SUIT B. Moran
Internet-Draft H. Tschofenig
Intended status: Standards Track Arm Limited
Expires: September 10, 2020 H. Birkholz
Fraunhofer SIT
K. Zandberg
Inria
March 09, 2020

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

draft-ietf-suit-manifest-04

Abstract

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

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 repositories; 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].

Most Update and Trusted Execution operations are composed of the same small set of fundamental operations, such as copying a firmware image from one place to another, checking that a firmware image is correct, verifying that the specified firmware is the correct firmware for the device, or unpacking a firmware. By using these fundamental operations in different orders and changing the parameters they use, a great many use cases can be supported by the same encoding. The SUIT manifest uses this observation to heavily optimize update metadata for consumption by constrained devices.

While the SUIT manifest is informed by and optimized 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.

The following terminology is used throughout this document.

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

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

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

- Manifest: A 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 Recipients.

- 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).

3. How to use this Document

This specification covers four aspects of firmware update: the background that has informed this specification, the behavior of a device consuming a manifest, the process of creating a manifest, and the structure of the manifest itself.

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

- Section 5 describes what actions a manifest processor should take.

- Section 6 describes the process of creating a manifest.

- Section 7 specifies the content of the manifest.

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

This section describes the logistical challenges, device constraints, use cases, and design principles that informed the structure of the manifest. For the security considerations of the manifest, see [I-D.ietf-suit-information-model].

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. IoT Firmware Update Constraints

The various constraints on IoT devices create 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 a bootloader, then the bootloader MUST also verify the signature of the manifest and the applicability of the manifest in order to implement secure boot workflows. The bootloader MAY add its own authentication, e.g. a MAC, to the manifest in order to prevent further verifications.

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

4.2.1. Pre-Authentication Compatibility Checks

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

If a device:

1. expects to receive many incompatible manifests.

2. expects to receive few manifests with failing signatures-for example if it is behind a gateway that checks signatures.

3. has a power budget that makes signature verification undesirable.

Then, the device MAY choose to parse and execute only the SUIT_Common section of the manifest prior to signature verification. The guidelines in Creating Manifests (Section 6) require that the common section contain the applicability checks, so this section is sufficient for applicability verification. The manifest parser MUST NOT execute any command with side-effects outside the parser (for example, Run, Copy, Swap, or Fetch commands) prior to authentication and any such command MUST result in an error.

4.3. SUIT Manifest Goals

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

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

- Simple to parse on a constrained node

- Simple to process on a constrained node

- Compact encoding

- Comprehensible by an intermediate system

- Expressive enough to enable advanced use cases on advanced nodes

- Extensible

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

- the Firmware Author to reason about releasing a firmware.

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

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

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

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

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

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

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

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

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

4.4. SUIT Manifest Design Summary

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

The specialized byte code approach gives benefits equivalent to those provided by a scripting language or conventional byte code, with two substantial differences. First, the language is extremely high level, consisting of only the operations that a device may perform during update and trusted execution of a firmware image. Second, the language specifies linear behavior, without reverse branches. Conditional processing is supported, and parallel and out-of-order processing may be performed by sufficiently capable devices.

By structuring the data in this way, the manifest processor becomes a very simple engine that uses a pull parser to interpret the manifest.

This pull parser invokes a series of command handlers that evaluate a Condition or execute a Directive. Most data is structured in a highly regular pattern, which simplifies the parser.

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

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

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

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

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

5. Interpreter Behavior

This section describes the behavior of the manifest interpreter. This section focuses primarily on interpreting commands in the manifest. However, there are several other important behaviors of the interpreter: encoding version detection , rollback protection, and authenticity verification are chief among these (see Section 5.1).

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 modeled 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 behavior.

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 behavior 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 Authorized (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 Behavior

The following table describes the behavior 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(isAuthorized())
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 Dependency command is encountered, the interpreter loads the dependency identified by the Current Dependency Index. The interpreter first executes the common-sequence section of the identified dependency, then it executes the section of the dependency that corresponds to the currently executing section of the dependent.

The interpreter also performs the checks described in Section 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 behavior.

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 example 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:
  - Set Current Component.
  - Set Parameters:
    * Vendor ID.
    * Class ID.
  - Check Vendor Identifier.
  - Check Class Identifier.

6.3. Use Case Template: XIP Secure Boot

- Common:
  - Set Current Component.
  - Override Parameters:
    * Digest.
    * Size.
- Run:
  - Set Current Component.
  - 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 enveloped in a CBOR map containing:

1. Authentication delegation chain(s)

2. The authentication wrapper (a list of COSE sign/MAC objects)

3. The manifest (a map)

    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. Reference URI

    4. Dependency resolution Reference or conditions/directives

    5. Payload fetch Reference or conditions/directives

    6. Installation Reference or conditions/directives

    7. Verification conditions/directives

    8. Load conditions/directives

    9. Run conditions/directives

    10. Text / Reference

    11. COSWID / Reference

4. Dependency resolution conditions/directives

5. Payload fetch conditions/directives

6. Installation conditions/directives

7. Text

8. COSWID

9. Inline Payload(s)

All elements in the outer map are wrapped in bstr.

```
   +--------------------+
   | Manifest Envelope  |
   +--------------------+
   | Delegation CWTs    |
   | COSE Envelopes     |
   | Manifest --------------------> +-----------------------+
   | Severable Elements |           | Manifest (bstr)       |
   +--------------------+           +-----------------------+
                                    | Structure Version     |
                                    | Sequence Number       |
   +-----------------------+ <------- Common Info           |
   | Common Info (bstr)    |        | Reference URI         |
   +-----------------------+        | Installation Commands ---+
   | Dependencies          |        | Invocation Commands -----+
   | Components IDs        |        | Protected Elements    |  |
   | Component References  |        +-----------------------+  |
   | Common Commands --------+                                 |
   +-----------------------+ |                                 |
                             +-> +-----------------------+ <---+
                                 | Commands (bstr)       |
                                 +-----------------------+
                                 | List of ( pairs of (  |
                                 |   * command ID code   |
                                 |   * argument          |
                                 | ))                    |
                                 +-----------------------+
```

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.

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. Envelope

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 envelope are contained in bstr objects. Wherever the manifest references an object in the envelope, the bstr is included in the digest calculation.

The CDDL that describes the envelope is below

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

SUIT_Delegation = [ + [ + CWT ] ]

SUIT_Authentication_Wrapper = [ + bstr .cbor SUIT_Authentication_Block ]

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

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

SUIT_Encryption_Wrapper = COSE_Encrypt_Tagged / COSE_Encrypt0_Tagged

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

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

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

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

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

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-reference-uri          => #6.32(tstr),
    * $$SUIT_Severable_Command_Sequences,
    * $$SUIT_Command_Sequences,
    * $$SUIT_Protected_Elements,
}

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

SUIT_Severable_Command_Sequence =
    SUIT_Digest / bstr .cbor SUIT_Command_Sequence

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

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

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

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 the Recipient. Because SUIT_Digest is a CBOR Array and each severable element is a CBOR bstr, it is straight-forward for a Recipient to determine whether an element is been severable. The key used for a severable element is the same in the SUIT_Manifest and in the

SUIT_Envelope so that a Recipient can easily identify the correct data in the envelope.

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

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

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 SUIT_Envelope. text is OPTIONAL.

suit-coswid is a digest that uniquely identifies the content of the concise-software-identifier that is packaged in the SUIT_Envelope. 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 behavior of another manifest, for example by directing the recipient to look at a different URI for the image or by changing the expected format, such as when a gateway performs decryption on behalf of a constrained device. 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	bstr	Component / Global	Vendor ID	A RFC4122 UUID representing the vendor of the device or component
2	bstr	Component / Global	Class ID	A RFC4122 UUID representing the class of the device or component
3	bstr	Component / Dependency	Image Digest	A SUIT_Digest
4	uint	Component / Global	Use Before	POSIX timestamp
5	uint	Component	Component Offset	Offset of the component
12	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.
13	boolean	Command Segment	Soft Failure	Condition failures only terminate the current command segment.
14	uint	Component /	Image Size	Integer size

		Dependency		
18	bstr	Component / Dependency	Encryption Info	A COSE object defining the encryption mode of a resource
19	bstr	Component	Compression Info	The information required to decompress the image
20	bstr	Component	Unpack Info	The information required to unpack the image
21	tstr	Component / Dependency	URI	A URI from which to fetch a resource
22	uint	Component	Source Component	A Component Index
23	bstr / nil	Component	Run Arguments	An encoded set of arguments for Run
24	bstr	Component / Global	Device ID	A RFC4122 UUID representing the device or component
25	uint	Global	Minimum Battery	A minimum battery level in mWh
26	int	Component / Global	Priority	The priority of the update
nint	int / bstr / tstr	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 modularization and division of responsibility within a pull parser. The same consideration does not apply to Directives because those elements are invoked with their arguments immediately

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 parallelize their handling of updates. Strict Order defaults to True. It MAY be set to False when the order of operations does not matter. When arriving at the end of a command sequence, ALL commands MUST have completed, regardless of the state of SUIT_Parameter_Strict_Order. If SUIT_Parameter_Strict_Order is returned to True, ALL preceding commands MUST complete before the next command is executed.

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.

7.7.1. 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
```

7.7.2. 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
```

7.7.3. SUIT_Parameters CDDL

The following CDDL describes all SUIT_Parameters.

```
SUIT_Parameters //= (suit-parameter-vendor-identifier => RFC4122_UUID)
SUIT_Parameters //= (suit-parameter-class-identifier => RFC4122_UUID)
SUIT_Parameters //= (suit-parameter-image-digest
    => bstr .cbor SUIT_Digest)
SUIT_Parameters //= (suit-parameter-image-size => uint)
SUIT_Parameters //= (suit-parameter-use-before => uint)
SUIT_Parameters //= (suit-parameter-component-offset => uint)

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

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

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


SUIT_Parameters //= (suit-parameter-uri-list
    => bstr .cbor SUIT_Component_URI_List)
SUIT_Parameters //= (suit-parameter-custom => int/bool/tstr/bstr)

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

RFC4122_UUID = bstr .size 16

SUIT_Condition_Version_Comparison_Value = [+int]

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

SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_gzip
SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_bzip2
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
```

7.8. SUIT_Command_Sequence

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

1. Dependency Resolution

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 is a command code identifier, followed by Nil. Each directive is composed of:

1. A command code identifier

2. An argument block or Nil

Argument blocks are defined for each type of directive.

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

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

7.9. SUIT_Condition

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

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

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

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

7.9.1. 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.

7.9.2. 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.

7.9.3. suit-condition-image-not-match

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

7.9.4. suit-condition-use-before

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

7.9.5. 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.

7.9.6. suit-condition-update-authorized

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

7.9.7. 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 occurred. For example, if the manifest defines a comparison, "Equal [1]", then this will match all version sequences starting with 1. If a manifest defines both "Greater or Equal [1,0]" and "Lesser [1,10]", then it will match versions 1.0.x up to, but not including 1.10.

The following CDDL describes SUIT_Condition_Version_Argument

```
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_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.

7.9.8. SUIT_Condition_Custom

SUIT_Condition_Custom describes any proprietary, application specific condition. This is encoded as a negative integer, chosen by the firmware developer. If additional information must be provided to the condition, it should be encoded in a custom parameter (a nint) as described in Section 7.6. SUIT_Condition_Custom is OPTIONAL to implement.

7.9.9. 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.

7.9.9.1. 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).

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

7.10. SUIT_Directive

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

Directive Code	Directive Name	Implementation
12	Set Component Index	REQUIRED if more than one component
13	Set Dependency Index	REQUIRED if dependencies used
14	Abort	OPTIONAL
15	Try Each	OPTIONAL
16	Reserved	N/A
17	Reserved	N/A
18	Process Dependency	OPTIONAL
19	Set Parameters	OPTIONAL
20	Override Parameters	REQUIRED
21	Fetch	REQUIRED for Updater
22	Copy	OPTIONAL
23	Run	REQUIRED for Bootloader
29	Wait	OPTIONAL
30	Run Sequence	OPTIONAL
32	Swap	OPTIONAL

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

7.10.1. 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
```

7.10.2. 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
```

7.10.3. suit-directive-abort

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

7.10.4. 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.

7.10.5. 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 initialized to True at the beginning of each sequence. If one sequence aborts due to a condition failure, the next is started. If no sequence completes without condition failure, then suit-directive-try-each returns an error. If a particular application calls for all sequences to fail and still continue, then an empty sequence (nil) can be added to the Try Each Argument.

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
]
```

7.10.6. 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
```

7.10.7. suit-directive-set-parameters

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

Available parameters are defined in Section 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 behavior 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 optimization: a way to set a parameter to the contents of another parameter, optionally with another component ID.

7.10.8. 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}

7.10.9. 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 behavior. If an argument is needed, it must be wrapped in a bstr.

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

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

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

SUIT_Directive_Fetch_Argument = nil/bstr

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

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

suit-directive-copy reads its source from SUIT_Parameter_Source_Component.

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

```
SUIT_Directive_Copy_Argument = nil
```

7.10.11. 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.

7.10.12. 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
```

7.10.13. suit-directive-wait

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

1. Authorization

2. External Power

3. Network availability

4. Other Device Firmware Version

5. Time

6. Time of Day

7. Day of Week

The following CDDL defines the encoding of these events.

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


SUIT_Wait_Event_Argument_Authorization = 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
```

7.10.14. 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_Try_Each_Argument = [
       + bstr .cbor SUIT_Command_Sequence,
       nil / bstr .cbor SUIT_Command_Sequence
   ]

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


   SUIT_Wait_Event_Argument_Authorization = 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
```

7.11. 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 7.9.9.1)
6	model-info	The information used to create the class-id (Section 7.9.9.1)
7	component-description	Free text description of each component in the manifest
8	json-source	The JSON-formatted document that was used to create the manifest
9	yaml-source	The yaml-formatted 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_Envelope = {
    suit-delegation            => bstr .cbor SUIT_Delegation
    suit-authentication-wrapper
        => bstr .cbor SUIT_Authentication_Wrapper / nil,
    $$SUIT_Manifest_Wrapped,
    * $$SUIT_Severed_Fields,
}

SUIT_Delegation = [ + [ + CWT ] ]

CWT = SUIT_Authentication_Block

SUIT_Authentication_Wrapper = [ + bstr .cbor SUIT_Authentication_Block ]

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

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

SUIT_Encryption_Wrapper = COSE_Encrypt_Tagged / COSE_Encrypt0_Tagged

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

```
COSE_Mac_Tagged = any
COSE_Sign_Tagged = any
COSE_Mac0_Tagged = any
COSE_Sign1_Tagged = any
COSE_Encrypt_Tagged = any
COSE_Encrypt0_Tagged = any

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

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

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

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

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

```
SUIT_Severable_Command_Sequence =
    SUIT_Digest / bstr .cbor SUIT_Command_Sequence

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

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

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

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

concise-software-identity = any

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

SUIT_Component_Identifier =  [* bstr]


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

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

SUIT_Command_Custom = (suit-command-custom, bstr/tstr/int/nil)
SUIT_Condition //= (suit-condition-vendor-identifier, nil)
SUIT_Condition //= (suit-condition-class-identifier,  nil)
SUIT_Condition //= (suit-condition-device-identifier, nil)
```

```
SUIT_Condition //= (suit-condition-image-match,       nil)
SUIT_Condition //= (suit-condition-image-not-match,   nil)
SUIT_Condition //= (suit-condition-use-before,        nil)
SUIT_Condition //= (suit-condition-minimum-battery,   nil)
SUIT_Condition //= (suit-condition-update-authorized, nil)
SUIT_Condition //= (suit-condition-version,           nil)
SUIT_Condition //= (suit-condition-component-offset,  nil)

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

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

SUIT_Wait_Event = { + SUIT_Wait_Events }

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

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

```
SUIT_Parameters //= (suit-parameter-vendor-identifier => RFC4122_UUID)
SUIT_Parameters //= (suit-parameter-class-identifier => RFC4122_UUID)
SUIT_Parameters //= (suit-parameter-image-digest
    => bstr .cbor SUIT_Digest)
SUIT_Parameters //= (suit-parameter-image-size => uint)
SUIT_Parameters //= (suit-parameter-use-before => uint)
SUIT_Parameters //= (suit-parameter-component-offset => uint)

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

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

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

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

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

RFC4122_UUID = bstr .size 16

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

```
SUIT_Condition_Version_Comparison_Types /=
    suit-condition-version-comparison-lesser

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

SUIT_Condition_Version_Comparison_Value = [+int]

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

SUIT_Compression_Algorithms /= SUIT_Compression_Algorithm_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_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 = {SUIT_Text_Keys => tstr}

SUIT_Text_Keys /= suit-text-manifest-description
SUIT_Text_Keys /= suit-text-update-description
SUIT_Text_Keys /= suit-text-vendor-name
SUIT_Text_Keys /= suit-text-model-name
```

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

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

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

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

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

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

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

suit-command-custom = nint

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

suit-condition-device-identifier        = 24
suit-condition-image-not-match          = 25
```

```
suit-condition-minimum-battery          = 26
suit-condition-update-authorized        = 27
suit-condition-version                  = 28

suit-directive-set-component-index      = 12
suit-directive-set-dependency-index     = 13
suit-directive-abort                    = 14
suit-directive-try-each                 = 15
;suit-directive-do-each                  = 16 ; TBD
;suit-directive-map-filter               = 17 ; TBD
suit-directive-process-dependency       = 18
suit-directive-set-parameters           = 19
suit-directive-override-parameters      = 20
suit-directive-fetch                    = 21
suit-directive-copy                     = 22
suit-directive-run                      = 23

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

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

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

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

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

suit-parameter-device-identifier = 24
suit-parameter-minimum-battery   = 26
```

```
suit-parameter-update-priority   = 27
suit-parameter-version           = 28
suit-parameter-wait-info         = 29

suit-parameter-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-----

Each example uses SHA256 as the digest function.

12.1. Example 0: Secure Boot

Secure boot and compatibility check.

```
{
    / authentication-wrapper / 2:h'81d28443a10126a058248202582064d8094
da3ef71c5971b7b84e7f4be1f56452c32fdde7bc1c70889112f1d5d9958407d637397e
12abdd41bc026a8e8a22f0f902a5b972e7786d570a37ac43c370b64a6946b0311f059c
a01d40f74d88d6fd7193baa36f5cf20aa57c46a0411a6b704' / [
        18([
                / protected / h'a10126' / {
                    / alg / 1:-7 / ES256 /,
                } /,
                / unprotected / {
                },
                / payload / h'8202582064d8094da3ef71c5971b7b84e7f4be1f
56452c32fdde7bc1c70889112f1d5d99' / [
                    / algorithm-id / 2 / sha256 /,
                    / digest-bytes /
h'64d8094da3ef71c5971b7b84e7f4be1f56452c32fdde7bc1c70889112f1d5d99'
                ] /,
                / signature / h'7d637397e12abdd41bc026a8e8a22f0f902a5b
972e7786d570a37ac43c370b64a6946b0311f059ca01d40f74d88d6fd7193baa36f5cf
20aa57c46a0411a6b704'
            ])
    ] /,
    / manifest / 3:h'a50101020103585ea20244818141000458548614a40150fa6
b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab4503820
2582000112233445566778899aabbccddeeff0123456789abcdeffedcba98765432100
e1987d001f602f60a438203f60c438217f6' / {
        / manifest-version / 1:1,
        / manifest-sequence-number / 2:1,
        / common / 3:h'a20244818141000458548614a40150fa6b4a53d5ad5fdfb
e9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab45038202582000112233
445566778899aabbccddeeff0123456789abcdeffedcba98765432100e1987d001f602f
6' / {
            / components / 2:h'81814100' / [
                [h'00']
            ] /,
            / common-sequence / 4:h'8614a40150fa6b4a53d5ad5fdfbe9de663
e4d41ffe02501492af1425695e48bf429b2d51f2ab4503820258200011223344556677
8899aabbccddeeff0123456789abcdeffedcba98765432100e1987d001f602f6' / [
                / directive-override-parameters / 20,{
                    / vendor-id /
1:h'fa6b4a53d5ad5fdfbe9de663e4d41ffe' / fa6b4a53-d5ad-5fdf-
be9d-e663e4d41ffe /,
                    / class-id / 2:h'1492af1425695e48bf429b2d51f2ab45'
/ 1492af14-2569-5e48-bf42-9b2d51f2ab45 /,
```

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

Total size of manifest without COSE authentication object: 116

Manifest:

a1035870a50101020103585ea20244818141000458548614a40150fa6b4a
53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab
45038202582000112233445566778899aabbccddeeff0123456789abcdef
fedcba98765432100e1987d001f602f60a438203f60c438217f6

Total size of manifest with COSE authentication object: 231

Manifest with COSE authentication object:

a202587081d28443a10126a058248202582064d8094da3ef71c5971b7b84
e7f4be1f56452c32fdde7bc1c70889112f1d5d9958407d637397e12abdd4
1bc026a8e8a22f0f902a5b972e7786d570a37ac43c370b64a6946b0311f0
59ca01d40f74d88d6fd7193baa36f5cf20aa57c46a0411a6b704035870a5
0101020103585ea20244818141000458548614a40150fa6b4a53d5ad5fdf
be9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab4503820258
2000112233445566778899aabbccddeeff0123456789abcdeffedcba9876
5432100e1987d001f602f60a438203f60c438217f6

12.2. Example 1: Simultaneous Download and Installation of Payload

Simultaneous download and installation of payload.

```
  {
      / authentication-wrapper / 2:h'81d28443a10126a0582482025820666b83f
```

```
  f516281903871704895355aa9441656d8a24401de6458595c42cb0165d58405cb310acb
  34f7ebb42acfffce430dbda94faa412900ce8e76650445e2c37e4cc132d8bb5f30ecf5
  f8130270bbf8d159f6d36e1cdf97b64229910fdb447538af1' / [
          18([
                  / protected / h'a10126' / {
                      / alg / 1:-7 / ES256 /,
                  } /,
                  / unprotected / {
                  },
                  / payload / h'82025820666b83ff516281903871704895355aa94
  41656d8a24401de6458595c42cb0165d' / [
                      / algorithm-id / 2 / sha256 /,
                      / digest-bytes /
  h'666b83ff516281903871704895355aa9441656d8a24401de6458595c42cb0165d'
                  ] /,
                  / signature / h'5cb310acb34f7ebb42acfffce430dbda94faa4
  12900ce8e76650445e2c37e4cc132d8bb5f30ecf5f8130270bbf8d159f6d36e1cdf97b
  64229910fdb447538af1'
              ])
      ] /,
      / manifest / 3:h'a50101020203585ea2024481814100045858614a40150fa6
  b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab4503820
  2582000112233445566778899aabbccddeeff0123456789abcdeffedcba98765432100
  e1987d001f602f60958258613a115781b687474703a2f2f6578616d706c652e636f6d2
  f66696c652e62696e15f603f60a438203f6' / {
          / manifest-version / 1:1,
          / manifest-sequence-number / 2:2,
          / common / 3:h'a2024481814100045858614a40150fa6b4a53d5ad5fdfb
  e9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab45038202582000112233 4
  45566778899aabbccddeeff0123456789abcdeffedcba98765432100e1987d001f602f
  6' / {
              / components / 2:h'81814100' / [
                  [h'00']
              ] /,
              / common-sequence / 4:h'8614a40150fa6b4a53d5ad5fdfbe9de663
  e4d41ffe02501492af1425695e48bf429b2d51f2ab45038202582000112233445566 77
  8899aabbccddeeff0123456789abcdeffedcba98765432100e1987d001f602f6' / [
                  / directive-override-parameters / 20,{
                      / vendor-id /
  1:h'fa6b4a53d5ad5fdfbe9de663e4d41ffe' / fa6b4a53-d5ad-5fdf-
  be9d-e663e4d41ffe /,
                      / class-id / 2:h'1492af1425695e48bf429b2d51f2ab45'
  / 1492af14-2569-5e48-bf42-9b2d51f2ab45 /,
                      / image-digest / 3:[
                          / algorithm-id / 2 / sha256 /,
                          / digest-bytes /
  h'00112233445566778899aabbccddeeff0123456789abcdeffedcba9876543210'
                      ],
```

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

Total size of manifest without COSE authentication object: 151

Manifest:

a1035893a50101020203585ea20244818141000458548614a40150fa6b4a
53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab
45038202582000112233445566778899aabbccddeeff0123456789abcdef
fedcba98765432100e1987d001f602f60958258613a115781b687474703a
2f2f6578616d706c652e636f6d2f66696c652e62696e15f603f60a438203
f6

Total size of manifest with COSE authentication object: 266

Manifest with COSE authentication object:

a202587081d28443a10126a0582482025820666b83ff5162819038717048
9535aa9441656d8a24401de6458595c42cb0165d58405cb310acb34f7ebb
42acfffce430dbda94faa412900ce8e76650445e2c37e4cc132d8bb5f30e
cf5f8130270bbf8d159f6d36e1cdf97b64229910fdb447538af1035893a5
0101020203585ea20244818141000458548614a40150fa6b4a53d5ad5fdf
be9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab4503820258
2000112233445566778899aabbccddeeff0123456789abcdeffedcba9876
5432100e1987d001f602f60958258613a115781b687474703a2f2f657861
6d706c652e636f6d2f66696c652e62696e15f603f60a438203f6

12.3. Example 2: Simultaneous Download, Installation, and Secure Boot

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

```
{
    / authentication-wrapper / 2:h'81d28443a10126a058248202582038df852
c98928fae9694fce5b6b51addd631bfde473eceb20c8b929ae6ec2d6c584050bba3dd9
b0ad6da91265cff1ec69c3a9e2e42ffd97e780e37c78ac78891406204398741 08ec527
1f3325988f2774f17339fcd61a5c08a3d15fb7fcdeef9294e' / [
        18([
                / protected / h'a10126' / {
                    / alg / 1:-7 / ES256 /,
                } /,
                / unprotected / {
                },
                / payload / h'8202582038df852c98928fae9694fce5b6b51add
d631bfde473eceb20c8b929ae6ec2d6c' / [
                    / algorithm-id / 2 / sha256 /,
                    / digest-bytes /
h'38df852c98928fae9694fce5b6b51addd631bfde473eceb20c8b929ae6ec2d6c'
                ] /,
                / signature / h'50bba3dd9b0ad6da91265cff1ec69c3a9e2e42
ffd97e780e37c78ac7889140620439874108ec5271f3325988f2774f17339fcd61a5c0
8a3d15fb7fcdeef9294e'
            ])
    ] /,
    / manifest / 3:h'a60101020303585ea2024481814100045854861 4a40150fa6
b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab4503820
2582000112233445566778899aabbccddeeff0123456789abcdeffedcba98765432100
e1987d001f602f60958258613a115781b687474703a2f2f6578616d706c652e636f6d2
f66696c652e62696e15f603f60a438203f60c438217f6' / {
        / manifest-version / 1:1,
        / manifest-sequence-number / 2:3,
        / common / 3:h'a2024481814100045854861 4a40150fa6b4a53d5ad5fdfb
e9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab45038202582000112233 4
45566778899aabbccddeeff0123456789abcdeffedcba98765432100e1987d001f602f
6' / {
            / components / 2:h'81814100' / [
                [h'00']
            ] /,
            / common-sequence / 4:h'8614a40150fa6b4a53d5ad5fdfbe9de663
e4d41ffe02501492af1425695e48bf429b2d51f2ab450382025820001122334455667 7
8899aabbccddeeff0123456789abcdeffedcba98765432100e1987d001f602f6' / [
                / directive-override-parameters / 20,{
                    / vendor-id /
1:h'fa6b4a53d5ad5fdfbe9de663e4d41ffe' / fa6b4a53-d5ad-5fdf-
be9d-e663e4d41ffe /,
```

```
                      / class-id / 2:h’1492af1425695e48bf429b2d51f2ab45’
  / 1492af14-2569-5e48-bf42-9b2d51f2ab45 /,
                      / image-digest / 3:[
                          / algorithm-id / 2 / sha256 /,
                          / digest-bytes /
  h’00112233445566778899aabbccddeeff0123456789abcdeffedcba9876543210’
                      ],
                      / image-size / 14:34768,
                  } ,
                  / condition-vendor-identifier / 1,F6 / nil / ,
                  / condition-class-identifier / 2,F6 / nil /
              ] /,
          } /,
          / install / 9:h’8613a115781b687474703a2f2f6578616d706c652e636f
  6d2f66696c652e62696e15f603f6’ / [
              / directive-set-parameters / 19,{
                  / uri / 21:’http://example.com/file.bin’,
              } ,
              / directive-fetch / 21,F6 / nil / ,
              / condition-image-match / 3,F6 / nil /
          ] /,
          / validate / 10:h’8203f6’ / [
              / condition-image-match / 3,F6 / nil /
          ] /,
          / run / 12:h’8217f6’ / [
              / directive-run / 23,F6 / nil /
          ] /,
      } /,
  }
```

Total size of manifest without COSE authentication object: 156

Manifest:

a1035898a60101020303585ea20244818141000458548614a40150fa6b4a
53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab
45038202582000112233445566778899aabbccddeeff0123456789abcdef
fedcba98765432100e1987d001f602f60958258613a115781b687474703a
2f2f6578616d706c652e636f6d2f66696c652e62696e15f603f60a438203
f60c438217f6

Total size of manifest with COSE authentication object: 271

Manifest with COSE authentication object:

```
a202587081d28443a10126a058248202582038df852c98928fae9694fce5
b6b51addd631bfde473eceb20c8b929ae6ec2d6c584050bba3dd9b0ad6da
91265cff1ec69c3a9e2e42ffd97e780e37c78ac788914062043987410 8ec
5271f3325988f2774f17339fcd61a5c08a3d15fb7fcdeef9294e035898a6
0101020303585ea20244818141000458548614a40150fa6b4a53d5ad5fdf
be9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab4503820258
2000112233445566778899aabbccddeeff0123456789abcdeffedcba9876
5432100e1987d001f602f60958258613a115781b687474703a2f2f657861
6d706c652e636f6d2f66696c652e62696e15f603f60a438203f60c438217
f6
```


12.4. Example 3: Load from External Storage

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

```
{
    / authentication-wrapper / 2:h'81d28443a10126a05824820258208ae1d4d
1846e82975dd5d7555ef0c3836e7e653a8bb1214466457781c0d2f2aa58401ef2d0ca6
aabf259feb880a1a4deb4e345cda314b2facf9983766da3744af825b3f98c74afdfa85
aed406b10315e0cc6c44ee19321681c69f911bc90bf8d22c0' / [
        18([
                / protected / h'a10126' / {
                    / alg / 1:-7 / ES256 /,
                } /,
                / unprotected / {
                },
                / payload / h'820258208ae1d4d1846e82975dd5d7555ef0c383
6e7e653a8bb1214466457781c0d2f2aa' / [
                    / algorithm-id / 2 / sha256 /,
                    / digest-bytes /
h'8ae1d4d1846e82975dd5d7555ef0c3836e7e653a8bb1214466457781c0d2f2aa'
                ] /,
                / signature / h'1ef2d0ca6aabf259feb880a1a4deb4e345cda3
14b2facf9983766da3744af825b3f98c74afdfa85aed406b10315e0cc6c44ee1932168
1c69f911bc90bf8d22c0'
            ])
    ] /,
    / manifest / 3:h'a701010204035863a2024782814100814101045856880c001
4a40150fa6b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f
2ab45038202582000112233445566778899aabbccddeeff0123456789abcdeffedcba9
8765432100e1987d001f602f6095827880c0013a115781b687474703a2f2f6578616d7
06c652e636f6d2f66696c652e62696e15f603f60a45840c0003f60b4b880c0113a1160
016f603f60c45840c0117f6' / {
        / manifest-version / 1:1,
        / manifest-sequence-number / 2:4,
        / common / 3:h'a2024782814100814101045856880c0014a40150fa6b4a5
3d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab45038202582
```

```
  00011223344556677889 9aabbccddeeff0123456789abcdeffedcba98765432100e198
  7d001f602f6' / {
              / components / 2:h'82814100814101' / [
                  [h'00'] ,
                  [h'01']
              ] /,
              / common-sequence / 4:h'880c0014a40150fa6b4a53d5ad5fdfbe9d
  e663e4d41ffe02501492af1425695e48bf429b2d51f2ab45038202582000112233445 5
  66778899aabbccddeeff0123456789abcdeffedcba98765432100e1987d001f602f6'
  / [
                  / directive-set-component-index / 12,0 ,
                  / directive-override-parameters / 20,{
                      / vendor-id /
  1:h'fa6b4a53d5ad5fdfbe9de663e4d41ffe' / fa6b4a53-d5ad-5fdf-
  be9d-e663e4d41ffe /,
                      / class-id / 2:h'1492af1425695e48bf429b2d51f2ab45'
  / 1492af14-2569-5e48-bf42-9b2d51f2ab45 /,
                      / image-digest / 3:[
                          / algorithm-id / 2 / sha256 /,
                          / digest-bytes /
  h'00112233445566778899aabbccddeeff0123456789abcdeffedcba9876543210'
                      ],
                      / image-size / 14:34768,
                  } ,
                  / condition-vendor-identifier / 1,F6 / nil / ,
                  / condition-class-identifier / 2,F6 / nil /
              ] /,
          } /,
          / install / 9:h'880c0013a115781b687474703a2f2f6578616d706c652e
  636f6d2f66696c652e62696e15f603f6' / [
              / directive-set-component-index / 12,0 ,
              / directive-set-parameters / 19,{
                  / uri / 21:'http://example.com/file.bin',
              } ,
              / directive-fetch / 21,F6 / nil / ,
              / condition-image-match / 3,F6 / nil /
          ] /,
          / validate / 10:h'840c0003f6' / [
              / directive-set-component-index / 12,0 ,
              / condition-image-match / 3,F6 / nil /
          ] /,
          / load / 11:h'880c0113a1160016f603f6' / [
              / directive-set-component-index / 12,1 ,
              / directive-set-parameters / 19,{
                  / source-component / 22:0 / [h'00'] /,
              } ,
              / directive-copy / 22,F6 / nil / ,
              / condition-image-match / 3,F6 / nil /
```

```
          ] /,
          / run / 12:h'840c0117f6' / [
              / directive-set-component-index / 12,1 ,
              / directive-run / 23,F6 / nil /
          ] /,
      } /,
  }
```

Total size of manifest without COSE authentication object: 180

Manifest:

a10358b0a7010102040358 63a2024782814100814101045856880c0014a4
0150fa6b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf42
9b2d51f2ab45038202582000112233445566778899aabbccddeeff012345
6789abcdeffedcba98765432100e1987d001f602f6095827880c0013a115
781b687474703a2f2f6578616d706c652e636f6d2f66696c652e62696e15
f603f60a45840c0003f60b4b880c0113a1160016f603f60c45840c0117f6

Total size of manifest with COSE authentication object: 295

Manifest with COSE authentication object:

a202587081d28443a10126a05824820258208ae1d4d1846e82975dd5d755
5ef0c3836e7e653a8bb1214466457781c0d2f2aa58401ef2d0ca6aabf259
feb880a1a4deb4e345cda314b2facf9983766da3744af825b3f98c74afdf
a85aed406b10315e0cc6c44ee19321681c69f911bc90bf8d22c00358b0a7
010102040358 63a2024782814100814101045856880c0014a40150fa6b4a
53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab
45038202582000112233445566778899aabbccddeeff0123456789abcdef
fedcba98765432100e1987d001f602f6095827880c0013a115781b687474
703a2f2f6578616d706c652e636f6d2f66696c652e62696e15f603f60a45
840c0003f60b4b880c0113a1160016f603f60c45840c0117f6

12.5. Example 4: Load and Decompress from External Storage

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

```
  {
      / authentication-wrapper / 2:h'81d28443a10126a0582482025820310798d
  3d8276a740505d1f017972e281d6d26c9967a658879ae6d07e6a238a958404d48f0059
  918c261bc1636b467b2b455801c4d211758a42e82a8f8fc245f21857d7c0e78f1b6d6a
  8ab1f0c9e147043066c0af53c1563070d4934faeec21bac55' / [
          18([
                  / protected / h'a10126' / {
                      / alg / 1:-7 / ES256 /,
                  } /,
```

```
                / unprotected / {
                },
                / payload / h'82025820310798d3d8276a740505d1f017972e28
  1d6d26c9967a658879ae6d07e6a238a9' / [
                    / algorithm-id / 2 / sha256 /,
                    / digest-bytes /
  h'310798d3d8276a740505d1f017972e281d6d26c9967a658879ae6d07e6a238a9'
                ] /,
                / signature / h'4d48f0059918c261bc1636b467b2b455801c4d
  211758a42e82a8f8fc245f21857d7c0e78f1b6d6a8ab1f0c9e147043066c0af53c1563
  070d4934faeec21bac55'
            ])
    ] /,
    / manifest / 3:h'a701010205035863a2024782814100814101045856880c001
  4a40150fa6b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f
  2ab45038202582000112233445566778899aabbccddeeff0123456789abcdeffedcba9
  8765432100e1987d001f602f6095827880c0013a115781b687474703a2f2f6578616d7
  06c652e636f6d2f66696c652e62696e15f603f60a45840c0003f60b4d880c0113a2130
  1160016f603f60c45840c0117f6' / {
        / manifest-version / 1:1,
        / manifest-sequence-number / 2:5,
        / common / 3:h'a2024782814100814101045856880c0014a40150fa6b4a5
  3d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab45038202582
  000112233445566778899aabbccddeeff0123456789abcdeffedcba98765432100e198
  7d001f602f6' / {
            / components / 2:h'82814100814101' / [
                [h'00'] ,
                [h'01']
            ] /,
            / common-sequence / 4:h'880c0014a40150fa6b4a53d5ad5fdfbe9d
  e663e4d41ffe02501492af1425695e48bf429b2d51f2ab450382025820001122334455
  66778899aabbccddeeff0123456789abcdeffedcba98765432100e1987d001f602f6'
  / [
                / directive-set-component-index / 12,0 ,
                / directive-override-parameters / 20,{
                    / vendor-id /
  1:h'fa6b4a53d5ad5fdfbe9de663e4d41ffe' / fa6b4a53-d5ad-5fdf-
  be9d-e663e4d41ffe /,
                    / class-id / 2:h'1492af1425695e48bf429b2d51f2ab45'
  / 1492af14-2569-5e48-bf42-9b2d51f2ab45 /,
                    / image-digest / 3:[
                        / algorithm-id / 2 / sha256 /,
                        / digest-bytes /
  h'00112233445566778899aabbccddeeff0123456789abcdeffedcba9876543210'
                    ],
                    / image-size / 14:34768,
                } ,
                / condition-vendor-identifier / 1,F6 / nil / ,
```

```
                / condition-class-identifier / 2,F6 / nil /
            ] /,
        } /,
        / install / 9:h'880c0013a115781b687474703a2f2f6578616d706c652e
  636f6d2f66696c652e62696e15f603f6' / [
            / directive-set-component-index / 12,0 ,
            / directive-set-parameters / 19,{
                / uri / 21:'http://example.com/file.bin',
            } ,
            / directive-fetch / 21,F6 / nil / ,
            / condition-image-match / 3,F6 / nil /
        ] /,
        / validate / 10:h'840c0003f6' / [
            / directive-set-component-index / 12,0 ,
            / condition-image-match / 3,F6 / nil /
        ] /,
        / load / 11:h'880c0113a21301160016f603f6' / [
            / directive-set-component-index / 12,1 ,
            / directive-set-parameters / 19,{
                / source-component / 22:0 / [h'00'] /,
                / compression-info / 19:1 / gzip /,
            } ,
            / directive-copy / 22,F6 / nil / ,
            / condition-image-match / 3,F6 / nil /
        ] /,
        / run / 12:h'840c0117f6' / [
            / directive-set-component-index / 12,1 ,
            / directive-run / 23,F6 / nil /
        ] /,
    } /,
}
```

Total size of manifest without COSE authentication object: 182

Manifest:

a10358b2a701010205035863a2024782814100814101045856880c0014a4
0150fa6b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf42
9b2d51f2ab45038202582000112233445566778899aabbccddeeff012345
6789abcdeffedcba98765432100e1987d001f602f6095827880c0013a115
781b687474703a2f2f6578616d706c652e636f6d2f66696c652e62696e15
f603f60a45840c0003f60b4d880c0113a21301160016f603f60c45840c01
17f6

Total size of manifest with COSE authentication object: 297

Manifest with COSE authentication object:

```
a202587081d28443a10126a0582482025820310798d3d8276a740505d1f0
```


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

Compatibility test, download, installation, and secure boot.

```
{
    / authentication-wrapper / 2:h'81d28443a10126a05824820258209a45659
58c6e09c92fc69feeb09081c875f113082245ba2025801fa46dc2280e58404604e6413
30d610fd0a0545b9b816f09c0767edf66fc57f40393cd4423e0807b36226e843e0f57b
f860a3cf542655048648dea81e62e39f19e7ac96652d3de90' / [
        18([
                / protected / h'a10126' / {
                    / alg / 1:-7 / ES256 /,
                } /,
                / unprotected / {
                },
                / payload / h'820258209a4565958c6e09c92fc69feeb09081c8
75f113082245ba2025801fa46dc2280e' / [
                    / algorithm-id / 2 / sha256 /,
                    / digest-bytes /
h'9a4565958c6e09c92fc69feeb09081c875f113082245ba2025801fa46dc2280e'
                ] /,
                / signature / h'4604e641330d610fd0a0545b9b816f09c0767e
df66fc57f40393cd4423e0807b36226e843e0f57bf860a3cf542655048648dea81e62e
39f19e7ac96652d3de90'
            ])
    ] /,
    / manifest / 3:h'a701010205035863a2024782814101814100045856880c011
4a40150fa6b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f
2ab45038202582000112233445566778899aabbccddeeff0123456789abcdeffedcba9
8765432100e1987d001f602f6085823840c0013a115781b687474703a2f2f6578616d7
06c652e636f6d2f66696c652e62696e094b880c0113a1160016f603f60a45840c0103f
60c45840c0117f6' / {
        / manifest-version / 1:1,
        / manifest-sequence-number / 2:5,
        / common / 3:h'a2024782814101814100045856880c0114a40150fa6b4a5
3d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab45038202582
```

```
  00112233445566778899aabbccddeeff0123456789abcdeffedcba98765432100e198
  7d001f602f6' / {
              / components / 2:h'82814101814100' / [
                  [h'01'] ,
                  [h'00']
              ] /,
              / common-sequence / 4:h'880c0114a40150fa6b4a53d5ad5fdfbe9d
  e663e4d41ffe02501492af1425695e48bf429b2d51f2ab45038202582000112233445
  566778899aabbccddeeff0123456789abcdeffedcba98765432100e1987d001f602f6'
  / [
                  / directive-set-component-index / 12,1 ,
                  / directive-override-parameters / 20,{
                      / vendor-id /
  1:h'fa6b4a53d5ad5fdfbe9de663e4d41ffe' / fa6b4a53-d5ad-5fdf-
  be9d-e663e4d41ffe /,
                      / class-id / 2:h'1492af1425695e48bf429b2d51f2ab45'
  / 1492af14-2569-5e48-bf42-9b2d51f2ab45 /,
                      / image-digest / 3:[
                          / algorithm-id / 2 / sha256 /,
                          / digest-bytes /
  h'00112233445566778899aabbccddeeff0123456789abcdeffedcba9876543210'
                      ],
                      / image-size / 14:34768,
                  } ,
                  / condition-vendor-identifier / 1,F6 / nil / ,
                  / condition-class-identifier / 2,F6 / nil /
              ] /,
          } /,
          / payload-fetch / 8:h'840c0013a115781b687474703a2f2f6578616d70
  6c652e636f6d2f66696c652e62696e' / [
              / directive-set-component-index / 12,0 ,
              / directive-set-parameters / 19,{
                  / uri / 21:'http://example.com/file.bin',
              }
          ] /,
          / install / 9:h'880c0113a1160016f603f6' / [
              / directive-set-component-index / 12,1 ,
              / directive-set-parameters / 19,{
                  / source-component / 22:0 / [h'01'] /,
              } ,
              / directive-copy / 22,F6 / nil / ,
              / condition-image-match / 3,F6 / nil /
          ] /,
          / validate / 10:h'840c0103f6' / [
              / directive-set-component-index / 12,1 ,
              / condition-image-match / 3,F6 / nil /
          ] /,
          / run / 12:h'840c0117f6' / [
```

```
              / directive-set-component-index / 12,1 ,
              / directive-run / 23,F6 / nil /
          ] /,
      } /,
  }
```

Total size of manifest without COSE authentication object: 176

Manifest:

a10358aca701010205035863a2024782814101814100045856880c0114a4
0150fa6b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf42
9b2d51f2ab45038202582000112233445566778899aabbccddeeff012345
6789abcdeffedcba98765432100e1987d001f602f6085823840c0013a115
781b687474703a2f2f6578616d706c652e636f6d2f66696c652e62696e09
4b880c0113a1160016f603f60a45840c0103f60c45840c0117f6

Total size of manifest with COSE authentication object: 291

Manifest with COSE authentication object:

a202587081d28443a10126a05824820258209a4565958c6e09c92fc69fee
b09081c875f113082245ba2025801fa46dc2280e58404604e641330d610f
d0a0545b9b816f09c0767edf66fc57f40393cd4423e0807b36226e843e0f
57bf860a3cf542655048648dea81e62e39f19e7ac96652d3de900358aca7
01010205035863a2024782814101814100045856880c0114a40150fa6b4a
53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab
45038202582000112233445566778899aabbccddeeff0123456789abcdef
fedcba98765432100e1987d001f602f6085823840c0013a115781b687474
703a2f2f6578616d706c652e636f6d2f66696c652e62696e094b880c0113
a1160016f603f60a45840c0103f60c45840c0117f6

12.7. Example 6: Two Images

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

```
  {
      / authentication-wrapper / 2:h’81d28443a10126a05824820258201d15a17
  13d3a4510ca392454adff987abb5425348e449618122ffa817012cc315840197a4a3a4
  188fe1dd8baa468ae9a35ac8e5ef462017530116eadd90892c96c6ab00825fcb45edb7
  57547733c14d3b637ea8a085ce7bfc782a0b2cd80d31b1294’ / [
          18([
                  / protected / h’a10126’ / {
                      / alg / 1:-7 / ES256 /,
                  } /,
                  / unprotected / {
                  },
```

```
                / payload / h’820258201d15a1713d3a4510ca392454adff987a
  bb5425348e449618122ffa817012cc31’ / [
                    / algorithm-id / 2 / sha256 /,
                    / digest-bytes /
  h’1d15a1713d3a4510ca392454adff987abb5425348e449618122ffa817012cc31’
                ] /,
                / signature / h’197a4a3a4188fe1dd8baa468ae9a35ac8e5ef4
  62017530116eadd90892c96c6ab00825fcb45edb757547733c14d3b637ea8a085ce7bf
  c782a0b2cd80d31b1294’
            ])
    ] /,
    / manifest / 3:h’a5010102030358 99a2024481814100045 88f8814a20150fa6
  b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab450f825
  82e8405f614a20382025820001122334455667788 99aabbccddeeff0123456789abcde
  ffedcba98765432100e1987d058308405f614a20382025820 0123456789abcdeffedcb
  a98765432100011223344556677889 9aabbccddeeff0e1a00012c2201f602f60958538
  60f8258248405f613a115781c687474703a2f2f6578616d706c652e636f6d2f66696c6
  5312e62696e58248405f613a115781c687474703a2f2f6578616d706c652e636f6d2f6
  6696c65322e62696e15f603f60a438203f6’ / {
        / manifest-version / 1:1,
        / manifest-sequence-number / 2:3,
        / common / 3:h’a2024481814100045 88f8814a20150fa6b4a53d5ad5fdfb
  e9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab450f82582e8405f614a20
  3820258200011223344556677889 9aabbccddeeff0123456789abcdeffedcba9876543
  2100e1987d058308405f614a20382025820 0123456789abcdeffedcba9876543210001
  1223344556677889 9aabbccddeeff0e1a00012c2201f602f6’ / {
            / components / 2:h’81814100’ / [
                [h’00’]
            ] /,
            / common-sequence / 4:h’8814a20150fa6b4a53d5ad5fdfbe9de663
  e4d41ffe02501492af1425695e48bf429b2d51f2ab450f82582e8405f614a203820258
  2000112233445566778899aabbccddeeff0123456789abcdeffedcba98765432100e19
  87d058308405f614a20382025820 0123456789abcdeffedcba98765432100011223344
  5566778899aabbccddeeff0e1a00012c2201f602f6’ / [
                / directive-override-parameters / 20,{
                    / vendor-id /
  1:h’fa6b4a53d5ad5fdfbe9de663e4d41ffe’ / fa6b4a53-d5ad-5fdf-
  be9d-e663e4d41ffe /,
                    / class-id / 2:h’1492af1425695e48bf429b2d51f2ab45’
  / 1492af14-2569-5e48-bf42-9b2d51f2ab45 /,
                } ,
                / directive-try-each / 15,[
                    h’8405f614a20382025820001122334455667788 99aabbccdd
  eeff0123456789abcdeffedcba98765432100e1987d0’ / [
                        / condition-component-offset / 5,F6 / nil / ,
                        / directive-override-parameters / 20,{
                            / image-digest / 3:[
                                / algorithm-id / 2 / sha256 /,
```

```
                                / digest-bytes /
  h'00112233445566778899aabbccddeeff0123456789abcdeffedcba9876543210'
                            ],
                            / image-size / 14:34768,
                        }
                    ] / ,
                    h'8405f614a20382025820012345678 9abcdeffedcba987654
  3210001122334455667788 99aabbccddeeff0e1a00012c22' / [
                        / condition-component-offset / 5,F6 / nil / ,
                        / directive-override-parameters / 20,{
                            / image-digest / 3:[
                                / algorithm-id / 2 / sha256 /,
                                / digest-bytes /
  h'0123456789abcdeffedcba987654321000112233445566778899aabbccddeeff'
                            ],
                            / image-size / 14:76834,
                        }
                    ] /
                ] ,
                / condition-vendor-identifier / 1,F6 / nil / ,
                / condition-class-identifier / 2,F6 / nil /
            ] /,
        } /,
        / install / 9:h'860f8258248405f613a115781c687474703a2f2f657861
  6d706c652e636f6d2f66696c65312e62696e58248405f613a115781c687474703a2f2f
  6578616d706c652e636f6d2f66696c65322e62696e15f603f6' / [
            / directive-try-each / 15,[
                h'8405f613a115781c687474703a2f2f6578616d706c652e636f6d
  2f66696c65312e62696e' / [
                    / condition-component-offset / 5,F6 / nil / ,
                    / directive-set-parameters / 19,{
                        / uri / 21:'http://example.com/file1.bin',
                    }
                ] / ,
                h'8405f613a115781c687474703a2f2f6578616d706c652e636f6d
  2f66696c65322e62696e' / [
                    / condition-component-offset / 5,F6 / nil / ,
                    / directive-set-parameters / 19,{
                        / uri / 21:'http://example.com/file2.bin',
                    }
                ] /
            ] ,
            / directive-fetch / 21,F6 / nil / ,
            / condition-image-match / 3,F6 / nil /
        ] /,
        / validate / 10:h'8203f6' / [
            / condition-image-match / 3,F6 / nil /
        ] /,
```

```
      } /,
  }
```

Total size of manifest without COSE authentication object: 256

Manifest:

a10358fca501010203035899a2024481814100045 88f8814a20150fa6b4a
53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab
450f82582e8405f614a2038202582000112233445566778899aabbccddee
ff0123456789abcdeffedcba98765432100e1987d058308405f614a20382
0258200123456789abcdeffedcba987654321000112233445566778899aa
bbccddeeff0e1a00012c2201f602f6095853860f8258248405f613a11578
1c687474703a2f2f6578616d706c652e636f6d2f66696c65312e62696e58
248405f613a115781c687474703a2f2f6578616d706c652e636f6d2f6669
6c65322e62696e15f603f60a438203f6

Total size of manifest with COSE authentication object: 371

Manifest with COSE authentication object:

a202587081d28443a10126a05824820258201d15a1713d3a4510ca392454
adff987abb5425348e449618122ffa817012cc315840197a4a3a4188fe1d
d8baa468ae9a35ac8e5ef462017530116eadd90892c96c6ab00825fcb45e
db757547733c14d3b637ea8a085ce7bfc782a0b2cd80d31b12940358fca5
01010203035899a202448181410004588f8814a20150fa6b4a53d5ad5fdf
be9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab450f82582e
8405f614a2038202582000112233445566778899aabbccddeeff01234567
89abcdeffedcba98765432100e1987d058308405f614a203820258200123
456789abcdeffedcba987654321000112233445566778899aabbccddeeff
0e1a00012c2201f602f6095853860f8258248405f613a115781c68747470
3a2f2f6578616d706c652e636f6d2f66696c65312e62696e58248405f613
a115781c687474703a2f2f6578616d706c652e636f6d2f66696c65322e62
696e15f603f60a438203f6

13. IANA Considerations

IANA is requested to setup a registry group for SUIT elements.

Within this group, IANA is requested to setup registries for SUIT keys:

- SUIT Envelope Elements

- SUIT Manifest Elements

- SUIT Common Elements

- SUIT Commands

- SUIT Parameters

- SUIT Text Values

- SUIT Algorithm Identifiers

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

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

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., Tschofenig, H., Brown, D., and M. Meriac, "A Firmware Update Architecture for Internet of Things", draft-ietf-suit-architecture-08 (work in progress), November 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-05 (work in progress), January 2020.

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

Koen Zandberg
Inria

EMail: koen.zandberg@inria.fr