344556677 8899aabbccddeeff0123456789abcdeffedcba987654
32100c1987d001f602f6095825860c0013a106781b687474703a2f2f6578616d706c652e
636f6d2f66696c652e62696e15f60c47860c0003f617f6

Total size of outer wrapper with COSE authentication object: 252

Signed Outer:

a2015854d28443a10126a1044874657374206b6579f658400014750c013f7e1cdbec6f14
b99b49195e081d1030508a6b8d271bd99dfb382a7767dc45f20c9943ed22a1eaac9d07a0
41ec1acfc10ad7e45e6424629ff3e3e50258a1a501010203035868a2024c818245466c61
736843003401045856861 3a40350fa6b4a53d5ad5fdfbe9de663e4d41ffe04501492af14
25695e48bf429b2d51f2ab450b58248202582000112233445566778899aabbccddeeff01
23456789abcdeffedcba98765432100c1987d001f602f6095825860c0013a106781b6874
74703a2f2f6578616d706c652e636f6d2f66696c652e62696e15f60c47860c0003f617f6

12.4. Example 3:

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

The following JSON shows the intended behaviour of the manifest.


```
{
    "structure-version": 1,
    "sequence-number": 4,
    "common": {
        "common-sequence": [
            {
                "directive-set-var": {
                    "vendor-id": "fa6b4a53-d5ad-5fdf-be9d-e663e4d41ffe",
                    "class-id": "1492af14-2569-5e48-bf42-9b2d51f2ab45"
                }
            },
            { "directive-set-component": 0 },
```

```
            {
                "directive-set-var": {
                    "digest": "00112233445566778899aabbccddeeff"
                              "0123456789abcdeffedcba9876543210",
                    "size": 34768
                }
            },
            { "directive-set-component": 1 },
            {
                "directive-set-var": {
                    "digest": "00112233445566778899aabbccddeeff"
                              "0123456789abcdeffedcba9876543210",
                    "size": 34768
                }
            },
            { "condition-vendor-id": null },
            { "condition-class-id": null }
        ],
        "components": [
            [
                "Flash",
                78848
            ],
            [
                "RAM",
                1024
            ]
        ]
    },
    "apply-image": [
        { "directive-set-component": 0 },
        {
            "directive-set-var": {
                "uri": "http://example.com/file.bin"
            }
        },
        { "directive-fetch": null }
    ],
    "run-image": [
        { "directive-set-component": 0 },
        { "condition-image": null },
        { "directive-set-component": 1 },
        {
            "directive-set-var": {
                "source-index": 0
            }
        },
        { "directive-fetch": null },
```

```
        { "condition-image": null },
        { "directive-run": null }
    ]
}
```

Converted into the SUIT manifest, this produces:


```
{
    / auth object / 1 : h'd28443a10126a1044874657374206b6579f6584070eb'
                        h'70f2552533fc954e934f50f42bdd9b6f7d4fd7e11463'
                        h'6b9cdbef2a065f9640243a7857f66c4389aea906c4f3'
                        h'b45150c8e55461e9bfda945904033fc70a84',
    / manifest / 2 : h'a5010102040358a3a20254828245466c6173684300340182'
                     h'4352414d4200040458898e13a20350fa6b4a53d5ad5fdfbe'
                     h'9de663e4d41ffe04501492af1425695e48bf429b2d51f2ab'
                     h'450c0013a20b58248202582000112233445566778899aabb'
                     h'ccddeeff0123456789abcdeffedcba98765432100c1987d0'
                     h'0c0113a20b582482025820001122334455667788 99aabbcc'
                     h'ddeeff0123456789abcdeffedcba98765432100c1987d001'
                     h'f602f6095825860c0013a106781b687474703a2f2f657861'
                     h'6d706c652e636f6d2f66696c652e62696e15f60c518e0c00'
                     h'03f60c0113a10a0015f603f617f6' \
    {
        / structure-version / 1 : 1,
        / sequence-number / 2 : 4,
        / common / 3 : h'a20254828245466c6173684300340182 4352414d420004'
                       h'0458898e13a20350fa6b4a53d5ad5fdfbe9de663e4d41f'
                       h'fe04501492af1425695e48bf429b2d51f2ab450c0013a2'
                       h'0b5824820258200011223344556677 8899aabbccddeeff'
                       h'0123456789abcdeffedcba98765432100c1987d00c0113'
                       h'a20b58248202582000112233445566778899aabbccddee'
                       h'ff0123456789abcdeffedcba98765432100c1987d001f6'
                       h'02f6' \ {
            / components / 2 : h'828245466c6173684300340182 4352414d4200'
                               h'04' \
            [
                [h'466c617368', h'003401'],
                [h'52414d', h'0004'],
            ],
            / common-sequence / 4 : h'8e13a20350fa6b4a53d5ad5fdfbe9de6'
                                    h'63e4d41ffe04501492af1425695e48bf'
                                    h'429b2d51f2ab450c0013a20b58248202'
                                    h'5820001122334455 66778899aabbccdd'
                                    h'eeff0123456789abcdeffedcba987654'
                                    h'32100c1987d00c0113a20b5824820258'
                                    h'2000112233445566778899aabbccddee'
                                    h'ff0123456789abcdeffedcba98765432'
                                    h'100c1987d001f602f6' \ [
```

```
                / set-vars / 19, {
                    / vendor-id / 3 : h'fa6b4a53d5ad5fdfbe9de663e4d41f'
                                      h'fe',
                    / class-id / 4 : h'1492af1425695e48bf429b2d51f2ab45',
                },
                / set-component-index / 12, 0,
                / set-vars / 19, {
                    / digest / 11 : h'8202582000112233445566778899aabb'
                                    h'ccddeeff0123456789abcdeffedcba98'
                                    h'76543210' \
                        [ 2, h'00112233445566778899aabbccddeeff01234567'
                             h'89abcdeffedcba9876543210' ],
                    / size / 12 : 34768,
                },
                / set-component-index / 12, 1,
                / set-vars / 19, {
                    / digest / 11 : h'8202582000112233445566778899aabb'
                                    h'ccddeeff0123456789abcdeffedcba98'
                                    h'76543210' \
                        [ 2, h'00112233445566778899aabbccddeeff01234567'
                             h'89abcdeffedcba9876543210' ],
                    / size / 12 : 34768,
                },
                / condition-vendor-id / 1, None,
                / condition-class-id / 2, None,
            ],
        },
        / apply-image / 9 : h'860c0013a106781b687474703a2f2f6578616d70'
                            h'6c652e636f6d2f66696c652e62696e15f6' \ [
            / set-component-index / 12, 0,
            / set-vars / 19, {
                / uri / 6 : http://example.com/file.bin,
            },
            / fetch / 21, None,
        ],
        / run-image / 12 : h'8e0c0003f60c0113a10a0015f603f617f6' \ [
            / set-component-index / 12, 0,
            / condition-image / 3, None,
            / set-component-index / 12, 1,
            / set-vars / 19, {
                / source-component / 10 : 0,
            },
            / fetch / 21, None,
            / condition-image / 3, None,
            / run / 23, None,
        ],
    }
}
```

Total size of outer wrapper without COSE authentication object: 236

Outer:

a201f60258e6a5010102040358a3a20254828245466c61736843003401824352414d4200
040458898e13a20350fa6b4a53d5ad5fdfbe9de663e4d41ffe04501492af1425695e48bf
429b2d51f2ab450c0013a20b58248202582000112233445566778899aabbccddeeff0123
456789abcdeffedcba98765432100c1987d00c0113a20b58248202582000112233445566
778899aabbccddeeff0123456789abcdeffedcba98765432100c1987d001f602f6095825
860c0013a106781b687474703a2f2f6578616d706c652e636f6d2f66696c652e62696e15
f60c518e0c0003f60c0113a10a0015f603f617f6

Total size of outer wrapper with COSE authentication object: 321

Signed Outer:

a2015854d28443a10126a1044874657374206b6579f6584070eb70f2552533fc954e934f
50f42bdd9b6f7d4fd7e114636b9cdbef2a065f9640243a7857f66c4389aea906c4f3b451
50c8e55461e9bfda945904033fc70a840258e6a5010102040358a3a20254828245466c61
736843003401824352414d4200040458898e13a20350fa6b4a53d5ad5fdfbe9de663e4d4
1ffe04501492af1425695e48bf429b2d51f2ab450c0013a20b5824820258200011223344
5566778899aabbccddeeff0123456789abcdeffedcba98765432100c1987d00c0113a20b
58248202582000112233445566778899aabbccddeeff0123456789abcdeffedcba987654
32100c1987d001f602f6095825860c0013a106781b687474703a2f2f6578616d706c652e
636f6d2f66696c652e62696e15f60c518e0c0003f60c0113a10a0015f603f617f6

12.5. Example 4:

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

The following JSON shows the intended behaviour of the manifest.

```
{
    "structure-version": 1,
    "sequence-number": 5,
    "common": {
        "common-sequence": [
            {
                "directive-set-var": {
                    "vendor-id": "fa6b4a53-d5ad-5fdf-be9d-e663e4d41ffe",
                    "class-id": "1492af14-2569-5e48-bf42-9b2d51f2ab45"
                }
            },
            { "directive-set-component": 0 },
            {
                "directive-set-var": {
                    "digest": "00112233445566778899aabbccddeeff"
```

```
                              "0123456789abcdeffedcba9876543210",
                    "size": 34768
                }
            },
            { "directive-set-component": 1 },
            {
                "directive-set-var": {
                    "digest": "0123456789abcdeffedcba9876543210"
                              "00112233445566778899aabbccddeeff",
                    "size": 34768
                }
            },
            { "condition-vendor-id": null },
            { "condition-class-id": null }
        ],
        "components": [
            [
                "Flash",
                78848
            ],
            [
                "RAM",
                1024
            ]
        ]
    },
    "apply-image": [
        { "directive-set-component": 0 },
        {
            "directive-set-var": {
                "uri": "http://example.com/file.bin"
            }
        },
        { "directive-fetch": null }
    ],
    "load-image": [
        { "directive-set-component": 0 },
        { "condition-image": null },
        { "directive-set-component": 1 },
        {
            "directive-set-var": {
                "source-index": 0,
                "compression-info": {
                    "algorithm": "gzip"
                }
            }
        },
        { "directive-copy": null }
```

```
    ],
    "run-image": [
        { "condition-image": null },
        { "directive-run": null }
    ]
}
```

Converted into the SUIT manifest, this produces:

```
{
    / auth object / 1 : h'd28443a10126a1044874657374206b6579f658403491'
                        h'5619c1ef02b4a7ffbbb69083e8b3fb82febd9ecd6feb'
                        h'f666d700fb981b208ec6d3df8735f36fd4a0a84e0189'
                        h'43ef80e25f57fc130a43e57c6634f337b7fa',
    / manifest / 2 : h'a6010102050358a3a20254828245466c6173684300340182'
                     h'4352414d4200040458898e13a20350fa6b4a53d5ad5fdfbe'
                     h'9de663e4d41ffe04501492af1425695e48bf429b2d51f2ab'
                     h'450c0013a20b5824820258200011223344556677889 9aabb'
                     h'ccddeeff0123456789abcdeffedcba98765432100c1987d0'
                     h'0c0113a20b58248202582001234567 89abcdeffedcba9876'
                     h'543210001122334455667788 99aabbccddeeff0c1987d001'
                     h'f602f6095825860c0013a106781b687474703a2f2f657861'
                     h'6d706c652e636f6d2f66696c652e62696e15f60b528a0c00'
                     h'03f60c0113a20843a101010a0016f60c458403f617f6' \
    {
        / structure-version / 1 : 1,
        / sequence-number / 2 : 5,
        / common / 3 : h'a20254828245466c6173684300340182435241 4d420004'
                       h'0458898e13a20350fa6b4a53d5ad5fdfbe9de663e4d41f'
                       h'fe04501492af1425695e48bf429b2d51f2ab450c0013a2'
                       h'0b5824820258200011223344556677889 9aabbccddeeff'
                       h'0123456789abcdeffedcba98765432100c1987d00c0113'
                       h'a20b582482025820012345678 9abcdeffedcba98765432'
                       h'100011223344556677889 9aabbccddeeff0c1987d001f6'
                       h'02f6' \ {
            / components / 2 : h'828245466c617368430034018243524 14d4200'
                               h'04' \
            [
                [h'466c617368', h'003401'],
                [h'52414d', h'0004'],
            ],
            / common-sequence / 4 : h'8e13a20350fa6b4a53d5ad5fdfbe9de6'
                                    h'63e4d41ffe04501492af1425695e48bf'
                                    h'429b2d51f2ab450c0013a20b58248202'
                                    h'582000112233445566778899aabbccdd'
                                    h'eeff0123456789abcdeffedcba987654'
                                    h'32100c1987d00c0113a20b5824820258'
                                    h'200123456789abcdeffedcba98765432'
```

```
                            h'10001122334455667788 99aabbccddee'
                            h'ff0c1987d001f602f6' \ [
                / set-vars / 19, {
                    / vendor-id / 3 : h'fa6b4a53d5ad5fdfbe9de663e4d41f'
                                      h'fe',
                    / class-id / 4 : h'1492af1425695e48bf429b2d51f2ab45',
                },
                / set-component-index / 12, 0,
                / set-vars / 19, {
                    / digest / 11 : h'8202582000112233445566778899aabb'
                                    h'ccddeeff0123456789abcdeffedcba98'
                                    h'76543210' \
                        [ 2, h'00112233445566778899aabbccddeeff01234567'
                             h'89abcdeffedcba9876543210' ],
                    / size / 12 : 34768,
                },
                / set-component-index / 12, 1,
                / set-vars / 19, {
                    / digest / 11 : h'820258200123456789abcdeffedcba98'
                                    h'7654321000112233445566778899aabb'
                                    h'ccddeeff' \
                        [ 2, h'0123456789abcdeffedcba987654321000112233'
                             h'445566778899aabbccddeeff' ],
                    / size / 12 : 34768,
                },
                / condition-vendor-id / 1, None,
                / condition-class-id / 2, None,
            ],
        },
        / apply-image / 9 : h'860c0013a106781b687474703a2f2f6578616d70'
                            h'6c652e636f6d2f66696c652e62696e15f6' \ [
            / set-component-index / 12, 0,
            / set-vars / 19, {
                / uri / 6 : http://example.com/file.bin,
            },
            / fetch / 21, None,
        ],
        / load-image / 11 : h'8a0c0003f60c0113a20843a101010a0016f6' \ [
            / set-component-index / 12, 0,
            / condition-image / 3, None,
            / set-component-index / 12, 1,
            / set-vars / 19, {
                / compression-info / 8 : h'a10101',
                / source-component / 10 : 0,
            },
            / copy / 22, None,
        ],
        / run-image / 12 : h'8403f617f6' \ [
```

```
            / condition-image / 3, None,
            / run / 23, None,
        ],
    }
}
```

Total size of outer wrapper without COSE authentication object: 244

Outer:

a201f60258eea6010102050358a3a20254828245466c6173684300340182435241 4d4200

Total size of outer wrapper with COSE authentication object: 329

Signed Outer:

a2015854d28443a10126a1044874657374206b6579f6584034915619c1ef02b4a7ffbbb6
9083e8b3fb82febd9ecd6febf666d700fb981b208ec6d3df8735f36fd4a0a84e018943ef
80e25f57fc130a43e57c6634f337b7fa0258eea6010102050358a3a20254828245466c61
7368430034018243524 14d4200040458898e13a20350fa6b4a53d5ad5fdfbe9de663e4d4
1ffe04501492af1425695e48bf429b2d51f2ab450c0013a20b58248202582000112233 44
5566778899aabbccddeeff0123456789abcdeffedcba98765432100c1987d00c0113a20b
58248202582001234 56789abcdeffedcba98765432100011223344556677 8899aabbccdd
eeff0c1987d001f602f6095825860c0013a106781b687474703a2f2f6578616d706c652e
636f6d2f66696c652e62696e15f60b528a0c0003f60c0113a20843a101010a0016f60c45
8403f617f6

12.6. Example 5:

Compatibility test, download, installation, and secure boot.

The following JSON shows the intended behaviour of the manifest.

```
{
    "structure-version": 1,
    "sequence-number": 6,
    "common": {
        "common-sequence": [
            {
                "directive-set-var": {
                    "vendor-id": "fa6b4a53-d5ad-5fdf-be9d-e663e4d41ffe",
                    "class-id": "1492af14-2569-5e48-bf42-9b2d51f2ab45"
```

```
                }
            },
            { "directive-set-component": 0 },
            {
                "directive-set-var": {
                    "digest": "00112233445566778899aabbccddeeff"
                              "0123456789abcdeffedcba9876543210",
                    "size": 34768
                }
            },
            { "directive-set-component": 1 },
            {
                "directive-set-var": {
                    "digest": "0123456789abcdeffedcba9876543210"
                              "00112233445566778899aabbccddeeff",
                    "size": 34768
                }
            },
            { "condition-vendor-id": null },
            { "condition-class-id": null }
        ],
        "components": [
            [
                "ext-Flash",
                78848
            ],
            [
                "Flash",
                1024
            ]
        ]
    },
    "apply-image": [
        { "directive-set-component": 0 },
        {
            "directive-set-var": {
                "uri": "http://example.com/file.bin"
            }
        },
        { "directive-fetch": null }
    ],
    "load-image": [
        { "directive-set-component": 1 },
        { "condition-not-image": null },
        { "directive-set-component": 0 },
        { "condition-image": null },
        { "directive-set-component": 1 },
        {
```

```
            "directive-set-var": {
                "source-index": 0
            }
        },
        { "directive-fetch": null }
    ],
    "run-image": [
        { "directive-set-component": 1 },
        { "condition-image": null },
        { "directive-run": null }
    ]
}
```

Converted into the SUIT manifest, this produces:

```
{
    / auth object / 1 : h'd28443a10126a1044874657374206b6579f65840a516'
                        h'466c62602aa017422f23d1469339e40c5cf06f9090da'
                        h'09bd9939ecfc4c1ffe3e6ce50e0620fe9948f76552da'
                        h'703a4c0bf2532d073be2d1f215ec83483f46',
    / manifest / 2 : h'a6010102060358a6a202578282467b1b4595ab2143003401'
                     h'8245466c617368420004045889 8e13a20350fa6b4a53d5ad'
                     h'5fdfbe9de663e4d41ffe04501492af1425695e48bf429b2d'
                     h'51f2ab450c0013a20b5824820258200011223344556677 88'
                     h'99aabbccddeeff0123456789abcdeffedcba98765432100c'
                     h'1987d00c0113a20b58248202582001234567 89abcdeffedc'
                     h'ba98765432100011223344556677 8899aabbccddeeff0c19'
                     h'87d001f602f6095825860c0013a106781b687474703a2f2f'
                     h'6578616d706c652e636f6d2f66696c652e62696e15f60b52'
                     h'8e0c011819f60c0003f60c0113a10a0015f60c47860c0103'
                     h'f617f6' \
    {
        / structure-version / 1 : 1,
        / sequence-number / 2 : 6,
        / common / 3 : h'a202578282467b1b4595ab21430034018245466c617368'
                       h'4200040458898e13a20350fa6b4a53d5ad5fdfbe9de663'
                       h'e4d41ffe04501492af1425695e48bf429b2d51f2ab450c'
                       h'0013a20b582482025820001122334455667788 99aabbcc'
                       h'ddeeff0123456789abcdeffedcba98765432100c1987d0'
                       h'0c0113a20b5824820258200123456789abcdeffedcba98'
                       h'765432100011223344556677 8899aabbccddeeff0c1987'
                       h'd001f602f6' \ {
            / components / 2 : h'8282467b1b4595ab21430034018245466c6173'
                               h'68420004' \
            [
                [h'7b1b4595ab21', h'003401'],
                [h'466c617368', h'0004'],
            ],
```

```
            / common-sequence / 4 : h'8e13a20350fa6b4a53d5ad5fdfbe9de6'
                                    h'63e4d41ffe04501492af1425695e48bf'
                                    h'429b2d51f2ab450c0013a20b58248202'
                                    h'58200011223344556677889 9aabbccdd'
                                    h'eeff0123456789abcdeffedcba987654'
                                    h'32100c1987d00c0113a20b5824820258'
                                    h'200123456789abcdeffedcba98765432'
                                    h'10001122334455667788 99aabbccddee'
                                    h'ff0c1987d001f602f6' \ [
                / set-vars / 19, {
                    / vendor-id / 3 : h'fa6b4a53d5ad5fdfbe9de663e4d41f'
                                      h'fe',
                    / class-id / 4 : h'1492af1425695e48bf429b2d51f2ab45',
                },
                / set-component-index / 12, 0,
                / set-vars / 19, {
                    / digest / 11 : h'8202582000112233445566778899aabb'
                                    h'ccddeeff0123456789abcdeffedcba98'
                                    h'76543210' \
                        [ 2, h'00112233445566778899aabbccddeeff01234567'
                             h'89abcdeffedcba9876543210' ],
                    / size / 12 : 34768,
                },
                / set-component-index / 12, 1,
                / set-vars / 19, {
                    / digest / 11 : h'820258200123456789abcdeffedcba98'
                                    h'7654321000112233445566778899aabb'
                                    h'ccddeeff' \
                        [ 2, h'0123456789abcdeffedcba987654321000112233'
                             h'445566778899aabbccddeeff' ],
                    / size / 12 : 34768,
                },
                / condition-vendor-id / 1, None,
                / condition-class-id / 2, None,
            ],
        },
        / apply-image / 9 : h'860c0013a106781b687474703a2f2f6578616d70'
                            h'6c652e636f6d2f66696c652e62696e15f6' \ [
            / set-component-index / 12, 0,
            / set-vars / 19, {
                / uri / 6 : http://example.com/file.bin,
            },
            / fetch / 21, None,
        ],
        / load-image / 11 : h'8e0c011819f60c0003f60c0113a10a0015f6' \ [
            / set-component-index / 12, 1,
            / condition-not-image / 25, None,
            / set-component-index / 12, 0,
```

```
            / condition-image / 3, None,
            / set-component-index / 12, 1,
            / set-vars / 19, {
                / source-component / 10 : 0,
            },
            / fetch / 21, None,
        ],
        / run-image / 12 : h'860c0103f617f6' \ [
            / set-component-index / 12, 1,
            / condition-image / 3, None,
            / run / 23, None,
        ],
    }
}
```

Total size of outer wrapper without COSE authentication object: 249

Outer:

a201f60258f3a6010102060358a6a202578282467b1b4595ab21430034018245466c6173
684200040458898e13a20350fa6b4a53d5ad5fdfbe9de663e4d41ffe04501492af142569
5e48bf429b2d51f2ab450c0013a20b5824820258200011223344556677889 9aabbccddee
ff0123456789abcdeffedcba98765432100c1987d00c0113a20b58248202582001234567
89abcdeffedcba987654321000112233445566778899aabbccddeeff0c1987d001f602f6
095825860c0013a106781b687474703a2f2f6578616d706c652e636f6d2f66696c652e62
696e15f60b528e0c011819f60c0003f60c0113a10a0015f60c47860c0103f617f6

Total size of outer wrapper with COSE authentication object: 334

Signed Outer:

a2015854d28443a10126a104487465737420 6b6579f65840a516466c62602aa017422f23
d1469339e40c5cf06f9090da09bd9939ecfc4c1ffe3e6ce50e0620fe9948f76552da703a
4c0bf2532d073be2d1f215ec83483f460258f3a6010102060358a6a202578282467b1b45
95ab21430034018245466c617368420004045889 8e13a20350fa6b4a53d5ad5fdfbe9de6
63e4d41ffe04501492af1425695e48bf429b2d51f2ab450c0013a20b58248202582000 11
2233445566778899aabbccddeeff0123456789abcdeffedcba98765432100c1987d00c01
13a20b58248202582001234567 89abcdeffedcba987654321000112233445566778899aa
bbccddeeff0c1987d001f602f6095825860c0013a106781b687474703a2f2f6578616d70
6c652e636f6d2f66696c652e62696e15f60b528e0c011819f60c0003f60c0113a10a0015
f60c47860c0103f617f6

12.7. Example 6:

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

The following JSON shows the intended behaviour of the manifest.

```
{
    "structure-version": 1,
    "sequence-number": 7,
    "common": {
        "common-sequence": [
            {
                "directive-set-var": {
                    "vendor-id": "fa6b4a53-d5ad-5fdf-be9d-e663e4d41ffe",
                    "class-id": "1492af14-2569-5e48-bf42-9b2d51f2ab45"
                }
            },
            { "directive-set-component": 0 },
            {
                "directive-set-var": {
                    "digest": "00112233445566778899aabbccddeeff"
                              "0123456789abcdeffedcba9876543210",
                    "size": 34768
                }
            },
            { "directive-set-component": 1 },
            {
                "directive-set-var": {
                    "digest": "0123456789abcdeffedcba9876543210"
                              "00112233445566778899aabbccddeeff",
                    "size": 76834
                }
            },
            { "condition-vendor-id": null },
            { "condition-class-id": null }
        ],
        "components": [
            [
                "Flash",
                78848
            ],
            [
                "Flash",
                132096
            ]
        ]
    },
    "apply-image": [
        { "directive-set-component": 0 },
        {
            "directive-set-var": {
                "uri": "http://example.com/file1.bin"
            }
        },
```

```
        { "directive-set-component": 1 },
        {
            "directive-set-var": {
                "uri": "http://example.com/file2.bin"
            }
        },
        { "directive-set-component": true },
        { "directive-fetch": null }
    ],
    "run-image": [
        { "directive-set-component": true },
        { "condition-image": null },
        { "directive-set-component": 0 },
        { "directive-run": null }
    ]
}
```

Converted into the SUIT manifest, this produces:

```
{
    / auth object / 1 : h'd28443a10126a1044874657374206b6579f658400d44'
                        h'c766566a88c5bbe61b544edd14effa7d53c9a6d43221'
                        h'99c6285490460b910c8e96c6a1065cc1be9cfa438f7b'
                        h'eeaffa9922e2ae440d6c8d0b9cb26bed2ffe',
    / manifest / 2 : h'a5010102070358a8a20257828245466c6173684300340182'
                     h'45466c617368430004020458 8b8e13a20350fa6b4a53d5ad'
                     h'5fdfbe9de663e4d41ffe04501492af1425695e48bf429b2d'
                     h'51f2ab450c0013a20b5824820258200011223344556677 88'
                     h'99aabbccddeeff0123456789abcdeffedcba98765432100c'
                     h'1987d00c0113a20b58248202582001234567 89abcdeffedc'
                     h'ba98765432100011223344556677 8899aabbccddeeff0c1a'
                     h'00012c2201f602f609584b8c0c0013a106781c687474703a'
                     h'2f2f6578616d706c652e636f6d2f66696c65312e62696e0c'
                     h'0113a106781c687474703a2f2f6578616d706c652e636f6d'
                     h'2f66696c65322e62696e0cf515f60c49880cf503f60c0017'
                     h'f6' \
    {
        / structure-version / 1 : 1,
        / sequence-number / 2 : 7,
        / common / 3 : h'a20257828245466c6173684300340182 45466c61736843'
                       h'00040204588b8e13a20350fa6b4a53d5ad5fdfbe9de663'
                       h'e4d41ffe04501492af1425695e48bf429b2d51f2ab450c'
                       h'0013a20b58248202582000112233445566778899aabbcc'
                       h'ddeeff0123456789abcdeffedcba98765432100c1987d0'
                       h'0c0113a20b5824820258200123456789abcdeffedcba98'
                       h'76543210001122334455667788 99aabbccddeeff0c1a00'
                       h'012c2201f602f6' \ {
            / components / 2 : h'828245466c6173684300340182 45466c617368'
```

```
                        h'43000402' \
            [
                [h'466c617368', h'003401'],
                [h'466c617368', h'000402'],
            ],
            / common-sequence / 4 : h'8e13a20350fa6b4a53d5ad5fdfbe9de6'
                                    h'63e4d41ffe04501492af1425695e48bf'
                                    h'429b2d51f2ab450c0013a20b58248202'
                                    h'58200011223344556677889aabbccdd'
                                    h'eeff0123456789abcdeffedcba987654'
                                    h'32100c1987d00c0113a20b5824820258'
                                    h'200123456789abcdeffedcba98765432'
                                    h'10001122334455667788899aabbccddee'
                                    h'ff0c1a00012c2201f602f6' \ [
                / set-vars / 19, {
                    / vendor-id / 3 : h'fa6b4a53d5ad5fdfbe9de663e4d41f'
                                      h'fe',
                    / class-id / 4 : h'1492af1425695e48bf429b2d51f2ab45',
                },
                / set-component-index / 12, 0,
                / set-vars / 19, {
                    / digest / 11 : h'820258200011223344556677889aabb'
                                    h'ccddeeff0123456789abcdeffedcba98'
                                    h'76543210' \
                        [ 2, h'00112233445566778899aabbccddeeff01234567'
                             h'89abcdeffedcba9876543210' ],
                    / size / 12 : 34768,
                },
                / set-component-index / 12, 1,
                / set-vars / 19, {
                    / digest / 11 : h'820258200123456789abcdeffedcba98'
                                    h'76543210001122334455667788899aabb'
                                    h'ccddeeff' \
                        [ 2, h'0123456789abcdeffedcba987654321000112233'
                             h'445566778899aabbccddeeff' ],
                    / size / 12 : 76834,
                },
                / condition-vendor-id / 1, None,
                / condition-class-id / 2, None,
            ],
        },
        / apply-image / 9 : h'8c0c0013a106781c687474703a2f2f6578616d70'
                            h'6c652e636f6d2f66696c65312e62696e0c0113a1'
                            h'06781c687474703a2f2f6578616d706c652e636f'
                            h'6d2f66696c65322e62696e0cf515f6' \ [
            / set-component-index / 12, 0,
            / set-vars / 19, {
                / uri / 6 : http://example.com/file1.bin,
```

```
            },
            / set-component-index / 12, 1,
            / set-vars / 19, {
                / uri / 6 : http://example.com/file2.bin,
            },
            / set-component-index / 12, True,
            / fetch / 21, None,
        ],
        / run-image / 12 : h'880cf503f60c0017f6' \ [
            / set-component-index / 12, True,
            / condition-image / 3, None,
            / set-component-index / 12, 0,
            / run / 23, None,
        ],
    }
}
```

Total size of outer wrapper without COSE authentication object: 272

Outer:

a201f602590109a5010102070358a8a20257828245466c6173684300340182454666c6173
684300040204588b8e13a20350fa6b4a53d5ad5fdfbe9de663e4d41ffe04501492af1425
695e48bf429b2d51f2ab450c0013a20b582482025820001122334455667788 99aabbccdd
eeff0123456789abcdeffedcba98765432100c1987d00c0113a20b58248202582001 2345
6789abcdeffedcba987654321000112233445566778899aabbccddeeff0c1a00012c2201
f602f609584b8c0c0013a106781c687474703a2f2f6578616d706c652e636f6d2f66696c
65312e62696e0c0113a106781c687474703a2f2f6578616d706c652e636f6d2f66696c65
322e62696e0cf515f60c49880cf503f60c0017f6

Total size of outer wrapper with COSE authentication object: 357

Signed Outer:

a2015854d28443a10126a10448746573742062b6579f658400d44c766566a88c5bbe61b54
4edd14effa7d53c9a6d4322199c6285490460b910c8e96c6a1065cc1be9cfa438f7beeaf
fa9922e2ae440d6c8d0b9cb26bed2ffe02590109a5010102070358a8a20257828245466c
6173684300340182454666c6173684300040204588b8e13a20350fa6b4a53d5ad5fdfbe9d
e663e4d41ffe04501492af1425695e48bf429b2d51f2ab450c0013a20b5824820258200 0
112233445566778899aabbccddeeff0123456789abcdeffedcba98765432100c1987d00c
0113a20b5824820258200123456789abcdeffedcba987654321000112233445566778899
aabbccddeeff0c1a00012c2201f602f609584b8c0c0013a106781c687474703a2f2f6578
616d706c652e636f6d2f66696c65312e62696e0c0113a106781c687474703a2f2f657861
6d706c652e636f6d2f66696c65322e62696e0cf515f60c49880cf503f60c0017f6

13. IANA Considerations

Several registries will be required for:

- standard Commands

- standard Parameters

- standard Algorithm identifiers

- standard text values

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 more detailed discussion about security can be found in the architecture document [I-D.ietf-suit-architecture] and in [I-D.ietf-suit-information-model].

15. Mailing List Information

The discussion list for this document is located at the e-mail address suit@ietf.org [1]. Information on the group and information on how to subscribe to the list is at https://www1.ietf.org/mailman/listinfo/suit [2]

Archives of the list can be found at: https://www.ietf.org/mail-archive/web/suit/current/index.html [3]

16. 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

17. References

17.1. 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>.

17.2. Informative References

[I-D.ietf-suit-architecture]
Moran, B., Meriac, M., Tschofenig, H., and D. Brown, "A Firmware Update Architecture for Internet of Things Devices", draft-ietf-suit-architecture-07 (work in progress), October 2019.

[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-04 (work in progress), October 2019.

17.3. URIs

[1] mailto:suit@ietf.org

[2] https://www1.ietf.org/mailman/listinfo/suit

[3] https://www.ietf.org/mail-archive/web/suit/current/index.html

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