SACM Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 6, 2018

H. Birkholz
Fraunhofer SIT
J. Fitzgerald-McKay
Department of Defense
C. Schmidt
The MITRE Corporation
D. Waltermire
NIST

March 05, 2018

Concise Software Identifiers
draft-ietf-sacm-coswid-04

Abstract

This document defines a concise representation of ISO/IEC 19770-2:2015 Software Identifiers (SWID tags) that is interoperable with the XML schema definition of ISO/IEC 19770-2:2015 and augmented for application in Constrained-Node Networks. Next to the inherent capability of SWID tags to express arbitrary context information, CoSWID support the definition of additional semantics via well-defined data definitions incorporated by extension points.

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 September 6, 2018.

Table of Contents

1. Introduction

SWID tags have several use-applications including but not limited to:

- Software Inventory Management, a part of the Software Asset Management [SAM] process, which requires an accurate list of discernible deployed software components.

- Vulnerability Assessment, which requires a semantic link between standardized vulnerability descriptions and IT-assets [X.1520].

- Remote Attestation, which requires a link between reference integrity measurements (RIM) and security logs of measured software components [I-D.birkholz-tuda].

SWID tags, as defined in ISO-19770-2:2015 [SWID], provide a standardized format for a record that identifies and describes a specific release of a software component. Different software components, and even different releases of a particular software component, each have a different SWID tag record associated with them. SWID tags are meant to be flexible and able to express a broad set of metadata about a software component.

Real-world instances of SWID tags can be fairly large, and the communication of SWID tags in use-applications such as those described earlier can cause a large amount of data to be transported. This can be larger than acceptable for constrained devices and networks. CoSWID tags significantly reduce the amount of data transported as compared to a typical SWID tag. This reduction is enable through the use of CBOR, which maps human-readable labels of that content to more concise integer labels (indices). This allows SWID tags to be part of an enterprise security solution for a wider range of endpoints and environments.

1.1. The SWID Tag Lifecycle

In addition to defining the format of these records, ISO/IEC 19770-2:2015 defines requirements concerning the SWID tag life-cycle. Specifically, when a software component is installed on an endpoint, that product's SWID tag is also installed. Likewise, when the product is uninstalled or replaced, the SWID tag is deleted or replaced, as appropriate. As a result, ISO/IEC 19770-2:2015 describes a system wherein there is a correspondence between the set of installed software products on an endpoint, and the presence on that endpoint of the SWID tags corresponding to those products.

The following is an excerpt (with some modifications and reordering) from NIST Interagency Report (NISTIR) 8060: Guidelines for the

Creation of Interoperable SWID Tags [SWID-GUIDANCE], which describes the tag types used within the lifecycle defined in ISO-19770-2:2015.

The SWID specification defines four types of SWID tags: primary, patch, corpus, and supplemental.

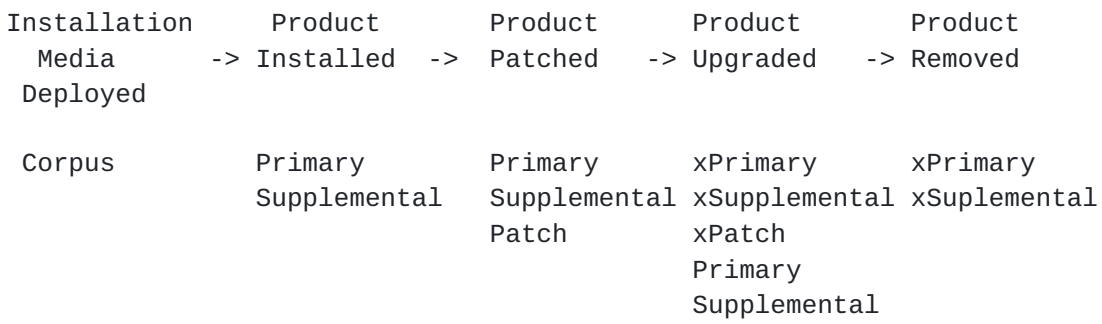
1. Primary Tag - A SWID tag that identifies and describes a software component is installed on a computing device.

2. Patch Tag - A SWID tag that identifies and describes an installed patch which has made incremental changes to a software component installed on a computing device.

3. Corpus Tag - A SWID tag that identifies and describes an installable software component in its pre-installation state. A corpus tag can be used to represent metadata about an installation package or installer for a software component, a software update, or a patch.

4. Supplemental Tag - A SWID tag that allows additional information to be associated with a referenced SWID tag. This helps to ensure that SWID Primary and Patch Tags provided by a software provider are not modified by software management tools, while allowing these tools to provide their own software metadata.

Corpus, primary, and patch tags have similar functions in that they describe the existence and/or presence of different types of software (e.g., software installers, software installations, software patches), and, potentially, different states of software components. In contrast, supplemental tags furnish additional information not contained in corpus, primary, or patch tags. All four tag types come into play at various points in the software lifecycle, and support software management processes that depend on the ability to accurately determine where each software component is in its lifecycle.



The figure above illustrates the steps in the software lifecycle and the relationships among those lifecycle events supported by the four types of SWID tags, as follows: - Software Deployment. Before the software component is installed (i.e., pre-installation), and while the product is being deployed, a corpus tag provides information about the installation files and distribution media (e.g., CD/DVD, distribution package). - Software Installation. A primary tag will be installed with the software component (or subsequently created) to uniquely identify and describe the software component. Supplemental tags are created to augment primary tags with additional site-specific or extended information. While not illustrated in the figure, patch tags may also be installed during software installation to provide information about software fixes deployed along with the base software installation. - Software Patching. When a new patch is applied to the software component, a new patch tag is provided, supplying details about the patch and its dependencies. While not illustrated in the figure, a corpus tag can also provide information about the patch installer, and patching dependencies that need to be installed before the patch. - Software Upgrading. As a software component is upgraded to a new version, new primary and supplemental tags replace existing tags, enabling timely and accurate tracking of updates to software inventory. While not illustrated in the figure, a corpus tag can also provide information about the upgrade installer, and dependencies that need to be installed before the upgrade. - Software Removal. Upon removal of the software component, relevant SWID tags are removed. This removal event can trigger timely updates to software inventory reflecting the removal of the product and any associated patch or supplemental tags.

Note: While not fully illustrated in the figure, supplemental tags can be associated with any corpus, primary, or patch tag to provide additional metadata about an installer, installed software, or installed patch respectively.

Each of the different SWID tag types of SWID tags provide different types of information. For example, a "corpus tag" is used to describe an application's installation image on an installation media, while a "patch tag" is meant to describe a patch that modifies some other application. While there are very few required fields in SWID tags, there are many optional fields that support different uses of these different types of tags. While a SWID tag that consisted only of required fields could be a few hundred bytes in size, a tag containing many of the optional fields could be many orders of magnitude larger.

1.2. Concise SWID Extensions

This document defines a more concise representation of SWID tags in the Concise Binary Object Representation (CBOR) [RFC7049]. This is described via the Concise Data Definition Language (CDDL) [I-D.greevenbosch-appsawg-cbor-cddl]. The resulting Concise SWID data definition is interoperable with the XML schema definition of ISO-19770-2:2015 [SWID]. The vocabulary, i.e., the CDDL names of the types and members used in the CoSWID data definition, is mapped to more concise labels represented as small integers. The names used in the CDDL data definition and the mapping to the CBOR representation using integer labels is based on the vocabulary of the XML attribute and element names defined in ISO/IEC 19770-2:2015.

This document specifies a standardized equivalent to the ISO-19770-2:2015 standard. The corresponding CoSWID data definition includes two kinds of augmentation.

- the explicit definition of types for attributes that are typically stored in the "any attribute" of an ISO-19770-2:2015 in XML representation. These are covered in the main body of this document.

- the inclusion of extension points in the CoSWID data definition that allow for additional uses of CoSWID tags that go beyond the original scope of ISO-19770-2:2015 tags. These are covered in appendices to this document.

1.3. Requirements Notation

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 RFC 2119, BCP 14 [RFC2119].

2. Concise SWID Data Definition

The following is a CDDL representation of the ISO/IEC 19770-2:2015 [SWID] XML schema definition of SWID tags. This representation includes every SWID tag field and attribute and thus supports all SWID tag use cases. The CamelCase notation used in the XML schema definition is changed to a hyphen-separated notation (e.g. ResourceCollection is named resource-collection in the CoSWID data definition). This deviation from the original notation used in the XML representation reduces ambiguity when referencing certain attributes in corresponding textual descriptions. An attribute referred by its name in CamelCase notation explicitly relates to XML SWID tags, an attribute referred by its name in hyphen-separated notation explicitly relates to CoSWID tags. This approach simplifies the composition of further work that reference both XML SWID and CoSWID documents.

Human-readable names of members in the CDDL data definition are mapped to integer indices via a block of rules at the bottom of the definition. The 67 character strings of the SWID vocabulary that would have to be stored or transported in full if using the original vocabulary are replaced.

Concise Software Identifiers are tailored to be used in the domain of constrained-node networks. A typical endpoint is capable of storing the CoSWID tag of installed software, a constrained-node might lack that capability. CoSWID address these constraints and the corresponding specification is augmented to retain their usefulness in the thing-2-thing domain. Specific examples include, but are not limited to limiting the scope of hash algorithms to the IANA Named Information tables or including firmware attributes addressing devices that do not necessarily provide a file-system to store a CoSWID tag in.

The following subsections describe the different parts of the CoSWID model.

2.1. The concise-software-identity Object

The CDDL for the main concise-software-identity object is as follows:

```
<CODE BEGINS>
concise-software-identity = {
  global-attributes,
  tag-id,
  tag-version,
  ? corpus,
  ? patch,
  ? supplemental,
  swid-name,
  ? software-version,
  ? version-scheme,
  ? media,
  ? software-meta-entry,
  ? entity-entry,
  ? link-entry,
  ? ( payload-entry / evidence-entry ),
  ? any-element-entry,
}
tag-id = (0: text)
swid-name = (1: text)
entity-entry = (2: entity / [ 2* entity ])
evidence-entry = (3: evidence)
link-entry = (4: link / [ 2* link ])
software-meta-entry = (5: software-meta / [ 2* software-meta ])
payload-entry = (6: payload)
any-element-entry = (7: any-element-map / [ 2* any-element-map ])
corpus = (8: bool)
patch = (9: bool)
media = (10: text)
supplemental = (11: bool)
tag-version = (12: integer)
software-version = (13: text)
version-scheme = (14: text)
<CODE ENDS>
```

The items are ordered ensure that tag metadata appears first, followed by general software metadata, entity information, link relations, and finally payload or evidence data. This ordering attempts to provide the most significant metadata that a parser may need first, followed by metadata that may support more specific use-applications. The following describes each child item of the concise-software-identity model.

- global-attributes: A list of items including an optional language definition to support the processing of text-string values and an unbounded set of any-attribute items. Described in Section 2.2.

- tag-id (label 0): An textual identifier uniquely referencing a (composite) software component. The tag identifier is intended to be globally unique. There are no strict guidelines on how this identifier is structured, but examples include a 16 byte GUID (e.g. class 4 UUID).

- tag-version (label 12): An integer value that indicates if a specific release of a software component has more than one tag that can represent that specific release. A typical use of this field may be to set an initial value to 0 and to monotonically increase the value for subsequent tags produced for the specific release. A change in the value of this field may be the case if a CoSWID tag producer creates and releases an incorrect tag that they subsequently want to fix, but with no underlying changes to the product the CoSWID tag represents. This could happen if, for example, a patch is distributed that has a link reference that does not cover all the various software releases it can patch. A newer CoSWID tag for that patch can be generated and the tag-version value incremented to indicate that the data is updated.

- corpus (label 8): A boolean value that indicates if the tag identifies and describes an installable software component in its pre-installation state. Installable software includes a installation package or installer for a software component, a software update, or a patch. If the Concise SWID tag represents installable software, the corpus item MUST be set to "true". If not provided the default value MUST be considered "false".

- patch (label 9): A boolean value that indicates if the tag identifies and describes an installed patch which has made incremental changes to a software component installed on a computing device. Typically, an installed patch has made a set of file modifications to pre-installed software, and does not alter the version number or the descriptive metadata of an installed software product. If a Concise SWID tag is for a patch, it MUST contain the patch item and its value MUST be set to "true". If not provided the default value MUST be considered "false".

- supplemental (label 11): A boolean value that indicates if the tag is providing additional information to be associated with another referenced SWID tag. Tags using this item help to ensure that primary and patch tags provided by a software provider are not modified by software management tools, while allowing these tools to provide their own software metadata for a software component. If a Concise SWID tag is a supplemntal tag, it MUST contain the supplemental item and its value MUST be set to "true". If not provided the default value MUST be considered "false".

- swid-name (label 1): This textual item provides the software component name as it would typically be referenced. For example, what would be seen in the add/remove software dialog in an operating system, or what is specified as the name of a packaged software component or a patch identifier name.

- software-version (label 13): A textual value representing the specific underlying release or development version of the software component.

- version-scheme (label 14): An 8-bit integer or textual value representing the versioning scheme used for the software-version item. If an integer value is used it MUST be a value from the registry (see section Section 4.1 or a value in the private use range: 32768-65,535.

- media (label 10): This text value is a hint to the tag consumer to understand what this SWID tag applies to. This item can also be included in the link item to represent a attributes defined by the W3C Media Queries Recommendation (see http://www.w3.org/TR/css3-mediaqueries/). A hint to the consumer of the link to what the target item is applicable for.

- software-meta-entry (label 5): An open-ended collection of key/value data related to this CoSWID. The attributes included in this Element are predefined attributes to ensure common usage across the industry. The schema allows for any additional attribute to be included in a CoSWID tag, though it is recommended that industry norms for new attributes are defined and followed to the degree possible. Described in Section 2.6.

- entity-entry (label 2): Specifies the organizations related to the software component referenced by this CoSWID tag. Described in Section 2.4.

- link-entry (label 4): A reference to any another item (can include details that are related to the CoSWID tag such as details on where specific resources can be found, e.g. vulnerability database associations, ROLIE feeds, MUD files, etc). This is modeled directly to match the HTML "link" element; it is critical for streamlining software discovery scenarios to ensure their consistency. Described in Section 2.5.

- payload-entry (label 6): The items that may be installed on a system entity when the software component is installed. Note that payload may be a superset of the items installed and - depending on optimization mechanisms in respect to that system entity - may or may not include every item that could be created or executed on

the corresponding system entitiy when software components are installed. In general, payload will be used to indicate the files that may be installed with a software component. Therefore payload will often be a superset of those files (i.e. if a particular optional sub-component is not installed, the files associated with that software component may be included in payload, but not installed in the system entity). Described in Section 2.7.3.

- evidence-entry (label 3): This item is used to provide results from a scan of a system where software that does not have a CoSWID tag is discovered. This information is not provided by the software-creator, and is instead created when a system is being scanned and the evidence for why software is believed to be installed on the device is provided in the evidence item. Described in Section 2.7.4.

- any-element-entry (label 7): A default map that can contain arbitrary map members and even nested maps (which would be also any-elements). In essence, the any-element allows items not defined in this CDDL data definition to be included in a Concise Software Identifier. Described in Section 2.3.

2.1.1. Determining the tag type

- Primary Tag: A CoSWID tag MUST be considered a primary tag if the corpus, patch, and supplemental items are "false".

- Patch Tag: A CoSWID tag MUST be considered a patch tag if the patch item is "true".

- Corpus Tag: A CoSWID tag MUST be considered a corpus tag if the corpus item is "true".

- Supplemental Tag: A CoSWID tag MUST be considered a supplemental tag if the supplemental item is set to "true".

If multiple of the corpus, patch, and supplemental items are "true", then the containing tag MUST be considered an unsupported tag type.

If the patch does modify the version number or the descriptive metadata of the software, then a new tag representing these details SHOULD be installed, and the old tag SHOULD be removed.

2.1.2. concise-software-identity Co-constraints

- Only one of the corpus, patch, and supplemental items MUST be set to "true", or all of the corpus, patch, and supplemental items MUST be set to "false" or be omitted.

- If the patch item is set to "true", the the tag SHOULD contain at least one link with the rel(ation) item value of "patches" and an href item specifying an association with the software that was patched.

- If the supplemental item is set to "true", the the tag SHOULD contain at least one link with the rel(ation) item value of "suplements" and an href item specifying an association with the software that is supplemented.

- If all of the corpus, patch, and supplemental items are "false", or if the corpus item is set to "true", then a software-version item MUST be included with a value set to the version of the software component. This ensure that primary and corpus tags have an identifiable software version.

2.2. The global-attributes Group

The global-attributes group provides a list of items including an optional language definition to support the processing of text-string values and an unbounded set of any-attribute items allowing for additional items to be provided as a general point of extension in the model.

The CDDL for the global-attributes is as follows:

```
<CODE BEGINS>
global-attributes = (
  ? lang,
  * any-attribute,
)

label = text / int

any-attribute = (
  label => text / int / [ 2* text ] / [ 2* int ]
)

lang = (15: text)
<CODE ENDS>
```

The following describes each child item of this object.

- lang (index 15): A language tag or corresponding IANA index integer that conforms with IANA Language Subtag Registry [RFC5646].

- any-attribute: This sub-group provides a means to include arbitrary information via label (key) item value pairs where both keys and values can be either a single integer or text string, or an array of integers or text strings.

2.3. The any-element-entry

The CDDL for the any-element-entry object is as follows:

```
<CODE BEGINS>
any-element-map = {
  global-attributes,
  * label => any-element-map / [ 2* any-element-map ],
}
any-element-entry = (7: any-element-map / [ 2* any-element-map ])
<CODE ENDS>
```

The following describes each child item of this object.

- global-attributes: The global-attributes group described in Section 2.2.

- label: a single or multiple

2.4. The entity Object

The CDDL for the entity object is as follows:

```
<CODE BEGINS>
entity = {
  global-attributes,
  entity-name,
  ? reg-id,
  role,
  ? thumbprint,
  extended-data,
}

any-uri = text

extended-data = (30: any-element-map / [ 2* any-element-map ])
entity-name = (31: text)
reg-id = (32: any-uri)
role = (33: text / [2* text])
thumbprint = (34: text)
<CODE ENDS>
```

The following describes each child item of this object.

- global-attributes: The global-attributes group described in Section 2.2.

- entity-name (index 32): The text-string name of the organization claiming a particular role in the CoSWID tag.

- reg-id (index 32): The registration id is intended to uniquely identify a naming authority in a given scope (e.g. global, organization, vendor, customer, administrative domain, etc.) that is implied by the referenced naming authority. The value of an registration ID MUST be a RFC 3986 URI. The scope SHOULD be the scope of an organization. In a given scope, the registration id MUST be used consistently.

- role (index 33): The relationship(s) between this organization and this tag. The role of tag creator is required for every CoSWID tag. The role of an entity may include any role value, but the per-defined roles include: "aggregator", "distributor", "licensor", "software-creator", "tag-creator". The enumerations of this will include a request to IANA in order to be reference-able via an integer index.

- thumbprint (index 34): This value provides a hexadecimal string that contains a hash (i.e. the thumbprint) of the signing entities certificate(s). .

- extended-data (index 30): An open-ended collection of elements that can be used to attach arbitrary metadata to an entity item.

2.5. The link Object

The CDDL for the link object is as follows:

```
<CODE BEGINS>
link = {
  global-attributes,
  ? artifact,
  href,
  ? media
  ? ownership,
  rel,
  ? media-type,
  ? use,
}
artifact = (37: text)
href = (38: any-uri)
media = (10: any-uri)
ownership = (39: "shared" / "private" / "abandon")
rel = (40: text)
media-type = (41: text)
use = (42: "optional" / "required" / "recommended")
<CODE ENDS>
```

The following describes each child item of this object.

- global-attributes: The global-attributes group described in Section 2.2.

- artifact (index: 37): For installation media (rel="installation-media") - dictates the canonical name for the file. Items with the same artifact name should be considered mirrors of each other (so download from wherever works).

- href (index 38): The link to the item being referenced. The href can point to several different things, and can be any of the following:

  * a relative uri (no scheme), which is interpreted depending on context (for example, "./folder/supplemental.coswid")

  * a physical file location with any system-acceptable URI scheme (e.g., file:// http:// https:// ftp://)

  * an URI with "coswid:" as the scheme, which refers to another CoSWID by tag-id. This URI would need to be resolved in the context of the system by software that can lookup other CoSWID tags (for example,

  * "coswid:2df9de35-0aff-4a86-ace6-f7dddd1ade4c"). an URI with "swidpath:" as the scheme, which refers to another CoSIWD via an XPATH query. This URI would need to be resolved in the context of the system entity via dedicated software components that can lookup other CoSWID tags and select the appropriate tag based on an XPATH query. Examples include:

  * swidpath://SoftwareIdentity[Entity/@regid='http://contoso.com'] would retrieve all CoSWID tags that include an entity where the regid was "Contoso".

  * swidpath://SoftwareIdentity[Meta/@persistentId='b0c55172-38e9-4e36-be86-92206ad8eddb'] would retrieve CoSWID tags that matched the persistent-id.

  * See XPATH query standard : http://www.w3.org/TR/xpath20/

- media (index 10): See media defined in Section 2.1.

- ownership (index 39): Determines the relative strength of ownership of the software components. Valid enumerations are: abandon, private, shared

- rel (index 40): The relationship between this CoSWID and the target file. Relationships can be identified by referencing the IANA registration library: https://www.iana.org/assignments/link-relations/link-relations.xhtml.

- media-type (index 41): The IANA MediaType for the target file; this provides the consumer with intelligence of what to expect. See http://www.iana.org/assignments/media-types/media-types.xhtml for more details on link type.

- use (index 42): Determines if the target software is a hard requirement or not. Valid enumerations are: required, recommended, optional.

2.6. The software-meta Object

The CDDL for the software-meta object is as follows:

```
<CODE BEGINS>

software-meta = {
  global-attributes,
  ? activation-status,
  ? channel-type,
  ? colloquial-version,
  ? description,
  ? edition,
  ? entitlement-data-required,
  ? entitlement-key,
  ? generator,
  ? persistent-id,
  ? product,
  ? product-family,
  ? revision,
  ? summary,
  ? unspsc-code,
  ? unspsc-version,
}
activation-status = (43: text)
channel-type = (44: text)
colloquial-version = (45: text)
description = (46: text)
edition = (47: text)
entitlement-data-required = (48: bool)
entitlement-key = (49: text)
generator = (50: text)
persistent-id = (51: text)
product = (52: text)
product-family = (53: text)
revision = (54: text)
summary = (55: text)
unspsc-code = (56: text)
unspsc-version = (57: text)
<CODE ENDS>
```

The following describes each child item of this object.

- global-attributes: The global-attributes group described in Section 2.2.

- activation-status (index 43): Identification of the activation status of this software title (e.g. Trial, Serialized, Licensed, Unlicensed, etc). Typically, this is used in supplemental tags.

- channel-type (index 44): Provides information on which channel this particular software was targeted for (e.g. Volume, Retail, OEM, Academic, etc). Typically used in supplemental tags.

- colloquial-version (index 45): The informal or colloquial version of the product (i.e. 2013). Note that this version may be the same through multiple releases of a software component where the version specified in entity is much more specific and will change for each software release. Note that this representation of version is typically used to identify a group of specific software releases that are part of the same release/support infrastructure (i.e. Fabrikam Office 2013). This version is used for string comparisons only and is not compared to be an earlier or later release (that is done via the entity version).

- description (index 46): A longer, detailed description of the software. This description can be multiple sentences (differentiated from summary, which is a very short, one-sentence description).

- edition (index 47): The variation of the product (Extended, Enterprise, Professional, Standard etc).

- entitlement-data-required (index 48): An indicator to determine if there should be accompanying proof of entitlement when a software license reconciliation is completed.

- entitlement-key (index 49): A vendor-specific textual key that can be used to reconcile the validity of an entitlement. (e.g. serial number, product or license key).

- generator (index 50): The name of the software tool that created a CoSWID tag. This item is typically used if tags are created on the fly or via a catalog-based analysis for data found on a computing device.

- persistent-id (index 51): A GUID used to represent products installed where the product are related, but may be different versions. For example, an "upgradeCode" (see http://msdn.microsoft.com/en-us/library/aa372375(v=vs.85).aspx as an reference for this example).

- product (index 52): The base name of the product (e.g.).

- product-family (index 53): The overall product family this software belongs to. Product family is not used to identify that a product is part of a suite, but is instead used when a set of products that are all related may be installed on multiple

different devices. For example, an enterprise backup system may consist of a backup services, multiple different backup services that support mail services, databases and ERP systems, as well as individual software components that backup client system entities. In such an usage scenario, all software components that are part of the backup system would have the same product-family name so they can be grouped together in respect to reporting systems.

- revision (index 54): The informal or colloquial representation of the sub-version of the given product (ie, SP1, R2, RC1, Beta 2, etc). Note that the version will provide very exact version details, the revision is intended for use in environments where reporting on the informal or colloquial representation of the software is important (for example, if for a certain business process, an organization recognizes that it must have, for example "ServicePack 1" or later of a specific product installed on all devices, they can use the revision data value to quickly identify any devices that do not meet this requirement). Depending on how a software organizations distributes revisions, this value could be specified in a primary (if distributed as an upgrade) or supplemental (if distributed as a patch) CoSWID tag.

- summary (index 55): A short (one-sentence) description of the software.

- unspsc-code (index 56): An 8 digit code that provides UNSPSC classification of the software component this SWID tag identifies. For more information see, http://www.unspsc.org/.

- unspsc-version (index 57): The version of the UNSPSC code used to define the UNSPSC code value. For more information see, http://www.unspsc.org/.

2.7. The Resource Collection Definition

2.7.1. The hash-entry Array

CoSWID add explicit support for the representation of hash entries using algorithms that are registered at the Named Information Hash Algorithm Registry via the hash-entry member (label 58).

```
hash-entry = (58: [ hash-alg-id: int, hash-value: bstr ] )
```

The number used as a value for hash-alg-id MUST refer the ID in the Named Information Hash Algorithm table; other hash algorithms MUST NOT be used. The hash-value MUST represent the raw hash value of the hashed resource generated using the hash algorithm indicated by the hash-alg-id.

2.7.2. The resource-collection Group

A list of items both used in evidence (discovered by an inventory process) and payload (installed in a system entity) content of a CoSWID tag document to structure and differentiate the content of specific CoSWID tag types. Potential content includes directories, files, processes, resources or firmwares.

The CDDL for the resource-collection group is as follows:

```
<CODE BEGINS>
resource-collection = (
  ? directory-entry,
  ? file-entry,
  ? process-entry,
  ? resource-entry
)

directory = {
  filesystem-item,
  path-elements,
}

file = {
  filesystem-item,
  ? size,
  ? file-version,
  ? hash-entry,
}

process = {
  global-attributes,
  process-name,
  ? pid,
}

resource = {
  global-attributes,
  type,
}

filesystem-item = (
  global-attributes,
  ? key,
  ? location,
  fs-name,
  ? root,
)
```

```
directory-entry = (16: directory / [ 2* directory ])
file-entry = (17: file / [ 2* file ])
process-entry = (18: process / [ 2* process ])
resource-entry = (19: resource / [ 2* resource ])
size = (20: integer)
file-version = (21: text)
key = (22: bool)
location = (23: text)
fs-name = (24: text)
root = (25: text)
path-elements = (26: { * file-entry,
                       * directory-entry,
                     }
                )
process-name = (27: text)
pid = (28: integer)
type = (29: text)
<CODE ENDS>
```

The following describes each child item or group for these groups.

- filesystem-item: A list of items both used in representing the nodes of a file-system hierarchy, i.e. directory items that allow one or more directories to be defined in the file structure, and file items that allow one or more files to be specified for a given location.

- global-attributes: The global-attributes group described in Section 2.2.

- directory-entry (index 16): A directory item allows one or more directories to be defined in the file structure.

- file-entry (index 17): A file element that allows one or more files to be specified for a given location.

- process-entry (index 18): Provides process (software component in execution) information for data that will show up in a devices process table.

- resource-entry (index 19): A set of items that can be used to provide arbitrary resource information about an application installed on a system entity, or evidence collected from a system entity.

- size (index 20): The file size in bytes of the file.

- file-version (index 21): The version of the file.

- key (index 22): Files that are considered important or required for the use of a software component. Typical key files would be those which, if not available on a system entity, would cause the software component not to execute or function properly. Key files will typically be used to validate that a software component referenced by the CoSWID tag document is actually installed on a specific system entity.

- location (index 23): The directory or location where a file was found or can expected to be located. This text-string is intended to include the filename itself. This SHOULD be the relative path from the location represented by the root item.

- fs-name (index 24): The file name or directory name without any path characters.

- root (index 25): A system-specific root folder that the location item is an offset from. If this is not specified the assumption is the root is the same folder as the location of the CoSWID tag. The text-string value represents a path expression relative to the CoSWID tag document location in the (composite) file-system hierarchy.

- path-elements (index 26): Provides the ability to apply a directory structure to the path expressions for files defined in a payload or evidence item.

- process-name (index 27): The process name as it will be found in the system entity's process table.

- pid (index 28): The process ID for the process in execution that can be included in the process item as part of an evidence tag.

- type (index 29): The type of resource represented via a text-string (typically, registry-key, port or root-uri).

2.7.3. The payload Object

The CDDL for the payload object is as follows:

```
payload = {
  global-attributes,
  resource-collection,
  * $$payload-extension
}
<CODE ENDS>
```

The following describes each child item of this object.

- global-attributes: The global-attributes group described in Section 2.2.

- resource-collection: The resource-collection group described in Section 2.7.2.

- $$payload-extension:

2.7.4. The evidence Object

The CDDL for the evidence object is as follows:

```
<CODE BEGINS>
evidence = {
  global-attributes,
  resource-collection,
  ? date,
  ? device-id,
  * $$evidence-extension
}
date = (35: time)
device-id = (36: text)
<CODE ENDS>
```

The following describes each child item of this object.

- global-attributes: The global-attributes group described in Section 2.2.

- resource-collection: The resource-collection group described in Section 2.7.2.

- date (index 35): The date and time evidence represented by an evidence item was gathered.

- device-id (index 36): A text-string identifier for a device evidence was gathered from.

- $$evidence-extension:

2.8. Full CDDL Definition

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

```
<CODE BEGINS>
concise-software-identity = {
```

```
    global-attributes,
    tag-id,
    tag-version,
    ? corpus,
    ? patch,
    ? supplemental,
    swid-name,
    ? software-version,
    ? version-scheme,
    ? media,
    ? software-meta-entry,
    ? entity-entry,
    ? link-entry,
    ? ( payload-entry / evidence-entry ),
    ? any-element-entry,
  }

  any-uri = text
  label = text / int

  any-attribute = (
    label => text / int / [ 2* text ] / [ 2* int ]
  )

  any-element-map = {
    global-attributes,
    * label => any-element-map / [ 2* any-element-map ],
  }

  global-attributes = (
    ? lang,
    * any-attribute,
  )

  resource-collection = (
    ? directory-entry,
    ? file-entry,
    ? process-entry,
    ? resource-entry
  )

  file = {
    filesystem-item,
    ? size,
    ? file-version,
    ? hash-entry,
  }
```

```
filesystem-item = (
  global-attributes,
  ? key,
  ? location,
  fs-name,
  ? root,
)

directory = {
  filesystem-item,
  path-elements,
}

process = {
  global-attributes,
  process-name,
  ? pid,
}

resource = {
  global-attributes,
  type,
}

entity = {
  global-attributes,
  entity-name,
  ? reg-id,
  role,
  ? thumbprint,
  extended-data,
}

evidence = {
  global-attributes,
  resource-collection,
  ? date,
  ? device-id,
  * $$evidence-extension
}

link = {
  global-attributes,
  ? artifact,
  href,
  ? media
  ? ownership,
  rel,
```

```
    ? media-type,
    ? use,
  }

  software-meta = {
    global-attributes,
    ? activation-status,
    ? channel-type,
    ? colloquial-version,
    ? description,
    ? edition,
    ? entitlement-data-required,
    ? entitlement-key,
    ? generator,
    ? persistent-id,
    ? product,
    ? product-family,
    ? revision,
    ? summary,
    ? unspsc-code,
    ? unspsc-version,
  }

  payload = {
    global-attributes,
    resource-collection,
    * $$payload-extension
  }

  tag-id = (0: text)
  swid-name = (1: text)
  entity-entry = (2: entity / [ 2* entity ])
  evidence-entry = (3: evidence)
  link-entry = (4: link / [ 2* link ])
  software-meta-entry = (5: software-meta / [ 2* software-meta ])
  payload-entry = (6: payload)
  any-element-entry = (7: any-element-map / [ 2* any-element-map ])
  corpus = (8: bool)
  patch = (9: bool)
  media = (10: text)
  supplemental = (11: bool)
  tag-version = (12: integer)
  software-version = (13: text)
  version-scheme = (14: text / int)
  lang = (15: text)
  directory-entry = (16: directory / [ 2* directory ])
  file-entry = (17: file / [ 2* file ])
  process-entry = (18: process / [ 2* process ])
```

```
   resource-entry = (19: resource / [ 2* resource ])
   size = (20: integer)
   file-version = (21: text)
   key = (22: bool)
   location = (23: text)
   fs-name = (24: text)
   root = (25: text)
   path-elements = (26: { * file-entry,
                          * directory-entry,
                        }
                   )
   process-name = (27: text)
   pid = (28: integer)
   type = (29: text)
   extended-data = (30: any-element-map / [ 2* any-element-map ])
   entity-name = (31: text)
   reg-id = (32: any-uri)
   role = (33: text / [2* text])
   thumbprint = (34: text)
   date = (35: time)
   device-id = (36: text)
   artifact = (37: text)
   href = (38: any-uri)
   ownership = (39: "shared" / "private" / "abandon")
   rel = (40: text)
   media-type = (41: text)
   use = (42: "optional" / "required" / "recommended")
   activation-status = (43: text)
   channel-type = (44: text)
   colloquial-version = (45: text)
   description = (46: text)
   edition = (47: text)
   entitlement-data-required = (48: bool)
   entitlement-key = (49: text)
   generator = (50: text)
   persistent-id = (51: text)
   product = (52: text)
   product-family = (53: text)
   revision = (54: text)
   summary = (55: text)
   unspsc-code = (56: text)
   unspsc-version = (57: text)
   hash-entry = (58: [ hash-alg-id: int,
                       hash-value: bstr,
                     ]
                )
   <CODE ENDS>
```

3. CoSWID Indexed Label Values

3.1. Version Scheme

The following are an initial set of values for use in the version-scheme item for the version schemes defined in the ISO/IEC 19770-2:2015 [SWID] specification. Index value in parens indicates the index value to use in the version-scheme item.

- multipartnumeric (index 0): Numbers separated by dots, where the numbers are interpreted as integers (e.g.,1.2.3, 1.4.5, 1.2.3.4.5.6.7)

- multipartnumeric+suffix (index 1): Numbers separated by dots, where the numbers are interpreted as integers with an additional string suffix(e.g., 1.2.3a)

- alphanumeric (index 2): Strictly a string, sorting is done alphanumerically

- decimal (index 3): A floating point number (e.g., 1.25 is less than 1.3)

- semver (index 4): Follows the [SEMVER] specification

The values above are registered in the "SWID/CoSWID Version Schema Values" registry defined in section Section 4.1. Additional valid values will likely be registered over time in this registry.

3.2. Entity Role Values

The following table indicates the index value to use for the entity roles defined in the ISO/IEC 19770-2:2015 [SWID] specification.

Index	Role Name
0	tagCreator
1	softwareCreator
2	aggregator
3	distributor
4	licensor

The values above are registered in the "SWID/CoSWID Entity Role Values" registry defined in section Section 4.2. Additional valid values will likely be registered over time. Additionally, the index values 226 through 255 have been reserved for private use.

4. IANA Considerations

This document will include requests to IANA:

- Integer indices for SWID content attributes and information elements.

- Content-Type for CoAP to be used in COSE.

This document has a number of IANA considerations, as described in the following subsections.

4.1. SWID/CoSWID Version Schema Values Registry

This document uses unsigned 16-bit index values to version-scheme item values. The initial set of version-scheme values are derived from the textual version scheme names defined in the ISO/IEC 19770-2:2015 specification [SWID].

This document defines a new a new registry entitled "SWID/CoSWID Version Schema Values". Future registrations for this registry are to be made based on [RFC8126] as follows:

Range	Registration Procedures
0-16383	Standards Action
16384-32767	Specification Required
32768-65535	Reserved for Private Use

Initial registrations for the SWID/CoSWID Version Schema Values registry are provided below.

Index	Role Name	Specification
0	multipartnumeric	See section 3.1
1	multipartnumeric+suffix	See section 3.1
2	alphanumeric	See section 3.1
3	decimal	See section 3.1
4-16383	Unassigned	
16384	semver	{{SEMVER}}
16385-32767	Unassigned	
32768-65535	Reserved for Private Use	

4.2. SWID/CoSWID Entity Role Values Registry

This document uses unsigned 8-bit index values to represent entity-role values. The initial set of Entity roles are derived from the textual role names defined in the ISO/IEC 19770-2:2015 specification [SWID].

This document defines a new a new registry entitled "SWID/CoSWID Entity Role Values". Future registrations for this registry are to be made based on [RFC8126] as follows:

Range	Registration Procedures
0-31	Standards Action
32-127	Specification Required
128-255	Reserved for Private Use

Initial registrations for the SWID/CoSWID Entity Role Values registry are provided below.

Index	Role Name	Specification
0	tagCreator	See section 3.2
1	softwareCreator	See section 3.2
2	aggregator	See section 3.2
3	distributor	See section 3.2
4	licensor	See section 3.2
5-49	Unassigned	
50-225	Unassigned	
225-255	Reserved for Private Use	

5. Security Considerations

SWID tags contain public information about software components and, as such, do not need to be protected against disclosure on an endpoint. Similarly, SWID tags are intended to be easily discoverable by applications and users on an endpoint in order to make it easy to identify and collect all of an endpoint's SWID tags. As such, any security considerations regarding SWID tags focus on the application of SWID tags to address security challenges, and the possible disclosure of the results of those applications.

A signed SWID tag whose signature is intact can be relied upon to be unchanged since it was signed. If the SWID tag was created by the software author, this generally means that it has undergone no change since the software application with which the tag is associated was

installed. By implication, this means that the signed tag reflects the software author's understanding of the details of that software product. This can be useful assurance when the information in the tag needs to be trusted, such as when the tag is being used to convey golden measurements. By contrast, the data contained in unsigned tags cannot be trusted to be unmodified.

SWID tags are designed to be easily added and removed from an endpoint along with the installation or removal of software components. On endpoints where addition or removal of software components is tightly controlled, the addition or removal of SWID tags can be similarly controlled. On more open systems, where many users can manage the software inventory, SWID tags may be easier to add or remove. On such systems, it may be possible to add or remove SWID tags in a way that does not reflect the actual presence or absence of corresponding software components. Similarly, not all software products automatically install SWID tags, so products may be present on an endpoint without providing a corresponding SWID tag. As such, any collection of SWID tags cannot automatically be assumed to represent either a complete or fully accurate representation of the software inventory of the endpoint. However, especially on devices that more strictly control the ability to add or remove applications, SWID tags are an easy way to provide an preliminary understanding of that endpoint's software inventory.

Any report of an endpoint's SWID tag collection provides information about the software inventory of that endpoint. If such a report is exposed to an attacker, this can tell them which software products and versions thereof are present on the endpoint. By examining this list, the attacker might learn of the presence of applications that are vulnerable to certain types of attacks. As noted earlier, SWID tags are designed to be easily discoverable by an endpoint, but this does not present a significant risk since an attacker would already need to have access to the endpoint to view that information. However, when the endpoint transmits its software inventory to another party, or that inventory is stored on a server for later analysis, this can potentially expose this information to attackers who do not yet have access to the endpoint. As such, it is important to protect the confidentiality of SWID tag information that has been collected from an endpoint, not because those tags individually contain sensitive information, but because the collection of SWID tags and their association with an endpoint reveals information about that endpoint's attack surface.

Finally, both the ISO-19770-2:2015 XML schema definition and the Concise SWID data definition allow for the construction of "infinite" SWID tags or SWID tags that contain malicious content with the intend if creating non-deterministic states during validation or processing

of SWID tags. While software product vendors are unlikely to do this, SWID tags can be created by any party and the SWID tags collected from an endpoint could contain a mixture of vendor and non-vendor created tags. For this reason, tools that consume SWID tags ought to treat the tag contents as potentially malicious and should employ input sanitizing on the tags they ingest.

6. Acknowledgments

7. Change Log

Changes from version 03 to version 04:

- Re-index label values in the CDDL.
- Added a section describing the CoSWID model in detail.
- Created IANA registries for entity-role and version-scheme

Changes from version 02 to version 03:

- Updated CDDL to allow for a choice between a payload or evidence
- Re-index label values in the CDDL.
- Added item definitions
- Updated references for COSE, CBOR Web Token, and CDDL.

Changes from version 01 to version 02:

- Added extensions for Firmware and CoSWID use as Reference Integrity Measurements (CoSWID RIM)
- Changes meta handling in CDDL from use of an explicit use of items to a more flexible unconstrained collection of items.
- Added sections discussing use of COSE Signatures and CBOR Web Tokens

Changes from version 00 to version 01:

- Added CWT usage for absolute SWID paths on a device
- Fixed cardinality of type-choices including arrays
- Included first iteration of firmware resource-collection

Changes since adopted as a WG I-D -00:

- Removed redundant any-attributes originating from the ISO-19770-2:2015 XML schema definition
- Fixed broken multi-map members
- Introduced a more restrictive item (any-element-map) to represent custom maps, increased restriction on types for the any-attribute, accordingly
- Fixed X.1520 reference
- Minor type changes of some attributes (e.g. NMTOKENS)
- Added semantic differentiation of various name types (e,g. fs-name)

Changes from version 00 to version 01:

- Ambiguity between evidence and payload eliminated by introducing explicit members (while still
- allowing for "empty" SWID tags)
- Added a relatively restrictive COSE envelope using cose_sign1 to define signed CoSWID (single signer only, at the moment)
- Added a definition how to encode hashes that can be stored in the any-member using existing IANA tables to reference hash-algorithms

Changes from version 01 to version 02:

- Enforced a more strict separation between the core CoSWID definition and additional usage by moving content to corresponding appendices.
- Removed artifacts inherited from the reference schema provided by ISO (e.g. NMTOKEN(S))
- Simplified the core data definition by removing group and type choices where possible
- Minor reordering of map members
- Added a first extension point to address requested flexibility for extensions beyond the any-element

8. Contributors

9. References

9.1. Normative References

[I-D.ietf-ace-cbor-web-token]
Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", draft-ietf-ace-cbor-web-token-12 (work in progress), February 2018.

[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>.

[RFC4108] Housley, R., "Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages", RFC 4108, DOI 10.17487/RFC4108, August 2005, <https://www.rfc-editor.org/info/rfc4108>.

[RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <https://www.rfc-editor.org/info/rfc5646>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

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

[SAM] "Information technology - Software asset management - Part 5: Overview and vocabulary", ISO/IEC 19770-5:2013, November 2013.

[SEMVER] Preston-Werner, T., "Semantic Versioning 2.0.0", n.d., <https://semver.org/spec/v2.0.0.html>.

[SWID] "Information technology - Software asset management - Part 2: Software identification tag", ISO/IEC 19770-2:2015, October 2015.

[SWID-GUIDANCE]
Waltermire, D., Cheikes, B., Feldman, L., and G. Witte, "Guidelines for the Creation of Interoperable Software Identification (SWID) Tags", NISTIR 8060, April 2016, <https://doi.org/10.6028/NIST.IR.8060>.

[X.1520] "Recommendation ITU-T X.1520 (2014), Common vulnerabilities and exposures", April 2011.

9.2. Informative References

[I-D.banghart-sacm-rolie-softwaredescriptor]
Waltermire, D. and S. Banghart, "Definition of the ROLIE Software Descriptor Extension", draft-banghart-sacm-rolie-softwaredescriptor-01 (work in progress), May 2017.

[I-D.birkholz-tuda]
Fuchs, A., Birkholz, H., McDonald, I., and C. Bormann, "Time-Based Uni-Directional Attestation", draft-birkholz-tuda-04 (work in progress), March 2017.

[I-D.greevenbosch-appsawg-cbor-cddl]
Birkholz, H., Vigano, C., and C. Bormann, "Concise data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-11 (work in progress), July 2017.

[I-D.ietf-sacm-terminology]
Birkholz, H., Lu, J., Strassner, J., Cam-Winget, N., and A. Montville, "Security Automation and Continuous Monitoring (SACM) Terminology", draft-ietf-sacm-terminology-14 (work in progress), December 2017.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <https://www.rfc-editor.org/info/rfc4949>.

[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>.

Appendix A. CoSWID Attributes for Firmware (label 60)

The ISO-19770-2:2015 specification of SWID tags assumes the existence of a file system a software component is installed and stored in. In the case of constrained-node networks [RFC7228] or network equipment this assumption might not apply. Concise software instances in the

form of (modular) firmware are often stored directly on a block device that is a hardware component of the constrained-node or network equipment. Multiple differentiable block devices or segmented block devices that contain parts of modular firmware components (potentially each with their own instance version) are already common at the time of this writing.

The optional attributes that annotate a firmware package address specific characteristics of pieces of firmware stored directly on a block-device in contrast to software deployed in a file-system. In essence, trees of relative path-elements expressed by the directory and file structure in CoSWID tags are typically unable to represent the location of a firmware on a constrained-node (small thing). The composite nature of firmware and also the actual composition of small things require a set of attributes to address the identification of the correct component in a composite thing for each individual piece of firmware. A single component also potentially requires a number of distinct firmware parts that might depend on each other (versions). These dependencies can be limited to the scope of the component itself or extend to the scope of a larger composite device. In addition, it might not be possible (or feasible) to store a CoSWID tag document (permanently) on a small thing along with the corresponding piece of firmware.

To address the specific characteristics of firmware, the extension points "$$payload-extension" and "$$evidence-extension" are used to allow for an additional type of resource description--firmware-entry--thereby increasing the self-descriptiveness and flexibility of CoSWID. The optional use of the extension points "$$payload-extension" and "$$evidence-extension" in respect to firmware MUST adhere to the following CDDL data definition.

```
<CODE BEGINS>
$$payload-extension  //= (firmware-entry,)
$$evidence-extension  //= (firmware-entry,)

firmware = {
  firmware-name,                  ; inherited from RFC4108
  ? firmware-version,
  ? firmware-package-identifier,  ; inherited from RFC4108
  ? dependency,                   ; inherited from RFC4108
  ? component-index,              ; equivalent to RFC4108 fwPkgType
  ? block-device-identifier,
  ? target-hardware-identifier,   ; an RFC4108 alternative to model-label
  model-label,
  ? hash-entry,                   ; a hash for a single, incl. NI hash-algo
index
  ? cms-firmware-package,         ; RCF4108, experimental, this is an actual
firmware blob!
}

firmware-entry = (60: firmware / [ 2* firmware ])
firmware-name = (61 : text)
firmware-version = (62 : text / int)
component-index = (63 : int)
model-label = (64 text / int)
block-device-identifier = (65 : text / int)
cms-firmware-package = (66: bstr)
firmware-package-identifier = (67: text)
target-hardware-identifier = (68: text)
dependency = (69: { ? firmware-name,
                    ? firmware-version,
                    ? firmware-package-identifier,
                  }
              )
<CODE ENDS>
```

The members of the firmware group that constitutes the content of the firmware-entry is based on the metadata about firmware Described in [RFC4108]. As with every semantic differentiation that is supported by the resource-collection type, the use of firmware-entry is optional. It is REQUIRED not to instantiate more than one firmware-entry, as the firmware group is used in a map and therefore only allows for unique labels.

The optional cms-firmware-package member allows to include the actual firmware in the CoSWID tag that also expresses its metadata as a byte-string. This option enables a CoSWID tag to be used as a container or wrapper that composes both firmware and its metadata in a single document (which again can be signed, encrypted and/or

compressed). In consequence, a CoSWID tag about firmware can be conveyed as an identifying document across endpoints or used as a

reference integrity measurement as usual. Alternatively, it can also convey an actual piece of firmware, serve its intended purpose as a SWID tag and then - due to the lack of a location to store it - be discarded.

Appendix B. Signed Concise SWID Tags using COSE

SWID tags, as defined in the ISO-19770-2:2015 XML schema, can include cryptographic signatures to protect the integrity of the SWID tag. In general, tags are signed by the tag creator (typically, although not exclusively, the vendor of the software component that the SWID tag identifies). Cryptographic signatures can make any modification of the tag detectable, which is especially important if the integrity of the tag is important, such as when the tag is providing reference integrity measurements for files.

The ISO-19770-2:2015 XML schema uses XML DSIG to support cryptographic signatures. CoSWID tags require a different signature scheme than this. COSE (CBOR Object Signing and Encryption) provides the required mechanism [RFC8152]. Concise SWID can be wrapped in a COSE Single Signer Data Object (cose-sign1) that contains a single signature. The following CDDL defines a more restrictive subset of header attributes allowed by COSE tailored to suit the requirements of Concise SWID.

```
<CODE BEGINS>
signed-coswid = #6.997(COSE-Sign1-coswid) ; see TBS7 in current COSE I-D

label = int / tstr  ; see COSE I-D 1.4.
values = any        ; see COSE I-D 1.4.

unprotected-signed-coswid-header = {
    1 => int,                   ; algorithm identifier
    3 => "application/coswid",  ; request for CoAP IANA registry to become an
int
    * label => values,
}

protected-signed-coswid-header = {
    4 => bstr,                  ; key identifier
    * label => values,
}

COSE-Sign1-coswid = [
    protected: bstr .cbor protected-signed-coswid-header,
    unprotected: unprotected-signed-coswid-header,
    payload: bstr .cbor concise-software-identity,
    signature: bstr,
]
<CODE ENDS>
```

Appendix C. CoSWID used as Reference Integrity Measurements (CoSWID RIM)

A vendor supplied signed CoSWID tag that includes hash-values for the files that compose a software component can be used as a RIM (reference integrity measurement). A RIM is a type of declarative guidance that can be used to assert the compliance of an endpoint by assessing the installed software. In the context of remote attestation based on an attestation via hardware rooted trust, a verifier can appraise the integrity of the conveyed measurements of software components using a CoSWID RIM provided by a source, such as [I-D.banghart-sacm-rolie-softwaredescriptor].

RIM Manifests (RIMM): A group of SWID tags about the same (sub-)system, system entity, or (sub-)component (compare [RFC4949]). A RIMM manifest is a distinct document that is typically conveyed en-block and constitutes declarative guidance in respect to a specific (target) endpoint (compare [I-D.ietf-sacm-terminology]).

If multiple CoSWID compose a RIMM, the following CDDL data definition SHOULD be used.

```
RIMM = [ + concise-software-identity / signed-coswid ]
```

Appendix D. CBOR Web Token for Concise SWID Tags

A typical requirement regarding specific instantiations of endpoints - and, as a result, specific instantiations of software components - is a representation of the absolute path of a CoSWID tag document in a file system in order to derive absolute paths of files represented in the corresponding CoSWID tag. The absolute path of an evidence CoSWID tag can be included as a claim in the header of a CBOR Web Token [I-D.ietf-ace-cbor-web-token]. Depending on the source of the token, the claim can be in the protected or unprotected header portion.

```
<CODE BEGINS>
 CDDL TBD
<CODE ENDS>
```

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
Darmstadt 64295
Germany

Email: henk.birkholz@sit.fraunhofer.de

Jessica Fitzgerald-McKay
Department of Defense
9800 Savage Road
Ft. Meade, Maryland
USA

Email: jmfitz2@nsa.gov

Charles Schmidt
The MITRE Corporation
202 Burlington Road
Bedford, Maryland 01730
USA

Email: cmschmidt@mitre.org

David Waltermire
National Institute of Standards and Technology
100 Bureau Drive
Gaithersburg, Maryland 20877
USA

Email: david.waltermire@nist.gov