SUIT
Internet-Draft
Intended status: Standards Track
Expires: January 14, 2021

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

July 13, 2020

A Concise Binary Object Representation (CBOR)-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest
draft-ietf-suit-manifest-09

Abstract

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

Commented [DT1]: This abstract implies that the manifest format can *only* support firmware, and *only* for IoT devices. As such, readers that see the abstract may never read the doc to find that this isn't true. Suggest updating the language here so that the last paragraph of section 1 doesn't contradict this.

E.g., this language implies that TEEP cannot use it

Status of This Memo

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

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

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

This Internet-Draft will expire on January 14, 2021.

Copyright Notice

Table of Contents

1. Introduction

A firmware update mechanism is an essential security feature for IoT devices to deal with vulnerabilities. While the transport of firmware images to the devices themselves is important, there are already various techniques available. Equally important is the inclusion of metadata about the conveyed firmware image (in the form of a manifest) and the use of a security wrapper to provide end-to-end security protection to detect modifications and (optionally) to make reverse engineering more difficult. End-to-end security allows the author, who builds the firmware image, to be sure that no other party (including potential adversaries) can install firmware updates on IoT devices without adequate privileges. For confidentiality protected firmware images it is additionally required to encrypt the firmware image. Starting security protection at the author is a risk mitigation technique so firmware images and manifests can be stored on untrusted repositories; it also reduces the scope of a compromise of any repository or intermediate system to be no worse than a denial of service.

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

Commented [DT2]: Same comment here as on the abstract

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

- Meet the requirements defined in [I-D.ietf-suit-information-model].
- Simple to parse on a constrained node
- Simple to process on a constrained node
- Compact encoding
- Comprehensible by an intermediate system

- Expressive enough to enable advanced use cases on advanced nodes
- Extensible

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

- ~~the~~ a Firmware Author to reason about releasing a firmware.
- ~~the~~ a Network Operator to reason about compatibility of a firmware.
- ~~the~~ a Device Operator to reason about the impact of a firmware.
- ~~the~~ a Device Operator to manage distribution of firmware to devices.
- ~~the~~ a Plant Manager to reason about timing and acceptance of firmware updates.
- ~~the~~ a device to reason about the authority & authenticity of a firmware prior to installation.
- ~~the~~ a device to reason about the applicability of a firmware.
- ~~the~~ a device to reason about the installation of a firmware.
- ~~the~~ a device to reason about the authenticity & encoding of a firmware at boot.

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

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

The design of this specification is based on an observation that the vast majority of operations that a device can perform during an update or secure boot are composed of a small group of operations:

- Copy some data from one place to another
- Transform some data
- Digest some data and compare to an expected value
- Compare some system parameters to an expected value

- Run some code

In ~~the SUIT manifest specification~~this document, these operations are called commands. Commands are classed as either conditions or directives. Conditions have no side-effects, while directives do have side-effects. Conceptually, a sequence of commands is like a script but the used language is tailored to software updates and secure boot.

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

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

2. Conventions and Terminology

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

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

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

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

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

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

Commented [DT3]: Same comment here as on the abstract

Also it looks odd for Firmware to be capitalized in soe definitions (like in Payload above) and un-capitalized in others, like here. Other terms, like “manifest” also appear un-capitalized in many places (e.g., definition of Update and Recipient, below), and capitalized in other places (e.g., definition of Manifest Processor). Same for “envelope” and “Envelope”, etc. My preference is un-capitalized, as I find it much easier to read.

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

- Update: One or more manifests that describe one or more payloads.

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

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

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

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

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

- Command: A Condition or a Directive.

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

- Directive: An action for the Recipient to perform.

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

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

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

- Report: A list of Records.

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

- Update Procedure: A procedure that updates a Recipient by fetching dependencies~~, software~~and images, and installing them.

- Boot Procedure: A procedure that boots a Recipient by verifying dependencies and images, loading images, and invoking one or more image.

Commented [DT4]: Undefined term, suggest providing a forward reference to a later section. (Also note that later sections use “severable” not “severed”)

Commented [DT5]: Bad grammar since the other parts of the AND have “fetching” and “installing” and there’s no verb here.
Does this mean “fetching software images”? And is the “them” at the end referring to both dependencies AND software images? See suggested fix, for symmetry with the wording of Boot Procedure immediately below.

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

- Firmware: Instructions and data that allow a Recipient to perform a useful function. Typically, changed infrequently, stored in nonvolatile memory, and small enough to apply to [RFC7228] Class 0-2 devices.

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

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

- Abort: The Manifest Processor immediately halts execution of the current Procedure. It creates a Record of an error condition.

3. How to use this Document

This specification covers five aspects of firmware update:

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

- Section 5 gives a general overview of the metadata structure to inform the following sections.

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

- Section 7 describes the process of creating a Manifest.

- Section 8 specifies the content of the Envelope and the Manifest.

To implement an updatable device, see Section 6 and Section 8. To implement a tool that generates updates, see Section 7 and Section 8.

The IANA consideration section~~,~~ ~~see~~ (Section 11)~~,~~ provides instructions to IANA to create several registries. This section also provides the CBOR labels for the structures defined in this document.

The complete CDDL description is provided in [full-cddl], examples are given in [examples] and a design rational is offered in [design-rationale]. Finally, [implementation-matrix] gives a summary~~ize~~ of the mandatory-to-implement features of this specification.

Commented [DT6]: Rather than duplicating the definition of Software, making it look like a cut-and-paste error, can we just say "**Software** that is typically changed infrequently, stored in …"

Commented [DT7]: Bad grammar. All the other definitions sound ok if you say "A <term> is <definition>" but this one doesn't.

Commented [DT8]: Undefined reference. I'm guessing you mean Appendix A.

Commented [DT9]: More undefined references…

4. Background

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

Commented [DT10]: I'd prefer "software" here, since the term above is SUIT not FUIT, and this sentence equally applies to TEEP.

4.1. IoT Firmware Update Constraints

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

Commented [DT11]: typo

- limited processing power and storage may require a simple representation of metadata.
- bandwidth constraints may require firmware compression or partial update support.
- bootloader complexity constraints may require simple selection between two bootable images.
- small internal storage may require external storage support.
- multiple microcontrollers may require coordinated update of all applications.
- large storage and complex functionality may require parallel update of many software components.
- extra information may need to be conveyed in the manifest in the earlier stages of the device lifecycle before those data items are stripped when the manifest is delivered~~dy~~ to a constrained device.

Commented [DT12]: typo

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. SUIT Workflow Model

There are several fundamental assumptions that inform the model of Update Procedure 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 Boot Procedure 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 Recipient installing an update can be broken down into five steps:

1. Verify the signature of the manifest.
2. Verify the applicability of the manifest.
3. Resolve dependencies.
4. Fetch payload(s).
5. Install payload(s).

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

1. Verify image(s).
2. Load image(s).
3. Run image(s).

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

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

5. Metadata Structure Overview

This section provides a high level overview of the manifest structure. The full description of the manifest structure is in Section 8.6.

The manifest is structured from several key components:

1. The Envelope (see Section 5.1) contains Delegation Chains, the Authentication Block, the Manifest, any Severable Elements, and any Integrated Payloads or Dependencies.

2. Delegation Chains (see Section 5.2) allow a Recipient to work from one of its Trust Anchors to an authority of the Authentication Block.

3. The Authentication Block (see Section 5.3) contains a list of signatures or MACs of the manifest.

4. The Manifest (see Section 5.4) contains all critical, non-severable metadata that the Recipient requires. It is further broken down into:

    1. Critical metadata, such as sequence number.

    2. Common metadata, including lists of dependencies and affected components.

    3. Command sequences, directing the Recipient how to install and use the payload(s).

    4. Integrity check values for severable fields.

5. Severable fields (see Section 5.5).

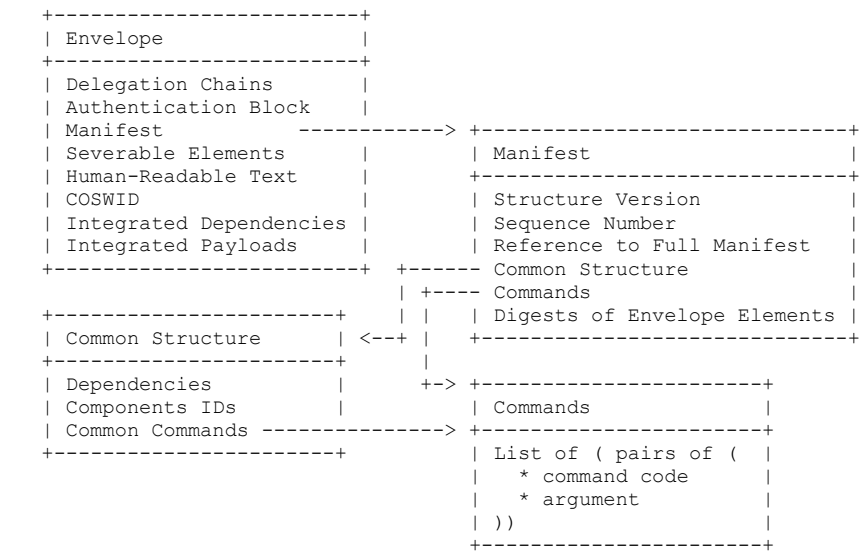
6. Integrated dependencies (see Section 5.6).

7. Integrated payloads (see Section 5.6).

Commented [DT13]: expand acronym per RFC style guide (since no asterisk next to the term in https://www.rfc-editor.org/materials/abbrev.expansion.txt)

Commented [DT14]: Section 5.5 is severable "elements" not "fields"

The diagram below illustrates the hierarchy of the Envelope.



5.1. Envelope

The SUIT Envelope is a container that encloses Delegation Chains, the Authentication Block, the Manifest, any Severable Elements, and any integrated payloads or dependencies. The Envelope is used instead of conventional cryptographic envelopes, such as COSE_Envelope because it allows modular processing, severing of elements, and integrated payloads in a way that would add substantial complexity with existing solutions. See Appendix C.1 for ~~a description of~~ the reasoning for this.

See Section 8.2 for more detail.

5.2. Delegation Chains

Delegation Chains allow a Recipient to validate intermediate Update Authorities against long-term a Trust Anchor. These are lists of CWTs, where the first in the list is signed by a Trust Anchor.

See Section 8.3 for more detail.

Commented [DT15]: Can't parse grammar, not sure what was intended

Commented [DT16]: First use of acronym. Expand acronym and add reference.

5.3. Authentication Block

The Authentication Block contains one or more COSE authentication blocks. These blocks are one of:

- COSE_Sign_Tagged
- COSE_Sign1_Tagged
- COSE_Mac_Tagged
- COSE_Mac0_Tagged

The payload element in each of these COSE elements is a SUIT_Digest Section 10.

See Section 8.4 for more detail.

5.4. Manifest

The Manifest contains most metadata about one or more images. The Manifest is divided into Critical Metadata, Common Metadata, Command Sequences, and Integrity Check Values.

See Section 8.6 for more detail.

5.4.1. Critical Metadata

Some metadata needs to be accessed before the manifest is processed. This metadata can be used to determine which the newest manifest is and whether the structure version is supported. It also MAY provide a URI for obtaining a canonical copy of the manifest and Envelope.

See Section 8.6.1, Section 8.6.2, and Section 8.6.3 for more detail.

5.4.2. Common

Some metadata is used repeatedly and in more than one command sequence. In order to reduce the size of the manifest, this metadata is collected into the Common section. Common is composed of three parts: a list of dependencies, a list of components referenced by the manifest, and a command sequence to execute prior to each other command sequence. The common command sequence is typically used to set commonly used values and perform compatibility checks. The common command sequence MUST NOT have any side-effects outside of setting parameter values.

See Section 8.7.2 and~~,~~ Section 8.7.2.1 for more detail.

Commented [DT17]: First use of acronym. Expand and add reference.

Commented [DT18]: … “as defined in”?

Commented [DT19]: Can’t parse grammar.
Do you mean “which manifest is newest”?

5.4.3. Command Sequences

Command sequences provide the instructions that a Recipient requires in order to install or use an image. These sequences tell a device to set parameter values, test system parameters, copy data from one place to another, transform data, digest data, and run code.

Command sequences are broken up into three groups: Common Command Sequence (see Section 5.4.2), update commands, and secure boot commands.

Update Command Sequences are: Dependency Resolution, Payload Fetch, and Payload Installation. An Update Procedure is the complete set of each Update Command Sequence, each preceded by the Common Command Sequence.

Boot Command Sequences are: System Validation, Image Loading, and Image Invocation. A Boot Procedure is the complete set of each Boot Command Sequence, each preceded by the Common Command Sequence.

Command Sequences are grouped into these sets to ensure that there is common coordination between dependencies and dependents on when to execute each command.

See Section 8.7.3 for more detail.

5.4.4. Integrity Check Values

To enable Section 5.5, there needs to be a mechanism to verify integrity of any metadata outside the manifest. Integrity Check Values are used to verify the integrity of metadata that is not contained in the manifest. This MAY include Severable Command Sequences, CoSWID, or Text data. Integrated Dependencies and Integrated Payloads are integrity-checked using Command Sequences, so they do not have Integrity Check Values present in the Manifest.

Commented [DT20]: Add reference

See Section 8.7.8 for more detail.

5.4.5. Human-Readable Text

Text is typically a Severable Element (Section 5.5). It contains all the text that describes the update. Because text is explicitly for human consumption, it is all grouped together so that it can be Severed easily. The text section has space both for describing the manifest as a whole and for describing each individual component.

See Section 8.6.4 for more detail.

5.5. Severable Elements

Severable Elements are elements of the Envelope (Section 5.1) that have Integrity Check Values (Section 5.4.4) in the Manifest (Section 5.4).

Because of this organisation, these elements can be discarded or "Severed" from the Envelope without changing the signature of the Manifest. This allows savings based on the size of the Envelope in several scenarios, for example:

- A management system sSevers the Text and CoSWID sections before sending an Envelope to a constrained Recipient, which saves Recipient bandwidth.

- A Recipient sSevers the Installation section after installing the Update, which saves storage space.

See Section 8.8 for more detail.

5.6. Integrated Dependencies and Payloads

In some cases, it is beneficial to include a dependency or a payload in the Envelope of a manifest. For example:

- When an update is delivered via a comparatively unconstrained medium, such as a removable mass storage device, it may be beneficial to bundle updates into single files.

- When a manifest requires encryption, it must be referenced as a dependency, so a trivial manifest may be used to enclose the encrypted manifest. The encrypted manifest may be contained in the dependent manifest's envelope.

- When a manifest transports a small payload, such as an encrypted key, that payload may be placed in the manifest's envelope.

See Section 7.8.1, Section 8.5 for more detail.

6. Interpreter Behavior

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

Commented [DT21]: Why was this capitalized?

Commented [DT22]: The terminology section defined a "Manifest Processor" but not an "Interpreter". What's the difference?

6.1. Interpreter Setup

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

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

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

- Test failed (e.g., Vendor ID/Class ID).
- Unsupported command encountered.
- Unsupported parameter encountered.
- Unsupported component ID encountered.
- Payload not available.
- Dependency not available.
- Application crashed when executed.
- Watchdog timeout occurred.
- Dependency or Payload verification failed.
- Missing component from a set.
- Required parameter not supplied.

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

Commented [DT23]: To confirm, so this means it MAY roll back to a version earlier than the last successful install, e.g., if the last successful install's manifest is unavailable or corrupted? (I'm especially thinking in a TEEP context)

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

6.2. Required Checks

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

When validating authenticity of manifests, the interpreter MAY use an ACL (see Section 9) to determine the extent of the rights conferred by that authenticity. Where a device supports only one level of access, it MAY choose to skip signature verification of dependencies, since they are referenced by digest. Where a device supports more than one trusted party, it MAY choose to defer the verification of signatures of dependencies until the list of affected components is known so that it can skip redundant signature verifications. For example, a dependency signed by the same author as the dependent does not require a signature verification. Similarly, if the signer of the dependent has full rights to the device, according to the ACL, then no signature verification is necessary on the dependency.

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

For each listed component, the interpreter MUST provide storage for the supported parameters. If the interpreter does not have sufficient temporary storage to process the parameters for all components, it MAY process components serially for each command sequence. See Section 6.6 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.

Commented [DT24]: I think I know what it means for a "component" to be supported, but what does it mean for a "component ID" to be supported?

Commented [DT25]: "Check Vendor Identifier command"?

Commented [DT26]: "Check Class Identifier command"?

Commented [DT27]: The table in 6.4 contains no such command.
Was this supposed to be "Set Component Index" maybe?

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.

If a Recipient supports groups of interdependent components (a Component Set), then it SHOULD require that all Components in the Component Set are specified by one manifest and its dependencies. This manifest is called the Root Manifest.

Commented [DT28]: Not sure how to parse this sentence. Does this mean some components can be in the dependencies? Reword to clarify.

6.2.1. Minimizing Signature Verifications

Signature verification can be energy and time expensive on a constrained device. MAC verification is typically unaffected by these concerns. A Recipient MAY choose to parse and execute only the SUIT_Common section of the manifest prior to signature verification, if all of the below apply:

- The Authentication Block contains a COSE_Sign_Tagged or COSE_Sign1_Tagged
- The Recipient can receive many incompatible or inapplicable manifests, and
- The Recipient has a power budget that makes signature verification Undesirable.

Commented [DT29]: Unclear. How does a Recipient know whether it can?

The guidelines in Creating Manifests (Section 7) require that the common section contains the applicability checks, so this section is sufficient for applicability verification. The parser MUST restrict acceptable commands to: Conditions, Override Parameters, Set Parameters, Try-Each, and Run Sequence ONLY. 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 Abort. The Common Sequence MUST be executed again in its entirety after authenticity validation.

Commented [DT30]: There is no Conditions command in the table in 6.4

Commented [DT31]: This is not hyphenated in the table in 6.4. Be consistent.

When executing Common prior to authenticity validation, the Manifest Processor MUST evaluate the integrity of the manifest using the SUIT_Digest present in the authentication block.

Alternatively, a Recipient MAY rely on network infrastructure to filter inapplicable manifests.

6.3. Interpreter Fundamental Properties

The interpreter has a small set of design goals:

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

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

3. Executing the same manifest on multiple Recipients 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.

Commented [DT32]: Suggest combining 1 & 2 into one bullet with "an update or secure boot", since as noted earlier the manifest can be used in other use cases, where keeping 2 bullets seems to imply you MUST use the manifest in secure boot if you use it in update, which should not be the case (e.g., for TEEP).

6.4. Abstract Machine Description

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

Commented [DT33]: The terminology section says they're processed by a Manifest Processor, not an interpreter.

There are two types of commands, namely those that modify state (directives) and those that perform tests (conditions). Parameters are used as the inputs to commands. Some directives offer control flow operations. Directives target a specific component or dependency. A dependency is another SUIT_Envelope that describes additional components. Dependencies are identified by digest, but referenced in commands by Dependency Index, the index into the array of Dependencies. A component is a unit of code or data that can be targeted by an update. Components are identified by Component Identifiers, i.e., arrays of binary strings, but referenced in commands by Component Index, the index into the array of Component Identifiers.

Commented [DT34]: Elsewhere mentions "Component ID". Is "ID" and "Identifier" the same? (I assume so). Pick one and be consistent throughout.

Conditions MUST NOT have any side-effects other than informing the interpreter of success or failure. The Interpreter does not Abort if the Soft Failure flag is set when a Condition reports failure.

Commented [DT35]: First mention of this term. Provide forward reference to a later section where this is defined.

Directives MAY have side-effects in the parameter table, the interpreter state, or the current component. The Interpreter MUST Abort if a Directive reports failure regardless of the Soft Failure flag.

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

Most commands operate on either a component or a dependency. Setting the Component Index clears the Dependency Index. Setting the Dependency Index clears the Component Index.

Command Name	Semantic of the Operation
Check Vendor Identifier	assert(binary-match(current, current.params[vendor-id]))
Check Class Identifier	assert(binary-match(current, current.params[class-id]))
Verify Image	assert(binary-match(digest(current), current.params[digest]))
Set Component Index	current := components[arg]
Override Parameters	current.params[k] := v for k,v in arg
Set Dependency Index	current := dependencies[arg]
Set Parameters	current.params[k] := v if not k in params for k,v in arg
Process Dependency	exec(current[common]); exec(current[current-segment])
Run	run(current)
Fetch	store(current, fetch(current.params[uri]))
Use Before	assert(now() < arg)
Check Component Offset	assert(offsetof(current) == arg)
Check Device Identifier	assert(binary-match(current, current.params[device-id]))
Check Image Not Match	assert(not binary-match(digest(current), current.params[digest]))
Check Minimum Battery	assert(battery >= arg)

Commented [DT36]: Section 7 says "On systems that support only a single component, Set Current Component has no effect"
Is "Set Current Component" and "Set Component Index" the same thing? If so, is this sentence wrong or is the one in section 7 wrong?

Commented [DT37]: "for-each"? (For comparison, the Try Each command below uses "for-each")

Commented [DT38]: "if k not in params, for-each"?

Check Update Authorized	assert(isAuthorized())
Check Version	assert(version_check(current, arg))
Abort	assert(0)
Try Each	break if exec(seq) is not error for-each seq in arg
Copy	store(current, current.params[src-component])
Swap	swap(current, current.params[src-component])
Wait For Event	until event(arg), wait
Run Sequence	exec(arg)
Run with Arguments	run(current, arg)

6.5. Special Cases of Component Index and Dependency Index

The interpreter MUST support a special case of Component Index if more than two or more components are supported: setting Component Index to True is allowed. When a command is invoked and the Component Index is True, the command MUST be invoked once for each Component, in the order listed in the array of Component Identifiers. The interpreter MUST support a special case of Dependency Index when two or more dependencies are supported. When a command is invoked and the Dependency Index is True, the command MUST be invoked once for each Dependency, in the order listed in the array of Dependencies.

This is represented by the following pseudocode.

```
if iscomponent(current):
    if current is true:
      cmd(component) for-each component in components
    else:
      cmd(current)
else:
    if current is true:
      cmd(dependency) for-each dependency in dependencies
    else:
      cmd(current)
```

Commented [DT39]: What does “break” mean? Stop the current manifest with success (potentially going on to the next manifest)? Clarify.

Commented [DT40]: What does “more than two or more” mean? “more than 2”? “2 or more”? “more than more”?

Commented [DT41]: The next line tests if current is true, implying that “iscomponent(true) == true”, implying “true” is a component. Is that intentional? This contradicts the definition of component in the terminology section which said “An updatable logical block of the Firmware, Software, configuration, or data of the Recipient” and “true” doesn’t meet that definition.

Try Each and Run Sequence are affected in the same way as other commands: they are invoked once for each possible Component or Dependency. This means that the sequences that are arguments to Try Each and Run Sequence are NOT invoked with Component Index = True or Dependency Index = True. They are only invoked with integer indices. The interpreter loops over the whole sequence, setting the Component Index or Dependency Index to each possible index in turn.

6.6. Serialized Processing Interpreter

In highly constrained devices, where storage for parameters is limited, the manifest processor MAY 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.

In order to operate in this mode, the manifest processor loops on each section for every supported component, simply ignoring commands when the current component is not selected.

When a serialized Manifest Processor encounters a component or dependency index of True, it does not ignore any commands. It applies them to the current component or dependency on each iteration.

6.7. Parallel Processing Interpreter

Advanced Recipients MAY make use of the Strict Order parameter and enable parallel processing of some Command Sequences, or it may reorder some Command Sequences. To perform parallel processing, once the Strict Order parameter is set to False, the Recipient 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 Recipient 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.

Commented [DT42]: Or thread? Why is a completely separate process needed?

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 begin with a Set Component Index directive. The interpreter MUST track each Set Component Index directive, and cause an Abort if more than one Set Component Index directive targets the same Component Index. When Strict Order = False, each suit-directive-run-sequence MUST begin with a Set Component Index directive. Any further Set Component Index directives MUST cause an Abort. This allows the interpreter that forks suit-directive-run-sequence processes to check that the first element is correct, then fork a process to handle the remainder of the sequence.

Commented [DT43]: MAY (is this a choice made by the manifest processor implementer, or who)?

Commented [DT44]: "Run Sequence command" is the term defined above. suit-directive-run-sequence isn't defined until much later in the doc.

Commented [DT45]: MAY?

6.8. Processing Dependencies

As described in Section 6.2, each manifest must invoke each of its dependencies sections from the corresponding section of the dependent. Any changes made to parameters by the dependency persist in the dependent.

When a Process Dependency command is encountered, the interpreter loads the dependency identified by the Current Dependency Index. The interpreter first executes the common-sequence section of the identified dependency, then it executes the section of the dependency that corresponds to the currently executing section of the dependent.

The Manifest Processor MUST also support a Dependency Index of True, which applies to every dependency, as described in Section 6.5.

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

6.9. Multiple Manifest Processors

When a system has multiple security domains they MAY require independent verification of authenticity or security policies. Security domains may be divided by separation technology such as Arm TrustZone, or Intel SGX. Security domains ~~may~~ might also be divided into separate processors and memory spaces, with a communication interface between them.

Commented [DT46]: Confusing grammar, "it" (a system)? Or "the security domains"?

Commented [DT47]: This doesn't seem like a choice made by the implementation of this RFC, so I suspect "might" is more appropriate here.

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

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

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

7. Creating Manifests

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

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

Commented [DT48]: Consider adding a reference to draft-ietf-teep-architecture

Commented [DT49]: Same comment, I am not sure this is a choice that is up to the manifest processor implementer.

Commented [DT50]: Section #?

Commented [DT51]: Expand acronym on first use

Commented [DT52]: This warrants discussion in Security Considerations about what is the trust model when they're in different security domains. Is it assumed that one trusts the other?

The aAuthor MUST ensure that all parameters consumed by a command are set prior to invoking that command. Where Component Index = True or Dependency Index = True, this means that the parameters consumed by each command MUST have been set for each Component or Dependency, respectively.

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

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

7.1. Compatibility Check Template

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

- Set Component Index directive (see Section 8.7.7.1)
- Set Parameters directive (see Section 8.7.7.6) for Vendor ID and Class ID (see Section 8.7.5)
- Check Vendor Identifier condition (see Section 8.7.5.1)
- Check Class Identifier condi~~ca~~tion (see Section 8.7.5.1)

7.2. Secure Boot Template

This template performs a secure boot operation.

The following ~~operations~~ commands are placed into the common block:

- Set Component Index directive (see Section 8.7.7.1)
- Override Parameters directive (see Section 8.7.7.7) for Image Digest and Image Size (see Section 8.7.5)

Then, the run block contains the following ~~operations~~commands:

- Set Component Index directive (see Section 8.7.7.1)
- Check Image Match condition (see Section 8.7.6.2)
- Run directive (see Section 8.7.7.13)

Commented [DT53]: No such command. "Set Component Index"?

Commented [DT54]: Seems like you need a sentence before this section to explain what a template is.

Commented [DT55]: "A manifest following the compatibility check template…"?

Commented [DT56]: typo

Commented [DT57]: No it doesn't, a *manifest that follows this template* can be used in performing a secure boot operation. A *template* doesn't have the actual parameters in it.

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

Commented [DT58]: This contradicts the sentence in section 7 that says "On systems that support only a single component, Set Current Component has no effect and can be omitted".

7.3. Firmware Download Template

This template triggers the download of firmware.

Commented [DT59]: See earlier comments about this wording. Make similar fixes to all other template sections.

The following operations are placed into the common block:

- Set Component Index directive (see Section 8.7.7.1)
- Override Parameters directive (see Section 8.7.7.7) for Image Digest and Image Size (see Section 8.7.5)

Then, the install block contains the following operations:

- Set Component Index directive (see Section 8.7.7.1)
- Set Parameters directive (see Section 8.7.7.6) for URI (see Section 8.7.5.12)
- Fetch directive (see Section 8.7.7.8)
- Check Image Match condition (see Section 8.7.6.2)

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

7.4. Install Template

This template modifies the Firmware Download template and adds an additional sequence. The Firmware Download operations are moved from the Payload Install sequence to the Payload Fetch sequence.

Then, the Install sequence contains the following ~~operations~~commands:

- Set Component Index directive (see Section 8.7.7.1)
- Set Parameters directive (see Section 8.7.7.6) for Source Component (see Section 8.7.5.13)
- Copy directive (see Section 8.7.7.10)
- Check Image Match condition (see Section 8.7.6.2)

7.5. Integrated Payload Template

This template triggers the installation of a payload included in the manifest envelope. It is identical to the Firmware Download template (Section 7.3) except that it places an added restriction on the URI passed to the Set Parameters directive.

An implementeer MAY choose to place a payload in the envelope of a manifest. The payload envelope key MAY be a positive or negative integer. The payload envelope key MUST NOT be a value between 0 and 24 and it MUST NOT be used by any other envelope element in the manifest. The payload MUST be serialized in a bstr element.

The URI for a payload enclosed in this way MUST be expressed as a fragment-only reference, as defined in [RFC3986], Section 4.4. The fragment identifier is the stringified envelope key of the payload. For example, an envelope that contains a payload a key 42 would use a URI "#42", key -73 would use a URI "#-73".

7.6. Load from Nonvolatile Storage Template

This directive loads an firmware image from external storage.

Commented [DT60]: This is a template not a directive

The following operations commands are placed into the load block:

- Set Component Index directive (see Section 8.7.7.1)
- Set Parameters directive (see Section 8.7.7.6) for Component Index (see Section 8.7.5)
- Copy directive (see Section 8.7.7.10)

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

7.7. Load & Decompress from Nonvolatile Storage Template

The following operations commands are placed into the load block:

- Set Component Index directive (see Section 8.7.7.1)
- Set Parameters directive (see Section 8.7.7.6) for Source Component Index and Compression Info (see Section 8.7.5)
- Copy directive (see Section 8.7.7.10)

This template is similar to Section 7.6 but additionally performs decompression. Hence, the only difference is in setting the Compression Info parameter.

7.8. Dependency Template

The following ~~operations~~ commands are placed into the dependency resolution block:

- Set Dependency Index directive (see Section 8.7.7.2)
- Set Parameters directive (see Section 8.7.7.6) for URI (see Section 8.7.5)
- Fetch directive (see Section 8.7.7.8)
- Check Image Match condition (see Section 8.7.6.2)
- Process Dependency directive (see Section 8.7.7.5)

Then, the validate block contains the following ~~operations~~commands:

- Set Dependency Index directive (see Section 8.7.7.2)
- Check Image Match condition (see Section 8.7.6.2)
- Process Dependency directive (see Section 8.7.7.5)

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

7.8.1. Composite Manifests

An implementeer MAY choose to place a dependency's envelope in the envelope of its dependent. The dependent envelope key for the dependency envelope MUST NOT be a value between 0 and 24 and it MUST NOT be used by any other envelope element in the dependent manifest.

The URI for a dependency enclosed in this way MUST be expressed as a fragment-only reference, as defined in [RFC3986], Section 4.4. The fragment identifier is the stringified envelope key of the dependency. For example, an envelope that contains a dependency at key 42 would use a URI "#42", key -73 would use a URI "#-73".

7.9. Encrypted Manifest Template

To use an encrypted manifest, create a plaintext dependent, and add the encrypted manifest as a dependency. The dependent can include very little information.

The following ~~operations~~ commands are placed into the dependency resolution block:

- Set Dependency Index directive (see Section 8.7.7.2)
- Set Parameters directive (see Section 8.7.7.6) for
  - URI (see Section 8.7.5)
  - Encryption Info (see Section 8.7.5)
- Fetch directive (see Section 8.7.7.8)
- Check Image Match condition (see Section 8.7.6.2)
- Process Dependency directive (see Section 8.7.7.5)

Then, the validate block contains the following ~~operations~~commands:

- Set Dependency Index directive (see Section 8.7.7.2)
- Check Image Match condition (see Section 8.7.6.2)
- Process Dependency directive (see Section 8.7.7.5)

A plaintext manifest and its encrypted dependency may also form a composite manifest (Section 7.8.1).

7.10. A/B Image Template

The following ~~operations~~ commands are placed in the common block:

- Set Component Index directive (see Section 8.7.7.1)
- Try Each
  - First Sequence:
    * Override Parameters directive (see Section 8.7.7.7, Section 8.7.5) for Offset A
    * Check Offset Condition (see Section 8.7.6.5)

        * Override Parameters directive (see Section 8.7.7.7) for Image Digest A and Image Size A (see Section 8.7.5)

    o Second Sequence:

        * Override Parameters directive (see Section 8.7.7.7, Section 8.7.5) for Offset B

        * Check Offset Condition (see Section 8.7.6.5)

        * Override Parameters directive (see Section 8.7.7.7) for Image Digest B and Image Size B (see Section 8.7.5)

The following ~~operations~~ commands are placed in the fetch block or install block

- Set Component Index directive (see Section 8.7.7.1)

- Try Each

    o First Sequence:

        * Override Parameters directive (see Section 8.7.7.7, Section 8.7.5) for Offset A

        * Check Offset Condition (see Section 8.7.6.5)

        * Set Parameters directive (see Section 8.7.7.7) for URI A (see Section 8.7.5)

    o Second Sequence:

        * Override Parameters directive (see Section 8.7.7.7, Section 8.7.5) for Offset B

        * Check Offset Condition (see Section 8.7.6.5)

        * Set Parameters directive (see Section 8.7.7.7) for URI B (see Section 8.7.5)

- Fetch

8. Metadata Structure

The metadata for SUIT updates is composed of several primary constituent parts: the Envelope, Delegation Chains, Authentication Information, Manifest, and Severable Elements.

For a diagram of the metadata structure, see Section 5.

8.1. Encoding Considerations

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

Wherever enumerations are used, they are started at 1. This allows detection of several common software errors that are caused by uninitialised variables. Positive numbers in enumerations are reserved for IANA registration. Negative numbers are used to identify application-specific implementations.

Commented [DT61]: "values"?

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.

8.2. Envelope

The Envelope contains each of the other primary constituent parts of the SUIT metadata. It allows for modular processing of the manifest by ordering components in the expected order of processing.

The Envelope is encoded as a CBOR Map. Each element of the Envelope is enclosed in a bstr, which allows computation of a message digest against known bounds.

8.3. Delegation Chains

The suit-delegation field MAY carry one or more CBOR Web Tokens (CWTs) [RFC8392], with [RFC8747] cnf claims. They can be used to perform enhanced authorization decisions. The CWTs are arranged into a list of lists. Each list starts with CWT authorized by a Trust Anchor, and finishes with a key used to authenticate the Manifest (see Section 8.4). This allows an Update Authority to delegate from a long term Trust Anchor, down through intermediaries, to a delegate without any out-of-band updates Trust Anchors.

Commented [DT62]: Can't parse grammar around here

A Recipient MAY choose to cache intermediaries and/or delegates. If an Update Distributor knows that a targeted Recipient has cached some intermediaries or delegates, it MAY choose to strip any cached intermediaries or delegates from the Delegation Chains in order to reduce bandwidth and energy.

8.4. Authenticated Manifests

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

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

8.5. Encrypted Manifests

To use an encrypted manifest, it must be a dependency of a plaintext manifest. This allows fine-grained control of what information is accessible to intermediate systems for the purposes of management, while still preserving the confidentiality of the manifest contents. This also means that a Recipient can process an encrypted manifest in the same way as an encrypted payload, allowing code reuse.

A template for using an encrypted manifest is covered in Encrypted Manifest Template (Section 7.9).

8.6. Manifest

The manifest contains:

- a version number (see Section 8.6.1)
- a sequence number (see Section 8.6.2)
- a reference URI (see Section 8.6.3)
- a common structure with information that is shared between command sequences (see Section 8.7.2)
- one or more lists of commands that the Recipient should perform (see Section 8.7.3)
- a reference to the full manifest (see Section 8.6.3)

- human-readable text describing the manifest found in the SUIT_Envelope (see Section 8.6.4)

- a Concise Software Identifier (CoSWID) found in the SUIT_Envelope (see Section 8.7.1)

The CoSWID, Text section, or any Command Sequence of the Update Procedure (Dependency Resolution, Image Fetch, Image Installation) can be either a CBOR structure or a SUIT_Digest. In each of these cases, the SUIT_Digest provides for a severable field. Severable fields are RECOMMENDED to implement. In particular, the human-readable text SHOULD be severable, since most useful text elements occupy more space than a SUIT_Digest, but are not needed by the Recipient. Because SUIT_Digest is a CBOR Array and each severable element is a CBOR bstr, it is straight-forward for a Recipient to determine whether an element has been severed. The key used for a severable element is the same in the SUIT_Manifest and in the SUIT_Envelope so that a Recipient can easily identify the correct data in the envelope. See Section 8.7.8 for more detail.

8.6.1. suit-manifest-version

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

8.6.2. suit-manifest-sequence-number

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

Commented [DT63]: Inappropriate use of MAY as this is not a choice made by an implementer. Perhaps rephrase as "For convenience, an implementer MAY use a UTC timestamp…"

8.6.3. suit-reference-uri

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

8.6.4. suit-text

suit-text SHOULD be a severable element. suit-text is a map of pairs. It MAY contain two different types of pairs:

- integer => text mappings

- SUIT_Component_Identifier => map mappings

Each SUIT_Component_Identifier => map entry contains a map of integer => text values. All SUIT_Component_Identifiers present in suit-text MUST also be present in suit-common (Section 8.7.2) or the suit-common of a dependency.

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

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

The following table describes the text fields available in suit-text:

CDDL Structure	Description
suit-text-manifest-description	Free text description of the manifest
suit-text-update-description	Free text description of the update
suit-text-manifest-json-source	The JSON-formatted document that was used to create the manifest
suit-text-manifest-yaml-source	The ~~yaml~~YAML-formatted document that was used to create the manifest

The following table describes the text fields available in each map identified by a SUIT_Component_Identifier.

Commented [DT64]: What does it mean if an implementer chooses to not follow this MAY? Can it contain other types of pairs?

Commented [DT65]: Expand acronym and add reference. (Unlike "JSON" which is in the RFC editor acronym list with an asterisk and so need not be expanded, YAML is not even on the list.)

CDDL Structure	Description
suit-text-vendor-name	Free text vendor name
suit-text-model-name	Free text model name
suit-text-vendor-domain	The domain used to create the vendor-id condition
suit-text-model-info	The information used to create the class-id condition
suit-text-component-description	Free text description of each component in the manifest
suit-text-component-version	A text version number
suit-text-version-required	A text expression of the required version number

suit-text is OPTIONAL to implement.

8.7. suit-text-version-required

suit-text-version-required is used to represent a version-based dependency on suit-parameter-version as described in Section 8.7.5.17 and Section 8.7.6.8. To describe a version dependency, a Manifest Author should populate the suit-text map with a SUIT_Component_Identifier key for the dependency component, and place in the corresponding map a suit-text-version-required key with a text expression that is representative of the version constraints placed on the dependency.

For example, to express a dependency on a component "['x', 'y']", where the version should be any v1.x later than v1.2.5, but not v2.0 or above, the author would add the following structure to the suit-text element. Note that this text is in cbor-diag notation.

```
" [h'78',h'79'] : { 7 : ">=1.2.5,<2" } "
```

8.7.1. suit-coswid

suit-coswid contains a Concise Software Identifier (CoSWID). This element SHOULD be made severable so that it can be discarded by the Recipient or an intermediary if it is not required by the Recipient.

Commented [DT66]: Does this mean it's not "free text" like the other text fields? If so, what restricted set of characters are legal? E.g., are letters legal ("1.2.5a")? ASCII only or is "1.2.5α" legal? Page 47 says "Versions are encoded as a CBOR list of integers" but that's in a section on suit-parameter-version, not suit-text-component-version

Commented [DT67]: What is the syntax for such expressions? E.g., can you do "equals" / "=" / "=="? Can you do "not equals" / "!=" / "<>" / "≠"? What about other operations? What about "≤" and "≥", or do you have to do "<=" and ">="?

suit-coswid is OPTIONAL to implement.

Commented [DT68]: On the processor side, the generator side, or both?
(same question on other commands that are listed as optional)

8.7.2. suit-common

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

suit-dependencies is a list of Section 8.7.2.1 blocks that specify manifests that must be present before the current manifest can be processed. suit-dependencies is OPTIONAL to implement.

suit-components is a list of SUIT_Component_Identifier (Section 8.7.2.2) blocks that specify the component identifiers that will be affected by the content of the current manifest. suit-components is REQUIRED to implement; at least one manifest in a dependency tree MUST contain a suit-components block.

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 Recipient identity and image digests when they are conditional (see Section 8.7.7.4 and Section 7.10 for more information on conditional sequences). suit-common-sequence is RECOMMENDED to implement. It is REQUIRED if the optimizations described in Section 6.2.1 will be used. Whenever a parameter or try-each is required by more than one Command Sequence, suit-common-sequence results in a smaller encoding.

Commented [DT69]: "Try Each command"?

8.7.2.1. Dependencies

SUIT_Dependency specifies a manifest that describes a dependency of the current manifest. The Manifest is identified, ~~however~~ but the Recipient should expect an Envelope when it acquires the dependency. This is because the Manifest is the one invariant element of the Envelope, where other elements may change by countersigning, adding authentication blocks, or severing elements.

The suit-dependency-digest specifies the dependency manifest uniquely by identifying a particular Manifest structure. This is identical to the digest that would be present as the payload of any suit-authentication-block in the dependency's Envelope. The digest is calculated over the Manifest structure instead of the COSE Sig_structure or Mac_structure. 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 (see Section 8.7.2.2). This specifies the scope at which the dependency operates. This allows the dependency to be forwarded on to a component that is capable of parsing its own manifests. It also allows one manifest to be deployed to multiple dependent Recipients without those Recipients needing consistent component hierarchy. This element is OPTIONAL.

Commented [DT70]: Optional to implement? Or optional to appear in any given manifest?

A dependency prefix can be used with a component identifier. This allows complex systems to understand where dependencies need to be applied. The dependency prefix can be used in one of two ways. The first simply prepends the prefix to all Component Identifiers in the dependency.

A dependency prefix can also be used to indicate when a dependency manifest needs to be processed by a secondary manifest processor, as described in Section 6.9.

8.7.2.2. SUIT_Component_Identifier

A component is a unit of code or data that can be targeted by an update. To facilitate composite devices, components are identified by a list of CBOR byte strings, which allows construction of hierarchical component structures. A dependency MAY declare a prefix to the components defined in the dependency manifest. Components are identified by Component Identifiers, i.e., arrays of binary strings, but referenced in commands

Commented [DT71]: … something missing at end here?

A Component Identifier can be trivial, such as the simple array [h'00']. It can also represent a filesystem path by encoding each segment of the path as an element in the list. For example, the path "/usr/bin/env" would encode to ['usr','bin','env'].

This hierarchical construction allows a component identifier to identify any part of a complex, multi-component system.

8.7.3. SUIT_Command_Sequence

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

1. Dependency Resolution: suit-dependency-resolution is a SUIT_Command_Sequence to execute in order to perform dependency resolution. Typical actions include configuring URIs of

dependency manifests, fetching dependency manifests, and validating dependency manifests' contents. suit-dependency-resolution is REQUIRED to implement and to use when suit-dependencies is present.

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

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

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

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

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

Goals 1,2,3 form the Update Procedure. Goals 4,5,6 form the Boot Procedure.

Each Command Sequence follows exactly the same structure to ensure that the parser is as simple as possible.

Lists of commands are constructed from two kinds of elements:

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

2. Directives that MUST be executed.

Each condition is a command code identifier, followed by a SUIT_Reporting_Policy (Section 8.7.4).

Each directive is composed of:

1. A command code identifier

2. An argument block or a reporting policy

Argument blocks are consumed only by flow-control directives:

- Set Component/Dependency Index

- Set/Override Parameters

- Try Each

- Run Sequence

Reporting policies provide a hint to the manifest processor of whether ~~or not~~ to add the success or failure of a command to any report that it generates.

Many conditions and directives apply to a given component, and these are 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,~~suit-directive-try-each (Section 8.7.7.4)~~,~~ 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. See Section 8.7.5.22.

Commented [DT72]: It's odd that conditions show the two pieces in sentence form whereas directives show the two pieces in bullet form. Also that conditions say "SUIT_Reporting_Policy" but directives say "reporting policy". Why the difference?

Commented [DT73]: manifest?

8.7.4. Reporting Policy

To facilitate construction of Reports that describe the success, or failure of a given Procedure, each command is given a Reporting Policy. This is an integer bitfield that follows the command and indicates what the Recipient should do with the Record of executing the command. The options are summarized in the table below.

Policy	Description
suit-send-record-on-success	Record when the command succeeds
suit-send-record-on-failure	Record when the command fails
suit-send-sysinfo-success	Add system information when the command succeeds
suit-send-sysinfo-failure	Add system information when the command fails

Any or all of these policies may be enabled at once.

If the component index is set to True when a command is executed with a non-zero reporting policy, then the Reporting Engine MUST receive one Record for each Component, in the order expressed in the Components list. If the dependency index is set to True when a command is executed with a non-zero reporting policy, then the Reporting Engine MUST receive one Record for each Dependency, in the order expressed in the Dependencies list.

SUIT does NOT REQUIRE a particular format of Records or Reports. SUIT only defines hints to the Reporting engine for which Records it should aggregate into the Report.

For example, a system using DICE certificates MAY use instances of suit-send-sysinfo-success to construct its certificates.

An OPTIONAL Record format, SUIT_Record is defined in [full-cddl]. It is encoded as a map, with the following elements.

Commented [DT74]: Undefined term, what's this?

Commented [DT75]: So this is yet another alternative to IPFIX, syslog, etc? State the rationale for defining a new format. (You might also say that it could be used in an octetArray in IPFIX if someone defined such a field)

If SUIT_Record is not something that would appear in a SUIT manifest, then I think it should not be in this document.

Element	Description
suit-record-success	The boolean or integer success or failure code of the command.
suit-record-component-id	The current component when the record was generated.
suit-record-dependency-id	The current dependency digest when the record was generated.
suit-record-command-sequence-id	The label of the Command Sequence that was executing when the record was generated.
suit-record-command-id	The label of the command that was in progress when the record was generated.
suit-record-params	The set of parameters that was consumed by the current command.
suit-record-actual	The value against which a suit-condition compared a parameter.

In Secure Boot operations, the Reporting engine MAY aggregate the Records produced in a Procedure into the evidence used for an attestation report.

Commented [DT76]: What's the rationale for doing so in an attestation report as opposed to some more typical logging/reporting mechanism that might integrate with existing error reporting mechanisms and trouble ticketing? I don't think this was motivated in either the suit architecture or the suit IM document, and I don't recall it ever being discussed on the SUIT list or WG meeting (but maybe it was and I forgot?)

8.7.5. SUIT_Parameters

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

Most parameters are scoped to a specific component. This means that setting a parameter for one component has no effect on the parameters of any other component. The only exceptions to this are two Manifest Processor parameters: Strict Order and Soft Failure.

The defined manifest parameters are described below.

Name	CDDL Structure	Reference

Vendor ID	suit-parameter-vendor-identifier	Section 8.7.5 .2
Class ID	suit-parameter-class-identifier	Section 8.7.5 .3
Image Digest	suit-parameter-image-digest	Section 8.7.5 .5
Image Size	suit-parameter-image-size	Section 8.7.5 .6
Use Before	suit-parameter-use-before	Section 8.7.5 .7
Component Offset	suit-parameter-component-offset	Section 8.7.5 .8
Encryption Info	suit-parameter-encryption-info	Section 8.7.5 .9
Compression Info	suit-parameter-compression-info	Section 8.7.5 .10
Unpack Info	suit-parameter-unpack-info	Section 8.7.5 .11
URI	suit-parameter-uri	Section 8.7.5 .12
Source Component	suit-parameter-source-component	Section 8.7.5 .13
Run Args	suit-parameter-run-args	Section 8.7.5 .14
Device ID	suit-parameter-device-identifier	Section 8.7.5 .4
Minimum Battery	suit-parameter-minimum-battery	Section 8.7.5 .15
Update Priority	suit-parameter-update-priority	Section 8.7.5 .16
Version	suit-parameter-version	Section 8.7.5 .17

Commented [DT77]: Why is this one out of order in the table?

Wait Info	suit-parameter-wait-info	Section 8.7.5.18
URI List	suit-parameter-uri-list	Section 8.7.5.19
Fetch Arguments	suit-parameter-fetch-arguments	Section 8.7.5.20
Strict Order	suit-parameter-strict-order	Section 8.7.5.21
Soft Failure	suit-parameter-soft-failure	Section 8.7.5.22
Custom	suit-parameter-custom	Section 8.7.5.23

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.

8.7.5.1. Constructing Identifiers

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

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

Identifiers are used for compatibility checks. They MUST NOT be used as assertions of identity. They are evaluated by identifier conditions (Section 8.7.6.1).

Commented [DT78]: So a vendor can't use an existing OUI or IANA private enterprise number that can be easily looked up? You have to instead use an opaque identifier with no way to look up its meaning? What is the rationale for inventing a new space rather than being able to reuse an existing one that can be looked up to get an org name, contact info, etc.? And a UUID takes up more bytes than either an OUI or a PEN, so is worse for constrained devices.

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

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

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

1. Hardware model/revision

2. OS

3. WiFi module model/revision

4. Application

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

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

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

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

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

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

- Model number.

- Hardware revision.

- Bootloader version (for immutable bootloaders).

8.7.5.2. suit-parameter-vendor-identifier

A RFC 4122 UUID representing the vendor of the device or component. The UUID is encoded as a 16 byte bstr, containing the raw bytes of the UUID. It MUST be constructed as described in Section 8.7.5.1.

8.7.5.3. suit-parameter-class-identifier

A RFC 4122 UUID representing the class of the device or component. The UUID is encoded as a 16 byte bstr, containing the raw bytes of the UUID. It MUST be constructed as described in Section 8.7.5.1.

8.7.5.4. suit-parameter-device-identifier

A RFC 4122 UUID representing the specific device or component. The UUID is encoded as a 16 byte bstr, containing the raw bytes of the UUID. It MUST be constructed as described in Section 8.7.5.1.

8.7.5.5. suit-parameter-image-digest

A fingerprint computed over the component itself, encoded in the Section 10 structure. The SUIT_Digest is wrapped in a bstr, as required in Section 8.7.5.

Commented [DT79]: Name of structure?

8.7.5.6. suit-parameter-image-size

The size of the firmware image in bytes. This size is encoded as a positive integer.

8.7.5.7. suit-parameter-use-before

An expiry date for the use of the manifest encoded as a POSIX timestamp; a positive integer. Implementations that use this parameter MUST use a 64-bit internal representation of the integer.

Commented [DT80]: Is there a reference for this that can be added?

8.7.5.8. suit-parameter-component-offset

This parameter sets the offset in a component. Some components support multiple possible Slots (offsets into a storage area). This parameter describes the intended Slot to use, identified by its offset into the component's storage area. This offset MUST be encoded as a positive integer.

8.7.5.9. 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.

8.7.5.10. suit-parameter-compression-info

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

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

8.7.5.11. suit-parameter-unpack-info

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

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

8.7.5.12. suit-parameter-uri

A URI from which to fetch a resource.

8.7.5.13. suit-parameter-source-component

This parameter sets the source component to be used with either Section 8.7.7.10 or with Section 8.7.7.14. The current Component, as set by suit-directive-set-component-index defines the destination, and suit-parameter-source-component defines the source.

8.7.5.14. suit-parameter-run-args

This parameter contains an encoded set of arguments for Section 8.7.7.11. The arguments MUST be provided as an implementation-defined bstr.

8.7.5.15. suit-parameter-minimum-battery

This parameter sets the minimum battery level in mWh. This parameter is encoded as a positive integer. Used with Section 8.7.6.6.

Commented [DT81]: Who is responsible for defining the format of this information (extensions)? And what is the expected mechanism for communicating it to those who author manifests?

Commented [DT82]: In what numbering space? (reference?)

Commented [DT83]: This sentence is normative (since there's no "for example", but instead "defines the use of"), but the reference is informative.

Commented [DT84]: These are hard to read with just a section number, list the actual name like suit-directive-copy. Same comment on the next couple sections that say "used with Section…"

8.7.5.16. suit-parameter-update-priority

This parameter sets the priority of the update. This parameter is encoded as an integer. It is used along with suit-condition-update-authorized [1] to ask an application for permission to initiate an update. This does not constitute a privilege inversion because an explicit request for authorization has been provided by the Update Authority in the form of the suit-condition-update-authorized command.

Applications MAY define their own meanings for the update priority. For example, critical reliability & vulnerability fixes MAY be given negative numbers, while bug fixes MAY be given small positive numbers, and feature additions MAY be given larger positive numbers, which allows an application to make an informed decision about whether and when to allow an update to proceed.

8.7.5.17. suit-parameter-version

Indicates allowable versions for the specified component. Allowable versions can be specified, either with a list or with range matching. This parameter is compared with version asserted by the current component when Section 8.7.6.8 is invoked. The current component may assert the current version in many ways, including storage in a parameter storage database, in a metadata object, or in a known location within the component itself.

The component version can be compared as:

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

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

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

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

suit-condition-version is OPTIONAL to implement.

Versions SHOULD be provided as follows:

1. The first integer represents the major number. This indicates breaking changes to the component.
2. The second integer represents the minor number. This is typically reserved for new features or large, non-breaking changes.
3. The third integer is the patch version. This is typically reserved for bug fixes.
4. The fourth integer is the build number.

Where Alpha (-3), Beta (-2), and Release Candidate (-1) are used, they are inserted as a negative number between Minor and Patch numbers. This allows these releases to compare correctly with final releases. For example, Version 2.0, RC1 should be lower than Version 2.0.0 and higher than any Version 1.x. By encoding RC as -1, this works correctly: [2,0,-1,1] compares as lower than [2,0,0]. Similarly, beta (-2) is lower than RC and alpha (-3) is lower than RC.

8.7.5.18. suit-parameter-wait-info

suit-directive-wait (Section 8.7.7.12) directs the manifest processor to pause until a specified event occurs. The suit-parameter-wait-info encodes the parameters needed for the directive.

The exact implementation of the pause is implementation-defined. For example, this could be done by blocking on a semaphore, registering an event handler and suspending the manifest processor, polling for a

notification, or aborting the update entirely, then restarting when a notification is received.

suit-parameter-wait-info is encoded as a map of wait events. When ALL wait events are satisfied, the Manifest Processor continues. The wait events currently defined are described in the following table.

Name	Encoding	Description
suit-wait-event-authorization	int	Same as Section 8.7.5.16
suit-wait-event-power	int	Wait until power state
suit-wait-event-network	int	Wait until network state
suit-wait-event-other-device-version	See below	Wait for other device to match version
suit-wait-event-time	uint	Wait until time (POSIX timestamp)
suit-wait-event-time-of-day	uint	Wait until seconds since 00:00:00
suit-wait-event-day-of-week	uint	Wait until days since Sunday

suit-wait-event-other-device-version reuses the encoding of suit-parameter-version-match. It is encoded as a sequence that contains an implementation-defined bstr identifier for the other device, and a list of one or more SUIT_Parameter_Version_Match.

8.7.5.19. suit-parameter-uri-list

Indicates a list of URIs from which to fetch a resource. The URI list is encoded as a list of tstr, in priority order. The Recipient should attempt to fetch the resource from each URI in turn, ruling out each, in order, if the resource is inaccessible or it is otherwise undesirable to fetch from that URI. suit-parameter-uri-list is consumed by Section 8.7.7.9.

Commented [DT85]: suit-parameter-update-priority

Commented [DT86]: UTC or local time?

Commented [DT87]: UTC or local time?

Commented [DT88]: Reference RFC 8610? e.g.: … as a list of “tstr” (text string, see [RFC8610])

8.7.5.20. suit-parameter-fetch-arguments

An implementation-defined set of arguments to Section 8.7.7.8. Arguments are encoded in a bstr.

Commented [DT89]: As before, please add the actual thing, not just the section reference. (They're not arguments to a document section, they're arguments to a command.)

8.7.5.21. 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.

Commented [DT90]: So what happens if a dependency sets it to False and doesn't reset it to true, this carries over to the dependent then, right? That sounds like a source of unexpected bugs.

See Section 6.7 for behavioral description of Strict Order.

8.7.5.22. suit-parameter-soft-failure

When executing a command sequence inside Section 8.7.7.4 or Section 8.7.7.13 and a condition failure occurs, the manifest processor aborts the sequence. For suit-directive-try-each, if Soft Failure is True, the next sequence in Try Each is invoked, otherwise suit-directive-try-each fails with the condition failure code. In suit-directive-run-sequence, if Soft Failure is True the suit-directive-run-sequence simply halts with no side-effects and the Manifest Processor continues with the following command, otherwise, the suit-directive-run-sequence fails with the condition failure code.

suit-parameter-soft-failure is scoped to the enclosing SUIT_Command_Sequence. Its value is discarded when SUIT_Command_Sequence terminates. It MUST NOT be set outside of suit-directive-try-each or suit-directive-run-sequence.

When suit-directive-try-each is invoked, Soft Failure defaults to True. An Update Author may choose to set Soft Failure to False if they require a failed condition in a sequence to force an Abort.

When suit-directive-run-sequence is invoked, Soft Failure defaults to False. An Update Author may choose to make failures soft within a suit-directive-run-sequence.

8.7.5.23. suit-parameter-custom

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

8.7.6. 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. All Conditions specify a Reporting Policy as described in Section 8.7.4. Conditions include:

Name	CDDL Structure	Reference
Vendor Identifier	suit-condition-vendor-identifier	Section 8.7.6.1
Class Identifier	suit-condition-class-identifier	Section 8.7.6.1
Device Identifier	suit-condition-device-identifier	Section 8.7.6.1
Image Match	suit-condition-image-match	Section 8.7.6.2
Image Not Match	suit-condition-image-not-match	Section 8.7.6.3
Use Before	suit-condition-use-before	Section 8.7.6.4
Component Offset	suit-condition-component-offset	Section 8.7.6.5
Minimum Battery	suit-condition-minimum-battery	Section 8.7.6.6
Update Authorized	suit-condition-update-authorized	Section 8.7.6.7
Version	suit-condition-version	Section 8.7.6.8
Custom Condition	SUIT_Condition_Custom	Section 8.7.6.9

Commented [DT91]: Why is this one capitalized differently from all the other rows?

The abstract description of these conditions is defined in Section 6.4.

Conditions compare parameters against properties of the system. These properties may be asserted in many different ways, including: calculation on-demand, volatile definition in memory, static definition within the manifest processor, storage in known location within an image, storage within a key storage system, storage in One-Time-Programmable memory, inclusion in mask ROM, or inclusion as a register in hardware. Some of these assertion methods are global in scope, such as a hardware register, some are scoped to an individual

component, such as storage at a known location in an image, and some assertion methods can be either global or component-scope, based on implementation.

Each condition MUST report a result code on completion. If a condition reports failure, then the current sequence of commands MUST terminate. A subsequent command or command sequence MAY continue executing if Section 8.7.5.22 is set. If a condition requires additional information, this MUST be specified in one or more parameters before the condition is executed. If a Recipient attempts to process a condition that expects additional information and that information has not been set, it MUST report a failure. If a Recipient encounters an unknown condition, it MUST report a failure.

Condition labels in the positive number range are reserved for IANA registration while those in the negative range are custom conditions reserved for proprietary use. See Section 11 for more details.

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

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

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.

Each identifier condition compares the corresponding identifier parameter to a parameter asserted to the Manifest Processor by the Recipient. Identifiers MUST be known to the Manifest Processor in order to evaluate compatibility.

Globally-scoped identifiers MUST match, regardless of current component index. Component-scoped identifiers match only when the current component index resolves to the component associated with the component-scoped identifier.

Commented [DT92]: A document section is never “set”, only parameters are set.

Commented [DT93]: Meaning they can be defined by the manifest processor implementer? Are they scoped to a particular vendor id or class id? I.e., when authoring a manifest for a given component on a given device, whose definition of these should I use? The component author’s, the device vendor’s, or the manifest processor author’s?

Commented [DT94]: What does it mean to not follow this MAY? I.e., what does it mean to NOT treat them as scoped by component or as global identifiers?

8.7.6.2. suit-condition-image-match

Verify that the current component matches the Section 8.7.5.5 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.

8.7.6.3. suit-condition-image-not-match

Verify that the current component does not match the Section 8.7.5.5. If no digest is specified, the condition fails. suit-condition-image-not-match is OPTIONAL to implement.

8.7.6.4. 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 (Section 8.7.5.7), which must have already been set as a parameter, encoded as a POSIX timestamp, that is seconds after 1970-01-01 00:00:00 UTC. Timestamp conditions MUST be evaluated in 64 bits, regardless of encoded CBOR size. suit-condition-use-before is OPTIONAL to implement.

8.7.6.5. suit-condition-component-offset

Verify that the offset of the current component matches the offset set in Section 8.7.5.8. This condition allows a manifest to select between several images to match a target offset.

8.7.6.6. suit-condition-minimum-battery

suit-condition-minimum-battery provides a mechanism to test a Recipient's battery level before installing an update. This condition is primarily 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. suit-condition-minimum-battery consumes Section 8.7.5.15.

8.7.6.7. suit-condition-update-authorized

Request Authorization from the application and fail if not authorized. This can allow a user to decline an update. Section 8.7.5.16 provides an integer priority level that the

Commented [DT95]: Same problem here (and in other sections below) about missing name

Commented [DT96]: Correct?

Commented [DT97]: And if a manifest processor doesn't implement the condition, does it automatically fail? Or automatically pass? Or result in a parse error?

application can use to determine whether or not to authorize the update. Priorities are application defined. suit-condition-update-authorized is OPTIONAL to implement.

Commented [DT98]: Same question about other OPTIONAL conditions… if not implemented, does it pass, fail, or result in a parse error?

8.7.6.8. suit-condition-version

suit-condition-version allows comparing versions of firmware. Verifying image digests is preferred to version checks because digests are more precise. suit-condition-version examines a component's version against the version info specified in Section 8.7.5.17.

8.7.6.9. SUIT_Condition_Custom

SUIT_Condition_Custom describes any proprietary, application specific condition. This is encoded as a negative integer, chosen by the firmware developer. If additional information must be provided to the condition, it should be encoded in a custom parameter (a nint) as described in Section 8.7.5. SUIT_Condition_Custom is OPTIONAL to implement.

8.7.7. SUIT_Directive

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

Name	CDDL Structure	Reference
Set Component Index	suit-directive-set-component-index	Section 8.7.1
Set Dependency Index	suit-directive-set-dependency-index	Section 8.7.2
Abort	suit-directive-abort	Section 8.7.3
Try Each	suit-directive-try-each	Section 8.7.4
Process Dependency	suit-directive-process-dependency	Section 8.7.5
Set Parameters	suit-directive-set-parameters	Section 8.7.6
Override Parameters	suit-directive-override-parameters	Section 8.7.7
Fetch	suit-directive-fetch	Section 8.7.8
Copy	suit-directive-copy	Section 8.7.10
Run	suit-directive-run	Section 8.7.11
Wait For Event	suit-directive-wait	Section 8.7.12
Run Sequence	suit-directive-run-sequence	Section 8.7.13
Swap	suit-directive-swap	Section 8.7.14
Fetch URI list	suit-directive-fetch-uri-list	Section 8.7.9

The abstract description of these commands is defined in Section 6.4.

Commented [DT99]: Why is this one out of order?

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

8.7.7.1. 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 suit-components. If the following commands apply to ALL components, then the boolean value "True" is used instead of an index. If the following commands 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.

If component index is set to True when a command is invoked, then the command applies to all components, in the order they appear in suit-common-components. When the Manifest Processor invokes a command while the component index is set to True, it must execute the command once for each possible component index, ensuring that the command receives the parameters corresponding to that component index.

8.7.7.2. 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.

If dependency index is set to True when a command is invoked, then the command applies to all dependencies, in the order they appear in suit-common-components. When the Manifest Processor invokes a command while the dependency index is set to True, it must execute the command once for each possible dependency index, ensuring that the command receives the parameters corresponding to that dependency index.

Typical operations that require suit-directive-set-dependency-index include setting a source URI or Encryption Information, invoking

"Fetch" or invoking "Process Dependency" for an individual dependency.

8.7.7.3. suit-directive-abort

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

8.7.7.4. suit-directive-try-each

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

Section 8.7.5.22 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 argument to suit-directive-try-each is a list of SUIT_Command_Sequence. suit-directive-try-each does not specify a reporting policy.

8.7.7.5. suit-directive-process-dependency

Execute the commands in the common section of the current dependency, followed by the commands in the equivalent section of the current dependency. For example, if the current section is "fetch payload," this will execute "common" in the current dependency, then "fetch payload" in the current dependency. Once this is complete, the command following suit-directive-process-dependency will be processed.

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

Commented [DT100]: Why not make it illegal?

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

8.7.7.6. suit-directive-set-parameters

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

parameters also allows a manifest to modify the behavior of its dependencies.

Available parameters are defined in Section 8.7.5.

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.

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

8.7.7.7. suit-directive-override-parameters

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

Available parameters are defined in Section 8.7.5.

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

8.7.7.8. suit-directive-fetch

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

suit-directive-fetch can target one or more manifests and one or more payloads. suit-directive-fetch retrieves each component and each manifest listed in component-index and dependency-index, respectively. If component-index or dependency-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 and set in suit-parameter-fetch-arguments.

suit-directive-fetch reads the URI 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.

8.7.7.9. suit-directive-fetch-uri-list

suit-directive-fetch-uri-list uses the same semantics as suit-directive-fetch (Section 8.7.7.8), ~~however~~ except that it iterates over the URI List (Section 8.7.5.19) to select a URI to fetch from.

8.7.7.10. 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.

Commented [DT101]: suit-directive-set-component-index?

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.

suit-directive-copy reads its source from Section 8.7.5.13.

8.7.7.11. 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 are provided to suit-directive-run through suit-parameter-run-arguments (Section 8.7.5.14) and 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.

8.7.7.12. 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

8.7.7.13. suit-directive-run-sequence

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

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

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

Section 8.7.5.22 defaults to False when suit-directive-run-sequence begins. Its value is discarded when suit-directive-run-sequence terminates.

Commented [DT102]: Name?

8.7.7.14. 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

Commented [DT103]: If the destination currently has no contents, does this command fail, or just move the source?

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.

8.7.8. Integrity Check Values

When the CoSWID, Text section, or any Command Sequence of the Update Procedure is made severable, it is moved to the Envelope and replaced with a SUIT_Digest. The SUIT_Digest is computed over the entire bstr enclosing the Manifest element that has been moved to the Envelope. Each element that is made severable from the Manifest is placed in the Envelope with an identical key, so that it matches the key of the corresponding Integrity Check Value.

Commented [DT104]: This sounds like you're saying that all severable elements have the same key

Each Integrity Check Value covers the corresponding Envelope Element as described in Section 8.8.

8.8. Severable Elements

Because the manifest can be used by different actors at different times, some parts of the manifest can be removed or "Severed" without affecting later stages of the lifecycle. 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 integrity 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 NOT 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.

9. 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 Recipient, authenticated by a trusted party or stored on the Recipient. This ACL grants access rights for specific component IDs or component ID prefixes to the listed identities or identity groups. Any identity ~~may~~ can verify an image digest, but fetching into or fetching from a component ID requires approval from the ACL.

Commented [DT105]: "can"? (not MAY, right?)

A third model allows a Recipient 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~~ can use in combination with that component ID.

10. SUIT Digest Container

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

The SUIT digest is a CBOR List containing two elements: a suit-digest-algorithm-id and a bstr containing the bytes of the digest.

11. IANA Considerations

IANA is requested to:

- allocate a CBOR tag for the SUIT Envelope and another for the SUIT Manifest.

Commented [DT106]: Need to follow the guidelines in https://tools.ietf.org/html/rfc8126#section-3.1

- allocate a media type for suit: application/suit-envelope

- set up several registries as described below

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

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

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

Commented [DT107]: Need to follow the guidelines in https://tools.ietf.org/html/rfc8126#section-2.2

Commented [DT108]: Need to follow the guidelines in https://tools.ietf.org/html/rfc8126#section-4.5

11.1. SUIT Commands

Label	Name
1	Vendor Identifier
2	Class Identifier
3	Image Match
4	Use Before
5	Component Offset
12	Set Component Index
13	Set Dependency Index
14	Abort
15	Try Each
16	Reserved
17	Reserved
18	Process Dependency
19	Set Parameters
20	Override Parameters
21	Fetch
22	Copy
23	Run

Commented [DT109]: I'd recommend adding a column for "Reference" so that any IANA values defined elsewhere require a reference. Same for other sub-registries below.

24	Device Identifier
25	Image Not Match
26	Minimum Battery
27	Update Authorized
28	Version
29	Wait For Event
30	Fetch URI List
31	Swap
32	Run Sequence
nint	Custom Condition

Commented [DT110]: What does it mean for this row to appear literally in the IANA registry? Same question for the other sub-registries below.

11.2. SUIT Parameters

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

11.3. SUIT Text Values

Label	Name
1	Manifest Description
2	Update Description
3	Manifest JSON Source
4	Manifest YAML Source
nint	Custom

11.4. SUIT Component Text Values

Label	Name
1	Vendor Name
2	Model Name
3	Vendor Domain
4	Model Info
5	Component Description
6	Component Version
7	Component Version Required
nint	Custom

11.5. SUIT Algorithm Identifiers

11.5.1. SUIT Digest Algorithm Identifiers

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

11.5.2. SUIT Compression Algorithm Identifiers

Label	Name
1	zlib
2	Brotli
3	zstd

11.5.3. Unpack Algorithms

Label	Name
1	HEX
2	ELF
3	COFF
4	SREC

12. Security Considerations

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

13. 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

14. References

14.1. 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>.

Commented [DT111]: "standardized way of delivering" seems wrong to me since SUIT charter is "This group will not define any new transport or discovery mechanisms, but may describe how to use existing mechanisms within the architecture."

That is, this is not necessarily part of a *standard* transport. (In TEEP it may be but in other contexts not.)

Commented [DT112]: This doesn't show up right in the txt version of the I-D

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[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>.

14.2. Informative References

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

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

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

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

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

[I-D.ietf-teep-architecture] Pei, M., Tschofenig, H., Thaler, D., and D. Wheeler, "Trusted Execution Environment Provisioning (TEEP) Architecture", draft-ietf-teep-architecture-11 (work in progress), July 2020.

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

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

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

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

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

[RFC8747] Jones, M., Seitz, L., Selander, G., Erdtman, S., and H. Tschofenig, "Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)", RFC 8747, DOI 10.17487/RFC8747, March 2020, <https://www.rfc-editor.org/info/rfc8747>.

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

14.3. URIs

[1] suit-condition-update-authorized

Commented [DT113]: I don't know what this means, since it's not a URI

A. Full CDDL

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

```
SUIT_Envelope = {
  ? suit-delegation => bstr .cbor SUIT_Delegation,
  ? suit-authentication-wrapper => bstr .cbor SUIT_Authentication,
  suit-manifest  => bstr .cbor SUIT_Manifest,
  SUIT_Severable_Manifest_Members,
  * $$SUIT_Envelope_Extensions,
  (int => bstr)
}

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

CWT = SUIT_Authentication_Block

SUIT_Authentication = [ + 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_Severable_Manifest_Members = (
  ? suit-dependency-resolution => bstr .cbor SUIT_Command_Sequence,
  ? suit-payload-fetch => bstr .cbor SUIT_Command_Sequence,
  ? suit-install => bstr .cbor SUIT_Command_Sequence,
  ? suit-text => bstr .cbor SUIT_Text_Map,
  ? suit-coswid => bstr .cbor concise-software-identity,
  * $$SUIT_severable-members-extensions,
)

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-extensions
]
```

```
; 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          => tstr,
    SUIT_Severable_Members,
    SUIT_Severable_Members_Digests,
    SUIT_Unseverable_Members,
    * $$SUIT_Manifest_Extensions,
}

SUIT_Unseverable_Members = (
  ? suit-validate => bstr .cbor SUIT_Command_Sequence,
  ? suit-load => bstr .cbor SUIT_Command_Sequence,
  ? suit-run => bstr .cbor SUIT_Command_Sequence,
  * $$unserverble-manifest-member-extensions,
)

SUIT_Severable_Members_Digests = (
  ? suit-dependency-resolution-digest => SUIT_Digest,
  ? suit-payload-fetch-digest => SUIT_Digest,
  ? suit-install-digest => SUIT_Digest,
  ? suit-text-digest => SUIT_Digest,
  ? suit-coswid-digest => SUIT_Digest,
  * $$severable-manifest-members-digests-extensions
)

SUIT_Common = {
    ? suit-dependencies           => SUIT_Dependencies,
```

```
    ? suit-components             => SUIT_Components,
    ? suit-common-sequence        => bstr .cbor SUIT_Common_Sequence,
    * $$SUIT_Common-extensions,
}

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

concise-software-identity = any

SUIT_Dependency = {
    suit-dependency-digest => SUIT_Digest,
    ? suit-dependency-prefix => SUIT_Component_Identifier,
    * $$SUIT_Dependency-extensions,
}

SUIT_Component_Identifier =  [* bstr]


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

SUIT_Common_Sequence = [
    + ( SUIT_Condition // SUIT_Common_Commands )
]

SUIT_Common_Commands //= (suit-directive-set-component-index,  uint/bool)
SUIT_Common_Commands //= (suit-directive-set-dependency-index, uint/bool)
SUIT_Common_Commands //= (suit-directive-run-sequence,
    bstr .cbor SUIT_Command_Sequence)
SUIT_Common_Commands //= (suit-directive-try-each,
    SUIT_Directive_Try_Each_Argument)
SUIT_Common_Commands //= (suit-directive-set-parameters,
    {+ SUIT_Parameters})
SUIT_Common_Commands //= (suit-directive-override-parameters,
    {+ SUIT_Parameters})


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, SUIT_Reporting_Policy)
SUIT_Condition //= (suit-condition-class-identifier,  SUIT_Reporting_Policy)
SUIT_Condition //= (suit-condition-device-identifier, SUIT_Reporting_Policy)
```

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

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,   SUIT_Reporting_Policy)
SUIT_Directive //= (suit-directive-set-parameters,
    {+ SUIT_Parameters})
SUIT_Directive //= (suit-directive-override-parameters,
    {+ SUIT_Parameters})
SUIT_Directive //= (suit-directive-fetch,                SUIT_Reporting_Policy)
SUIT_Directive //= (suit-directive-copy,                 SUIT_Reporting_Policy)
SUIT_Directive //= (suit-directive-swap,                 SUIT_Reporting_Policy)
SUIT_Directive //= (suit-directive-run,                  SUIT_Reporting_Policy)
SUIT_Directive //= (suit-directive-wait,                 SUIT_Reporting_Policy)
SUIT_Directive //= (suit-directive-abort,                SUIT_Reporting_Policy)
SUIT_Directive //= (suit-directive-fetch-uri-list,       SUIT_Reporting_Policy)

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

SUIT_Reporting_Policy = uint .bits suit-reporting-bits

suit-reporting-bits = &(
    suit-send-record-success : 0,
    suit-send-record-failure : 1,
    suit-send-sysinfo-success : 2,
    suit-send-sysinfo-failure : 3
)

SUIT_Command_ID /= suit-command-custom
SUIT_Command_ID /= suit-condition-vendor-identifier
SUIT_Command_ID /= suit-condition-class-identifier
SUIT_Command_ID /= suit-condition-image-match
SUIT_Command_ID /= suit-condition-use-before
SUIT_Command_ID /= suit-condition-component-offset
SUIT_Command_ID /= suit-condition-device-identifier
```

```
SUIT_Command_ID /= suit-condition-image-not-match
SUIT_Command_ID /= suit-condition-minimum-battery
SUIT_Command_ID /= suit-condition-update-authorized
SUIT_Command_ID /= suit-condition-version
SUIT_Command_ID /= suit-directive-set-component-index
SUIT_Command_ID /= suit-directive-set-dependency-index
SUIT_Command_ID /= suit-directive-abort
SUIT_Command_ID /= suit-directive-try-each
;SUIT_Command_ID /= suit-directive-do-each
;SUIT_Command_ID /= suit-directive-map-filter
SUIT_Command_ID /= suit-directive-process-dependency
SUIT_Command_ID /= suit-directive-set-parameters
SUIT_Command_ID /= suit-directive-override-parameters
SUIT_Command_ID /= suit-directive-fetch
SUIT_Command_ID /= suit-directive-copy
SUIT_Command_ID /= suit-directive-run
SUIT_Command_ID /= suit-directive-wait
SUIT_Command_ID /= suit-directive-run-sequence
SUIT_Command_ID /= suit-directive-swap
SUIT_Command_ID /= suit-directive-fetch-uri-list

suit-record = {
  suit-record-success             => bool/int,
  ? suit-record-component-id      => SUIT_Component_ID,
  ? suit-record-dependency-id     => SUIT_Digest,
  ? suit-record-command-sequence-id => (
      suit-common-sequence /
      suit-dependency-resolution /
      suit-payload-fetch /
      suit-install /
      suit-validate /
      suit-load /
      suit-run /
      * $$suit-command-sequence-list-extensions
  ),
  ? suit-record-interpeter-offset => uint,
  ? suit-record-command-id        => SUIT_Command_ID,
  ? suit-record-params            => SUIT_Parameters,
  ? suit-record-actual            => SUIT_Parameters,
  * $$suit-record-extensions
}

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: [ + SUIT_Parameter_Version_Match ]
]

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)

SUIT_Parameters //= (suit-parameter-uri-list =>
    bstr .cbor SUIT_URI_List)

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_Info-extensions,
}

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

SUIT_Compression_Algorithm_zlib = 1
SUIT_Compression_Algorithm_brotli = 2
SUIT_Compression_Algorithm_zstd = 3

SUIT_Unpack_Info = {
    suit-unpack-algorithm => SUIT_Unpack_Algorithms,
    * $$SUIT_Unpack_Info-extensions,

}

SUIT_Unpack_Algorithms /= SUIT_Unpack_Algorithm_Hex
SUIT_Unpack_Algorithms /= SUIT_Unpack_Algorithm_Elf
SUIT_Unpack_Algorithms /= SUIT_Unpack_Algorithm_Coff
```

```
SUIT_Unpack_Algorithms /= SUIT_Unpack_Algorithm_Srec

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

SUIT_URI_List = [+ tstr ]

SUIT_Text_Map = {
    ? suit-text-components =>
    [
        + {
            1 => SUIT_Component_Identifier
            SUIT_Text_Component_Keys
        }
    ],
    SUIT_Text_Keys
}

SUIT_Text_Component_Keys = (
    ? suit-text-vendor-name           => tstr,
    ? suit-text-model-name            => tstr,
    ? suit-text-vendor-domain         => tstr,
    ? suit-text-model-info            => tstr,
    ? suit-text-component-description => tstr,
    ? suit-text-component-version     => tstr,
    ? suit-text-version-required      => tstr,
    * $$suit-text-component-key-extensions
)

SUIT_Text_Keys = (
    ? suit-text-manifest-description => tstr,
    ? suit-text-update-description   => tstr,
    ? suit-text-manifest-json-source => tstr,
    ? suit-text-manifest-yaml-source => tstr,
    * $$suit-text-key-extensions
)

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

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-fetch-uri-list           = 30
```

```
suit-directive-swap                    = 31
suit-directive-run-sequence            = 32

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

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

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

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

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

suit-parameter-custom = nint

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

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

suit-text-manifest-description  = 1
suit-text-update-description    = 2
suit-text-manifest-json-source  = 3
suit-text-manifest-yaml-source  = 4
```

```
suit-text-vendor-name           = 1
suit-text-model-name            = 2
suit-text-vendor-domain         = 3
suit-text-model-info            = 4
suit-text-component-description = 5
suit-text-component-version     = 6
suit-text-version-required      = 7
```

B. Examples

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

The examples are signed using the following ECDSA secp256r1 key:

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

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

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

Each example uses SHA256 as the digest function.

Note that reporting policies are declared for each non-flow-control command in these examples. The reporting policies used in the examples are described in the following tables.

Policy	Label
suit-send-record-on-success	Rec-Pass
suit-send-record-on-failure	Rec-Fail
suit-send-sysinfo-success	Sys-Pass
suit-send-sysinfo-failure	Sys-Fail

Command	Sys-Fail	Sys-Pass	Rec-Fail	Rec-Pass
suit-condition-vendor-identifier	1	1	1	1
suit-condition-class-identifier	1	1	1	1
suit-condition-image-match	1	1	1	1
suit-condition-component-offset	0	1	0	1
suit-directive-fetch	0	0	1	0
suit-directive-copy	0	0	1	0
suit-directive-run	0	0	1	0

B.1. Example 0: Secure Boot

This example covers the following templates:

- Compatibility Check (Section 7.1)

- Secure Boot (Section 7.2)

It also serves as the minimum example.

```
{
    / authentication-wrapper / 2:h'81588fd28443a10126a0584482025840356
330393765663634626633626239623439346537316531663234313865656638643436 3
663633930326636333961383535656339616633653965646462393958409334 7ceebc1
209a2d660bfbbe78e461079f1952c614e1ae8f734ff0ea438110d056c1a0cce6b0599d
b54e6704847de49efe60e9a7b821215d83368a2c8c7c088' / [
        h'd28443a10126a0584482025840356330393765663634626633626239623 4
393465373165316632343138656566386434363663393032663633396138353565 63
396166336539656464623939584093347ceebc1209a2d660bfbbe78e461079f1952c61
4e1ae8f734ff0ea438110d056c1a0cce6b0599db54e6704847de49efe60e9a7b821215
d83368a2c8c7c088' / 18([
                / protected / h'a10126' / {
                    / alg / 1:-7 / "ES256" /,
                } /,
                / unprotected / {
                },
```

```
                / payload / h'8202584035633039376566363462663362623962
  343934653731653166323431386565663864343636636339303266363339613835356 5
  633961663365396564646239 39' / [
                    / algorithm-id / 2 / "sha256" /,
                    / digest-bytes / h'3563303937656636346266336262396
  234393465373165316632343138656566386434363663633930326636333961383535 6
  563396166336539656464623939'
                ] /,
                / signature / h'93347ceebc1209a2d660bfbbe78e461079f195
  2c614e1ae8f734ff0ea438110d056c1a0cce6b0599db54e6704847de49efe60e9a7b82
  1215d83368a2c8c7c088'
            ]) /
        ] /,
        / manifest / 3:h'a50101020003585fa202818141000458568614a40150fa6b4
  a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab450358248
  202582000112233445566778899aabbccddeeff0123456789abcdeffedcba987654321
  00e1987d0010f020f0a4382030f0c43821702' / {
            / manifest-version / 1:1,
            / manifest-sequence-number / 2:0,
            / common / 3:h'a202818141000458568614a40150fa6b4a53d5ad5fdfbe9
  de663e4d41ffe02501492af1425695e48bf429b2d51f2ab450358248202582000112233
  3445566778899aabbccddeeff0123456789abcdeffedcba98765432100e1987d0010f0
  20f' / {
                / components / 2:[
                    [h'00']
                ],
                / common-sequence / 4:h'8614a40150fa6b4a53d5ad5fdfbe9de663
  e4d41ffe02501492af1425695e48bf429b2d51f2ab450358248202582000112233445 5
  66778899aabbccddeeff0123456789abcdeffedcba98765432100e1987d0010f020f'
  / [
                    / directive-override-parameters / 20,{
                        / vendor-id /
  1:h'"fa6b4a53d5ad5fdfbe9de663e4d41ffe"' / fa6b4a53-d5ad-5fdf-
  be9d-e663e4d41ffe /,
                        / class-id /
  2:h'"1492af1425695e48bf429b2d51f2ab45"' /
  1492af14-2569-5e48-bf42-9b2d51f2ab45 /,
                        / image-digest / 3:h'8202582000112233445566778899a
  abbccddeeff0123456789abcdeffedcba9876543210' / [
                            / algorithm-id / 2 / "sha256" /,
                            / digest-bytes /
  h'00112233445566778899aabbccddeeff0123456789abcdeffedcba9876543210'
                        ] /,
                        / image-size / 14:34768,
                    } ,
                    / condition-vendor-identifier / 1,15 ,
                    / condition-class-identifier / 2,15
                ] /,
```

```
          } /,
          / validate / 10:h'82030f' / [
              / condition-image-match / 3,15
          ] /,
          / run / 12:h'821702' / [
              / directive-run / 23,2
          ] /,
      } /,
  }
```

Total size of Envelope without COSE authentication object: 117

Envelope:

```
a1035871a50101020003585fa202818141000458568614a40150fa6b4a53
d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab45
0358248202582000112233445566778899aabbccddeeff0123456789abcd
effedcba98765432100e1987d0010f020f0a4382030f0c43821702
```

Total size of Envelope with COSE authentication object: 266

Envelope with COSE authentication object:

```
a202589281588fd28443a10126a0584482025840356330393765663634 62
6633626239623439346537316531663234313865656638643436366363 39
3032663633396138353565633961663365396564646239395840 93347cee
bc1209a2d660bfbbe78e461079f1952c614e1ae8f734ff0ea438110d056c
1a0cce6b0599db54e6704847de49efe60e9a7b821215d83368a2c8c7c088
035871a50101020003585fa202818141000458568614a40150fa6b4a53d5
ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab4503
582482025820001122334455667788 99aabbccddeeff0123456789abcdef
fedcba98765432100e1987d0010f020f0a4382030f0c43821702
```

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

This example covers the following templates:

- Compatibility Check (Section 7.1)

- Firmware Download (Section 7.3)

Simultaneous download and installation of payload. No secure boot is present in this example to demonstrate a download-only manifest.

```
{
    / authentication-wrapper / 2:h'81588fd28443a10126a0584482025840393
837656563383566613939666433316433333233383162393831306693062303563326
530643466323834613666343231313230376564303066666637353058404931df82e15
```

```
  3bf1e3af5a59800216d8a47c33a37839e7d63d9f526fd369aa8359daae18f7619c9591
  23e7f7f928ee92a9893afedd35d06a936d6ed3d5843bf2a' / [
          h'd28443a10126a0584482025840393837656563383566613939666433316 4
  333332333831623938313066393062303563326530643466323834613666343231313 2
  303765643030666666373530584049 31df82e153bf1e3af5a59800216d8a47c33a3783
  9e7d63d9f526fd369aa8359daae18f7619c959123e7f7f928ee92a9893afedd35d06a9
  36d6ed3d5843bf2a' / 18([
                  / protected / h'a10126' / {
                      / alg / 1:-7 / "ES256" /,
                  } /,
                  / unprotected / {
                  },
                  / payload / h'8202584039383765656338356661393966643331
  643333323338316239383130663930623035632653064346632383461366634323131
  323037656430306666666373530' / [
                      / algorithm-id / 2 / "sha256" /,
                      / digest-bytes / h'393837656563383566139396664333
  164333332333831623938313066393062303563326530643466323834613666343231 3
  132303765643030666666373530'
                  ] /,
                  / signature / h'4931df82e153bf1e3af5a59800216d8a47c33a
  37839e7d63d9f526fd369aa8359daae18f7619c959123e7f7f928ee92a9893afedd35d
  06a936d6ed3d5843bf2a'
              ]) /
      ] /,
      / manifest / 3:h'a50101020103585fa202818141000458568614a40150fa6b4
  a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab450358248
  202582000112233445566778899aabbccddeeff0123456789abcdeffedcba987654321
  00e1987d0010f020f0958258613a115781b687474703a2f2f6578616d706c652e636f6
  d2f66696c652e62696e1502030f0a4382030f' / {
          / manifest-version / 1:1,
          / manifest-sequence-number / 2:1,
          / common / 3:h'a202818141000458568614a40150fa6b4a53d5ad5fdfbe9
  de663e4d41ffe02501492af1425695e48bf429b2d51f2ab4503582482025820001122 3
  3445566778899aabbccddeeff0123456789abcdeffedcba98765432100e1987d0010f0
  20f' / {
              / components / 2:[
                  [h'00']
              ],
              / common-sequence / 4:h'8614a40150fa6b4a53d5ad5fdfbe9de663
  e4d41ffe02501492af1425695e48bf429b2d51f2ab450358248202582000112233445 5
  66778899aabbccddeeff0123456789abcdeffedcba98765432100e1987d0010f020f'
  / [
                  / directive-override-parameters / 20,{
                      / vendor-id /
  1:h'"fa6b4a53d5ad5fdfbe9de663e4d41ffe"' / fa6b4a53-d5ad-5fdf-
  be9d-e663e4d41ffe /,
                      / class-id /
```

```
  2:h'"1492af1425695e48bf429b2d51f2ab45"' /
  1492af14-2569-5e48-bf42-9b2d51f2ab45 /,
                      / image-digest / 3:h'8202582000112233445566778899a
  abbccddeeff0123456789abcdeffedcba9876543210' / [
                           / algorithm-id / 2 / "sha256" /,
                           / digest-bytes /
  h'00112233445566778899aabbccddeeff0123456789abcdeffedcba9876543210'
                      ] /,
                      / image-size / 14:34768,
                  } ,
                  / condition-vendor-identifier / 1,15 ,
                  / condition-class-identifier / 2,15
              ] /,
          } /,
          / install / 9:h'8613a115781b687474703a2f2f6578616d706c652e636f
  6d2f66696c652e62696e1502030f' / [
              / directive-set-parameters / 19,{
                  / uri / 21:'http://example.com/file.bin',
              } ,
              / directive-fetch / 21,2 ,
              / condition-image-match / 3,15
          ] /,
          / validate / 10:h'82030f' / [
              / condition-image-match / 3,15
          ] /,
      } /,
  }
```

Total size of Envelope without COSE authentication object: 152

Envelope:

```
a1035894a50101020103585fa202818141000458568614a40150fa6b4a53
d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab45
035824820258200011223344556677889 9aabbccddeeff0123456789abcd
effedcba98765432100e1987d0010f020f0958258613a115781b68747470
3a2f2f6578616d706c652e636f6d2f66696c652e62696e1502030f0a4382
030f
```

Total size of Envelope with COSE authentication object: 301

Envelope with COSE authentication object:

```
   a202589281588fd28443a10126a0584482025840393837656563383566661
   393966643331643333323338316239383130663930623035633265306434
   663238346136663432313132303765643030666666373530584049 31df82
   e153bf1e3af5a59800216d8a47c33a37839e7d63d9f526fd369aa8359daa
   e18f7619c959123e7f7f928ee92a9893afedd35d06a936d6ed3d5843bf2a
   035894a50101020103585fa202818141000458568614a40150fa6b4a53d5
   ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab4503
   582482025820001122334455667788 99aabbccddeeff0123456789abcdef
   fedcba98765432100e1987d0010f020f0958258613a115781b687474703a
   2f2f6578616d706c652e636f6d2f66696c652e62696e1502030f0a438203
   0f
```

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

This example covers the following templates:

- Compatibility Check (Section 7.1)
- Secure Boot (Section 7.2)
- Firmware Download (Section 7.3)

This example also demonstrates severable elements (Section 5.5), and text (Section 8.6.4).

```
  {
     / authentication-wrapper / 2:h'81588fd28443a10126a0584482025840373
  536383535373961383362616264373165633865663232666134396163383733663738
  137303861343361363734653738326164333062363539386431376158 40faca70796c3
  19ce6dae69690a64ced3ab91b9bb7f3e9a5004122d629d2816216a870448424ce4410d
  658b80215185e32d8ec6feb15c7275d64437c36418463e4' / [
          h'd28443a10126a0584482025840373536383535373961383362616264373 1
  656338656632326661343961633837336637386137303861343361363734653738326 1
  643330623635393864313761584 0faca70796c319ce6dae69690a64ced3ab91b9bb7f3
  e9a5004122d629d2816216a870448424ce4410d658b80215185e32d8ec6feb15c7275d
  64437c36418463e4' / 18([
                  / protected / h'a10126' / {
                      / alg / 1:-7 / "ES256" /,
                  } /,
                  / unprotected / {
                  },
                  / payload / h'8202584037353638353537396138336261626437
  316563386566323266613439616338373366373861373038613433613637346537383 2
  616433306236353938643137 61' / [
                      / algorithm-id / 2 / "sha256" /,
                      / digest-bytes / h'373536383535373961383362616264 3
  731656338656632326661343961633837336637386137303861343361363734653738 3
```

```
  261643330623635393864313761'
                ] /,
                / signature / h'faca70796c319ce6dae69690a64ced3ab91b9b
  b7f3e9a5004122d629d2816216a870448424ce4410d658b80215185e32d8ec6feb15c7
  275d64437c36418463e4'
            ]) /
    ] /,
    / manifest / 3:h'a70101020203585fa202818141000458568614a40150fa6b4
  a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab450358248
  202582000112233445566778899aabbccddeeff0123456789abcdeffedcba987654321
  00e1987d0010f020f09820258203ee96dc79641970ae46b929ccf0b72ba9536dd84602
  0dbdc9f949d84ea0e18d20a4382030f0c438217020d8202582023f48b2e2838650f43c
  144234aee18401ffe3cce4733b23881c3a8ae2d2b66e8' / {
        / manifest-version / 1:1,
        / manifest-sequence-number / 2:2,
        / common / 3:h'a202818141000458568614a40150fa6b4a53d5ad5fdfbe9
  de663e4d41ffe02501492af1425695e48bf429b2d51f2ab450358248202582002011223
  3445566778899aabbccddeeff0123456789abcdeffedcba98765432100e1987d0010f0
  20f' / {
            / components / 2:[
                [h'00']
            ],
            / common-sequence / 4:h'8614a40150fa6b4a53d5ad5fdfbe9de663
  e4d41ffe02501492af1425695e48bf429b2d51f2ab450358248202582000112233445 5
  66778899aabbccddeeff0123456789abcdeffedcba98765432100e1987d0010f020f'
  / [
                / directive-override-parameters / 20,{
                    / vendor-id /
  1:h'"fa6b4a53d5ad5fdfbe9de663e4d41ffe"' / fa6b4a53-d5ad-5fdf-
  be9d-e663e4d41ffe /,
                    / class-id /
  2:h'"1492af1425695e48bf429b2d51f2ab45"' /
  1492af14-2569-5e48-bf42-9b2d51f2ab45 /,
                    / image-digest / 3:h'8202582000112233445566778899a
  abbccddeeff0123456789abcdeffedcba9876543210' / [
                        / algorithm-id / 2 / "sha256" /,
                        / digest-bytes /
  h'00112233445566778899aabbccddeeff0123456789abcdeffedcba9876543210'
                    ] /,
                    / image-size / 14:34768,
                } ,
                / condition-vendor-identifier / 1,15—,
                / condition-class-identifier / 2,15
            ] /,
        } /,
        / install / 9:[
            / algorithm-id / 2 / "sha256" /,
            / digest-bytes /
```

```
  h'3ee96dc79641970ae46b929ccf0b72ba9536dd846020dbdc9f949d84ea0e18d2'
          ],
          / validate / 10:h'82030f' / [
              / condition-image-match / 3,15
          ] /,
          / run / 12:h'821702' / [
              / directive-run / 23,2
          ] /,
          / text / 13:[
              / algorithm-id / 2 / "sha256" /,
              / digest-bytes /
  h'23f48b2e2838650f43c144234aee18401ffe3cce4733b23881c3a8ae2d2b66e8'
          ],
      } /,
      / install / 9:h'8613a1157832687474703a2f2f6578616d706c652e636f6d2f
  766572792f6c6f6e672f706174682f746f2f66696c652f66696c652e62696e1502030f
  ' / [
          / directive-set-parameters / 19,{
              / uri /
  21:'http://example.com/very/long/path/to/file/file.bin',
          } ,
          / directive-fetch / 21,2,
          / condition-image-match / 3,15
      ] /,
      / text / 13:h'a1814100a2036761726d2e636f6d0578525468697320636f6d70
  6f6e656e74206973206120646566d6f6e737472617469f6e2e20546865206469676573
  74206973206120736163706c65207061747465726e2c206e6f742061207265616c206f
  6e652e' / {
          [h'00']:{
                  / vendor-domain / 3:'arm.com',
                  / component-description / 5:'This component is a
  demonstration. The digest is a sample pattern, not a real one.',
              }
      } /,
  }
```

Commented [DT114]: Why is there sometimes a space in places like this? Typo?

Commented [DT115]: Shouldn't this say "suit-text-vendor-domain"?
Or at minimum there should be an explanation of how these relate to the actual terms in the doc.

```
   Total size of the Envelope without COSE authentication object or
   Severable Elements: 191

   Envelope:

   a10358bba70101020203585fa202818141000458568614a40150fa6b4a53
   d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab45
   035824820258200011223344556677889 9aabbccddeeff0123456789abcd
   effedcba98765432100e1987d0010f020f09820258203ee96dc79641970a
   e46b929ccf0b72ba9536dd846020dbdc9f949d84ea0e18d20a4382030f0c
   438217020d8202582023f48b2e2838650f43c144234aee18401ffe3cce47
   33b23881c3a8ae2d2b66e8
```

Total size of the Envelope with COSE authentication object but without Severable Elements: 340

Envelope:

```
a202589281588fd28443a10126a0584482025840373536383535373961383362616264373165633865663232666134396163383733663738613730386134336136373465373832616433306236353938643137615840faca70796c319ce6dae69690a64ced3ab91b9bb7f3e9a5004122d629d2816216a870448424ce4410d658b80215185e32d8ec6feb15c7275d64437c36418463e40358bba70101020203585fa202818141000458568614a40150fa6b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab4503582482025820001122334455667788 99aabbccddeeff0123456789abcdeffedcba98765432100e1987d0010f020f09820258203ee96dc79641970ae46b929ccf0b72ba9536dd846020dbdc9f949d84ea0e18d20a4382030f0c438217020d8202582023f48b2e2838650f43c144234aee18401ffe3cce4733b23881c3a8ae2d2b66e8
```

Total size of Envelope with COSE authentication object: 923

Envelope with COSE authentication object:

```
   a402589281588fd28443a10126a0584482025840373536383535373961 38
   3362616264373165633865663232666134396163383733663738613730 38
   6134336136373465373832616433306236353938643137615840faca7079
   6c319ce6dae69690a64ced3ab91b9bb7f3e9a5004122d629d2816216a870
   448424ce4410d658b80215185e32d8ec6feb15c7275d64437c36418463e4
   0358bba70101020203585fa202818141000458568614a40150fa6b4a53d5
   ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab4503
   582482025820001122334455667788 99aabbccddeeff0123456789abcdef
   fedcba98765432100e1987d0010f020f09820258203ee96dc79641970ae4
   6b929ccf0b72ba9536dd846020dbdc9f949d84ea0e18d20a4382030f0c43
   8217020d8202582023f48b2e2838650f43c144234aee18401ffe3cce4733
   b23881c3a8ae2d2b66e809583c8613a1157832687474703a2f2f6578616d
   706c652e636f6d2f766572792f6c6f6e672f706174682f746f2f66696c65
   2f66696c652e62696e1502030f0d590204a20179019d2323204578616d70
   6c6520323a2053696d756c74616e656f757320446f776e6c6f61642c2049
   6e7374616c6c6174696f6e2c2053656375726520426f6f742c2053657665
   726564204669656c64730a0a2020202054686973206578616d706c652063
   6f766572732074686520666f6c6c6f77696e672074656d706c617465733a
   0a202020200a202020202a20436f6d7061746962696c6974792043686563
   6b20287b7b74656d706c6174652d636f6d7061746962696c6974792d6368
   65636b7d7d290a202020202a2053656375726520426f6f7420287b7b7465
   6d706c6174652d7365637572652d626f6f747d7d290a202020202a204669
   726d7761726520446f776e6c6f616420287b7b6669726d776172652d646f
   776e6c6f61642d74656d706c6174657d7d290a202020200a202020205468
   6973206578616d706c6520616c736f2064656d6f6e737472617465732073
   6576657261626c6520656c656d656e747320287b7b6f76722d7365766572
   61626c657d7d292c20616e64207465787420287b7b6d616e69666573742d
   6469676573742d746578747d7d292e814100a2036761726d2e636f6d0578
   525468697320636f6d706f6e656e74206973206120646 56d6f6e73747261
   74696f6e2e2054686520646967657374206973206120736d706c652070
   61747465726e2c206e6f742061207265616c206f6e652e
```

B.4. Example 3: A/B images

This example covers the following templates:

- Compatibility Check (Section 7.1)
- Secure Boot (Section 7.2)
- Firmware Download (Section 7.3)
- A/B Image Template (Section 7.10)

```
  {
      / authentication-wrapper / 2:h'81588fd28443a10126a0584482025840616
  530633165613638396339383030613834333535306633383739366236666462643532 6
  130633738626535643236303131643865373834646134336434376358401022 2ddbce4
```

```
  e82a85f6ec7b72db34d7c5be8d2e822e4b2d099a4cf1d08aa2174c56c2e93bf20c785b
  ca298900208d92d352faf86e6cddc902a726bbc443c21ff' / [
          h'd28443a10126a0584482025840616530633165613638396339383030613838
  343335353066333837393662366664626435326130633738626535643236303131643838
  653738346461343364343763584010222ddbce4e82a85f6ec7b72db34d7c5be8d2e822
  e4b2d099a4cf1d08aa2174c56c2e93bf20c785bca298900208d92d352faf86e6cddc90
  2a726bbc443c21ff' / 18([
                  / protected / h'a10126' / {
                      / alg / 1:-7 / "ES256" /,
                  } /,
                  / unprotected / {
                  },
                  / payload / h'820258406165306331656136383963393830306138
  383433353530663338373936623666646264353261306337386265356432363031316438
  386537383464613433643437763' / [
                      / algorithm-id / 2 / "sha256" /,
                      / digest-bytes / h'6165306331656136383963393830306
  138343335353066333837393662366664626435326130633738626535643236303131643
  8386537383464613433643437763'
                  ] /,
                  / signature / h'10222ddbce4e82a85f6ec7b72db34d7c5be8d2
  e822e4b2d099a4cf1d08aa2174c56c2e93bf20c785bca298900208d92d352faf86e6cd
  dc902a726bbc443c21ff'
              ]) /
      ] /,
      / manifest / 3:h'a5010102030358aaa202818141000458a18814a20150fa6b4
  a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab450f82583
  68614a105198400050514a203582482025820001122334455667788 99aabbccddeeff0
  123456789abcdeffedcba98765432100e1987d0583a8614a1051a00084400050514a20
  3582482025820012345678 9abcdeffedcba9876543210001122334455667788 99aabbc
  cddeeff0e1a00012c22010f020f095861860f82582a8613a105198400050513a115781
  c687474703a2f2f6578616d706c652e636f6d2f66696c65312e62696e582c8613a1051
  a00084400050513a115781c687474703a2f2f6578616d706c652e636f6d2f66696c653
  22e62696e1502030f0a4382030f' / {
          / manifest-version / 1:1,
          / manifest-sequence-number / 2:3,
          / common / 3:h'a202818141000458a18814a20150fa6b4a53d5ad5fdfbe9
  de663e4d41ffe02501492af1425695e48bf429b2d51f2ab450f8258368614a10519840
  0050514a203582482025820001122334455667788 99aabbccddeeff0123456789abcde
  ffedcba98765432100e1987d0583a8614a1051a00084400050514a2035824820258200
  123456789abcdeffedcba98765432100011223344556 6778899aabbccddeeff0e1a000
  12c22010f020f' / {
              / components / 2:[
                  [h'00']
              ],
              / common-sequence / 4:h'8814a20150fa6b4a53d5ad5fdfbe9de663
  e4d41ffe02501492af1425695e48bf429b2d51f2ab450f8258368614a1051984000505
  14a203582482025820001122334455667788 99aabbccddeeff0123456789abcdeffedc
```

```
  ba98765432100e1987d0583a8614a1051a00084400050514a203582482025820012345
  6789abcdeffedcba9876543210001122334455667788 99aabbccddeeff0e1a00012c22
  010f020f' / [
                  / 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'8614a105198400050514a203582482025820001122334455
  66778899aabbccddeeff0123456789abcdeffedcba98765432100e1987d0' / [
                        / directive-override-parameters / 20,{
                           / offset / 5:33792,
                        } ,
                        / condition-component-offset / 5,5 ,
                        / directive-override-parameters / 20,{
                           / image-digest / 3:h'820258200011223344556
  6778899aabbccddeeff0123456789abcdeffedcba9876543210' / [
                                / algorithm-id / 2 / "sha256" /,
                                / digest-bytes /
  h'00112233445566778899aabbccddeeff0123456789abcdeffedcba9876543210'
                           ] /,
                           / image-size / 14:34768,
                        }
                     ] / ,
                     h'8614a1051a00084400050514a203582482025820012345678
  9abcdeffedcba9876543210001122334455667788 99aabbccddeeff0e1a00012c22'
  / [
                        / directive-override-parameters / 20,{
                           / offset / 5:541696,
                        } ,
                        / condition-component-offset / 5,5 ,
                        / directive-override-parameters / 20,{
                           / image-digest / 3:h'820258200123456789abc
  deffedcba9876543210001122334455667788 99aabbccddeeff' / [
                                / algorithm-id / 2 / "sha256" /,
                                / digest-bytes /
  h'0123456789abcdeffedcba9876543210001122334455667788 99aabbccddeeff'
                           ] /,
                           / image-size / 14:76834,
                        }
                     ] /
                  ] ,
                  / condition-vendor-identifier / 1,15 ,
                  / condition-class-identifier / 2,15
```

```
            ] /,
        } /,
        / install / 9:h'860f82582a8613a105198400050513a115781c68747470
  3a2f2f6578616d706c652e636f6d2f66696c65312e62696e582c8613a1051a00084400
  050513a115781c687474703a2f2f6578616d706c652e636f6d2f66696c65322e62696e
  1502030f' / [
            / directive-try-each / 15,[
                h'8613a105198400050513a115781c687474703a2f2f6578616d70
  6c652e636f6d2f66696c65312e62696e' / [
                    / directive-set-parameters / 19,{
                        / offset / 5:33792,
                    } ,
                    / condition-component-offset / 5,5 ,
                    / directive-set-parameters / 19,{
                        / uri / 21:'http://example.com/file1.bin',
                    }
                ] / ,
                h'8613a1051a00084400050513a115781c687474703a2f2f657861
  6d706c652e636f6d2f66696c65322e62696e' / [
                    / directive-set-parameters / 19,{
                        / offset / 5:541696,
                    } ,
                    / condition-component-offset / 5,5 ,
                    / directive-set-parameters / 19,{
                        / uri / 21:'http://example.com/file2.bin',
                    }
                ] /
            ] ,
            / directive-fetch / 21,2 ,
            / condition-image-match / 3,15
        ] /,
        / validate / 10:h'82030f' / [
            / condition-image-match / 3,15
        ] /,
    } /,
}
```

Total size of Envelope without COSE authentication object: 288

Envelope:

```
a10359011ba5010102030358aaa202818141000458a18814a20150fa6b4a
53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab
450f8258368614a105198400050514a203582482025820001122334455 66
778899aabbccddeeff0123456789abcdeffedcba98765432100e1987d058
3a8614a1051a00084400050514a203582482025820012345678 9abcdeffe
dcba9876543210001122334455667788 99aabbccddeeff0e1a00012c2201
0f020f095861860f82582a8613a105198400050513a115781c687474703a
2f2f6578616d706c652e636f6d2f66696c65312e62696e582c8613a1051a
00084400050513a115781c687474703a2f2f6578616d706c652e636f6d2f
66696c65322e62696e1502030f0a4382030f
```

Total size of Envelope with COSE authentication object: 437

Envelope with COSE authentication object:

```
a202589281588fd28443a10126a0584482025840616530633165613638 39
6339383030613834333535306633383739366236666462643532613063 37
3862653564323630313164386537383464613433643437635840102 22ddb
ce4e82a85f6ec7b72db34d7c5be8d2e822e4b2d099a4cf1d08aa2174c56c
2e93bf20c785bca298900208d92d352faf86e6cddc902a726bbc443c21ff
0359011ba5010102030358aaa202818141000458a18814a20150fa6b4a53
d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab45
0f8258368614a105198400050514a203582482025820001122334455667 7
8899aabbccddeeff0123456789abcdeffedcba98765432100e1987d0583a
8614a1051a00084400050514a203582482025820012345678 9abcdeffedc
ba9876543210001122334455667788 99aabbccddeeff0e1a00012c22010f
020f095861860f82582a8613a105198400050513a115781c687474703a2f
2f6578616d706c652e636f6d2f66696c65312e62696e582c8613a1051a00
084400050513a115781c687474703a2f2f6578616d706c652e636f6d2f66
696c65322e62696e1502030f0a4382030f
```

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

This example covers the following templates:

- Compatibility Check (Section 7.1)
- Secure Boot (Section 7.2)
- Firmware Download (Section 7.3)
- Install (Section 7.4)
- Load & Decompress (Section 7.7)

```
{
    / authentication-wrapper / 2:h'81588fd28443a10126a0584482025840346
  2346337633863306664613736633963393539316139646231363039313865326233633
```

```
  936613538623061356534393834666434653866393335396139323858 40d7063361f65
  3d57e63691e1bd9c856058c773b94e488bff58d599c45277788e90eb92fbef666f584e
  8d35b3b20ceef50a69b94dcff12beee92e426a06ea31320' / [
          h'd28443a10126a0584482025840346234633763386330666461373663396 3
  393539316139646231363039313865326233633936613538623061356534393834666 4
  346538663933353961393238 5840d7063361f653d57e63691e1bd9c856058c773b94e4
  88bff58d599c45277788e90eb92fbef666f584e8d35b3b20ceef50a69b94dcff12beee
  92e426a06ea31320' / 18([
                  / protected / h'a10126' / {
                      / alg / 1:-7 / "ES256" /,
                  } /,
                  / unprotected / {
                  },
                  / payload / h'8202584034623463376338633066646137366339
  6339353931613964623136303931386532623363396613538623061356534393834666
  643465386639333539613932 38' / [
                      / algorithm-id / 2 / "sha256" /,
                      / digest-bytes / h'346234633763386330666461373663 3
  963393539316139646231363039313865326233633936613538623061356534393834 6
  664346538663933353961393238'
                  ] /,
                  / signature / h'd7063361f653d57e63691e1bd9c856058c773b
  94e488bff58d599c45277788e90eb92fbef666f584e8d35b3b20ceef50a69b94dcff12
  beee92e426a06ea31320'
              ]) /
      ] /,
      / manifest / 3:h'a801010204035867a202838141008141028141010458588 80
  c0014a40150fa6b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2
  d51f2ab450358248202582000112233445566778899aabbccddeeff0123456789abcde
  ffedcba98765432100e1987d0010f020f085827880c0113a115781b687474703a2f2f6
  578616d706c652e636f6d2f66696c652e62696e1502030f094b880c0013a1160116020
  30f0a45840c00030f0b583a880c0213a4035824820258200123456789abcdeffedcba9
  876543210001122334455667788 99aabbccddeeff0e1a00012c2213011600160203 0f0
  c45840c021702' / {
          / manifest-version / 1:1,
          / manifest-sequence-number / 2:4,
          / common / 3:h'a202838141008141028141010458588 80c0014a40150fa6
  b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab4503582
  482025820001122334455667788 99aabbccddeeff0123456789abcdeffedcba9876543
  2100e1987d0010f020f' / {
              / components / 2:[
                  [h'00'] ,
                  [h'02'] ,
                  [h'01']
              ],
              / common-sequence / 4:h'880c0014a40150fa6b4a53d5ad5fdfbe9d
  e663e4d41ffe02501492af1425695e48bf429b2d51f2ab450358248202582000112233
  445566778899aabbccddeeff0123456789abcdeffedcba98765432100e1987d0010f02
```

```
  0f' / [
                / directive-set-component-index / 12,0 ,
                / directive-override-parameters / 20,{
                    / vendor-id /
  1:h'"fa6b4a53d5ad5fdfbe9de663e4d41ffe"' / fa6b4a53-d5ad-5fdf-
  be9d-e663e4d41ffe /,
                    / class-id /
  2:h'"1492af1425695e48bf429b2d51f2ab45"' /
  1492af14-2569-5e48-bf42-9b2d51f2ab45 /,
                    / image-digest / 3:h'8202582000112233445566778899a
  abbccddeeff0123456789abcdeffedcba9876543210' / [
                        / algorithm-id / 2 / "sha256" /,
                        / digest-bytes /
  h'00112233445566778899aabbccddeeff0123456789abcdeffedcba9876543210'
                    ] /,
                    / image-size / 14:34768,
                } ,
                / condition-vendor-identifier / 1,15 ,
                / condition-class-identifier / 2,15
            ] /,
        } /,
        / payload-fetch / 8:h'880c0113a115781b687474703a2f2f6578616d70
  6c652e636f6d2f66696c652e62696e1502030f' / [
            / directive-set-component-index / 12,1 ,
            / directive-set-parameters / 19,{
                / uri / 21:'http://example.com/file.bin',
            } ,
            / directive-fetch / 21,2 ,
            / condition-image-match / 3,15
        ] /,
        / install / 9:h'880c0013a116011602030f' / [
            / directive-set-component-index / 12,0 ,
            / directive-set-parameters / 19,{
                / source-component / 22:1 / [h'02'] /,
            } ,
            / directive-copy / 22,2 ,
            / condition-image-match / 3,15
        ] /,
        / validate / 10:h'840c00030f' / [
            / directive-set-component-index / 12,0 ,
            / condition-image-match / 3,15
        ] /,
        / load / 11:h'880c0213a4035824820258200123456789abcdeffedcba98
  76543210001122334455667788 99aabbccddeeff0e1a00012c22130116001602030f'
  / [
            / directive-set-component-index / 12,2 ,
            / directive-set-parameters / 19,{
                / image-digest / 3:h'820258200123456789abcdeffedcba987
```

```
  6543210001122334455667788 99aabbccddeeff' / [
                      / algorithm-id / 2 / "sha256" /,
                      / digest-bytes /
  h'0123456789abcdeffedcba9876543210001122334455667788 99aabbccddeeff'
                  ] /,
                  / image-size / 14:76834,
                  / source-component / 22:0 / [h'00'] /,
                  / compression-info / 19:1 / "gzip" /,
              } ,
              / directive-copy / 22,2 ,
              / condition-image-match / 3,15
          ] /,
          / run / 12:h'840c021702' / [
              / directive-set-component-index / 12,2 ,
              / directive-run / 23,2
          ] /,
      } /,
  }
```

Total size of Envelope without COSE authentication object: 245

Envelope:

```
a10358f1a801010204035867a2028381410081410281410104585 8880c00
14a40150fa6b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48
bf429b2d51f2ab4503582482025820001122334455667788 99aabbccddee
ff0123456789abcdeffedcba98765432100e1987d0010f020f085827880c
0113a115781b687474703a2f2f6578616d706c652e636f6d2f66696c652e
62696e1502030f094b880c0013a116011602030f0a45840c00030f0b583a
880c0213a4035824820258200123456789abcdeffedcba98765432100011
2233445566778899aabbccddeeff0e1a00012c2213011600160 2030f0c45
840c021702
```

Total size of Envelope with COSE authentication object: 394

Envelope with COSE authentication object:

Commented [DT116]: Section 11.5.1 defines 2 as "SHA256" not "sha256". Be consistent throughout.

Commented [DT117]: Section 11.5.2 defines 1 as "zlib" not "gzip", this looks like a bug

```
   a202589281588fd28443a10126a0584482025840346234633763386330666
   6461373663396339353931613964623136303931386532623363393661356
   3862306135653439383466643465386639333539613932385840d7063361
   f653d57e63691e1bd9c856058c773b94e488bff58d599c45277788e90eb9
   2fbef666f584e8d35b3b20ceef50a69b94dcff12beee92e426a06ea31320
   0358f1a801010204035867a202838141008141028141010458588800c0014
   a40150fa6b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf
   429b2d51f2ab45035824820258200011223344556677889 9aabbccddeeff
   0123456789abcdeffedcba98765432100e1987d0010f020f085827880c01
   13a115781b687474703a2f2f6578616d706c652e636f6d2f66696c652e62
   696e1502030f094b880c0013a116011602030f0a45840c00030f0b583a88
   0c0213a4035824820258200123456789abcdeffedcba9876543210001122
   33445566778899aabbccddeeff0e1a00012c22130116001602030f0c4584
   0c021702
```

B.6. Example 5: Two Images

This example covers the following templates:

- Compatibility Check (Section 7.1)

- Secure Boot (Section 7.2)

- Firmware Download (Section 7.3)

Furthermore, it shows using these templates with two images.

```
  {
      / authentication-wrapper / 2:h'81588fd28443a10126a0584482025840323
  1306231323835306332333930393164386538326330653965393130363632623638616
  3383432343538613634313865333366363730316564353833343263584 0b5b8cb30c2b
  bb646c4d32426d72768668d6d6af54c26ac46c4020ca37ada47b9468340b4d0b2ddd15
  db824a7e6b0bc233e753940dfb7131fa145ddc456da3cf6' / [
          h'd28443a10126a0584482025840323130623132383530632333930393164
  3865383263306539653931303636326236386163383432343538613634313865333366
  3637303165643538333432635840b5b8cb30c2bbb646c4d32426d72768668d6d6af54c
  26ac46c4020ca37ada47b9468340b4d0b2ddd15db824a7e6b0bc233e753940dfb7131f
  a145ddc456da3cf6' / 18([
                  / protected / h'a10126' / {
                      / alg / 1:-7 / "ES256" /,
                  } /,
                  / unprotected / {
                  },
                  / payload / h'8202584032313062313238353063323339303931
  6438653832633065396539313036363262363861633834323435386136343138653333
  6636373031656435383343263' / [
                      / algorithm-id / 2 / "sha256" /,
                      / digest-bytes / h'323130623132383530633233393039 3
```

```
  164386538326330653965393130363632623638616338343234353861363431386533
  366363730316564353833343263'
                  ] /,
                  / signature / h'b5b8cb30c2bbb646c4d32426d72768668d6d6a
  f54c26ac46c4020ca37ada47b9468340b4d0b2ddd15db824a7e6b0bc233e753940dfb7
  131fa145ddc456da3cf6'
              ]) /
      ] /,
      / manifest / 3:h'a601010205035895a2028281410081410104588 98c0c0014a
  40150fa6b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2a
  b450358248202582000112233445566778899aabbccddeeff0123456789abcdeffedcb
  a98765432100e1987d0010f020f0c0114a203582482025820012345 6789abcdeffedcb
  a9876543210001122334455667788 99aabbccddeeff0e1a00012c2209584f900c0013a
  115781c687474703a2f2f6578616d706c652e636f6d2f66696c65312e62696e1502030
  f0c0113a115781c687474703a2f2f6578616d706c652e636f6d2f66696c65322e62696
  e1502030f0a49880c00030f0c01030f0c47860c0017021702' / {
          / manifest-version / 1:1,
          / manifest-sequence-number / 2:5,
          / common / 3:h'a202828141008141010458898c0c0014a40150fa6b4a53d
  5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b2d51f2ab4503582482025
  8200011223344556677889 9aabbccddeeff0123456789abcdeffedcba98765432100e1
  987d0010f020f0c0114a203582482025820012345 6789abcdeffedcba9876543210001
  12233445566778899aabbccddeeff0e1a00012c22' / {
              / components / 2:[
                  [h'00'] ,
                  [h'01']
              ],
              / common-sequence / 4:h'8c0c0014a40150fa6b4a53d5ad5fdfbe9d
  e663e4d41ffe02501492af1425695e48bf429b2d51f2ab4503582482025820001122 33
  445566778899aabbccddeeff0123456789abcdeffedcba98765432100e1987d0010f02
  0f0c0114a203582482025820012345 6789abcdeffedcba9876543210001122334455 66
  778899aabbccddeeff0e1a00012c22' / [
                  / directive-set-component-index / 12,0 ,
                  / directive-override-parameters / 20,{
                      / vendor-id /
  1:h'"fa6b4a53d5ad5fdfbe9de663e4d41ffe"' / fa6b4a53-d5ad-5fdf-
  be9d-e663e4d41ffe /,
                      / class-id /
  2:h'"1492af1425695e48bf429b2d51f2ab45"' /
  1492af14-2569-5e48-bf42-9b2d51f2ab45 /,
                      / image-digest / 3:h'8202582000112233445566778899a
  abbccddeeff0123456789abcdeffedcba9876543210' / [
                          / algorithm-id / 2 / "sha256" /,
                          / digest-bytes /
  h'00112233445566778899aabbccddeeff0123456789abcdeffedcba9876543210'
                      ] /,
                      / image-size / 14:34768,
                  } ,
```

```
                / condition-vendor-identifier / 1,15 ,
                / condition-class-identifier / 2,15 ,
                / directive-set-component-index / 12,1 ,
                / directive-override-parameters / 20,{
                    / image-digest / 3:h'82025820012345678 9abcdeffedcb
  a9876543210001122334455667788 99aabbccddeeff' / [
                       / algorithm-id / 2 / "sha256" /,
                       / digest-bytes /
  h'0123456789abcdeffedcba9876543210001122334455667788 99aabbccddeeff'
                    ] /,
                    / image-size / 14:76834,
                }
            ] /,
        } /,
        / install / 9:h'900c0013a115781c687474703a2f2f6578616d706c652e
  636f6d2f66696c65312e62696e1502030f0c0113a115781c687474703a2f2f6578616d
  706c652e636f6d2f66696c65322e62696e1502030f' / [
            / directive-set-component-index / 12,0 ,
            / directive-set-parameters / 19,{
                / uri / 21:'http://example.com/file1.bin',
            } ,
            / directive-fetch / 21,2 ,
            / condition-image-match / 3,15 ,
            / directive-set-component-index / 12,1 ,
            / directive-set-parameters / 19,{
                / uri / 21:'http://example.com/file2.bin',
            } ,
            / directive-fetch / 21,2 ,
            / condition-image-match / 3,15
        ] /,
        / validate / 10:h'880c00030f0c01030f' / [
            / directive-set-component-index / 12,0 ,
            / condition-image-match / 3,15 ,
            / directive-set-component-index / 12,1 ,
            / condition-image-match / 3,15
        ] /,
        / run / 12:h'860c0017021702' / [
            / directive-set-component-index / 12,0 ,
            / directive-run / 23,2 ,
            / directive-run / 23,2
        ] /,
    } /,
}
```

Total size of Envelope without COSE authentication object: 264

Envelope:

```
a103590103a601010205035895a2028281410081410104588 98c0c0014a4
0150fa6b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf42
9b2d51f2ab450358248202582000112233445566778899aabbccddeeff01
23456789abcdeffedcba98765432100e1987d0010f020f0c0114a2035824
8202582001234567 89abcdeffedcba9876543210001122334455667788 99
aabbccddeeff0e1a00012c2209584f900c0013a115781c687474703a2f2f
6578616d706c652e636f6d2f66696c65312e62696e1502030f0c0113a115
781c687474703a2f2f6578616d706c652e636f6d2f66696c65322e62696e
1502030f0a49880c00030f0c01030f0c47860c0017021702
```

Total size of Envelope with COSE authentication object: 413

Envelope with COSE authentication object:

```
a202589281588fd28443a10126a0584482025840323130623132383 53063
3233393039316438653832633065396539313036362622636386163383432
3435386136343138653333663637303165643538333432635840b5b8cb30
c2bbb646c4d32426d72768668d6d6af54c26ac46c4020ca37ada47b94683
40b4d0b2ddd15db824a7e6b0bc233e753940dfb7131fa145ddc456da3cf6
03590103a601010205035895a2028281410081410104588 98c0c0014a401
50fa6b4a53d5ad5fdfbe9de663e4d41ffe02501492af1425695e48bf429b
2d51f2ab450358248202582000112233445566778899aabbccddeeff0123
456789abcdeffedcba98765432100e1987d0010f020f0c0114a203582482
0258200123456789abcdeffedcba9876543210001122334455667788 99aa
bbccddeeff0e1a00012c2209584f900c0013a115781c687474703a2f2f65
78616d706c652e636f6d2f66696c65312e62696e1502030f0c0113a11578
1c687474703a2f2f6578616d706c652e636f6d2f66696c65322e62696e15
02030f0a49880c00030f0c01030f0c47860c0017021702
```

C. Design Rationale

Commented [DT118]: typo

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.

C.1. ~~C.1~~ Design Rationale: Envelope

Commented [DT119]: redundant

The Envelope is used instead of a COSE structure for several reasons:

1. This enables the use of Severable Elements (Section 8.8)

2. This enables modular processing of manifests, particularly with large signatures.

3. This enables multiple authentication schemes.

4. This allows integrity verification by a dependent to be unaffected by adding or removing authentication structures.

Modular processing is important because it allows a Manifest Processor to iterate forward over an Envelope, processing Delegation Chains and Authentication Blocks, retaining only intermediate values, without any need to seek forward and backwards in a stream until it gets to the Manifest itself. This allows the use of large, Post-Quantum signatures without requiring retention of the signature itself, or seeking forward and backwards.

Four authentication objects are supported by the Envelope:

- COSE_Sign_Tagged
- COSE_Sign1_Tagged
- COSE_Mac_Tagged
- COSE_Mac0_Tagged

The SUIT Envelope allows an Update Authority or intermediary to mix and match any number of different authentication blocks it wants without any concern for modifying the integrity of another authentication block. This also allows the addition or removal of an authentication blocks without changing the integrity check of the Manifest, which is important for dependency handling. See Section 6.2.

Commented [DT120]: bad grammar (“an …blocks”)

C.2. ~~C.2~~ Byte String Wrappers

Byte string wrappers are used in several places in the ~~suit~~ SUIT manifest. The primary reason for wrappers ~~it~~ is to limit the parser extent when invoked at different times, with a possible loss of context.

The elements of the suit envelope are wrapped both to set the extents used by the parser and to simplify integrity checks by clearly defining the length of each element.

The common block is re-parsed in order to find components identifiers from their indices, to find dependency prefixes and digests from their identifiers, and to find the common sequence. The common sequence is wrapped so that it matches other sequences, simplifying the code path.

Commented [DT121]: remove s? or add apostrophe if possessive?

A severed SUIT command sequence will appear in the envelope, so it must be wrapped as with all envelope elements. For consistency, command sequences are also wrapped in the manifest. This also allows

the parser to discern the difference between a command sequence and a SUIT_Digest.

Parameters that are structured types (arrays and maps) are also wrapped in a bstr. This is so that parser extents can be set correctly using only a reference to the beginning of the parameter. This enables a parser to store a simple list of references to parameters that can be retrieved when needed.

D. Implementation Conformance Matrix

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

Commented [DT122]: The same table applies to both a manifest processor and a manifest generator?

The subsequent table shows the conditions.

Name	Reference	Implementation
Vendor Identifier	Section 8.7.5.1	REQUIRED
Class Identifier	Section 8.7.5.1	REQUIRED
Device Identifier	Section 8.7.5.1	OPTIONAL
Image Match	Section 8.7.6.2	REQUIRED
Image Not Match	Section 8.7.6.3	OPTIONAL
Use Before	Section 8.7.6.4	OPTIONAL
Component Offset	Section 8.7.6.5	OPTIONAL
Minimum Battery	Section 8.7.6.6	OPTIONAL
Update Authorized	Section 8.7.6.7	OPTIONAL
Version	Section 8.7.6.8	OPTIONAL
Custom Condition	Section 8.7.6.9	OPTIONAL

The subsequent table shows the directives.

Name	Reference	Implementation
Set Component Index	Section 8.7.7.1	REQUIRED if more than one component
Set Dependency Index	Section 8.7.7.2	REQUIRED if dependencies used
Abort	Section 8.7.7.3	OPTIONAL
Try Each	Section 8.7.7.4	OPTIONAL
Process Dependency	Section 8.7.7.5	OPTIONAL
Set Parameters	Section 8.7.7.6	OPTIONAL
Override Parameters	Section 8.7.7.7	REQUIRED
Fetch	Section 8.7.7.8	REQUIRED for Updater
Copy	Section 8.7.7.10	OPTIONAL
Run	Section 8.7.7.11	REQUIRED for Bootloader
Wait For Event	Section 8.7.7.12	OPTIONAL
Run Sequence	Section 8.7.7.13	OPTIONAL
Swap	Section 8.7.7.14	OPTIONAL
Fetch URI List	Section 8.7.7.9	OPTIONAL

The subsequent table shows the parameters.

Name	Reference	Implementation
Vendor ID	Section 8.7.5.2	REQUIRED
Class ID	Section 8.7.5.3	REQUIRED
Image Digest	Section 8.7.5.5	REQUIRED
Image Size	Section 8.7.5.6	REQUIRED
Use Before	Section 8.7.5.7	RECOMMENDED
Component Offset	Section 8.7.5.8	OPTIONAL
Encryption Info	Section 8.7.5.9	RECOMMENDED
Compression Info	Section 8.7.5.10	RECOMMENDED
Unpack Info	Section 8.7.5.11	RECOMMENDED
URI	Section 8.7.5.12	REQUIRED for Updater
Source Component	Section 8.7.5.13	OPTIONAL
Run Args	Section 8.7.5.14	OPTIONAL
Device ID	Section 8.7.5.4	OPTIONAL
Minimum Battery	Section 8.7.5.15	OPTIONAL
Update Priority	Section 8.7.5.16	OPTIONAL
Version Match	Section 8.7.5.17	OPTIONAL
Wait Info	Section 8.7.5.18	OPTIONAL
URI List	Section 8.7.5.19	OPTIONAL
Strict Order	Section 8.7.5.21	OPTIONAL
Soft Failure	Section 8.7.5.22	OPTIONAL
Custom	Section 8.7.5.23	OPTIONAL

Authors' Addresses

Brendan Moran
Arm Limited

EMail: Brendan.Moran@arm.com

Hannes Tschofenig
Arm Limited

EMail: hannes.tschofenig@arm.com

Henk Birkholz
Fraunhofer SIT

EMail: henk.birkholz@sit.fraunhofer.de

Koen Zandberg
Inria

EMail: koen.zandberg@inria.fr