DICE Endorsement Architecture for Devices

Version 1.0
Revision 0.38
March 28, 2022

Contact: admin@trustedcomputinggroup.org

PUBLIC REVIEW

Work in Progress
This document is an intermediate draft for comment only and is subject to change without notice. Readers should not design products based on this document.
DISCLAIMERS, NOTICES, AND LICENSE TERMS

THIS SPECIFICATION IS PROVIDED “AS IS” WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

Without limitation, TCG disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification and to the implementation of this specification, and TCG disclaims all liability for cost of procurement of substitute goods or services, lost profits, loss of use, loss of data or any incidental, consequential, direct, indirect, or special damages, whether under contract, tort, warranty or otherwise, arising in any way out of use or reliance upon this specification or any information herein.

This document is copyrighted by Trusted Computing Group (TCG), and no license, express or implied, is granted herein other than as follows: You may not copy or reproduce the document or distribute it to others without written permission from TCG, except that you may freely do so for the purposes of (a) examining or implementing TCG specifications or (b) developing, testing, or promoting information technology standards and best practices, so long as you distribute the document with these disclaimers, notices, and license terms.

Contact the Trusted Computing Group at www.trustedcomputinggroup.org for information on specification licensing through membership agreements.

Any marks and brands contained herein are the property of their respective owners.
# CHANGE HISTORY

<table>
<thead>
<tr>
<th>REVISION</th>
<th>DATE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00/0.01</td>
<td>June 23, 2020</td>
<td>Starting template</td>
</tr>
<tr>
<td>1.00/0.02</td>
<td>June 23, 2020</td>
<td>Fleshed out the introduction section</td>
</tr>
<tr>
<td>1.00/0.03</td>
<td>July 17, 2020</td>
<td>Fleshed out sections 6.1 – 6.4, restructured 6.4, added diagrams to 6.5 and 6.7</td>
</tr>
<tr>
<td>1.00/0.04</td>
<td>July 24, 2020</td>
<td>Added section 6.4.1 - 6.4.3 and section 6.5 – 6.7</td>
</tr>
<tr>
<td>1.00/0.05</td>
<td>October 23, 2020</td>
<td>Added diagrams for revocation; added CDDL for revocation, Claims extension, entity-entry thumbprint extension and tag collection which allows multiple tags per manifest. Added more references to the references section and cleaned up cross references. Restructured scope outline. Added acronyms. Removed some content that will appear in the TCG Framework. Marked non-normative sections as such.</td>
</tr>
<tr>
<td>1.00/0.06</td>
<td>October 28, 2020</td>
<td>Tweaked definition of element name in section 5.2. Added non-normative section to 5.3 that gives examples of measurable hardware. Added section for a matching algorithm.</td>
</tr>
<tr>
<td>1.00/0.07</td>
<td>October 29, 2020</td>
<td>Reworked CDDL for Claims-entry; changed deny-list-entry to use a deny-id rather than using tag-id. Added deny-id to the top-level manifest class.</td>
</tr>
<tr>
<td>1.00/0.08</td>
<td>November 11, 2020</td>
<td>Added appendices for CDDL content. Updated UML diagrams to reflect updated CDDL. Divided section 5.4 into sub-sections. Updated UML diagrams.</td>
</tr>
<tr>
<td>1.00/0.09</td>
<td>November 19, 2020</td>
<td>Updated UML diagrams r10 and CDDL r10. Detailed editing pass affecting normative and informative text.</td>
</tr>
<tr>
<td>1.00/0.10</td>
<td>March 10, 2021</td>
<td>Updated CDDL appendix to the r11 schema; swapped section 4.4 with 4.3 for readability; reworded section 5.1 intro; Reworked section 5.2 to focus on the information model for corim; Added new sections for element value and CoSWID information models with diagrams.</td>
</tr>
<tr>
<td>1.00/0.11</td>
<td>March 26, 2021</td>
<td>Edits applied to sections in 5.2 on Claims, Element Name, Element Value and CoSWID schema. Added tag relationship normative. Reworded Section 4.4 and 5.3 intro. Added data model sections (5.3 – 5.3.5) with partial content. Accepted formatting changes and content changes reviewed at the March 25th WG meeting.</td>
</tr>
<tr>
<td>1.00/0.12</td>
<td>April 7, 2021</td>
<td>Corim table cleanup; Updated UML; rearranged sections; added device-to-tag-mapping diagram to section 5.1; Added diagram for multi-endorser manifests; moved sections on multi-endorser manifests, multi-tag modules into section 5.3; Word smithed these sections. Fleshed out section 5.4 on coswid; accepted formatting changes and other minor changes.</td>
</tr>
<tr>
<td>1.00/0.13</td>
<td>April 13, 2021</td>
<td>Accepted all changes from r12; changed element-name-map.type to element-name-map.class-id and updated related descriptions; Added / modified comid and corim roles; added entity-maps for corim and comid; Updated section on xcorim to include data model contents following conventions for other sections.</td>
</tr>
<tr>
<td>1.00/0.14</td>
<td>April 14, 2021</td>
<td>Misc tweaks waterfalled from corim CDDL effort. Added linked tags relationship examples in section 5.3.3</td>
</tr>
<tr>
<td>1.00/0.15</td>
<td>April 21, 2021</td>
<td>Removed CoMID-to-CoSWID ‘supplements’ relationship. Removed comid.distributor and comid.licensor role types. Added sections 6.2, 6.3 and 6.4; Moved verification example to appendix;</td>
</tr>
<tr>
<td>1.00/0.16</td>
<td>April 29, 2021</td>
<td>Merged content in sections 6.4.2 – 6.1.4 into 6.4.1. Applied schema changes driven by CDDL discussions. Changed $concise-reference-integrity-manifest-group-choice to $concise-reference-integrity-manifest-type-choice. Split instance-claim-map into two, ref-instance-claim-map and end-instance-claim-map. Added element-value-map to both as instance data could be reported in raw-value-map and digests-map. Removed COSE_KeySet and replaced with list of COSE_Key. Updated UML diagrams. Updated figure 5-3 and 5-4.</td>
</tr>
<tr>
<td>1.00/0.17</td>
<td>June 23, 2021</td>
<td>Accepted all changes from previous revisions. Used strikethrough text to preliminarily remove includes, or-includes, runs-on-module and requires-module link relationships. Applied name changes (element-name-map to class-map, element-value-map to measurement-map, device-id-type-choice to instance-id-type-choice, Claims-map to triples-map). Replaced reference-claim-map with reference-triple-record, endorsed-claim-map with endorsed-triple-record and identity-claim-map with identity-triple-record. Removed ref-instance-claim-map and end-instance-claim-map. Merged class and instance measurements into measurement-map. Added attest-key-triple-record. Added verification-key-map that associates a PKIX cert chain to a COSE_Key. Removed ‘updates’ linked tag relationship as this can be added later via extension. Removed module-name from concise-mid-tag-map. Cleaned up diagrams to reflect various changes.</td>
</tr>
</tbody>
</table>
| 1.00/0.18| June 24, 2021 | Removed strikethrough text (feature dropped from first revision). Relocated section 5.4.5 and 5.4.6 to be subsections of 5.4.2. Updated IM diagrams to reflect r17 changes. Wording and
<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00/0.19</td>
<td>June 25, 2021</td>
<td>Accepted changes from r0.18. Reworked Appendix A. minor cleanup. Added measurement-map as a parent to measurement-values-map. Removed the glossary section. Various wordsmithing. Fixed figure references.</td>
</tr>
<tr>
<td>1.00/0.20</td>
<td>June 30, 2021</td>
<td>Accepted all changes from r0.19. Updated references. Addressed reviewer feedback. Added overview of sections in section 4. Added flowchart diagram in section 6. Added Appendix B heading for a RIM example. Corrected diagrams.</td>
</tr>
<tr>
<td>1.00/0.21</td>
<td>July 6, 2021</td>
<td>Added figures 4.2 and 4.3 with descriptions. Removed historical sidebar comments. Added content for Appendix B.</td>
</tr>
<tr>
<td>1.00/0.22</td>
<td>July 7, 2021</td>
<td>Added ASCII CBOR content to Appendix B. Minor edits to section 6.1. Minor informative changes to Appendix B.</td>
</tr>
<tr>
<td>1.00/0.23</td>
<td>July 12, 2021</td>
<td>Replaced #6.X global code points with IANA assigned values. Replaced COSE_Key with base64 encoded PKIX key. Removed COSE_Key map.</td>
</tr>
<tr>
<td>1.00/0.24</td>
<td>July 12, 2021</td>
<td>Added reference for RFC5912. Cleaned up tables. Replaced temporary global code points with IANA registered code points. Corrected linked tags processing in section 6.3. Minor wordsmithing.</td>
</tr>
<tr>
<td>1.00/0.25</td>
<td>August 20, 2021</td>
<td>Formatting revisions. Renumbered sections (section 5 became 4 and 6 became 5).</td>
</tr>
<tr>
<td>1.00/0.26</td>
<td>September 27, 2021</td>
<td>Removed impl-id-type as it is defined in PSA profile. Cleaned up references. Added section 4.6 for certificate considerations; changed raw-value to be a tagged-type-choice. Added section 1.3 Audience. Removed implementation ID types. Changed raw-value-group to a type-choice. Improved wording in section 3.2 and section 4. Included citations for References as a comment in section 1. Added references as comments where needed. Changed comid.entity to comid.entities. Added rid validity to unsigned comid. Created manifest signer map. Removed ‘signed’ from unprotected corim header in COSE signature map. Changed protected-signed-corim-header-map to make corim-meta-map an opaque CBOR object. Renumbered the tables.</td>
</tr>
<tr>
<td>1.00/0.27</td>
<td>September 28, 2021</td>
<td>Added table for xcorim-signer-map, removed ‘signed’ string from the unprotected-signed-xcorim-header-map name, made xcorim-meta-map an opaque CBOR block. Changed xcorim-entity-map to allow a single role from one or more roles. Changed xcorim-map from xcorim.creator to xcorim.entity and changed multiplicity to one or more. Removed the xcorim.deny-list-signer type choice. Renumbered the tables. Added IANA reference. Added Appendix C containing sample CDDL. Added table of tables.</td>
</tr>
<tr>
<td>1.00/0.28</td>
<td>September 30, 2021</td>
<td>Removed ‘signed’ from protected-signed-corim-header-map.</td>
</tr>
<tr>
<td>1.00/0.29</td>
<td>October 1, 2021</td>
<td>Misc cleanup.</td>
</tr>
<tr>
<td>1.00/0.30</td>
<td>October 5, 2021</td>
<td>Alphabetized corim and xcorim types tables. Misc edits to table entries. Made xcorim section 4.5.2 prose into non-normative block. Updated UML diagram for xcorim. Xcorim tables cleaned up to match CDDL.</td>
</tr>
<tr>
<td>1.00/0.31</td>
<td>October 8, 2021</td>
<td>Updated model to model-name in measurement-values-map. Added appendix D for UML diagrams. Cleaned up tables in Xcorim and comid sections. Added model-name to measurement-values-map. Fixed $verification-key-map-extension name. changed comid.$role-type-choice to $comid-role-type-choice.</td>
</tr>
<tr>
<td>1.00/0.32</td>
<td>October 13, 2021</td>
<td>Fixed typo in normative in section 4.4.2.3.2. removed Relying Party and last sequence from Fig 16. Replaced ‘globally unique’ with ‘statistically unique’. Added definitions for each flag in $operational-flags type definition. Capitalized role names. Capitalized role names, role messages and other framework terms.</td>
</tr>
<tr>
<td>1.00/0.33</td>
<td>November 01, 2021</td>
<td>Changed integrity-protected to not-integrity-protected and replay-protected to not-replay-protected in flags-type. Added acknowledgements section. Added IANA reference. Fixed several broken references. Added omitted CDDL to appendix C.</td>
</tr>
<tr>
<td>1.00/0.33</td>
<td>November 02, 2021</td>
<td>Fixed typo in Acknowledgements. Fixed CDDL for measurement-values-map.name.</td>
</tr>
<tr>
<td>1.00/0.34</td>
<td>November 05, 2021</td>
<td>Included global tag in signature payload block. Clarified $operational-flags-type-choice as bit position indicators. Added $measured-element-type-choice as uint. Added bit mask to flags-type. Applied reviewer feedback, various spelling and grammatical fixes. Updated tables to repeat header row across page breaks.</td>
</tr>
<tr>
<td>1.00/0.35</td>
<td>February 23, 2022</td>
<td>Revisions based on TC feedback.</td>
</tr>
<tr>
<td>1.00/0.36</td>
<td>February 24, 2022</td>
<td>Updated flags-type to flags-map, key-type changed to $key-type-choice and cert-type to $cert-type-choice; Accepted changes. Updated UML diagrams.</td>
</tr>
<tr>
<td>1.00/0.37</td>
<td>March 4, 2022</td>
<td>Updated verification-key-map with $crypto-key-type-choice and related tables, diagrams and CDDL.</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENT

The TCG wishes to thank all those who contributed to this specification. This document builds on work done in various working groups in the TCG and the industry at large.

<table>
<thead>
<tr>
<th>NAME</th>
<th>COMPANY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Henk Birkholz</td>
<td>Fraunhofer SIT</td>
</tr>
<tr>
<td>Yogesh Deshpande</td>
<td>ARM Holdings</td>
</tr>
<tr>
<td>Thomas Fossati</td>
<td>ARM Holdings</td>
</tr>
<tr>
<td>Dennis Mattoon</td>
<td>Microsoft Corporation</td>
</tr>
<tr>
<td>Ned Smith</td>
<td>Intel Corporation</td>
</tr>
</tbody>
</table>
CONTENTS

DISCLAIMERS, NOTICES, AND LICENSE TERMS ......................................................................................................................... 1
CHANGE HISTORY .............................................................................................................................................................................. 2
ACKNOWLEDGEMENT ........................................................................................................................................................................ 4
FIGURES ........................................................................................................................................................................................... 7
TABLES ............................................................................................................................................................................................ 8
1 SCOPE ........................................................................................................................................................................................... 10
  1.1 Key Words ................................................................................................................................................................................. 10
  1.2 Statement Type .......................................................................................................................................................................... 10
2 REFERENCES .................................................................................................................................................................................. 11
3 TERMS AND DEFINITIONS .......................................................................................................................................................... 14
  3.1 Acronyms .................................................................................................................................................................................... 14
4 INTRODUCTION ............................................................................................................................................................................. 15
  4.1 Endorsement ................................................................................................................................................................................. 15
  4.2 Attestation Verification ............................................................................................................................................................... 16
  4.3 Endorsement Manifests ............................................................................................................................................................... 17
  4.4 Endorsement Manifest Schemas and Technology .................................................................................................................. 17
5 ENDORSEMENT MANIFEST ARCHITECTURE .......................................................................................................................... 19
  5.1 Endorsement Manifest Requirements ........................................................................................................................................ 19
  5.2 Information Model for Endorsement Manifests ........................................................................................................................ 20
    5.2.1 Attester Device Composition ............................................................................................................................................... 20
    5.2.2 Manifest Information Model .............................................................................................................................................. 21
    5.2.3 CoMID Tag ............................................................................................................................................................................. 21
    5.2.4 Claims ....................................................................................................................................................................................... 23
    5.2.5 Environments ......................................................................................................................................................................... 24
    5.2.6 Measurements ................................................................................................................................................................. 25
    5.2.7 CoSWID Information Model .............................................................................................................................................. 27
  5.3 Device Composition Using Multiple Tags ................................................................................................................................ 28
    5.3.1 Tag Linking .............................................................................................................................................................................. 29
    5.3.2 Multi-Tag Manifests ............................................................................................................................................................ 31
    5.3.3 Multi-Endorser Manifests ................................................................................................................................................... 31
  5.4 Archive Files ............................................................................................................................................................................... 32
  5.5 Data Model for Concise Reference Integrity Manifest (CoRIM) ............................................................................................... 32
    5.5.1 Data Model Conventions ................................................................................................................................................... 33
    5.5.2 CoRIM Schema ................................................................................................................................................................. 34
    5.5.3 A Schema for CoMID Tags ............................................................................................................................................. 38
5.5.4 A Schema for CoSWID Tags ................................................................. 46
5.6 Xcorim Data Model for Deny Lists....................................................... 48
  5.6.1 Xcorim Signature and Map Structures ............................................ 48
  5.6.2 Xcorim Data Types ........................................................................ 50
  5.6.3 Xcorim Content Type and File Extension ...................................... 50
5.7 Certificate Considerations ................................................................. 50

6 PROCESSING ENDORSEMENT CLAIMS ...................................................... 52
  6.1 Verifier Provisioning ......................................................................... 53
  6.2 Attester Authentication ..................................................................... 54
  6.3 Resolution of Linked Tags .................................................................. 54
    6.3.1 Algorithm for Linked Tag Resolution .......................................... 55
  6.4 Evidence Processing .......................................................................... 56
    6.4.1 Evidence Matching Algorithm .................................................... 56

APPENDIX A – ATTESTATION EXAMPLE .................................................... 60
APPENDIX B – ENDORSEMENT MANIFEST EXAMPLE ............................. 62
APPENDIX C – CORIM CDDL EXAMPLES ..................................................... 66
APPENDIX D – UML DIAGRAMS ................................................................. 71
<table>
<thead>
<tr>
<th>FIGURES</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1: Endorsements message from Endorser to Verifier</td>
<td>15</td>
</tr>
<tr>
<td>Figure 2: CoRIM Schema informs CoRIM creation</td>
<td>17</td>
</tr>
<tr>
<td>Figure 3: Manifest authoring and verification</td>
<td>18</td>
</tr>
<tr>
<td>Figure 4: Module to Tag Mapping</td>
<td>20</td>
</tr>
<tr>
<td>Figure 5: Concise Endorsement Manifest Information Model</td>
<td>21</td>
</tr>
<tr>
<td>Figure 6: Concise Module ID Tag Information Model</td>
<td>22</td>
</tr>
<tr>
<td>Figure 7: Claim Triple Information Model</td>
<td>24</td>
</tr>
<tr>
<td>Figure 8: Environment Name Information Model</td>
<td>25</td>
</tr>
<tr>
<td>Figure 9: Class and Instance Measurements</td>
<td>26</td>
</tr>
<tr>
<td>Figure 10: Software ID Tags Information Model</td>
<td>28</td>
</tr>
<tr>
<td>Figure 11: The ‘REPLACES’ relationship example</td>
<td>30</td>
</tr>
<tr>
<td>Figure 12: The ‘SUPPLEMENTS’ relationship example</td>
<td>30</td>
</tr>
<tr>
<td>Figure 13: Example Multi-Endorser Manifest</td>
<td>32</td>
</tr>
<tr>
<td>Figure 14: Processing Endorsements and Reference Values flow</td>
<td>53</td>
</tr>
<tr>
<td>Figure 15: Evidence matching algorithm flowchart</td>
<td>59</td>
</tr>
<tr>
<td>Figure 16: Attestation example</td>
<td>61</td>
</tr>
<tr>
<td>Figure 17: CoRIM data model in UML</td>
<td>71</td>
</tr>
<tr>
<td>Figure 18: CoMID data model in UML</td>
<td>72</td>
</tr>
<tr>
<td>Figure 19: CoRIM and Xcorim data types in UML</td>
<td>73</td>
</tr>
<tr>
<td>Figure 20: CoSWID data model in UML</td>
<td>74</td>
</tr>
<tr>
<td>Figure 21: Xcorim - a CoRIM deny list in UML</td>
<td>75</td>
</tr>
</tbody>
</table>
TABLES

Table 1: Notation for data model definition ................................................................. 33
Table 2: Typographical notation conventions ............................................................... 34
Table 3: Format of concise-reference-integrity-manifest structure. .............................. 35
Table 4: Format of corim-map structure ....................................................................... 35
Table 5: Format of corim-locator-map structure .......................................................... 35
Table 6: Format of signed-corim structure ................................................................... 36
Table 7: Format of COSE-Sign1-corim structure. ......................................................... 36
Table 8: Format of protected-corim-header-map structure .......................................... 36
Table 9: Format of unprotected-corim-header-map structure. ...................................... 36
Table 10: Format of corim-meta-map structure ............................................................. 37
Table 11: Format of validity-map structure .................................................................... 37
Table 12: Format of corim-signer-map structure ......................................................... 37
Table 13: Format of corim-entity-map structure .......................................................... 37
Table 14: Format of concise-mid-tag-map structure .................................................... 39
Table 15: Format of tag-identity-map structure ............................................................ 39
Table 16: Format of linked-tag-map structure ............................................................. 39
Table 17: Format of triples-map structure .................................................................... 39
Table 18: Format of version-map structure .................................................................. 39
Table 19: Format of environment-map structure ......................................................... 40
Table 20: Format of reference-triple-record structure ................................................ 40
Table 21: Format of endorsed-triple-record structure ................................................. 40
Table 22: Format of identity-triple-record structure .................................................... 40
Table 23: Format of attest-key-triple-record structure ................................................. 40
Table 24: Format of class-map structure ...................................................................... 41
Table 25: Format of measurements-map structure ...................................................... 41
Table 26: Format of measurement-values-map structure .............................................. 41
Table 27: Format of flags-map structure ...................................................................... 42
Table 28: Format of raw-value-group structure ........................................................... 42
Table 29: Format of digests-type structure ................................................................... 42
Table 30: Format of entity-map structure ..................................................................... 42
Table 31: CoMID linked tag relationships ................................................................... 43
Table 32: CoRIM and CoMID type definitions ............................................................ 45
Table 33: Format of concise-swid-tag CoSWID structure. ........................................... 46
Table 34: Format of software-meta-entry CoSWID structure ..................................... 46
Table 35: Format of entity-entry CoSWID structure. .................................................... 46
Table 36: Format of link-entry CoSWID structure ....................................................... 46
Table 37: Format of a corim revocation structure. ....................................................... 48
Table 38: Format of signed-xcorim Xcorim structure. ............................................... 48
Table 39: Format of COSE-Sign1-xcorim structure. ...................................................... 48
Table 40: Format of protected-xcorim-header-map structure ................................. 49
Table 41: Format of unprotected-xcorim-header-map structure ................................. 49
Table 42: Format of xcorim-meta-map structure ......................................................... 49
Table 43: Format of xcorim-map structure ................................................................. 49
Table 44: Format of xcorim-entity-map structure ....................................................... 49
Table 45: Format of xcorim-signer-map structure ....................................................... 50
Table 46: Xcorim data types .................................................................................... 50
1 SCOPE

This specification is intended for software and hardware architects, developers, manufacturers, vendors, service providers, and data engineers seeking to build solutions leveraging DICE and integrating endorsement manifests. It describes the role of endorsement structures to attestation, the composition of an endorsement manifest schema that describes hardware (devices and components), how vendors might define relevant Claim sets, and how those Claim sets can be represented in an interoperable, machine-readable format. It further describes how to construct manifests that describe devices having multiple components and multiple component vendors that each might issue endorsement manifests.

1.1 Key Words

The key words “MUST,” “MUST NOT,” “REQUIRED,” “SHALL,” “SHALL NOT,” “SHOULD,” “SHOULD NOT,” “RECOMMENDED,” “MAY,” and “OPTIONAL” in this document normative statements are to be interpreted as described in RFC-2119, Key words for use in RFCs to Indicate Requirement Levels.

1.2 Statement Type

Please note a very important distinction between different sections of text throughout this document. There are two distinctive kinds of text: informative comments and normative statements. Because most of the text in this specification will be of the kind normative statements, the authors have informally defined it as the default and, as such, have specifically called out text of the kind informative comment. They have done this by flagging the beginning and end of each informative comment and highlighting its text in gray. This means that unless text is specifically marked as of the kind informative comment, it can be considered a kind of normative statement.

EXAMPLE: Start of informative comment

This is the first paragraph of 1–n paragraphs containing text of the kind informative comment ...

This is the second paragraph of text of the kind informative comment ...

This is the nth paragraph of text of the kind informative comment ...

To understand the TCG specification the user must read the specification. (This use of MUST does not require any action).

End of informative comment
2 REFERENCES


3 TERMS AND DEFINITIONS

For the purposes of this specification, the following terms and definitions apply. This specification uses attestation related terminology defined in the Trusted Computing Group Attestation Framework Requirements [1]. Additionally, some terms are defined in the Trusted Computing Group Glossary [2].

This specification assumes the reader is familiar with the TCG Attestation Framework Requirements [1] that defines attestation roles (i.e., Attester, Verifier, Relying Party, Endorser, Verifier Owner, and Relying Party Owner) and role messages (i.e., Evidence, Attestation Results, Endorsements, Appraisal Policy for Evidence, and Appraisal Policy for Attestation Results). Similar terminology definitions are found in the IETF Remote Attestation Procedures (RATS) architecture (see [3]).

This specification may use the terms device and component interchangeably or may use them together to express decomposition semantics. Additionally, tag and module may be used interchangeably.

3.1 Acronyms

<table>
<thead>
<tr>
<th>ABBREVIATIONS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBOR</td>
<td>Constrained Binary Object Representation [4]</td>
</tr>
<tr>
<td>CDDL</td>
<td>Concise Data Definition Language [5]</td>
</tr>
<tr>
<td>CoRIM</td>
<td>Concise Reference Integrity Manifest (see Section 5.5.2)</td>
</tr>
<tr>
<td>CoMID</td>
<td>Concise Module Identifier (see Section 5.5.3)</td>
</tr>
<tr>
<td>COSE</td>
<td>CBOR Object Signing and Encryption [6]</td>
</tr>
<tr>
<td>CoSWID</td>
<td>Concise Software Identifier [7]</td>
</tr>
<tr>
<td>DICE</td>
<td>Device Identifier Composition Engine [8]</td>
</tr>
<tr>
<td>IDevID</td>
<td>Initial Device Identity [9]</td>
</tr>
<tr>
<td>JOSE</td>
<td>JSON Object Signing and Encryption</td>
</tr>
<tr>
<td>OCM</td>
<td>Original Component Manufacturer</td>
</tr>
<tr>
<td>ODM</td>
<td>Original Device Manufacturer</td>
</tr>
<tr>
<td>OEM</td>
<td>Original Equipment Manufacturer</td>
</tr>
<tr>
<td>RIM</td>
<td>Reference Integrity Manifest [10]</td>
</tr>
<tr>
<td>SWID</td>
<td>Software Identifier Tags [11]</td>
</tr>
<tr>
<td>TCG</td>
<td>Trusted Computing Group</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier [12]</td>
</tr>
</tbody>
</table>
4 INTRODUCTION

Start of informative comment

This specification defines an attestation endorsement architecture, schema, and manifest profile that informs and supports verification and appraisal of Evidence obtained from a DICE layered device [13] [14] [15]. A primary objective of this specification is the interoperation of Endorsements supplied by one or more Endorsers to one or more Verifiers.

This introductory section provides an overview of concepts related to attestation Endorsements. Endorsements are also discussed in the IETF Attestation Architecture [3] and by the TCG Attestation Framework Requirements Specification [1].

Section 5 describes an Endorsement Manifest Architecture. An information model for the manifest is presented in sections 5.2 and 5.3 followed by a data model definition in sections 5.5 and 5.6.

Section 6 provides guidance for attestation Verifiers with particular attention given to verification of Evidence from DICE certificates [14] [15].

End of informative comment

4.1 Endorsement

Start of informative comment

Endorsers are supply chain entities that supply Endorsements to Verifiers (see Figure 1). Some examples of supply chain entities are device manufacturers, original component manufacturers (OCMs), original device manufacturers (ODMs), original equipment manufacturers (OEMs), firmware vendors, software vendors, pre-release validators, post-release validators, test labs, and quality compliance labs. Some Endorsers define and create components and other ingredients that make up a device (or platform, see [2]). The device, as the Attester, supplies Evidence that a Verifier uses to appraise the Attester’s trustworthiness. The Verifier compares the Claims in Evidence with Endorsements to determine whether the expected Reference Values and Endorsed Values differ from the actual values in Evidence.

Endorsements can include Reference Values and Endorsed Values. Reference Values are Claims that the Verifier can expect to find in Evidence from an Attester. Reference Values in Endorsements are compared directly to Claims in Evidence from the Attester. Endorsed Values describe trustworthiness properties that apply to the Attester but are not reported in Evidence from the Attester. Endorsed Values indicate intrinsic properties of a device that are asserted by the Endorser.

The Endorser that produces a module may be the most authoritative entity for making assertions about the module’s intrinsic trustworthiness properties or to generate Reference Values for the module. Verifiers should have a clear policy for identifying which Endorsers are authorized to provide Endorsements.

Claims are data provided as Evidence by an Attester. However, the term Claim may also sometimes be used to refer to values in Endorsements. For example: a Verifier may compare a firmware measurement Claim in Evidence to a Reference Value Claim in Endorsements. In this context the term Claim refers to a statement in Evidence, but also to the Reference Value in Endorsements against which the Evidence Claim is compared.

Figure 1: Endorsements message from Endorser to Verifier
Endorsements have the following objectives:

- Define a logical representation of the device and its sub-components.
- Define a logical representation of the device that is machine-readable.
- Define machine-readable Reference Values that are to be matched with Evidence.
- Define machine-readable values for Attester Claims that are not expected to be matched with Evidence.
- Construct Endorsements that can be authenticated by Verifiers.
- Construct device identities that can be authenticated (e.g., IDevIDs [9]).

Generally, Endorsements are information provided by manufacturers and suppliers about the Attester. The type of information provided generally fits into the following three categories:

(i) Class Endorsements
(ii) Instance Endorsements
(iii) Identity Endorsements

Class Endorsements apply to a class of device or component, of which there may be millions or even billions of mass-produced clones. Consequently, the same endorsement document (e.g., manifest) could be reused for all devices in the class.

Instance Endorsements are specific to a device instance. Each device instance requires a different Endorsement Manifest containing the information that is specific for each. Due to the potential for differential scalability and manageability properties, mixing class and instance information in the same Endorsement Manifest should be avoided.

Identity Endorsements describe the relationship between cryptographic keys and identifiers. For example, a device may have an embedded cryptographic key pair used to authenticate the device or to sign attestation Evidence.

For the purposes of this specification, Endorsements are contained in Endorsement Manifests. Endorsements may also be contained in other types of documents as well, for example, identity certificates [16], attribute certificates, or other manifests [10] [17].

End of informative comment

4.2 Attestation Verification

Endorsements and Evidence are used to determine which Claims accurately describe the Attester. If Endorsements do not accurately describe the Attester, or the Verifier is unable to process Endorsements, the Verifier could become confused, resulting in incorrect Attestation Results.

Verifiers accept Endorser-supplied Claims if the Endorser is trusted. Typically, this means the Verifier policy contains a trust anchor for the Endorser. Endorsements contain:

1. Reference Values – Claims that must match Evidence before Evidence is accepted as valid.
2. Endorsed Values – Assertions from the Endorser about the Attester that are accepted as valid if Attester Evidence is accepted as valid.

Verifiers obtain an Appraisal Policy for Evidence to determine which Endorsements are relevant for a particular Attester. Accepted Claims are those Claims received from an Attester that a Verifier can verify using Endorsements. By default, all accepted Claims are relevant. However, for a given Relying Party, application, or network deployment, certain Claims might insufficiently or extraneously describe the Attester. The Appraisal Policy for Evidence specifies which of the accepted Claims are required.

End of informative comment
4.3 Endorsement Manifests

Start of informative comment

An Endorsement Manifest is a machine-readable (and possibly human-readable) document. The manifest may have a schema for expressing Claims and relating them to the Attester. An Endorsement Manifest also includes measurements that can be compared with Attester Evidence.

Manifests can be integrity protected and authenticated using cryptography. The Endorsement Manifest signer is the entity that asserts the Endorsement Claims.

In a complex supply chain, it is likely multiple Endorsers will produce Endorsement Manifests. For example, an OCM, being the most authoritative Endorser, may be required to sign a manifest describing the component it produces. Endorsers working independently may complete the manufacture of parts at different times making it impractical for one to become the signer of a single manifest for all components. OEMs often build systems from components supplied by OCMs and software vendors. Another aspect of Endorsement Manifests is that manifests provide links between various other manifests such that the combination of manifests describes a device class.

End of informative comment

4.4 Endorsement Manifest Schemas and Technology

Start of informative comment

Endorsement schemas facilitate interoperability between different Endorsers and between Endorsers and Verifiers. They specify metadata that provides an abstract representation of a class of attestable devices. This specification uses the term device template to refer to the human-readable, and possibly machine-readable, expression of the abstract device class. If multiple Endorsers cooperate to produce a device, each supplying different device components or Reference Values, the device template defines a profile of the Endorsements schema to provide a common understanding of the device class.

The CoRIM schema, expressed in CDDL, is used to generate the CoRIM file that typically is rendered in CBOR [18] or JSON [19] format. This is illustrated in Figure 2.

![Figure 2: CoRIM Schema informs CoRIM creation.](image)

A SWID Tag [11] is both a schema and an encoding (i.e., XML [20]) that models software asset management. A SWID Tag is adapted to attestation use cases by [21]. CoSWID [7] is a concise representation of a SWID Tag using CDDL [5] as the data definition language. CoSWID defines an extension to a SWID Tag and can be implemented in a memory efficient manner. CoSWID supports at least two encodings, CBOR and JSON. A SWID Tag and CoSWID define a software ‘tag’ structure that contains metadata and Claims about software packages.
In contrast, Concise Module ID (CoMID) (see Section 5.5.3) tags contain metadata and Claims about hardware and firmware modules. CoMID can be used to describe platforms, devices, and components. Multiple CoMID tags can describe the composition of a device. Multiple vendors may cooperate to issue linked CoMID tags that help describe a supply chain.

CoRIM [22] is a manifest schema that envelopes CoSWID and CoMID expressions. A signed CoRIM may be revoked using a deny list defined by Xcorim (see Section 5.6).

CoRIM is intended to be Root of Trust agnostic but anticipates DICE layering architectures and compositions involving multiple DICE Roots of Trust and Trusted Computing Base (TCB) layers. The TCG Reference Integrity Manifest (RIM) specification [10] uses some of the extensibility features in SWID Tags to reference TPM-centric manifest schemas [10] [17].

For the purposes of this specification, the CoRIM schema is inclusive of both CoMID and CoSWID schemas. CoRIM manifests may be authored using a variety of popular human-readable formats. Often there is rich tooling available as well. The CoRIM Schema is a CDDL representation that can be used with tooling to automate schema compliance checking as part of the CoRIM authoring process (see Figure 3). Authoring tools generate a CoRIM manifest file that is conveyed to Verifiers. CoRIM files are formatted in either CBOR or JSON.

When Verifiers receive the CoRIM file, the manifest schema (CoRIM CDDL) is used to ensure the manifest file complies with expected CoRIM Schema. Having schema compliant manifests ensures interoperability between Endorsers and Verifiers. Verifiers create an internal representation of the information contained in the manifest to complete Evidence appraisal.

---

**Figure 3: Manifest authoring and verification**

---

**End of informative comment**
5 ENDORSEMENT MANIFEST ARCHITECTURE

Start of informative comment

The focus of this Endorsement Manifest architecture is to define the information, data models, and encapsulation for exchanging information between the Endorser and Verifier Roles [1]. Endorsements are information produced by Endorsers and consumed by Verifiers.

Endorsement Manifests contain tags that describe the composition and measurements of a platform, device, component, or software. This specification contains an information model for Endorsement Manifests (See Section 5.2). For brevity, the term *module* may be used instead of platform, device, component, or package. Modules have attributes that are described by *tags*. Software tags describe attributes related to software and software lifecycle. Hardware tags describe attributes related to hardware and firmware. A tag may reference other tags that together model module architecture. Tags contain Claims that describe the trustworthiness properties of the module. Claims consist of a subject, object, and predicate. The predicate defines semantics for relating a subject to an object. Typical Claim expressions relate a Target Environment to a set of Reference Values (e.g., measurements).

Endorsement Manifest Claims can have varying properties, including:

- Claims may apply to a class of device or software where a reference measurement is matched to Evidence.
- Claims may be unique to a module instance.
- Claims may apply to a class of device or software where no matching Evidence is expected.
- Claims may assert a cryptographic key and identifier that can be used to authenticate the module.
- Claims may assert that a key can be used to sign Evidence.

This specification defines both an Endorsement Manifest information model and a data model. The data model may be realized using data definition language. There are several popular data definition languages such as CDDL and ASN.1. Data definition languages generate data encoded output suitable for conveyance over a wire protocol. Common data encoding formats may include JSON, CBOR, BER, and XML. This specification does not specify a data encoding format. Nor does it require a specific data definition language, although it uses CDDL.

Claims about software are described by the SWID Tag and CoSWID data models. Claims about hardware or firmware are described by the CoMID data model (see Section 5.5).

This specification combines CoSWID and CoMID with COSE [6] signatures (and alternatively JOSE [23] signatures) to form a CoRIM manifest.

End of informative comment

5.1 Endorsement Manifest Requirements

The Endorsement Manifest (CoRIM) and encapsulated tags (CoMID, CoSWID) have the following interoperability and security requirements:

1) Endorsement Manifests MUST be machine-readable.
2) Endorsement Manifests MUST be authenticated by an Endorser (entity).
3) Endorsement Manifest signatures MUST have a validity period.
4) Endorsement Manifests MUST support multiple tags.
5) Module Tags MUST support linked tags relationships where the relationship type is explicitly stated.
6) Module Tags MUST support reference and endorsed Claims. (See Section 5.2.3)
7) Module Tags SHOULD support module identity Claims.
8) Module Tags MUST support Reference Values Claims that match DiceTcbInfo Evidence.
Start of informative comment

Designers of Endorsements infrastructure should consider scalability challenges given the potential for millions or billions of instance Endorsements.

Designers of Endorsements infrastructure should anticipate multiple Endorsers (i.e., supply chain entities) working independently to produce complex devices and Endorsements structures.

End of informative comment

5.2 Information Model for Endorsement Manifests

This section describes various aspects of the information model for Endorsement Manifests. Normative language in the information model applies to the Endorsement Manifest data model as defined in section 5.5.

5.2.1 Attester Device Composition

Start of informative comment

Endorsement Manifests rely on module vendors to define Claims in a manner that allows universal verification. An interoperable representation of the module definition can be provided via CoMID tags. Tag creation involves mapping the trustworthiness properties of real-world modules to a tag representation that contains trustworthiness Claims (see Figure 4).

Tags have a tag identifier (e.g., tag-id) that identifies tag instances. Tags have a lifecycle. They are created, replaced, updated, and deleted. As part of tag lifecycle management, one tag might link to another tag. For example, a tag that replaces another tag might link to the replaced tag for historical continuity. A linked tag relationship defines link semantics. Additionally, tags are linked when the complete set of Claims is divided across multiple tags.

Software can exist independently of the module it runs on. Software ID (SWID) Tags describe software packages. A software tag has a tag identifier (e.g., tag-id). The software package is identified by its software name.

CoRIM manifests contain both CoMID and CoSWID tags.

End of informative comment

![Figure 4: Module to Tag Mapping](image-url)
5.2.2 Manifest Information Model

This specification defines CoRIM, which is a concise representation of an Endorsement Manifest. The overall structure is illustrated in Figure 5. A CoRIM is identified by a universally unique manifest identifier. A CoRIM has a manifest locator. Verifiers may not have all the information needed to verify Evidence. A manifest locator contains Web references to services containing additional or updated manifests. A CoRIM can have both hardware and software tags. ComID tags contain Claims about hardware and firmware. Claims are statements that relate measurements to Target Environments. CoSWID tags contain Claims about software.

Typically, a manifest is digitally signed. Signing signifies that the Claims contained in tags are valid according to the signer.

End of informative comment

A CoRIM manifest MUST contain a statistically unique Manifest ID.

5.2.3 CoMID Tag

The CoMID Tag information model (see Figure 6) includes a Tag ID that is universally unique. Tag metadata includes a tag version and a list of supply chain entities with their tag lifecycle role. Tags also contain Claims and links to other tags.

The Tag ID and tag version identify tags regardless of the entity that created them. Tag IDs are statistically unique: this property enables a tag lifecycle that may involve recycling previously issued tags. The tag version is part of the Tag metadata field in Figure 6, but is not shown explicitly.

CoMID defines several types of Claims:

- Reference Claims contain Reference Values that are expected to match attested Evidence. Evidence omitting Claims found in Reference Claims, or Evidence containing additional Claims not found in Reference Claims, could be an indication of Attester compromise.
• Endorsed Claims contain assertions about the module that do not require Evidence matching to be accepted by the Verifier. Endorsed Claims may describe properties that are immutable or may reflect assurance properties derived from testing or other supply chain processes.
• Identity Claims contain cryptographic credentials used for authentication. The CoMID tag containing Identity Claims is used by Verifiers to cryptographically challenge the Attester. Credentials could identify a single device or component instance or could identify a group. Identity Claims include cryptographic keying material where the security property of the keying material determines group or singleton identity semantics.
• Attestation Key Claims contain cryptographic keys used to integrity protect Evidence. Verifiers use these Claims to verify signed Evidence.

CoMID tags that contain both class and instance Claims could have scalability and privacy implications. Class Claims can be applied to a large community of modules while instance Claims are unique to a specific module. Instance Claims may have privacy considerations if Claims are used to track individuals.

CoMID tags can link to other tags (i.e., linked tags). Linking involves identifying the linked tag that declares a linking relationship. Link tag relationships describe how to process the linked tag.

End of informative comment

5.2.3.1 Module Metadata
Each CoMID tag MUST contain a statistically unique Tag ID.

CoMID tag metadata MUST include a tag version.

5.2.3.2 Claims
CoMID tags MUST be able to include Reference Claims and Endorsed Claims.

CoMID tags MUST be able to include Claims with class measurements.

CoMID tags SHOULD be able to include Claims with cryptographic identities.

CoMID tags SHOULD support Claims with instance measurements.
5.2.4 Claims

Start of informative comment

Claim triples (see Figure 7) are expressions that consist of a subject, object, and predicate. The predicate relates the subject to the object.

The Claim triple for Reference Values relates reference measurements to a Target Environment. For Reference Value Claims, the subject identifies a Target Environment, the object contains measurements, and the predicate asserts that these are the expected (a.k.a., reference) measurements for the Target Environment.

The Claim triple for Endorsed Values declares additional measurements that are valid when a Target Environment has been verified against reference measurements. For Endorsed Value Claims, the subject is either a Target or Attesting Environment, the object contains measurements, and the predicate defines semantics for how the object relates to the subject.

The Claim triple for Device Identity relates one or more cryptographic keys to a device. The subject of an Identity triple uses an instance or class identifier to refer to a device, and a cryptographic key is the object. The predicate asserts that the identity is authenticated by the key. A common application for this triple is device identity.

The Claim triple for Attestation Keys relates one or more cryptographic keys to an Attesting Environment. The Attestation Key triple subject is an Attesting Environment whose object is a key. The predicate asserts that the Attesting Environment signs Evidence that can be verified using the key.

Target Environments may be described by class attributes or by instance identifiers. Class attributes describe a collection of modules. Instance identifiers distinguish each instance of a module.

End of informative comment

This specification initially defines four triples:

1) REFERENCE-VALUES triple:
   a. Subject: Target Environment ($TE) – identified by class name or instance identifier
   b. Object: One or more measurement values ($MV)
   c. Predicate: The $TE environment has these reference measurements $MV

2) ENDORSED-VALUES triple:
   a. Subject: A Target or Attesting Environment ($E) – identified by class name or instance identifier
   b. Object: One or more measurement values ($MV)
   c. Predicate: The $E environment has these endorsed measurements $MV

3) DEVICE-IDENTITY triple:
   a. Subject: The device ($D) - identified by class name or instance identifier
   b. Object: One or more device identity keys ($K)
   c. Predicate: The $D device is authenticated by the key ($K)

4) ATTESTATION-VERIFICATION-KEY triple:
   a. Subject: Attesting Environment ($AE) – identified by class name or instance identifier
   b. Object: One or more keys ($K)
   c. Predicate: The $AE signs Evidence that can be verified with key ($K)
This specification defines Reference Value Claims that correspond to the DICE tcbinfo Evidence extension (see [14]). The tcbinfo Evidence Claims include class attributes and measurements that easily match the Claims described by the REFERENCE-VALUES triple.

A Claim MUST be a logical triple having a subject, object, and predicate, where the predicate relates the subject to the object.

A Claim triple MUST have at least one qualifying attribute as part of its predicate.

A set of Claims MUST be populated with data matching any of the defined triples.

The data definition for Claim triples created by the Endorser MUST be extensible.

5.2.5 Environments

The information model describing an Environment (see Figure 8) consists of a Class Name and/or an Instance Name. The Class Name consists of class attributes that distinguish the class of environment from other classes. The class attributes include class-id, vendor, model, layer, and index. The Endorser determines which attributes are needed.

- The class-id attribute identifies the environment via well-known identifier. Typically, the class-id is an object identifier (OID) or universally unique identifier (UUID). Use of this attribute is preferred.
- The vendor attribute identifies the entity responsible for choosing values for the other class attributes that do not already have naming authority.
- The model attribute describes a product, generation, and family.
- The layer attribute is used to capture where in a sequence the environment exists. For example, the order in which bootstrap code is executed may have security relevance.
- The index attribute is used when there are clones (i.e., multiple instances) of the same class of environment. Each clone is given a different index value to disambiguate it from the other clones. For example, given a chassis with several network interface controllers (NIC), each NIC can be given a different index value.

The Class Name SHALL consist of at least one of five attributes as follows:

- class-id – An environment class identifier. An environment SHOULD have a class-id.
For a class-id example, see hwType as defined in RFC4108 [24].

The Instance Name SHALL consist of one of the following attributes:

- uuid – A universally unique identifier that is reliably bound to the environment.
- Ueid – A unique enough identifier that is reliably bound to the environment.

Measurements can be of a variety of things including software, firmware, configuration files, read-only memory, fuses, IO ring configuration, partial reconfiguration regions etc. Measurements comprise raw values, digests, or status information.

An environment has one or more measurable elements. Each element can have a dedicated measurement or multiple elements could be combined into a single measurement. Measurements are identified by a class or instance identifier: this is the key in a key-value pair. Figure 9 illustrates structures containing class and instance measurement values. The left-most structure in Figure 9 illustrates a measurement key-value pair. Figure 9 shows both a class and instance key-value pair even though, in reality, a measurement key-value pair represents either a class measurement or an instance measurement, not both.

Class measurements apply generally to a class of something. Instance measurements apply to a specific instance of something. Environments identified by a class identifier have measurements that are common to the class. Environments identified by an instance identifier may have measurements that are specific to that instance. A cryptographic group key is an example of a class identifier, given a group size larger than one. A cryptographic key is an example of an instance identifier.
5.2.6.1 Class Measurements

This specification defines a set of class measurements consisting of version, security version number (SVN), flags, digests, raw value, and raw value mask.

The version typically changes whenever an environment is updated.

The svn (security version number) typically changes only when a security relevant change is made to an environment.

The flags field describes security relevant operational modes. For example, security relevant operational modes include whether the environment is in a debug mode, recovery mode, not fully configured, not secure, not replay protected or not integrity protected. The flags field indicates which operational modes are relevant to the module.

The digests field contains the digest and hash algorithm (see [25]) used to generate a digest of an environment.

The raw value field contains the actual (not hashed) value of the element. The raw value mask indicates which bits in the raw value field are relevant for verification. A mask of all ones (“1”) means all bits in the raw value field are relevant. Multiple values could be combined to create a single raw value attribute. The vendor determines how to pack multiple values into a single raw value structure. The same packing format is used when collecting Evidence so that Reference Values and collected values are bit-wise comparable. The vendor determines the encoding of raw value and the corresponding raw value mask.

The set of possible class measurements is not limited to those defined here.

Class measurements SHALL include at least the following attributes:

- version – A revision control identifier.
- svn – A value that increases with each security relevant change.

A minimum SVN identifies the lowest possible SVN below which the Target Environment is determined to be unacceptable.
• **flags** – A value that identifies a security relevant operational mode. The operational mode SHOULD encompass several modes including debug, recovery, not-secure, not-configured, not-replay-protected, not-integrity-protected. A bit value of 1 means the mode that corresponds to that bit is operational. A bit value of 0 means the mode that corresponds to that bit is not operational. If the flags measurement is omitted the operational mode is unknown and not expected.

• **digests** – A set of tuples containing a digest and a hash algorithm identifier. The digest of a raw value is computed using the hash algorithm identifier.

• **raw value** – A bit array of the raw (actual) values.

• **raw value mask** – A bit array mask. A bit value corresponds to a bit value in the raw value at the same array position as the mask bit. A mask value of 1 exposes the corresponding bit in the raw value. A mask value of 0 hides the corresponding bit in raw value. The raw value mask SHOULD NOT be included in Evidence.

5.2.6.2 Instance Measurements

**Start of informative comment**

There are a variety of possible instance measurements. This specification defines five instance measurements that include: **MAC address**, **IP address**, **serial number**, **UUID**, and **UEID**. The measurements data structure (see Figure 9) could be extended to include additional instance measurements.

The Claim triple defines semantics for when it is appropriate (or inappropriate) to include instance measurements.

The set of possible instance measurements is not limited to those defined here.

**End of informative comment**

Instance measurements expressible in a measurement values map SHALL include at least the following attributes:

• MAC address – A EUI-48 or EUI-64 MAC address
• IP address – An IPv4 or IPv6 address
• Serial Number – A product serial number
• UUID – A universally unique identifier
• UEID – A unique enough identifier

The measurement values map created by the Endorser MUST be extensible.

5.2.7 CoSWID Information Model

**Start of informative comment**

The concise software ID tag (CoSWID) information model is formally documented in [7]. This section summarizes the CoSWID schema that consists of a software ID tag structure (see Figure 10) that contains three primary sub-structures:

(i) **package** and **tag metadata**,  
(ii) **payload**, and  
(iii) **links** to related CoSWID or CoMID tags.

The payload consists of a set of Claims called a resource collection. There are three types of Claims:

(i) **path elements**,  
(ii) **resources**, and  
(iii) **processes**.

Path elements consist of directories and files. Directories have a directory name. Files have a filename and measurement values. Path elements describes pathnames to the files that, when hashed, produce a digest that is
CoSWID resource collections may also contain resources. A resource is anything that can be named that isn’t otherwise described by path elements or processes. This specification does not anticipate use of CoSWID resources.

CoSWID resource collections may contain process Claims. A process Claim is a measurement over contents of memory. This specification does not anticipate use of CoSWID process.

CoSWID schema also defines Evidence which inherits the payload schema. However, because this specification defines Endorsement Manifests, CoSWID Evidence is out-of-scope.

End of informative comment

5.3 Device Composition Using Multiple Tags

A device can be partitioned into its component parts (e.g., components, modules, environments, elements etc.). Sub-components can be separately attested or reported. The various sub-components may have Endorsement Claims and may also have different suppliers that may create separate CoRIM tags.
Recall that a tag is a collection of Claims about the device or its components. One or more tags could be used to describe the device or its components. Multiple manifests can be used to authenticate, and integrity protect the tags.

A CoRIM might contain multiple tags that describe different devices or components. Tags that describe the same device or component are associated through linking relationships. This section explains linked tags relationships. Tag linking semantics are the same whether working with multi-chip modules or across multiple different modules.

A multi-tag module refers to a device or component that employs multiple tag structures. A multi-tag manifest refers to a single manifest (e.g., CoRIM) that contains multiple tags.

5.3.1 Tag Linking

Linked tags are used to create multi-tag compositions and manage tag lifecycles. Tag linking can occur between any tag type (e.g., CoMID, CoSWID) according to tag relationship semantics.

Linked tag relationships can express device compositions. A multi-tag module can describe a hierarchical composition where a root tag might link to second layer tags, that in turn link to third layer tags etc.

Linked tag relationships help describe multiple Endorsers, each asserting different Claims about a composition. Tag link relationships can also help to correct malformed tags and update or patch previously minted tags.

Module tags MUST support inter-tag linking.

Linked tags MUST contain a tag identifier and a link relationship identifier.

A child tag MUST have at least one parent tag.

Linking SHOULD be acyclic.

Linked tags MUST support CoMID-to-CoMID linking.

Linked tags SHOULD support CoMID-to-CoSWID, CoSWID-to-CoSWID and CoSWID-to-CoMID linking.

5.3.1.1 Linked Tag Relationship Types

Linked tag relationship types are specific to the type of tag originating the link and the type of tag that is the target of the link.

CoMID to CoMID and CoMID to CoSWID relationship types:

- 'replaces' to correct erroneous tags
- 'supplements' to augment existing Claims with additional Claims

CoSWID relationship types are documented in [7].

5.3.1.2 Linked Tag Relationship Examples

This section describes examples of linked tags to highlight a particular relationship type. It uses the two relationship types (i.e., replaces, supplements) that apply to CoMID to CoMID and CoMID to CoSWID linking.
5.3.1.2.1 Replaces Relationship Example

Start of informative comment

In this example, Alice created a tag in error. Alice corrects the mistake by recreating the tag (tag-id: 11) and increments the tag version so the tag containing erroneous content can be ignored or deleted. Note that, in Figure 11, the position of the tag creator relative to the graphical representation of the tag itself is irrelevant.

End of informative comment

![Figure 11: The ‘REPLACES’ relationship example](image)

5.3.1.2.2 Supplements Relationship Example

Start of informative comment

In this example, Bob provides a firmware validation service for one of Alice’s devices. Upon completion of the validation tests, Bob creates Claim-B to describe the validation test results. The device tested in Claim-B matches the device manufactured by Alice in Claim-A. Both tags may be archived for later inspection. Note that, in Figure 12, the position of a tag creator relative to the graphical representation of the tag is irrelevant.

End of informative comment

![Figure 12: The ‘SUPPLEMENTS’ relationship example](image)

5.3.1.3 Linked Tags Relationship Requirements

CoMID linked tag relationship types created by the Endorser MUST be extensible.

Start of informative comment

The extensibility provides an Endorser with the flexibility to define its own relationship between tags in the supply chain as per its own use case requirements.

End of informative comment

CoMID to CoMID linked tags MUST support supplements and replaces relationship types.

CoMID to CoSWID linked tags MUST support the supplements relationship type.

CoSWID tags MAY link to CoMID tags using the supplemental relationship type.

Given the supplements relationship type, the CoMID tag MUST contain Claims that augment the linked tag’s Claim set.

Given the ‘replaces’ relationship type, the CoMID tag MUST contain corrections to the linked tag. The tag ID MUST remain the same. The tag version MUST be incremented.

See also Section 5.5.3.2 and Section 5.5.4.6.
5.3.2 Multi-Tag Manifests

A multi-tag manifest contains multiple tags that may or may not be linked. Non-linked tags contain Claims about modules that are unrelated. Linked tags describe related modules. Linking semantics are at the tag level. Linked tags relationships describe the purpose for linking multiple tags. Often, tags are linked because they describe a common device. Tags contain Claims that describe module components and their measurements.

A device consisting of several components can be described using multiple tags where the Claims associated with different tags describe a device and its sub-components. For example, a top-level tag (e.g., the device tag) might link to tags that describe each subcomponent. A CoMID containing linked tags is known as a multi-tag module. Multi-tag modules typically describe the composition of modules because each supply chain entity creates a tag for the modules it produces, iteratively. Alternatively, a supply chain entity might merge the various Claim triples into a single tag that fully describes a multi-module device.

A corim-map SHALL be linked if the tags describe the same device or related components. Concise-tags in a corim-map SHALL NOT be linked if the tags describe unrelated devices or components.

5.3.3 Multi-Endorser Manifests

A multi-tag manifest could be divided into multiple manifests signed by different Endorsers. Each of those multiple manifests is called a multi-Endorser manifest. Each multi-Endorser manifest could itself be a multi-tag manifest that might be divided into multi-Endorser manifests as well.

For example, a supply chain might consist of a device vendor that works with several component suppliers that produce printed circuit boards (PCBs) and discrete logic components for the PCBs. There may be add-in PCBs that attach to the main board that further consist of discrete logic components. Each supplier independently issues Endorsement Manifests containing tags for each module.

A tag creator might link to other tags created by other entities. The system of linked tags is a composition. The tag creator ensures the multi-tag module is partitioned correctly to ensure the correct supplier can assert the desired Claims when manifests are issued (i.e., digitally signed). Typically, the entity that issues a manifest should be the entity that provides the most credibility for the Claims in that manifest.

Additionally, a composer could collapse a multi-Endorser manifest into a single Endorser manifest by copying Claims from the various multi-Endorser manifests into a tag created by a singleton Endorser. It is assumed that the composer is trusted by the Verifier community to combine the tags and Claims. The Verifier Appraisal Policy for Evidence determines which Endorsers must issue which manifests.

An example multi-Endorser manifest (see Figure 13) describes a device (see Figure 4) that produces two tags and five Claim sets (D1, C1, C2, S1, S2). D1 and C1 are included in Tag-1 in manifest M1 that is issued by Endorser E1. Claim sets C2, S1 and S2 are included in Tag-2 in manifest M2 that is issued by Endorser E2. There is a supplements linked tag relationship between Tag-1 and Tag-2. The exact details of the relationship depend upon the exact nature of the Claims.

The third Endorser E3 adds valuable information (such as validation of device compliance) to the device described by M1 and M2 such as compliance validation. The results of compliance validation are expressed in E3 Claims in Tag-3. The Endorser E3 issues a third manifest M3 that links Tag-3 to Tag-1 where the E3 Claims identify C1 as a related module.
5.4 Archive Files

Verifiers need to obtain the manifests, certificates, and trust anchors necessary for appraisal. CoRIM locator maps (e.g., corim-locator-map and link-entry) may facilitate the discovery and download of such files.

The collection of manifests and certificates used for verification can be made available to Verifiers by archiving them in a file format such as TAR [26]. Alternatively, the manifests that describe device composition and measurements may be provisioned to a Verifier by an Endorser.

Additionally, a Verifier may be provisioned with trust anchors to verify the manifest signature to ensure the integrity and authenticity of CoRIM’s that are not directly provisioned.

The security of unsigned manifests and certificates in a file archive is out of scope for this specification.

Multiple CoRIM files can be packaged into a file archive format but note that specification of archive formats is outside the scope of this specification.

Archive files might contain related certificates or other files useful to Verifiers.

5.5 Data Model for Concise Reference Integrity Manifest (CoRIM)

This section describes a data model definition for CoRIM, also known as ‘CoRIM schema’. The Endorsement Manifest data model consists of a top-level map structure containing either a signed CoRIM or an unsigned CoRIM. A CoRIM describes one or more devices and their respective components, and sub-components (i.e., modules).

A single manifest could describe multiple modules.
Multiple manifests could describe the same module.

A CoRIM signature indicates that the signer Endorses the Claims.

The entity that created the manifest could be different from the entity that signed it. Therefore, CoRIM can specify these entities and which actions each can perform.

An unsigned CoRIM endorses Claims by a method (such as a conveyance protocol or an enveloped data structure) that is not defined in this specification.

**End of informative comment**

### 5.5.1 Data Model Conventions

#### 5.5.1.1 Data Model Notation

Start of informative comment

The Concise Reference Integrity Manifest (CoRIM) data model is defined in tabular notation (see Table 1) and in Unified Modeling Language (UML). The data model captures the semantics of the CoRIM schema while allowing adaptation to various data definition languages (DDL) and encoding formats. A CDDL definition of the CoRIM schema is found in [22]. An example CBOR encoding is found in APPENDIX A – ATTESTATION EXAMPLE.

Data model names use a dash character (‘-’) to separate names consisting of multiple strings. However, UML prohibits inclusion of the dash characters in class names. Therefore, an underscore (‘_’) is used instead.

The schema in tabular form in Table 1 captures several properties of a description grammar.

End of informative comment

<table>
<thead>
<tr>
<th>NUMERIC KEY (NK)</th>
<th>STRING KEY (SK)</th>
<th>VALUE / TYPE</th>
<th>OPTIONAL (OPT)</th>
<th>MULTIPlicity (MUL)</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric identifier</td>
<td>String identifier</td>
<td>Type specifier or data</td>
<td>‘T’ is optional; otherwise, tuple is required.</td>
<td>‘*’ – Zero or more</td>
<td>Additional information</td>
</tr>
</tbody>
</table>

Start of informative comment

Data model conventions follow a `<key><value>` tuple syntax.

The `<key>` portion of a tuple can be represented by a Numeric Key (NK) and/or String Key (SK). Depending on the data encoding, either the numeric key or the string key might be used in a `<key><value>` tuple.

The `<value>` portion of a tuple describes its value or type. Types may refer to other types that are recursively substituted to construct the `<value>`. Additionally, a `<value>` may refer to yet another `<key><value>` tuple, data type, boolean, or scalar.

The format for a statement in the schema (expressed here as a row in a table similar to Table 1) contains columns for keys (numeric and string), value, optionality, multiplicity, and description information.

The key columns are *Numeric Key* and *String Key*. They contain the `<key>` portion of the `<key><value>` pair. Binary encodings such as CBOR use the numeric keys and text encodings, such as JSON, use string keys.

The *Value Type* column describes the `<value>` portion of a given tuple. It contains a statement defined in another table in the schema or a type structure that is defined in the data types table (see 5.5.3.3). Note that braces (‘{}’) contain additional information to support correct encodings.

The *Optional* column in the table indicates whether a tuple is optional or required. The character ‘T’ in this column indicates the statement is optional, otherwise the statement is required.
The **Multiplicity** column indicates whether a statement can occur multiple consecutive times. If a single instance of a statement is intended, then the multiplicity column is empty. If the statement can occur multiple times, this column contains either the ‘+’ character (i.e., 1 or more) or the ‘*’ character (i.e., zero or more).

The **Description** column contains normative statements corresponding to the `<key>`<value> tuple.

End of informative comment

5.5.1.2 Typographical Notation

Start of informative comment

The tabular notation (see Table 1) uses a ‘key’ to ‘value’ (or ‘key’ to ‘type’) mapping convention where the key is expressed in both numeric and text representations. DDLs that define numeric code points, like CBOR, use the numeric key while those using text keys, like JSON use the string key. The ‘type’ describes the type or range of possible data allowed. Structure naming follows the typographical conventions contained in Table 2.

A ‘type’ is a data type or constant data value. A ‘type choice’ is an extensible data type where differently typed data can subsequently be added to the schema. A non-extensible ‘type choice’ is a data type definition that cannot be extended by the specification. A ‘tagged type’ is a ‘type’ that is prefaced with a well-known data type tag such as the Concise Binary Object Representation (CBOR) Tags in the IANA registry (See [27]). A ‘map’ is a collection of name-value pairs. A ‘group’ is set of name-value pairs that can be added to a map. A ‘group choice’ is an extensible group. A ‘flags’ structure enumerates a set of key names.

End of informative comment

<table>
<thead>
<tr>
<th>TYPE TRAIT</th>
<th>CDDL EXAMPLE</th>
<th>FORMAT FOR VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>int</td>
<td>NAME-type</td>
</tr>
<tr>
<td>Type choice (extensible)</td>
<td>int / text</td>
<td>$NAME-type-choice</td>
</tr>
<tr>
<td>Type choice (non-extensible)</td>
<td>int / text</td>
<td>NAME-type-choice</td>
</tr>
<tr>
<td>Tagged type</td>
<td>#6.123(int)</td>
<td>tagged-$NAME-type</td>
</tr>
<tr>
<td>Map</td>
<td>{ 1 =&gt; int, 2 =&gt; text }</td>
<td>NAME-map</td>
</tr>
<tr>
<td>Group</td>
<td>( 1 =&gt; int, 2 =&gt; text )</td>
<td>NAME-group</td>
</tr>
<tr>
<td>Group choice</td>
<td>( 1 =&gt; int // 2 =&gt; text )</td>
<td>$$NAME-group-choice</td>
</tr>
<tr>
<td>Flags</td>
<td>&amp; (a: 1, b: 2)</td>
<td>NAME-flags</td>
</tr>
<tr>
<td>Flags (extensible)</td>
<td>&amp; (a: 1, b: 2)</td>
<td>$NAME-flags</td>
</tr>
</tbody>
</table>

Table 2: Typographical notation conventions

Start of informative comment

A typographical convention for key names prepends a namespace identifier to the string key with the ‘.’ character as a separator. For example, given a namespace identifier of corim and a key name of device, a qualified key name would be corim.device.

End of informative comment

5.5.2 CoRIM Schema

Start of informative comment

The creator of a CoRIM constructs either an **unsigned-corim** or a **signed-corim** (see Figure 17). This section and its subsections describe the data model of the CoRIM schema. Later sections describe the data model for the manifest tags, either CoMID (see Section 5.5.3) or CoSWID (see Section 5.5.4).

An example of CDDL for CoRIM is in APPENDIX C – CORIM CDDL EXAMPLES, see also [22].

End of informative comment

5.5.2.1 concise-reference-integrity-manifest

The top-level CoRIM structure is described in Table 3.
5.5.2.2 corim-map

The structure of an unsigned CoRIM is described in Table 4.

Start of informative comment

An unsigned CoRIM contains a manifest identifier, CoMID or CoSWID tags, and optionally references to services where additional related CoRIMs, certificates or other Endorsements can be obtained. The unsigned CoRIM also may identify profiles that this CoRIM satisfies. A profile specifies which of the optional parts of a CoRIM are required, which are prohibited, and which extension points are exercised and how.

An unsigned CoRIM payload isn’t integrity protected or authenticated. Integrity protection and authentication can be applied in a variety of ways. This specification defines one possibility using COSE signing (see Section 5.5.2.3).

Note that a CoRIM may contain tags that describe multiple unrelated modules.

End of informative comment

An unsigned CoRIM SHOULD NOT be trusted unless contained in a signed CoRIM or its integrity and authenticity has been established by some other means.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>corim.id</td>
<td></td>
<td></td>
<td>MUST identify a manifest instance</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>corim.tags</td>
<td></td>
<td>+</td>
<td>One or more CoMID or CoSWID tag</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>corim.dependent-rims</td>
<td>T</td>
<td></td>
<td>An Internet service that supplies additional, possibly dependent, manifests or other related files</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>corim.profile</td>
<td>T</td>
<td>+</td>
<td>Profiles that if specified MUST be processed before processing any tags. If a profile isn’t recognized the tags MUST NOT be processed.</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>corim.rim-validity</td>
<td>T</td>
<td></td>
<td>The validity period of the RIM contents.</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>corim.entities</td>
<td>T</td>
<td>+</td>
<td>A list of entities involved in the CoRIM lifecycle.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>corim-map-extension</td>
<td>*</td>
<td></td>
<td>Extension point for future expansion</td>
</tr>
</tbody>
</table>

Table 4: Format of corim-map structure.

5.5.2.2.1 corim-locator-map

The CoRIM locator structure is described in Table 5.

Start of informative comment

The CoRIM locator map contains pointers to repositories where dependent manifests, certificates or other information can be retrieved by a Verifier. The Verifier may have additional resources it checks to obtain the Endorsements it needs.

End of informative comment

The CoRIM SHOULD reference other manifests, certificates, or Endorsements upon which this CoRIM depends.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>corim.href</td>
<td></td>
<td></td>
<td>Pointer to a service that supplies dependent files or records</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>corim.thumbprint</td>
<td>T</td>
<td></td>
<td>Digest of the file or record referenced by corim.href.</td>
</tr>
</tbody>
</table>

Table 5: Format of corim-locator-map structure.
5.5.2.3 signed-corim
The structure of a signed CoRIM is described in Table 6.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>corim.cose-signature</td>
<td>#6.18(COSE-Sign1-corim)</td>
<td></td>
<td></td>
<td>An IANA global content tag is defined for COSE-Sign1-corim</td>
</tr>
</tbody>
</table>

Table 6: Format of signed-corim structure.

5.5.2.3.1 COSE-Sign1-corim
The CoRIM signature structure is described in Table 7.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>protected</td>
<td>bstr .cbor protected-corim-header-map</td>
<td></td>
<td></td>
<td>A CBOR encoded COSE header that is protected by the COSE signature</td>
</tr>
<tr>
<td>N/A</td>
<td>unprotected</td>
<td>unprotected-corim-header-map</td>
<td></td>
<td></td>
<td>A COSE header that is not protected by the COSE signature.</td>
</tr>
<tr>
<td>N/A</td>
<td>payload</td>
<td>bstr .cbor tagged-corim-map</td>
<td></td>
<td></td>
<td>A CBOR encoded tagged CoRIM</td>
</tr>
<tr>
<td>N/A</td>
<td>signature</td>
<td>bstr</td>
<td></td>
<td></td>
<td>A COSE signature block</td>
</tr>
</tbody>
</table>

Table 7: Format of COSE-Sign1-corim structure.

5.5.2.3.2 protected-corim-header-map
The protected CoRIM header structure is described in Table 8.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>corim.alg-id</td>
<td>integer</td>
<td></td>
<td></td>
<td>Signature algorithm identifier (see [25])</td>
</tr>
<tr>
<td>3</td>
<td>corim.content-type</td>
<td>&quot;application/rim+cbor&quot;</td>
<td></td>
<td></td>
<td>MIMIE content type identifier</td>
</tr>
<tr>
<td>4</td>
<td>corim.issuer-key-id</td>
<td>bstr</td>
<td></td>
<td></td>
<td>Issuer key identifier</td>
</tr>
<tr>
<td>8</td>
<td>corim.meta-key</td>
<td>bstr .cbor corim-meta-map</td>
<td></td>
<td></td>
<td>CBOR encoded metadata associated with a signed CoRIM</td>
</tr>
</tbody>
</table>

Table 8: Format of protected-corim-header-map structure.

5.5.2.3.3 unprotected-corim-header-map
The unprotected CoRIM header structure is described in Table 9.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>{int}</td>
<td>coro-label</td>
<td>{text}</td>
<td>coro-values</td>
<td>{any}</td>
<td>* Additional data in the COSE signature block</td>
</tr>
</tbody>
</table>

Table 9: Format of unprotected-corim-header-map structure.

5.5.2.3.4 corim-meta-map
The CoRIM meta map structure is described in Table 10.

Start of informative comment
The CoRIM meta map identifies the entity (or entities) that create and sign the CoRIM. This ensures the consumer of the CoRIM is able to identify credentials used to authenticate its creator / signer.

End of informative comment
5.5.2.3.5 validity-map
The CoRIM validity structure is described in Table 11.

Table 10: Format of corim-meta-map structure.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>corim.signer</td>
<td></td>
<td></td>
<td>The entity (or entities) that creates and signs the CoRIM. The entity name SHOULD reference the manifest issuer’s entity name as contained in the manifest issuer’s signing credential (e.g., if the issuer credential is an X.509 certificate, it is the subjectName, subjectAltName, or SubjectPublicKeyInfo).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>corim.signature-validity</td>
<td>validity-map</td>
<td>T</td>
<td>The validity period of a signed manifest</td>
</tr>
</tbody>
</table>

5.5.2.3.6 corim-signer-map
The CoRIM signer structure is described in Table 12.

Table 11: Format of validity-map structure.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>corim.not-before</td>
<td>time-type</td>
<td>T</td>
<td>Beginning of the validity period</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>corim.not-after</td>
<td>time-type</td>
<td></td>
<td>End of the validity period</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>corim.signer-name</td>
<td>$entity-name-type-choice</td>
<td></td>
<td>The name of an organization that performs the signer role.</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>corim.signer-uri</td>
<td>uri-type</td>
<td>T</td>
<td>The registration identifier for the organization that manages the namespace for corim.signer-name.</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>corim.role</td>
<td>$corim-role-type-choice</td>
<td></td>
<td>Manifest lifecycle role.</td>
</tr>
</tbody>
</table>

5.5.2.3.7 corim-entity-map
The CoRIM entity structure is described in Table 13.

Table 12: Format of corim-signer-map structure.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>corim.entity-name</td>
<td>$entity-name-type-choice</td>
<td></td>
<td>The name of an organization that performs the signer role.</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>corim.reg-id</td>
<td>uri-type</td>
<td>T</td>
<td>The registration identifier for the organization that manages the namespace for corim.entity-name.</td>
</tr>
</tbody>
</table>

5.5.2.3.8 Signed CoRIM Requirements
The following requirements apply to the signed CoRIM structure:

1) A signed-corim SHALL identify its signer with a CoRIM meta map structure that has an optional validity period.
2) A corim.signer SHOULD identify the Endorser that creates the manifest.
3) A corim.signer MUST identify the Endorser that issues (signs) the manifest.
4) The corim.signer MUST be identified and SHOULD use a name that is statistically unique.
5) A signed-corim or corim-map SHOULD include corim-locator-map references to the CoRIMs that the current manifest depends upon.
6) A corim-meta-map SHOULD contain a validity-map that defines the manifest validity period.
7) A signed-corim SHALL be invalid after the corim.not-after time is exceeded.
8) If the validity-map is omitted, the signed-corim SHALL be immediately valid.
9) If a CoRIM signing key is compromised, the signing certificate MUST be revoked.
10) A signed-corim SHALL be invalid if the key used to sign the CoRIM is revoked.

5.5.2.4 CoRIM Manifest Content Type and File Extension
The content type for a CoRIM manifest SHALL be "application/rim+cbor".

The filename extension for a file containing a CoRIM manifest SHALL be ".corim".

5.5.2.5 CoRIM Manifest URI/URL Fragments

An example of a tag-id field, the string "1234" is the value of the tag-id:

{ HYPERLINK https://my.manifest-server.com/my-manifests/manifestfilefile.corim#tag-id=1234 }

5.5.3 A Schema for CoMID Tags

An example of CDDL for CoMID is in APPENDIX C – CORIM CDDL EXAMPLES.

5.5.3.1 concise-mid-tag-map
The CoMID tag structure is described in Table 14.
5.5.3.1.1 tag-identity-map
The CoMID tag identity structure is described in Table 15.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>comid.tag-id</td>
<td>§</td>
<td></td>
<td>CoMID tag ID MUST be statistically unique</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>comid.tag-version</td>
<td></td>
<td></td>
<td>CoMID tag version</td>
</tr>
</tbody>
</table>

Table 15: Format of tag-identity-map structure.

5.5.3.1.2 linked-tag-map
The CoMID linked tag structure is described in Table 16.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>comid.linked-tag-id</td>
<td>§</td>
<td></td>
<td>Tag ID of the linked tag; MAY be a CoMID or a CoSWID tag</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>comid.tag-rel</td>
<td>§</td>
<td></td>
<td>CoMID linked tag relationship (see Section 5.3.1)</td>
</tr>
</tbody>
</table>

Table 16: Format of linked-tag-map structure.

5.5.3.1.3 triples-map
The structure for a CoMID triples map is described in Table 17.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>comid.reference-triples</td>
<td>§</td>
<td></td>
<td>If supplied, MUST contain one or more reference Claims</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>comid.endorsed-triples</td>
<td>§</td>
<td></td>
<td>If supplied, MUST contain one or more endorsed Claims</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>comid.attest-key-triples</td>
<td>§</td>
<td></td>
<td>If supplied, MUST contain one or more attestation key Claims</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>comid.identity-triples</td>
<td>§</td>
<td></td>
<td>If supplied, MUST contain one or more identity Claims</td>
</tr>
<tr>
<td>{int}</td>
<td>{text}</td>
<td>$$triples-map-extension</td>
<td>§</td>
<td></td>
<td>Extension point for future expansion</td>
</tr>
</tbody>
</table>

Table 17: Format of triples-map structure.

The triples-map MUST contain at least one entry.

5.5.3.1.4 version-map
The CoMID version structure is described in Table 18.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>comid.version</td>
<td>§</td>
<td></td>
<td>Dot separated version string</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>comid.version-scheme</td>
<td>§</td>
<td></td>
<td>If supplied, MUST contain version string formatting rules. See [7].</td>
</tr>
</tbody>
</table>

Table 18: Format of version-map structure.
5.5.3.1.5 environment-map
The CoMID environment structure is described in Table 19.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>comid.class</td>
<td>T</td>
<td></td>
<td>If supplied, MUST identify an environment class attributes</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>comid.instance</td>
<td>T</td>
<td></td>
<td>If supplied, MUST identify an environment by instance id</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>comid.group</td>
<td>T</td>
<td></td>
<td>If supplied, MUST identify an environment by group id</td>
</tr>
</tbody>
</table>

Table 19: Format of environment-map structure.

5.5.3.1.6 reference-triple-record
The CoMID reference triple is described in Table 20.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>comid.environment</td>
<td></td>
<td>+</td>
<td>Identifies an Attester Target Environment. Attester Evidence MUST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>comid.measurement</td>
<td></td>
<td>+</td>
<td>match the class or instance values of environment-map.</td>
</tr>
</tbody>
</table>

Table 20: Format of reference-triple-record structure.

5.5.3.1.7 endorsed-triple-record
The CoMID endorsed triple is described in Table 21.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>comid.environment</td>
<td></td>
<td></td>
<td>Identifies a Target Environment or Attesting Environment. The endorsed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>comid.measurement</td>
<td></td>
<td>+</td>
<td>value Claims for environment-map MUST match an environment-map from a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>reference-triple-record of the Attester before it can be associated with</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>the Attester. See Section 6.</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>comid.measurement</td>
<td></td>
<td>+</td>
<td>Contains one or more endorsed measurements.</td>
</tr>
</tbody>
</table>

Table 21: Format of endorsed-triple-record structure.

5.5.3.1.8 identity-triple-record
The CoMID identity triple is described in Table 22.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>comid.environment</td>
<td></td>
<td></td>
<td>Identifies an environment. The environment MUST prove it is the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>comid.measurement</td>
<td></td>
<td></td>
<td>environment-map by authenticating with the key identified by comid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>verification-key</td>
<td></td>
<td></td>
<td>verification-key (e.g., by signing a challenge).</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>+</td>
<td>A set of keys where the entity identified by environment-map possesses the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>key material identified by $crypto-key-type-choice. See [29].</td>
</tr>
</tbody>
</table>

Table 22: Format of identity-triple-record structure.

5.5.3.1.9 attest-key-triple-record
The CoMID attestation key triple is described in Table 23.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>comid.environment</td>
<td></td>
<td></td>
<td>Identifies an Attester's Attesting Environment. The Attesting Environment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>comid.measurement</td>
<td></td>
<td></td>
<td>MUST sign Evidence using the key identified by comid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>verification-key</td>
<td></td>
<td></td>
<td>verification-key.</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>+</td>
<td>A set of keys where the entity identified by environment-map possesses the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>key material identified by $crypto-key-type-choice.</td>
</tr>
</tbody>
</table>

Table 23: Format of attest-key-triple-record structure.
5.5.3.1.10 class-map
The CoMID class map structure is described in Table 24.

The **class-map** MUST contain either the **comid.class-id** or both **comid.vendor** and **comid.model**.

The **class-map** SHOULD contain the **comid.class-id**.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description (normative)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>comid.class-id</td>
<td>$class-id-type-choice</td>
<td>T</td>
<td></td>
<td>The environment’s class identifier.</td>
</tr>
<tr>
<td>1</td>
<td>comid.vendor</td>
<td>vendor-type</td>
<td>T</td>
<td></td>
<td>The environment’s vendor, manufacturer, creator, etc.</td>
</tr>
<tr>
<td>2</td>
<td>comid.model</td>
<td>model-type</td>
<td>T</td>
<td></td>
<td>The environment’s class identifier.</td>
</tr>
<tr>
<td>3</td>
<td>comid.layer</td>
<td>layer-type</td>
<td>T</td>
<td></td>
<td>Identifies the environment’s sequence relative to other environments’ layer value.</td>
</tr>
<tr>
<td>4</td>
<td>comid.index</td>
<td>index-type</td>
<td>T</td>
<td></td>
<td>Identifies clones of the same environment class.</td>
</tr>
</tbody>
</table>

*Table 24: Format of class-map structure.*

5.5.3.1.11 measurement-map
The CoMID measurement map structure is described in Table 25.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>comid.mkey</td>
<td>$measured-element-type-choice</td>
<td>T</td>
<td></td>
<td>Key that identifies the type of element measurement-values-map represents: can be used to describe multiple elements that are measured as a single block.</td>
</tr>
<tr>
<td>1</td>
<td>comid.mval</td>
<td>measurement-values-map</td>
<td>T</td>
<td></td>
<td>Measurement values</td>
</tr>
</tbody>
</table>

*Table 25: Format of measurements-map structure.*

5.5.3.1.12 measurement-values-map
The CoMID measurement values map is described in Table 26.

The **measurement-values-map** MUST contain at least one measurement value.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>comid.ver</td>
<td>version-map</td>
<td>T</td>
<td></td>
<td>A version number measurement value</td>
</tr>
<tr>
<td>1</td>
<td>comid.svn</td>
<td>$svn-type-choice</td>
<td>T</td>
<td></td>
<td>A security related version number measurement value</td>
</tr>
<tr>
<td>2</td>
<td>comid.digests</td>
<td>digests-type</td>
<td>T</td>
<td></td>
<td>A digest of the raw value measurement</td>
</tr>
<tr>
<td>3</td>
<td>comid.flags</td>
<td>flags-map</td>
<td>T</td>
<td></td>
<td>Operational modes that are made permanent at manufacturing time and do not change during normal operation of the Attester.</td>
</tr>
<tr>
<td>N/A</td>
<td>N/A</td>
<td>raw-value-group</td>
<td>T</td>
<td></td>
<td>A raw value measurement (See Section 5.5.3.1.14)</td>
</tr>
<tr>
<td>6</td>
<td>comid.mac-addr</td>
<td>$mac-addr-type-choice</td>
<td>T</td>
<td></td>
<td>EUI-48 or EUI-64 MAC address see [30]</td>
</tr>
<tr>
<td>7</td>
<td>comid.ip-addr</td>
<td>$ip-addr-type-choice</td>
<td>T</td>
<td></td>
<td>IPv4 or IPv6 address</td>
</tr>
<tr>
<td>8</td>
<td>comid.serial-number</td>
<td>serial-number-type</td>
<td>T</td>
<td></td>
<td>Serial number in ASCII</td>
</tr>
<tr>
<td>9</td>
<td>comid.uuid</td>
<td>uuid-type</td>
<td>T</td>
<td></td>
<td>Unique Enough ID</td>
</tr>
<tr>
<td>10</td>
<td>comid.uid</td>
<td>uuid-type</td>
<td>T</td>
<td></td>
<td>Universally Unique ID</td>
</tr>
<tr>
<td>11</td>
<td>comid.name</td>
<td>text</td>
<td>T</td>
<td></td>
<td>Human readable information (e.g., model name)</td>
</tr>
<tr>
<td>{int}</td>
<td>{text}</td>
<td>$$measurement-values-map-extension</td>
<td>*</td>
<td></td>
<td>Extension point for future expansion</td>
</tr>
</tbody>
</table>

*Table 26: Format of measurement-values-map structure.*

5.5.3.1.13 Flags-map
Start of informative comment

The **flags-map** measurement describes Boolean operational modes.
If a flags-map value is not specified, then the operational mode is unknown.

End of informative comment

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>comid.operational-flag-configured</td>
<td>boolean</td>
<td>T</td>
<td></td>
<td>The Target Environment is fully configured for normal operation if the flag is true.</td>
</tr>
<tr>
<td>1</td>
<td>comid.operational-flag-secure</td>
<td>boolean</td>
<td>T</td>
<td></td>
<td>The Target Environment’s configurable security settings are fully enabled if the flag is true.</td>
</tr>
<tr>
<td>2</td>
<td>comid.operational-flag-recovery</td>
<td>boolean</td>
<td>T</td>
<td></td>
<td>The Target Environment is NOT in a recovery state if the flag is true.</td>
</tr>
<tr>
<td>3</td>
<td>comid.operational-flag-debug</td>
<td>boolean</td>
<td>T</td>
<td></td>
<td>The Target Environment is in a debug enabled state if the flag is true.</td>
</tr>
<tr>
<td>4</td>
<td>comid.operational-flag-replay-protected</td>
<td>boolean</td>
<td>T</td>
<td></td>
<td>The Target Environment is protected from replay by a previous image that differs from the current image if the flag is true.</td>
</tr>
<tr>
<td>5</td>
<td>comid.operational-flag-integrity-protected</td>
<td>boolean</td>
<td>T</td>
<td></td>
<td>The Target Environment is protected from unauthorized update if the flag is true.</td>
</tr>
</tbody>
</table>

Table 27: Format of flags-map structure.

5.5.3.1.14 raw-value-group

The raw value CoMID structure is described in Table 28.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>comid.raw-value</td>
<td>$raw-value-type-choice</td>
<td></td>
<td></td>
<td>Bit positions in raw-value-type correspond to bit positions in raw-value-mask-type.</td>
</tr>
<tr>
<td>5</td>
<td>comid.raw-value-mask</td>
<td>raw-value-mask-type</td>
<td>T</td>
<td></td>
<td>The raw-value-mask-type bit MUST be 1 to evaluate the corresponding raw-value-type bit.</td>
</tr>
</tbody>
</table>

Table 28: Format of raw-value-group structure.

5.5.3.1.15 digests-type

The CoMID digest structure is described in Table 29.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>comid.hash-entry</td>
<td>hash-entry</td>
<td></td>
<td>+</td>
<td>A list of hash-entry values: used when multiple hash algorithms generate different digests for the same input data.</td>
</tr>
</tbody>
</table>

Table 29: Format of digests-type structure.

5.5.3.1.16 entity-map

The CoMID entity map structure is described in Table 30.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>comid.entity-name</td>
<td>$entity-name-type-choice</td>
<td></td>
<td></td>
<td>The name of an organization that performs the role as defined in $comid-role-type-choice.</td>
</tr>
<tr>
<td>1</td>
<td>comid.reg-id</td>
<td>uri-type</td>
<td></td>
<td>T</td>
<td>The registration identifier for the organization that manages the namespace for comid.entity-name.</td>
</tr>
<tr>
<td>2</td>
<td>comid.role</td>
<td>$comid-role-type-choice</td>
<td></td>
<td>+</td>
<td>The list of CoMID entity roles. The entity that creates the concise-mid-tag SHOULD include a comid.$module-role-type-choice value of comid.tag-creator.</td>
</tr>
</tbody>
</table>

Table 30: Format of entity-map structure.
5.5.3.2 CoMID Linked Tag Relationships

Start of informative comment

This specification defines two linked tags relationships. Other relationships may be added by other specifications as needed. The `comid.supplements` relationship is especially useful for multi-tag modules and manifests. The `comid.replaces` relationship is useful for managing tag lifecycle.

CoMID tags can link to other CoMID tags or to CoSWID tags. The linking relationships differ for each.

Relationship types are defined by `$tag-rel-type-choice`.

End of informative comment

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>supplements</td>
<td>CoMID: The current CoMID tag contains Claims that augment the linked module’s Claim set, identified by the linked tag. The linked tag module name MUST match the subject of the Claim in the current tag before the new Claims can be combined with the linked tag. CoSWID: N/A</td>
</tr>
<tr>
<td>replaces</td>
<td>CoMID: The current CoMID tag contains corrections to a previously issued tag. The current tag replaces the linked tag. The linked tag MAY be deleted. The tag-id of the current tag MUST remain unchanged from the tag-id of the linked tag. The tag-version of the current tag MUST be greater than the tag-version of the linked tag. CoSWID: N/A</td>
</tr>
</tbody>
</table>

Table 31: CoMID linked tag relationships.

5.5.3.3 CoRIM and CoMID Type Definitions

Start of informative comment

Type definitions describe the type of data expected. Types may also be constant data values.

The type definitions for CoRIM and CoMID are in Table 32. The types provided in Table 32 are provided as guidance, additional types may also be defined.

Table 32 substitutes Type Name with Type Value recursively until it reaches a termination point. In some cases, there are multiple potential terminating Type Values. The Choice column indicates that a choice of multiple Type Values is expected. Only one of the possible choices is selected when Choice is true. The ‘C’ character in the Choice column indicates Choice is true. Additionally, the Type Value column may contain hints for data models that focus on data encoding and representation.

End of informative comment

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Type Value</th>
<th>Choice</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$class-id-type-choice</code></td>
<td>tagged-oid-type</td>
<td>C</td>
<td>The value selected MUST be used in Evidence.</td>
</tr>
<tr>
<td></td>
<td>tagged-uuid-type</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>tagged-int-type</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>$comid.module-role-type-choice</code></td>
<td>comid.tag-creator</td>
<td>C</td>
<td>CoMID entity role types</td>
</tr>
<tr>
<td></td>
<td>comid.creator</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>comid.maintainer</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>comid.tag-creator</strong></td>
<td>Constant : 1</td>
<td></td>
<td>Creator of the CoMID tag</td>
</tr>
<tr>
<td><strong>comid.creator</strong></td>
<td>Constant : 2</td>
<td></td>
<td>Creator of the module</td>
</tr>
<tr>
<td><strong>comid.maintainer</strong></td>
<td>Constant : 3</td>
<td></td>
<td>Person or organization making changes to a module, used when different from the module creator.</td>
</tr>
<tr>
<td>Type Name</td>
<td>Type Value</td>
<td>Choice</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>$version-scheme-type-choice</td>
<td>$coswid.version-scheme</td>
<td></td>
<td>Note that CoMID reuses the CoSWID defined version-scheme; See [7]</td>
</tr>
<tr>
<td>$concise-reference-integrity-manifest-type-choice</td>
<td>tagged-corim-map #6.502(signed-corim)</td>
<td>C</td>
<td>An IANA global content tag that identifies one of corim-map or signed-corim</td>
</tr>
<tr>
<td>$concise-tag-type-choice</td>
<td>corim.concise-mid-tag corim.concise-swid-tag</td>
<td>C</td>
<td>CoRIM tag types</td>
</tr>
<tr>
<td>corim.concise-mid-tag</td>
<td>#6.506(concise-mid-tag-map)</td>
<td></td>
<td>concise-mid-tag-map with IANA global identifier code point</td>
</tr>
<tr>
<td>corim.concise-swid-tag</td>
<td>#6.505(concise-swid-tag-map)</td>
<td></td>
<td>concise-swid-tag-map with IANA global identifier code point</td>
</tr>
<tr>
<td>$corim-id-type-choice</td>
<td>text uuid-type</td>
<td>C</td>
<td>A manifest identifier; SHOULD use UUID</td>
</tr>
<tr>
<td>$corim-role-type-choice</td>
<td>corim.manifest-creator</td>
<td>C</td>
<td>Roles related to manifest lifecycle</td>
</tr>
<tr>
<td>corim.manifest-creator</td>
<td>1</td>
<td></td>
<td>Creator of the CoRIM</td>
</tr>
<tr>
<td>$entity-name-type-choice</td>
<td>text</td>
<td>C</td>
<td>Entities, such as supply chain entities</td>
</tr>
<tr>
<td>$group-id-type-choice</td>
<td>tagged-uuid-type</td>
<td>C</td>
<td>A group identifier</td>
</tr>
<tr>
<td>$vector-type</td>
<td>integer</td>
<td></td>
<td>A clone environment item number</td>
</tr>
<tr>
<td>$instance-id-type-choice</td>
<td>tagged-uuid-type</td>
<td>C</td>
<td>The value selected MUST be used to construct Evidence.</td>
</tr>
<tr>
<td>$ip-addr-type-choice</td>
<td>ip4-addr-type</td>
<td>C</td>
<td>IPv4 or IPv6 address types</td>
</tr>
<tr>
<td>ip4-addr-type</td>
<td>bytes .size 4</td>
<td></td>
<td>IPv4 address, see [30]</td>
</tr>
<tr>
<td>ip6-addr-type</td>
<td>bytes .size 6</td>
<td></td>
<td>IPv6 address, see [31]</td>
</tr>
<tr>
<td>layer-type</td>
<td>integer</td>
<td></td>
<td>An environment layering sequence number</td>
</tr>
<tr>
<td>$mac-address-type-choice</td>
<td>mac48-addr-type</td>
<td>C</td>
<td>MAC address types</td>
</tr>
<tr>
<td>mac48-addr-type</td>
<td>bytes .size 6</td>
<td></td>
<td>EUI-48 MAC address, see [32]</td>
</tr>
<tr>
<td>mac64-addr-type</td>
<td>bytes .size 8</td>
<td></td>
<td>EUI-64 MAC address, see [32]</td>
</tr>
<tr>
<td>$min-svn</td>
<td>svn-type</td>
<td></td>
<td>A minimum security version number</td>
</tr>
<tr>
<td>model-type</td>
<td>text</td>
<td></td>
<td>An environment class identifier string</td>
</tr>
<tr>
<td>$measured-element-type-choice</td>
<td>tagged-oid-type</td>
<td>C</td>
<td>The type of identifier used to identify the measurement contained in comid.mval.</td>
</tr>
<tr>
<td>oid-type</td>
<td>bytes</td>
<td></td>
<td>Object Identifier, see [4]</td>
</tr>
<tr>
<td>$crypto-key-type-choice</td>
<td>tagged-pkix-base64-key-type</td>
<td></td>
<td>A tagged structure containing a base64 encoded pkix public key.</td>
</tr>
<tr>
<td></td>
<td>tagged-pkix-base64-cert-type</td>
<td></td>
<td>A tagged structure containing a base64 encoded pkix certificate that contains a public key.</td>
</tr>
<tr>
<td></td>
<td>tagged-pkix-base64-cert-path-type</td>
<td></td>
<td>A tagged structure containing a base64 encoded pkix certificate path that contains an end entity public key certificate and other certificates in the certificate path.</td>
</tr>
<tr>
<td>tagged-pkix-base64-key-type</td>
<td>#6.554(tstr)</td>
<td></td>
<td>A raw key in DER format base64 encoded. See [29]</td>
</tr>
<tr>
<td>Type Name</td>
<td>Type Value</td>
<td>Choice</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------</td>
<td>--------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>tagged-pkix-base64-cert-type</td>
<td>#6.555(tstr)</td>
<td></td>
<td>An X.509 certificate in DER format base64 encoded. See [33]</td>
</tr>
<tr>
<td>tagged-pkix-base64-cert-path-type</td>
<td>#6.556(tstr)</td>
<td></td>
<td>An X.509 certificate path in DER format base64 encoded.</td>
</tr>
<tr>
<td>profile-type-choice</td>
<td>uri-type, tagged-oid-type</td>
<td>C</td>
<td>CoRIM profile identifier: choice of URI or OID</td>
</tr>
<tr>
<td>$raw-value-type-choice</td>
<td>raw-value-type</td>
<td>C</td>
<td>The extensible type choice that can be bytes, text or yet to be defined tagged values</td>
</tr>
<tr>
<td>raw-value-type</td>
<td>#6.560(bytes)</td>
<td></td>
<td>Bit array</td>
</tr>
<tr>
<td>raw-value-mask-type</td>
<td>bytes</td>
<td></td>
<td>Bit array</td>
</tr>
<tr>
<td>serial-number-type</td>
<td>text</td>
<td></td>
<td>A product serial number</td>
</tr>
<tr>
<td>svn</td>
<td>svn-type</td>
<td></td>
<td>A security version number</td>
</tr>
<tr>
<td>$svn-type</td>
<td>unsigned integer</td>
<td></td>
<td>A monotonically increasing value</td>
</tr>
<tr>
<td>$svn-type-choice</td>
<td>tagged-svn, tagged-min-svn</td>
<td>C</td>
<td>A tagged security version number</td>
</tr>
<tr>
<td>$tag-id-type-choice</td>
<td>text, uuid-type</td>
<td>C</td>
<td>SHOULD use a UUID</td>
</tr>
<tr>
<td>$tag-rel-type-choice</td>
<td>comid-supplements, comid-replaces</td>
<td></td>
<td>See also Section 5.3.1</td>
</tr>
<tr>
<td>comid.supplements</td>
<td>Constant : 0</td>
<td></td>
<td>A CoMID linked tags relationship type</td>
</tr>
<tr>
<td>comid.replaces</td>
<td>Constant : 1</td>
<td></td>
<td>A CoMID linked tags relationship type</td>
</tr>
<tr>
<td>tag-version-type</td>
<td>integer .default 0</td>
<td></td>
<td>A version string for CoMID tags</td>
</tr>
<tr>
<td>tagged-corim-map</td>
<td>#6.501(corim-map)</td>
<td></td>
<td>A corim-map with IANA global identifier code point</td>
</tr>
<tr>
<td>tagged-int-type</td>
<td>#6.551(int)</td>
<td></td>
<td>Numeric identifier with IANA global identifier code point</td>
</tr>
<tr>
<td>tagged-min-svn</td>
<td>#6.553(min-svn)</td>
<td></td>
<td>An IANA global content tag that identifies a security version number that is evaluated with greater than or equals semantics</td>
</tr>
<tr>
<td>tagged-oid-type</td>
<td>#6.111(oid-type)</td>
<td></td>
<td>OID with IANA global code point</td>
</tr>
<tr>
<td>tagged-svn</td>
<td>#6.552(svn)</td>
<td></td>
<td>An IANA global content tag that identifies a security version number that is evaluated with equivalence semantics</td>
</tr>
<tr>
<td>tagged-euid-type</td>
<td>#6.550(euid-type)</td>
<td></td>
<td>UUID with IANA global identifier code point</td>
</tr>
<tr>
<td>tagged-uuid-type</td>
<td>#6.37(uuid-type)</td>
<td></td>
<td>UUID with IANA global identifier code point</td>
</tr>
<tr>
<td>tagged-xcorim-map</td>
<td>#6.526(xcorim-map)</td>
<td></td>
<td>A xcorim-map with IANA global identifier code point</td>
</tr>
<tr>
<td>time-type</td>
<td>time</td>
<td></td>
<td>Coordinated Universal Time</td>
</tr>
<tr>
<td>uid-type</td>
<td>uuid</td>
<td></td>
<td>Unique Enough ID, see [34]</td>
</tr>
<tr>
<td>uri-type</td>
<td>URI</td>
<td></td>
<td>Universal Resource Identifier, see [12]</td>
</tr>
<tr>
<td>uuid-type</td>
<td>bytes .size 16</td>
<td></td>
<td>Universally Unique ID, see [35]</td>
</tr>
<tr>
<td>vendor-type</td>
<td>text</td>
<td></td>
<td>Vendor name or identifier</td>
</tr>
<tr>
<td>version-type</td>
<td>text .default ‘0.0.0’</td>
<td></td>
<td>A version string that MAY follow a version string encoding scheme (see coswid.$version-scheme).</td>
</tr>
</tbody>
</table>

Table 32: CoRIM and CoMID type definitions.
5.5.4 A Schema for CoSWID Tags

Start of informative comment

This section describes normative requirements and extensions to CoSWID that are unique to CoRIM. The Reader is directed to existing specifications that describe normative SWID Tag [11] and CoSWID [7] syntax and semantics. Additionally, Figure 20 contains a simplified UML description of CoSWID.

End of informative comment

5.5.4.1 concise-swid-tag

The CoSWID tag structure is described in Table 33.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>coswid.tag-id</td>
<td>$tag-id-type-choice</td>
<td></td>
<td></td>
<td>A value that identifies a tag.</td>
</tr>
<tr>
<td></td>
<td>coswid.software-name</td>
<td>text</td>
<td></td>
<td></td>
<td>A value that identifies a software component.</td>
</tr>
<tr>
<td></td>
<td>coswid.software-version</td>
<td>text</td>
<td></td>
<td></td>
<td>A version string that identifies the software version.</td>
</tr>
<tr>
<td></td>
<td>coswid.version-scheme</td>
<td>$version-scheme</td>
<td></td>
<td></td>
<td>A version string formatting convention.</td>
</tr>
<tr>
<td></td>
<td>coswid.software-meta</td>
<td>software-meta-entry</td>
<td></td>
<td></td>
<td>A map containing additional information about the software tag.</td>
</tr>
</tbody>
</table>

Table 33: Format of concise-swid-tag CoSWID structure.

5.5.4.2 software-meta-entry

The software meta map structure is described in Table 34.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>coswid.persistent-id</td>
<td>text</td>
<td>T</td>
<td></td>
<td>MUST be a statistically unique identifier that identifies the software.</td>
</tr>
<tr>
<td></td>
<td>coswid.product</td>
<td>text</td>
<td>T</td>
<td></td>
<td>A software component product name.</td>
</tr>
</tbody>
</table>

Table 34: Format of software-meta-entry CoSWID structure.

5.5.4.3 entity-entry

The CoSWID entity name structure is described in Table 35.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>coswid.entity-name</td>
<td>text</td>
<td></td>
<td></td>
<td>The name of an organization that performs the role as defined in $role-type-choice.</td>
</tr>
<tr>
<td></td>
<td>coswid.role</td>
<td>$role-type-choice</td>
<td>+</td>
<td></td>
<td>The name of an entity role. The entity that creates the concise-swid-tag SHOULD include a role value of coswid.tag-creator.</td>
</tr>
</tbody>
</table>

Table 35: Format of entity-entry CoSWID structure.

5.5.4.4 link-entry

The CoSWID link tag relationship type structure is described in Table 36.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>coswid.rel</td>
<td>$rel</td>
<td></td>
<td></td>
<td>Linked tag relationship types for both CoSWID-to-CoSWID and CoSWID-to-CoMID linked tags.</td>
</tr>
</tbody>
</table>

Table 36: Format of link-entry CoSWID structure.

5.5.4.5 Tag Identification

The coswid.tag-id field in concise-swid-tag uniquely identifies a tag. The tag-id SHOULD be statistically unique. The corim.signer is the entity that vouches for the process that produces a unique coswid.tag-id value.

CoSWID concise-swid-tag.tag-id SHOULD use the same tag-id format as is used by CoMID concise-mid-tag-map.tag-identity-map.tag-id.
5.5.4.6 CoSWID Linked Tags Relationships

Start of informative comment

CoSWID tags can link to other CoSWID tags or to CoMID tags. The linking relationships differ for each. CoSWID to CoSWID linked tags relationships are defined using the link-entry map. CoSWID to CoMID linked tags relationships use the CoSWID relationships in the $rel type choice.

End of informative comment

5.5.4.7 CoSWID Claims

The concise-swid-tag map contains Endorsements and Evidence. This specification defines Endorsements only. Therefore, only payload-entry maps are recognized by this schema.

CoSWID payloads (payload-entry map) are normally interpreted as Reference Value Claims.

CoSWID payloads might contain additional information that does not have matching Reference Value Claims. This information is interpreted as Endorsed Value Claims. Endorsed Value Claims are accepted as definitive because of the Verifier’s trust in the Endorser. Nevertheless, Verifiers MUST establish that Endorsed Value Claims pertain to the Attester. Typically, this is accomplished by validating Evidence according to the Reference Value Claims, then processing the Endorsed Value Claims.

Evidence encoding formats need to align with Reference Claims formats to ensure proper verification. CoSWID can be used as an Evidence encoding format when software about which the Verifier requires Evidence is installed on the Attester.

A CoSWID evidence-entry map MUST NOT be used to describe CoRIM Claims.

A CoSWID payload-entry map SHOULD be used to construct Endorsements Claims and MAY contain both Endorsed and Reference Values Claims.

CoMID links to CoSWID tags MUST use a link relationship of comid.supplements.

5.5.4.8 CoRIM Locator using CoSWID

Start of informative comment

The CoSWID link-entry map contains URI references to providers of CoRIM and possibly other manifests. The link-entry map contains an href to the entity supplying manifests for linked tags. The href identifies both the manifest provider and the tag-id of the linked tag. The href requestor verifies that the tag-id in the returned manifest matches the tag-id in the request. A tag thumbprint digest contained in the link-entry map may be used to verify that the expected tag contents have not been manipulated.

A Verifier uses CoRIM to reconstruct the composition of a multi-tag device. Attesters can provide URI references to CoRIM providers for use by Verifiers to aid in the reconstruction.

End of informative comment

Manifests containing linked CoSWID tags SHOULD have a corim-locator-map that contains a corim.href to a provider of the manifest containing the linked CoSWID tag.

Manifests containing CoSWID tags with linked CoMID tags SHOULD have a link-entry map that contains a corim.href to a provider of the manifest containing the linked CoMID tag. The link-entry relationship ($rel) SHOULD be supplemental.

A link-entry map SHOULD include a thumbprint digest of the linked tag contents.
5.6 Xcorim Data Model for Deny Lists

Start of informative comment

This section contains normative requirements for a CoRIM deny list called Xcorim. A UML description is found in Figure 19 and Figure 21. If a CoRIM manifest is determined to be invalid and the validity-entry has not yet expired, a deny list can be used to invalidate that CoRIM manifest. An Xcorim deny list consists of the entity that originally issued the manifest and a list of CoRIM identifiers (e.g., corim.id) to be invalidated.

Typically, the same entity that issues the CoRIM will issue the Xcorim.

The integrity of the deny list is established via digital signature.

A service entity maintains the set of deny lists for use by the Verifier community.

Once a deny-id is placed on a deny list and distributed, the denial is considered permanent.

End of informative comment

5.6.1 Xcorim Signature and Map Structures

5.6.1.1 Xcorim

The top-level Xcorim structure is described in Table 37.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>xcorim</td>
<td></td>
<td></td>
<td>The top-level Xcorim structure that is either a signed-xcorim or an xcorim-map</td>
</tr>
</tbody>
</table>

Table 37: Format of a corim revocation structure.

5.6.1.2 signed-xcorim

The signed Xcorim structure is described in Table 38.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>xcorim.cose-signature</td>
<td></td>
<td></td>
<td>IANA defines a global content tag for COSE-Sign1-xcorim</td>
</tr>
</tbody>
</table>

Table 38: Format of signed-xcorim Xcorim structure.

5.6.1.3 COSE-Sign1-xcorim

The Xcorim signature structure is described in Table 39.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td></td>
<td>protected</td>
<td></td>
<td></td>
<td>A CBOR encoded COSE header that is protected by the COSE signature</td>
</tr>
<tr>
<td>N/A</td>
<td></td>
<td>unprotected</td>
<td></td>
<td></td>
<td>A COSE header that is not protected by the COSE signature</td>
</tr>
<tr>
<td>N/A</td>
<td></td>
<td>payload</td>
<td></td>
<td></td>
<td>A CBOR encoded tagged Xcorim</td>
</tr>
<tr>
<td>N/A</td>
<td></td>
<td>signature</td>
<td></td>
<td></td>
<td>A COSE signature block</td>
</tr>
</tbody>
</table>

Table 39: Format of COSE-Sign1-xcorim structure.

5.6.1.4 protected-xcorim-header-map

The Xcorim protected header structure is described in Table 40.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>xcorim.alg-id</td>
<td></td>
<td></td>
<td>signature algorithm identifier</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>xcorim.content-type</td>
<td></td>
<td></td>
<td>MIMIE content type identifier</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>xcorim.issuer-key-id</td>
<td></td>
<td></td>
<td>Issuer key identifier</td>
</tr>
</tbody>
</table>
5.6.1.5 unprotected-xcorim-header-map
The Xcorim unprotected header structure is described in Table 41.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(int)</td>
<td>cose-label {text}</td>
<td>cose-values {any}</td>
<td>*</td>
<td>Additional information contained in the COSE signature block</td>
<td></td>
</tr>
</tbody>
</table>

Table 41: Format of unprotected-xcorim-header-map structure.

5.6.1.6 xcorim-meta-map
The Xcorim meta map structure is described in Table 42.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>xcorim.signer</td>
<td>xcorim-signer-map</td>
<td>T</td>
<td>+</td>
<td>The entity that signs the Xcorim. The entity name SHOULD reference the issuer's entity name as contained in the issuer's signing credential, e.g. the subjectName, subjectAltName, or SubjectPublicKeyInfo field of a X.509 certificate. The entity that issues the signed-xcorim MUST be identified by xcorim.signer-name in xcorim-signer-map.</td>
</tr>
<tr>
<td>1</td>
<td>xcorim.timestamp</td>
<td>corim.time-type</td>
<td></td>
<td></td>
<td>The date and time the deny list was issued.</td>
</tr>
</tbody>
</table>

Table 42: Format of xcorim-meta-map structure.

5.6.1.7 xcorim-map
The Xcorim entity map structure is described in Table 43.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>xcorim.entity</td>
<td>xcorim-entity-map</td>
<td>T</td>
<td>+</td>
<td>The entity that constructs the unsigned Xcorim. The entity in xcorim-entity-map MUST have a type of xcorim.deny-list-creator for CoRIM revocation.</td>
</tr>
<tr>
<td>1</td>
<td>xcorim.deny-list</td>
<td>$corim-id-type-choice</td>
<td>+</td>
<td>An array of $corim-id-type-choice values.</td>
<td></td>
</tr>
<tr>
<td>(int)</td>
<td>(text)</td>
<td>xcorim.deny-list-map-extension</td>
<td>*</td>
<td>Extension point for future expansion</td>
<td></td>
</tr>
</tbody>
</table>

Table 43: Format of xcorim-map structure.

5.6.1.8 xcorim-entity-map
The Xcorim entity name structure is described in Table 44.

<table>
<thead>
<tr>
<th>NK</th>
<th>SK</th>
<th>Value Type</th>
<th>Opt</th>
<th>Mul</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>xcorim.entity-name</td>
<td>$entity-name-type-choice</td>
<td></td>
<td></td>
<td>The name of an organization that performs the role defined in $corim-role-type-choice.</td>
</tr>
<tr>
<td>1</td>
<td>xcorim.reg-id</td>
<td>uri-type</td>
<td>T</td>
<td></td>
<td>The registration entity that manages the namespace for xcorim.entity-name.</td>
</tr>
<tr>
<td>2</td>
<td>xcorim.role</td>
<td>$xcorim-role-type-choice</td>
<td></td>
<td></td>
<td>The Xcorim entity role</td>
</tr>
<tr>
<td>(int)</td>
<td>(text)</td>
<td>$xcorim-entity-map-extension</td>
<td>*</td>
<td>Extension point for future expansion</td>
<td></td>
</tr>
</tbody>
</table>

Table 44: Format of xcorim-entity-map structure.

5.6.1.9 xcorim-signer-map
The Xcorim signer structure is described in Table 45.
5.6.2 Xcorim Data Types

This section contains Xcorim type definitions in Table 46. These types are not already defined by CoRIM type definitions (see Table 32). See also Figure 19.

End of informative comment

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Type Value</th>
<th>Choice</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$corim-revocation-type-choice</td>
<td>tagged-xcorim-map</td>
<td>C</td>
<td>An IANA global content tag identifying either: xcorim-map, or signed-xcorim</td>
</tr>
<tr>
<td>$entity-name-type-choice</td>
<td>text</td>
<td>C</td>
<td>Entities, such as supply chain entities, that de-assert the trustworthiness of claims contained in Xcorim.</td>
</tr>
<tr>
<td>$xcorim-role-type-choice</td>
<td>xcorim.deny-list-creator</td>
<td>C</td>
<td>Xcorim roles</td>
</tr>
<tr>
<td>xcorim.deny-list-creator</td>
<td>1</td>
<td></td>
<td>The entity that created the xcorim.</td>
</tr>
</tbody>
</table>

Table 46: Xcorim data types.

5.6.3 Xcorim Content Type and File Extension

The content type for a CoRIM deny list SHALL be "application/xrim+cbor".

The filename extension for a file containing a CoRIM deny list SHALL be ".xcorim".

5.7 Certificate Considerations

The creator of CoRIM and Xcorim documents may use X.509v3 certificates to certify the signing key. Certificates are another form of Endorsements message that can be conveyed from an Endorser to a Verifier. An Endorser might issue manifests and device certificates. Manifests and device identities may be signed by an Endorser with a certificate. The certificate path may also be included in the Endorsements. A Verifier might trust certificates based on a trust anchor policy that recognizes the root CA as trusted. The Verifier needs to be able to determine which certificate is authorized for signing Endorsement Manifests vs. other types of data.

The X.509v3 certificate contains an extended key usage extension that indicates the key is used for signing manifests and manifest revocations (see [14]). The certificate key usage extension is expected to specify a digitalSignature usage.

Additional extended key usage extension object identifiers for manifest signing are defined below.

End of informative comment

tcg-dice-kp-manifestSign OBJECT IDENTIFIER ::= {tcg-dice-kp 13}

The tcg-dice-kp-manifestSign object identifier MUST be included as a KeyPurposeId in the Extended Key Usage extension (see [33] ) in certificates used to sign CoRIM documents.
tcg-dice-kp-manifestRevoke OBJECT IDENTIFIER ::= {tcg-dice-kp 14}

The `tcg-dice-kp-manifestRevoke` object identifier MUST be included as a KeyPurposeId in the Extended Key Usage extension in certificates used to sign Xcorim documents.
6 PROCESSING ENDORSEMENT CLAIMS

Start of informative comment

This section describes the steps needed to process Endorsement Manifest Claims. Processing is divided into five steps (See Figure 14):

1) Provisioning – Installs appropriate default and Verifier Owner authorized trust anchors used to verify Endorsement Manifests, certificates, or other information obtained from supply chain entities. Provisioning also installs other policies used during the appraisal of attestation Evidence.

2) Endorsements Discovery – Locates, downloads, and caches manifests, certificates, or other information used during Evidence appraisal.

3) Authentication – Authenticates the Attester endpoint which may include establishing a communication channel and a security context. Typically, authentication requires use of module identity credentials such as a device identity certificate.

4) Evidence collection and appraisal – Obtain fresh attestation Evidence from the attester. The Attester may be a multi-module device and may have multiple sets of Evidence that involve repeated matching and appraisal. Evidence Claims are matched against Reference Claims and associated with Endorsed Claims that may be necessary to properly apply an Appraisal Policy for Evidence. Evidence may also contain instance data (i.e., un-authenticated unique identifiers) that could be matched with instance Claims as directed by policy.

5) Attestation results propagation (not shown in Figure 14) – Attestation results are created and distributed to interested relying parties. Relying parties show interest in Attestation Results in a variety of ways that are out of scope for this specification.

In this example, (see Figure 14) each attestation role is performed by a separate entity. Actual deployments may combine multiple roles on the same entity or distribute a role across multiple entities. Although there will likely be a variety of attestation Verifiers ranging from special purpose, embedded, and generalized, this example assumes a generalized Verifier. Verifiers might, for example, operate as a service or as an installed application.

End of informative comment
6.1 Verifier Provisioning

Start of informative comment

This section describes steps (1), (2), and (3) in Figure 14.

A generalized Verifier can interoperate with many supply chain Endorsers by being configured or provisioned with multiple trust anchors as needed to reflect supply chain dynamics. Endorsement Manifests, manifest deny lists, certificates or other information are presumed to be digitally signed using asymmetric keys that are either certified by public key infrastructure with certificates that chain to one of the configured trust anchors or have individual signing keys that anchor signature verification operations. Manifest deny lists should be checked in addition to checking manifest signatures.

Verifiers may benefit from other policies that facilitate Evidence appraisal. Endorsements are assertions from supply chain entities that vouch for the integrity of modules as manufactured, tested and supplied. However, a Verifier Owner could apply admission control, asset tracking, auditing or other management function that affects appraisal results. This specification doesn’t define an Appraisal Policy for Evidence structure or semantics. This specification presumes that an Appraisal Policy for Evidence is provisioned at the start of Endorsements and Evidence evaluation: therefore, the potential impact of updated policy is not considered. The provisioning step is performed by the Verifier Owner, and in Figure 14 the Verifier Owner is represented as the Actor marked ‘Owner’.

The supply chain ecosystem may be multi-faceted and dynamic such that multiple manifests and certificates are needed to bring together a complete picture of the device or platform. As such, supply chain services support generalized verifiers. Supply chain services can facilitate discovery, delivery and caching of manifests and certificates. Generalized Verifiers may need the provisioning or configuration of support services as a prerequisite to appraisal. This specification does not define standardized supply chain support services.
Supply chain Endorsers might support attestation throughout a device’s lifecycle resulting in changes to the manifest or certificates after original manufacture. Additionally, manifest and certificate creation and issuance during manufacturing may occur at different times and in different locations making it challenging to coordinate a monolithic approach to creation and issuance.

End of informative comment

6.2 Attester Authentication

Start of informative comment

This section describes steps (4) and (5) in Figure 14.

The Verifier authenticates the Attester and may establish a secure channel to ensure attestation Evidence comes from the Attester. The Attester may consist of multiple components that each might consist of multiple Attesting and/or Target Environments. The device authentication credentials will have layer or device scope, while a composition of modules may have multiple Attesting Environments, each with distinct attestation keys. Consequently, the Verifier expects the device authentication credentials to be securely bound to the attestation credentials. In a DICE layering endpoint, at least one of the layers contains a device authentication credential. The other layers may contain additional attestation credentials. DICE layers are cryptographically related using Compound Device Identifier (CDI) values that are layer specific. In a composite device model, there may be multiple Target Environments tied to a common Attesting Environment. Each Target Environment might report distinct Evidence protected using a common attestation key or might use a different attestation key for each Target Environment.

Attester policy determines Attester endpoint composition and key hierarchy. The Verifier depends on the Endorsement manifest for endpoint composition details that it uses to validate composition details supplied with attestation Evidence. In some cases, attestation Evidence may include endpoint composition details that might assist the Verifier in obtaining the correct Endorsement manifest and tags. However, all information obtained from the Attester is by default presumed to be suspicious until proven otherwise.

Device authentication credentials may contain device information that helps narrow the field of possible related Endorsement manifests, tags and policies. A Verifier might realize scalability improvements by focusing its search for related attestation Endorsements to the current attesting device.

End of informative comment

6.3 Resolution of Linked Tags

Start of informative comment

This section describes step (6) in Figure 14.

Attester devices might require multiple Endorsement tags to comprehensively describe endpoint composition. An Endorsement tag can describe a top-level device that corresponds to the device authentication credential. The device credential can be correlated with the device Endorsement tag using the following common techniques:

- Include device class identifiers in the device ID credential, such as vendor, model, and class-id that are already included in the top-level endorsement tag.
- Include a reference to the top-level Endorsement tag in the device ID credential.
- Include a reference to the device ID credential in the top-level Endorsement tag.
- Include a device identity Claim in the top-level Endorsement tag.
- Include Endorsements in the linked tag that matches a reference measurement that is the expected source of Claims in the linked tag.

Multi-module device compositions rely on multiple tags each containing module specific Claims. Different types of Claims may pertain to a common module, for example, different supply chain Endorsers may mint separate tags
for reference, endorsed, instance and identity Claims. The union of these represents the complete set of Claims about the module.

Multiple Endorsers could cooperate to assert multiple Claims of the same type. For example, two suppliers may develop reference Claims derived from a common bill of materials. The ‘supplements’ link relationship can be used to model both situations.

Multi-module device compositions rely on linked tags to model device composition involving either singleton or multiple Endorsers.

Device lifecycle considerations could result in tags that either update, patch or replace existing tags.

The above considerations may require resolving linked tags in order to identify the most relevant Claims when matched with Evidence and when applying appraisal policies.

End of informative comment

6.3.1 Algorithm for Linked Tag Resolution

Start of informative comment

This section describes a possible algorithm for resolving linked tags that results in an accepted set of Claims for a given device or component. Claims are “accepted” in that they are received by the Verifier and determined to be relevant to the Appraisal process. The set of accepted Claims is a subset of the set of all Claims known to the Verifier. The set of all known Claims is referred to as the CLAIMSET. The algorithm could be applied once for a large set of possible compositions, or it could be integrated with Evidence matching to better optimize algorithm performance.

**Step 0:**

The algorithm presumes the top-level Endorsement tag TCURR has been successfully identified and placed into the CLAIMSET. If TCURR has no linked tags entries then processing stops, otherwise the next unprocessed tag is selected. The selected tag is named TLINK. TCURR has a link relationship with TLINK that determines how TCURR processes TLINK.

**Step 1:**

*IF link relationship is ‘replaces’ AND*

\[
\text{TCURR.tag-id} = \text{TLINK.tag-id AND TCURR.tag-version} > \text{TLINK.tag-version}
\]

THEN

TCURR replaces TLINK in CLAIMSET

TLINK may be audited

TLINK may be deleted

**Step 2:**

*IF link relationship is supplements AND TLINK.tag-id} != TCURR.tag-id*

THEN

Add TLINK to CLAIMSET

End of informative comment
6.4 Evidence Processing

Start of informative comment

This section describes steps (7), (8), (9), and (10) in Figure 14. Section

Section 6.4.1 contains an algorithm for matching a set of related modules and their associated Evidence Claims, i.e., the CLAIMSET. CoMID Claims can describe compositional relationships that are security relevant, for example a device with DICE layering may have an upper layer that depends on a lower layer. Evidence could describe a particular layering, but the Endorser might describe an expected layering. A Verifier might rely on Endorsements for composition detail rather than inferring composition through Evidence as a strategy for avoiding replacement attacks.

The Verifier starts with a set of accepted Claims that are initialized from Evidence, referred to as Accepted CLAIMS. Accepted Claims are a subset of the full Claim set and include Reference and Endorsed Values, and device identity Claims that the Verifier expects to find in Evidence from the Attester. The Verifier processes one or more tags from the Claim set that might match the Evidence. For each tag, the Verifier compares Reference Claims in CLAIMSET against Evidence in Accepted CLAIMS. If all Reference Claims match Accepted CLAIMS, then the Evidence matches, and all values in the CLAIMSET are added to the Accepted CLAIMS.

Evidence is obtained from an authenticated Attester and evaluated to find matching endorsed Claims contained in the CLAIMSET. Accepted CLAIMS are appraised according to an Appraisal Policy for Evidence. Other Claims, such as Endorsed Claims, are included in the Accepted CLAIMS if they share a common claims-map subject.

The Accepted CLAIMS are appraised by an Appraisal Policy for Evidence to determine Attestation Results. For example, if an Appraisal Policy for Evidence expects a set of Claims that is not a subset of the Accepted CLAIMS, then the Attestation Results might be the empty set.

Compare operations are applied at the attribute level for each Claim (e.g., ‘vendor’, ‘class-id’, ‘raw-value’ etc...). The endianness for CoRIM data is big endian. The endianness for attribute data is expected to be big endian unless endianness is specified by a related standard. CBOR encoded data is either big endian or does not have endianness constraints.

End of informative comment

6.4.1 Evidence Matching Algorithm

Start of informative comment

This section contains an algorithm (See also Figure 15) for matching Claims contained in Evidence to Endorsed Claims contained in Accepted CLAIMS. Evidence is expected to be cryptographically bound to the currently authenticated Attester. A cryptographic binding could be in the form of a digital signature by an attestation key that is certified by the device identity key.

Step 0:

The CLAIMSET is a collection of structures (consisting of environment-measurement associations).

There may be four maps within CLAIMSET that correspond to the four types of Claims triples (reference, endorsed, identity, and attestation key).

The environment fields that name a module class include vendor, class-id, model, layer, and index.

Note that class-id in CoMID class-map is matched with type in the tcbinfo Evidence extension in [14]. If class-id is a $class-id-type-choice of the form tagged-uuid-type or tagged-oid-type then the
IANA global code point is not included. If the Evidence creator uses both forms such that there is a possibility of namespace collisions, the Vendor or model fields can be used to avoid namespace collisions.

If mkey is used, then match mkey in Evidence with mkey in Reference Values.

The measurement fields include digest, raw-value, version, svn, flags and other measurements.

The raw-value in CoMID class-map is synonymous with vendorinfo in the tcbinfo Evidence extension in [14].

If the measurement fields contain instance measurements, the environment fields may be described by an instance identifier uuid or ueid.

For uuid or ueid in CoMID $instance-id-type-choice, the type field in the tcbinfo Evidence extension contains the UUID. Otherwise, ueid is contained in the TcbUeid Evidence extension. When comparing Evidence with CoMID, the IANA global code point is ignored.

The ACCEPTED_CLAIMS is initialized to empty.

This algorithm assumes the Verifier has a method for walking the Evidence and correlating it with the CLAIMSET.

The Claims in Evidence are called $CEV$.

The Claims in CLAIMSET may contain reference Claims called $CREF$, endorsed Claims called $CEND$, and identity Claims called $CID$.

**Step 1:**

Check that, for a given Claim $CX$ in Evidence $CEV$, there isn't another Claim $CY$ in Evidence with the same environment but different measurements, otherwise fail.

If evidence contains an element key then check that the comid.mkey in evidence has the same value as the corresponding comid.mkey in reference values (endorsements).

**Step 2:**

The Verifier has Evidence that it has obtained from the Attester device. Each piece of Evidence $CEV$ contains a collection of Claims.

For each Claim $CX$ in $CREF$ and $CY$ in $CEV$:

If the environment of $CY$ matches any $CX$ environment in $CREF$ then assign $CY$ to $CZ$.

If there is no matching Claim ($CZ$) then add $CY$ to PENDING_CLAIMS and get the next $CZ$.

If $CZ$ has a layer value that is greater than the layer value for any already ACCEPTED_CLAIMS, and it has the same environment (i.e., class-map or instance-id) and, if used, the same element (i.e., comid.mkey), and different or down-revision measurements (i.e., measurement-values-map), then fail Claims verification.

Compare the measurement values of $CX$ and $CZ$.

If $CX$ contains raw-value and $CZ$ contains raw-value with a different binary value after applying the raw-value-mask then fail Claims verification. If the raw-value-mask length is different from the raw-value length, then fail Claims verification.

If $CX$ contains digest and $CZ$ contains digest with a different binary value, then fail Claims verification. Note that there should be at least one intersecting algorithm for digests. For every intersecting algorithm the digests must be the same.
If Cx contains version and Cz contains version with a different or down-revision binary value, then fail Claims verification.

If Cx contains svn and Cz contains svn with a different integer value, then fail Claims verification. Note that svn refers to a tagged-svn that is stripped of the IANA global tag.

If Cx contains min-svn and Cz contains svn that is less than min-svn then fail Claims verification. Note that min-svn refers to a tagged-min-svn that is stripped of the IANA global tag.

If Cx contains flags and Cz contains flags with a different binary value, then fail Claims verification. When ref doesn’t contain flags then Evidence must not contain flags.

Etc…. continue checking for other values in Cx and Cz.

If all value fields match as described above, then add Cx Claim to the ACCEPTED_CLAIMS.

Note that the current attesting device may have Evidence Claims that were not anticipated by the Reference Claims. If the device is found to be trustworthy based on the Reference Claims, then it can assert additional Claims that are immediately accepted as definitive. These Claims are held in PENDING_CLAIMS until the device is found to be trustworthy.

Step 3:
If the currently attesting device has associated endorsed Claims.

For each Claim Cx in CEND and Cy in ACCEPTED_CLAIMS:
If the Cx environment matches the Cy environment, then add Cx to ACCEPTED_CLAIMS

Step 4:
If the current attesting device has identity Claims.

For each Claim Cx in CID and Cy in ACCEPTED_CLAIMS:
If the Cx environment instance id matches the Cy environment instance id then add Cx to ACCEPTED_CLAIMS

Step 5:
Apply the Appraisal Policy for Evidence to the ACCEPTED_CLAIMS

Step 6:
If the ACCEPTED_CLAIMS describes a valid trusted Attester, then apply the Appraisal Policy for Evidence to the PENDING_CLAIMS

Produce Attestation Results.

End of informative comment
Figure 15: Evidence matching algorithm flowchart
APPENDIX A – ATTESTATION EXAMPLE

The attestation example (See Figure 16) shows an Attester consisting of four environments: a Root of Trust; an Attesting Environment, and two Target Environments. The Root of Trust environment collects Claims about environment E1, constructs Evidence for E1, creates an attestation key for E1 and issues a certificate with E1 Evidence and attestation public key E1. The Attesting Environment E1 collects Claims for two Target Environments E2 and E3 and creates Evidence for both. Evidence might take the form of a signed document or an attestation protocol payload.

The Attester conveys the certificate E1 and Evidence E2 and E3 to a Verifier for appraisal.

There are two Endorsers which issue manifests. Manifest M0 contains endorsed value Claims that describe the Root of Trust environment and a certificate that certifies the Root of Trust attestation key. Manifest M1 contains reference Claims for the Attesting Environment E1. Manifest M2 contains reference Claims for the Target Environments E2 and E3. The Endorsers deposit their manifests containing the various Claim sets in a repository for easy access by Verifiers.

The Verifier is configured to discover manifests, or the Attester could provide a pointer to a different repository. The Verifier inspects the Evidence and Claims for information about which manifests are needed. For example, the E1 certificate might reference the RoT certificate or an environment class identifier from one of the Claim sets could be used to formulate a query to the repository. The manifest M0 might have a locator for M1 and M1 might have a locator for M2.

Once the Verifier has the manifests, it constructs a working set of Claims, (i.e., RoT Endorsed Value Claims, E1, E2, E3 Reference Claims), for the related (linked) tags. To construct a Claim set, a Verifier might need to request Claims from multiple sources. The schema provides a mechanism to find related Claims via linked tags (see Section 5.3.1). The working set (i.e., CLAIMSET) determines whether the measurements from each source have corresponding (matching) reference measurements.

The Verifier policy determines whether to trust the Endorser keys that signed the manifests and issued the Root of Trust certificate. If the Root of Trust certificate is trusted and the certificate path associated with the signed Evidence is verified, the Root of Trust endorsed Claims are appraised using Verifier policy. If the Root of Trust and Attester attestation keys are valid and all the Evidence is accepted by the reference Claims, the Verifier generates an Attestation Result.
Figure 16: Attestation example
APPENDIX B – ENDORSMENT MANIFEST EXAMPLE

The example CoRIM in ‘CBOR diagnostic format’ depicts the example from Appendix A where ACME creates a manifest containing endorsed value Claims about the ACME RoadRunner Root of Trust and Reference Values Claims for the ACME RoadRunner Firmware, and two instances of the ABC Trusted OS. The single unsigned manifest containing a single CoMID tag with all four Claim sets is presumed to be issued by ACME Inc.

The manifest is represented in CBOR diagnostic format that is JSON-like with the exception that the tag name and tag numeric are provided to the left of the colon that separates tag information from data values. Tag name values are separated by beginning and ending slash characters (e.g., /name/).

Example CoRIM in CBOR Diagnostic format:

```
/ corim / 500(
  / corim-map / 501({
    / corim.id / 0 : h'284e6c3e5d9f4f6b851f5a4247f243a7',
    / corim.tags / 1 :
      / concise-mid-tag / 506( <<
        / concise-mid-tag / {
          / comid.tag-identity / 1 : {
            / comid.tag-id / 0 : h'3f06af63a93c11e49779700505690773f'
          },
          / comid.entities / 2 : {
            / comid.entity-name / 0 : "ACME Inc.",
            / comid.reg-id / 1 : 32("https://acme.example"),
            / comid.role / 2 : 0 / tag-creator /
          },
          / comid.triples / 4 : {
            / comid.reference-triples / 0 : [
              / environment-map / {
                / comid.class / 0 : {
                  / comid.class-id / 0 :
                    / tagged-uuid-type / 37({
                      h'a71b3e388d454a0581f352e58c832c5c'
                    })
                  / comid.vendor / 1 : "ACME Inc.",
                  / comid.model / 2 : "ACME RoadRunner Firmware",
                  / comid.layer / 3 : 1
                },
                / measurement-map / {
                  / comid.mval / 1 : {
                    / comid.digests / 2 : [
                      / hash-alg-id / 1, / sha256 /
                      / hash-value / h'bb71198ed6095dc3c619e555c2c0b8d7564a38031b034a195892591c65365b0'
                    ]
                  }
                }
              },
              / environment-map / {
                / comid.class / 0 : {
                  / comid.class-id / 0 :
                    / tagged-uuid-type / 37({
                      h'a71b3e388d454a0581f352e58c832c5c'
                    })
                  / comid.vendor / 1 : "ABC Inc.",
                  / comid.model / 2 : "ABC Trusted OS",
                  / comid.layer / 3 : 2,
                  / comid.index / 4 : 0
                },
                / measurement-map / {
                  / comid.mval / 1 : {
                    / comid.digests / 2 : [
                      / hash-alg-id / 1, / sha256 /
                      / hash-value / h'bb71198ed6095dc3c619e555c2c0b8d7564a38031b034a195892591c65365b0'
                    ]
                  }
                }
              }
            ]
          }
        })
      })
  })
)```
Example CoRIM in ASCII CBOR format:

d9 1f4
d9 1f5
a2
00
50
2846e35d9f4f6851e4247f243a7
01
d9 1fa
a3
01
a1
00
50
3f06af3a93c11e49797b0505690773f
# 3f06af3a93c11e49797b0505690773f
# unsigned(2)

# tag(500)
# tag(501)
# map(2)
# unsigned(0)
# bytes(16)
# "\x06\xAF\x11\x04\x97\x00\x90\x73"
# unsigned(1)
# tag(506)
# map(3)
# unsigned(1)
# map(1)
# unsigned(0)
# bytes(16)
# 02
a3                                   # map(3)
00                                # unsigned(0)
69                                # text(9)
41434d4520496e632e             # "ACME Inc."
01                                # unsigned(1)
d8 20                             # text(20)
74                                # text(20)
6874747073a2f2f61636d6578616d706c65
# "https://acme.example"
02                                # unsigned(2)
00                                # unsigned(0)
04                                # unsigned(4)
a2                                   # map(2)
00                                # unsigned(0)
83                                # array(3)
82                             # array(2)
a1                          # map(1)
00                                # unsigned(0)
a2                          # map(4)
00                                # unsigned(0)
d8 25                             # tag(37)
50                                # bytes(16)
67b28b6c34cc40a19117ab505911e37
# "g\xB2\x8B\x14\xCC\xA1\xA9\xA7\xA5\x05\x91\x1E7"
01                                # unsigned(1)
69                                # text(9)
41434d4520496e632e             # "ACME Inc."
02                                # unsigned(2)
78 18                             # text(24)
41434d520526f616452756e6572046669726d77617265
# "ACME RoadRunner Firmware"
03                                # unsigned(3)
01                                # unsigned(1)
a1                          # map(1)
01                                # unsigned(1)
a1                          # map(1)
02                                # unsigned(2)
82                             # array(2)
a1                          # map(1)
00                                # unsigned(0)
a5                          # map(5)
00                                # unsigned(0)
d8 25                             # tag(37)
50                                # bytes(16)
44a336af4cb14a879432e53dd6571c7fa9bcfafb75f4882592626ea3a4d91b
# "D\xAD3j\xF4\xCB\x14\xA8\xC7\xF7\xA9\xB7\x82\x8F\x83\x87"
82                             # array(2)
a1                          # map(1)
00                                # unsigned(0)
a5                          # map(5)
00                                # unsigned(0)
d8 25                             # tag(37)
50                                # bytes(16)
a71b3e388d45a0581f352e58c832c5c
# "\xA7\xe8\x8DE\xA0\xF3R\xA9\xF8\xA3,\x83,\"
01                                # unsigned(1)
6a                                # text(10)
57594c494520496e632e
# "ABC Inc."
02                                # unsigned(2)
77                                # text(23)
57594c494520436f79664652054727573746564204f53
# "ABC Trusted OS"
03                                # unsigned(3)
02                                # unsigned(2)
04                                # unsigned(4)
00                                # unsigned(0)
a1                          # map(1)
01                                # unsigned(1)
a1                          # map(1)
02                                # unsigned(2)
82                             # array(2)
01                                # unsigned(1)
58 20                             # bytes(32)
bb71198ed60a95dc3619e555c2c0b8d7564a38031b034a195892591c65365b0
# "\xBBq\x19\x8E\xD6\n\x95\xDC<\xa\x9EU\\\",\v\x8Dud\xA3\x801\xB04\xA1\x95\x89%\x91\xC6Se\xB0"
82                             # array(2)
  a1                          # map(1)
    00                       # unsigned(0)
    a5                       # map(5)
    00                       # unsigned(0)
    d8 25                    # tag(37)
    50                       # bytes(16)
      a71b3e388d45a0581f352e58c832c5c
a7                             # "\xBBq\x19\x8E\xD6\n\x95\xDC<\xa\x9EU\\\",\v\x8Dud\xA3\x801\xB04\xA1\x95\x89%\x91\xC6Se\xB0"
82                             # array(2)
  a1                          # map(1)
    00                       # unsigned(0)
    a5                       # map(5)
    00                       # unsigned(0)
    d8 25                    # tag(37)
    50                       # bytes(16)
      a71b3e388d45a0581f352e58c832c5c
a7                             # "\xBBq\x19\x8E\xD6\n\x95\xDC<\xa\x9EU\\\",\v\x8Dud\xA3\x801\xB04\xA1\x95\x89%\x91\xC6Se\xB0"
82                             # array(2)
  a1                          # map(1)
    00                       # unsigned(0)
    a5                       # map(5)
    00                       # unsigned(0)
    d8 25                    # tag(37)
    50                       # bytes(16)
      a71b3e388d45a0581f352e58c832c5c
a7                             # "\xBBq\x19\x8E\xD6\n\x95\xDC<\xa\x9EU\\\",\v\x8Dud\xA3\x801\xB04\xA1\x95\x89%\x91\xC6Se\xB0"
82                             # array(2)
  a1                          # map(1)
    00                       # unsigned(0)
    a5                       # map(5)
    00                       # unsigned(0)
    d8 25                    # tag(37)
    50                       # bytes(16)
      a71b3e388d45a0581f352e58c832c5c
01                             # unsigned(1)
  6a                       # text(10)
    57594c49520496e632e
02                             # unsigned(2)
  77                       # text(23)
    57594c49520496e632e
03                             # unsigned(3)
  02                       # unsigned(2)
  04                       # unsigned(4)
  01                       # unsigned(1)
  a1                          # map(1)
    01                       # unsigned(1)
    a1                       # map(1)
    02                       # unsigned(2)
    82                       # array(2)
      01                       # unsigned(1)
      58 20                    # bytes(32)
        bb71198ed60a95dc3cc619e555c2c08d7564a3801b034a1958291c65365b0
01                             # unsigned(1)
  82                       # unsigned(2)
    a1                          # map(1)
    00                       # unsigned(0)
    a4                       # map(4)
    00                       # unsigned(0)
    d8 25                    # tag(37)
    50                       # bytes(16)
      67b28b6c34cc40a19117ab5b0391e37
01                             # unsigned(1)
  69                       # text(9)
    41434d4520496e632e     # "ACME Inc."
    02                       # unsigned(2)
    72                       # text(18)
    41434d4520526f6f74206f662054727573746564204f53
03                             # unsigned(3)
  00                      # unsigned(0)
    a1                          # map(1)
    01                       # unsigned(1)
    a1                          # map(1)
    01                       # unsigned(1)
    d9 228                    # tag(552)
    01                       # unsigned(1)
APPENDIX C – CORIM CDDL EXAMPLES

This section contains sample CDDL for selected CoRIM structures.

corim = #6.500($concise-reference-integrity-manifest-type-choice)
tagged-corim-map = #6.501(corim-map)
$concise-reference-integrity-manifest-type-choice /= tagged-corim-map
$concise-reference-integrity-manifest-type-choice /= #6.502(signed-corim)

corim-map = {
corim.id => $corim-id-type-choice
corim.tags => [ + $concise-tag-type-choice ]
? corim.dependent-rims => [ + corim-locator-map ]
? corim.profile => [ + profile-type-choice ]
? corim.rim-validity => validity-map
? corim.entities => [ + corim-entity-map ]
* $$corim-map-extension
}

profile-type-choice = uri / tagged-oid-type

corim-locator-map = {
corim.href => uri
? corim.thumbprint => hash-entry
}

$concise-tag-type-choice /= #6.505(bytes .cbor concise-swid-tag)
$concise-tag-type-choice /= #6.506(bytes .cbor concise-mid-tag)

corim-entity-map = {
corim.entity-name => $entity-name-type-choice
? corim.reg-id => uri
corim.role => $corim-role-type-choice
* $$corim-entity-map-extension
}

$corim-role-type-choice /= corim.manifest-creator

signed-corim = #6.18(COSE-Sign1-corim)

protected-corim-header-map = {
corim.alg-id => int
corim.content-type => "application/rim+cbor"
corim.issuer-key-id => bstr
corim.meta => bstr .cbor corim-meta-map
* cose-label => cose-values
}

corim-meta-map = {
corim.signer => corim-signer-map
? corim.signature-validity => validity-map
}

corim-signer-map = {

}
corim.signer-name => $entity-name-type-choice
? corim.signer-uri => uri
* $$corim-signer-map-extension
}

validity-map = {
? corim.not-before => time
corim.not-after => time
}

unprotected-corim-header-map = {
* cose-label => cose-values
}

COSE-Sign1-corim = [
  protected: bstr .cbor protected-corim-header-map
  unprotected: unprotected-corim-header-map
  payload: bstr .cbor tagged-corim-map
  signature: bstr
]

concise-mid-tag = {
? comid.language => language-type
comid.tag-identity => tag-identity-map
? comid.entities => [ + entity-map ]
? comid.linked-tags => [ + linked-tag-map ]
comid.triples => triples-map
* $$concise-mid-tag-extension
}

language-type = text

tag-identity-map = {
  comid.tag-id => $tag-id-type-choice
? comid.tag-version => tag-version-type
}

$tag-id-type-choice /= tstr
$tag-id-type-choice /= uuid-type

tag-version-type = uint .default 0

entity-map = {
  comid.entity-name => $entity-name-type-choice
? comid.reg-id => uri
comid.role => [ + $comid-role-type-choice ]
* $$entity-map-extension
}

$comid-role-type-choice /= comid.tag-creator
$comid-role-type-choice /= comid.creator
$comid-role-type-choice /= comid.maintainer

linked-tag-map = {
comid.linked-tag-id => $tag-id-type-choice
comid.tag-rel => $tag-rel-type-choice

$tag-rel-type-choice /= comid.supplements
$tag-rel-type-choice /= comid.replaces

triples-map = non-empty<{
  ? comid.reference-triples => [ + reference-triple-record ]
  ? comid.endorsed-triples => [ + endorsed-triple-record ]
  ? comid.attest-key-triples => [ + attest-key-triple-record ]
  ? comid.identity-triples => [ + identity-triple-record ]
  * $triples-map-extension
}>

; REFERENCE-VALUE triple
; "target environment $TE" "has reference measurements" "$RV"
reference-triple-record = [ environment-map ; target environment
[ + measurement-map ] ; reference measurements ]

; ENDORSED-VALUE triple
; "environment $E" "has endorsed measurements" "$EV"
endorsed-triple-record = [ environment-map ; (target or attesting) environment
[ + measurement-map ] ; endorsed measurements ]

; ATTESTATION-VERIFICATION-KEY triple
; "attesting environment $AE" "signs Evidence that can be verified with key" "$K"
attest-key-triple-record = [ environment-map ; attesting environment
[ + $crypto-key-type-choice ] ; attestation verification key(s) ]

; DEVICE-IDENTITY triple
; "device $D" "is identified by key" "$K"
identity-triple-record = [ environment-map ; device identifier (instance or class)
[ + $crypto-key-type-choice ] ; DevID, or semantically equivalent ]

; Public key in DER format base64-encoded.
$key-type-choice /= tstr

; Optional X.509 certificate chain corresponding to the public key
; in comid.key, encoded as an array of one or more base64-encoded
; DER PKIX certificates.
$cert-type-choice /= tstr

environment-map = non-empty<{
  ? comid.class => class-map
  ? comid.instance => $instance-id-type-choice
? comid.group => $group-id-type-choice
}

class-map = non-empty<{
  ? comid.class-id => $class-id-type-choice
  ? comid.vendor => tstr
  ? comid.model => tstr
  ? comid.layer => uint
  ? comid.index => uint
}

$class-id-type-choice /= tagged-oid-type
$class-id-type-choice /= tagged-uuid-type
$instance-id-type-choice /= tagged-ueid-type
$instance-id-type-choice /= tagged-uuid-type
$group-id-type-choice /= tagged-uuid-type

oid-type = bytes
tagged-oid-type = #6.111(oid-type)
;
; github.com/lucas-clemente/cbor-specs/blob/master/uuid.md
;
tagged-uuid-type = #6.37(uuid-type)
;
; From draft-ietf-rats-eat
;
ueid-type = bytes .size 33
tagged-ueid-type = #6.550(ueid-type)

$measured-element-type-choice /= tagged-oid-type
$measured-element-type-choice /= tagged-uuid-type
$measured-element-type-choice /= uint

measurement-map = {
  ? comid.mkey => $measured-element-type-choice
  comid.mval => measurement-values-map
}

measurement-values-map = non-empty<{
  ? comid.ver => version-map
  ? comid.svn => svn-type-choice
  ? comid.digests => digests-type
  ? comid.flags => flags-map
  ? raw-value-group
  ? comid.mac-addr => mac-addr-type-choice
  ? comid.ip-addr => ip-addr-type-choice
  ? comid.serial-number => serial-number-type
  ? comid.ueid => ueid-type
  ? comid.uuid => uuid-type
  ? comid.name => tstr
version-map = {
    comid.version => version-type
    ? comid.version-scheme => $version-scheme
}
version-type = text .default '0.0.0'
version-scheme is defined in CoSWID

svn-type = uint
svn = svn-type
min-svn = svn-type
tagged-svn = #6.552(svn)
tagged-min-svn = #6.553(min-svn)
svn-type-choice = tagged-svn / tagged-min-svn

flags-map = {
    ? comid.operational-flag-configured => bool
    ? comid.operational-flag-secure => bool
    ? comid.operational-flag-recovery => bool
    ? comid.operational-flag-debug => bool
    ? comid.operational-flag-replay-protected => bool
    ? comid.operational-flag-integrity-protected => bool
    * $$flags-map-extension
}

raw-value-group = {
    comid.raw-value => $raw-value-type-choice
    ? comid.raw-value-mask => raw-value-mask-type
}
$raw-value-type-choice /= #6.560(bytes)
raw-value-mask-type = bytes

ip-addr-type-choice = ip4-addr-type / ip6-addr-type
ip4-addr-type = bytes .size 4
ip6-addr-type = bytes .size 16
mac-addr-type-choice = eui48-addr-type / eui64-addr-type
eui48-addr-type = bytes .size 6
eui64-addr-type = bytes .size 8
serial-number-type = text

;  Notes:
;  - hash-entry is defined in CoSWID schema
digests-type = [ + hash-entry ]
APPENDIX D – UML DIAGRAMS

This section contains Unified Modeling Language (UML) diagrams for CoRIM, CoMID, CoSWID (simplified), and Xcorim.

**Figure 17: CoRIM data model in UML**
Figure 18: CoMID data model in UML
### Concise Reference Integrity Manifest – CoRIM Types

? means (0 or 1); *n* means (0 to n); ’+’ means (1 to n)

<table>
<thead>
<tr>
<th>$corim$-types</th>
</tr>
</thead>
<tbody>
<tr>
<td>type definitions</td>
</tr>
<tr>
<td>$class-id-type-choice : tagged-oid-type</td>
</tr>
<tr>
<td>$class-id-type-choice : tagged-uuid-type</td>
</tr>
<tr>
<td>$class-id-type-choice : tagged-int-type</td>
</tr>
<tr>
<td>$command-module-role-type-choice : comando-tag-creator 1</td>
</tr>
<tr>
<td>$command-module-role-type-choice : comando.creator 2</td>
</tr>
<tr>
<td>$command-module-role-type-choice : comando.miscellaneous 3</td>
</tr>
<tr>
<td>$command.version.scheme-type-choice : $coswld.version.scheme</td>
</tr>
<tr>
<td>$concrete-reference-integrity-manifest-type-choice : $6.501(coswld-map)</td>
</tr>
<tr>
<td>$concrete-reference-integrity-manifest-type-choice : $6.502(xcorim)</td>
</tr>
<tr>
<td>$concrete-tag-type-choice : corim.concrete-mid-tag / corim.concrete-swld-tag</td>
</tr>
<tr>
<td>corim.concrete-mid-tag : $6.506 (concrete-mid-tag)</td>
</tr>
<tr>
<td>corim.concrete-swld-tag : $6.505 (concrete-swld-tag)</td>
</tr>
<tr>
<td>$corim-id-type-choice : text / uuid-type</td>
</tr>
<tr>
<td>$corim-revocation-type-choice : $6.526(xcorim-map)</td>
</tr>
<tr>
<td>$corim-revocation-type-choice : $6.527(xcorim)</td>
</tr>
<tr>
<td>$corim-role-type-choice : corim.manifest-creator 1</td>
</tr>
<tr>
<td>$entity-type-name-type-choice : text</td>
</tr>
<tr>
<td>$group-id-type-choice : tagged-uuid-type</td>
</tr>
<tr>
<td>$index-type : integer</td>
</tr>
<tr>
<td>$instance-id-type-choice : tagged-uuid-type / tagged-uuid-type</td>
</tr>
<tr>
<td>$ip4-addr-type-choice : ip4-addr-type / ip6-addr-type</td>
</tr>
<tr>
<td>$ip4-addr-type : bytes.size 4</td>
</tr>
<tr>
<td>$ip6-addr-type : bytes.size 6</td>
</tr>
<tr>
<td>$layer-type : integer</td>
</tr>
<tr>
<td>$mac48-addr-type-choice : mac48-addr-type / mac64-addr-type</td>
</tr>
<tr>
<td>$mac48-addr-type : bytes.size 6</td>
</tr>
<tr>
<td>$mac64-addr-type : bytes.size 8</td>
</tr>
<tr>
<td>$min-svn : integer</td>
</tr>
<tr>
<td>$model-type : text</td>
</tr>
<tr>
<td>$oid-type : bstr</td>
</tr>
<tr>
<td>$crypto-key-type-choice : tagged-pki-base64-key-type : $6.554(ustr)</td>
</tr>
<tr>
<td>$crypto-key-type-choice : tagged-pki-base64-cert-type : $6.555(ustr)</td>
</tr>
<tr>
<td>$crypto-key-type-choice : tagged-pki-base64-cert-path-type : $6.556(ustr)</td>
</tr>
<tr>
<td>$profile-type-choice : uri-type / tagged-oid-type</td>
</tr>
<tr>
<td>$raw-value-type-choice : raw-value-type</td>
</tr>
<tr>
<td>$raw-value-type : #6.560(ustr)</td>
</tr>
<tr>
<td>$raw-value-mask : bytes</td>
</tr>
<tr>
<td>$serial-number-type : text</td>
</tr>
<tr>
<td>$svn : integer</td>
</tr>
<tr>
<td>$svn-type-choice : tagged-svn / tagged-min-svn</td>
</tr>
<tr>
<td>$tag-id-type-choice : text / uuid-type</td>
</tr>
<tr>
<td>$tag-rel-type-choice : comando-supplements 0</td>
</tr>
<tr>
<td>$tag-rel-type-choice : comando-replaces 1</td>
</tr>
<tr>
<td>$tag-version-type : uint.default 0</td>
</tr>
<tr>
<td>$tagged-int-type : #6.551(integer)</td>
</tr>
<tr>
<td>$tagged-min-svn : #6.553(min-svn)</td>
</tr>
<tr>
<td>$tagged-oid-type : #6.111(oid-type)</td>
</tr>
<tr>
<td>$tagged-svn : #6.552(svn)</td>
</tr>
<tr>
<td>$tagged-uuid-type : #6.553(ustr)</td>
</tr>
<tr>
<td>$time-type : UTC-time</td>
</tr>
<tr>
<td>$uuid-type : bytes.size 16</td>
</tr>
<tr>
<td>$uri-type : text</td>
</tr>
<tr>
<td>$vendor-type : text</td>
</tr>
<tr>
<td>$version-type : text.default 0.0.0</td>
</tr>
<tr>
<td>$xcorim-role-type-choice : xcorim.deny-list-creator 1</td>
</tr>
</tbody>
</table>

---

**Figure 19:** CoRIM and Xcorim data types in UML.
This diagram may require magnification with high resolution display.
CoRIM Deny List

'?' means (0 or 1); '*' means (0 to n); '+' means (1 to n)

Figure 21: Xcorim - a CoRIM deny list in UML.